M.A.R.S. : un modèle opérationnel de conception de simulations pédagogiques

Jean-Philippe Pernin

THESE

présentée par

Jean-Philippe PERNIN

pour obtenir le titre de

Docteur de l'Université Joseph Fourier - Grenoble I
(arrêté ministériel du 5 Juillet 1984 et du 30 Mars 1992)
Spécialité : INFORMATIQUE

M.A.R.S.
Un modèle opérationnel de conception de simulations pédagogiques

Date de soutenance : 22 Janvier 1996

Composition du Jury :

Président :	Joëlle COUTAZ
Rapporteurs :	Jean-Claude BOUSSARD
	Bertrand T. DAVID
Examinateurs :	Eddy FORTE
	Jean-Pierre GIRAUDIN
Directeurs :	Viviane GUERAUD
	Jean-Pierre PEYRIN

THESE PREPAREE
AU SEIN DU LABORATOIRE CLIPS - IMAG
A L'UNIVERSITE JOSEPH FOURIER - GRENOBLE I

Je tiens à remercier :

Mme *Joëlle Coutaz,* Professeur à l'Université Joseph Fourier, pour m'avoir fait l'honneur de présider le jury de cette thèse ;

MM. *Jean-Claude Boussard,* Professeur à l'Université de Nice - Sophia Antipolis, et *Bertrand T. David,* Professeur à l'Ecole Centrale de Lyon, pour avoir accepté de juger ce travail et pour l'intérêt qu'ils lui ont accordé ;

M. *Eddy Forte,* et M. *Jean-Pierre Giraudin,* pour avoir accepté de participer à ce jury ;

Jean-Pierre Peyrin, Professeur à l'Université Joseph Fourier, et *Viviane Guéraud,* Maître de Conférences à l'Université Stendhal, pour avoir assuré ensemble la direction de cette thèse. Je remercie Jean-Pierre pour sa grande ouverture d'esprit qui m'a permis, il y a déjà longtemps, de reprendre des études, et pour la confiance qu'il m'a toujours accordée. Je remercie Viviane pour tout l'intérêt qu'elle a porté à mon travail, pour ses critiques minitieuses et pour l'énergie qu'elle a déployée pour mener cette co-direction parallèlement à ses autres activités.

Cette thèse a été réalisée dans le cadre du projet MELISA, projet de partenariat entre le laboratoire CLIPS-IMAG et la division T.P.E.C de la société Hewlett-Packard. Je tiens particulièrement à remercier *Franck Bugnet*, pour son assistance et pour les idées qu'il m'a apportées.

Je tiens également à remercier tous les membres de l'équipe ARCADE qui ont contribué à l'aboutissement de ce travail, et en particulier :

Jean-Michel Cagnat, pour la justesse de ses critiques et pour toute l'aide qu'il m'a apportée ;

Jean-Pierre David, avec qui ont été entreprises les réflexions qui ont abouti à cette thèse ;

Frédérique Coudret, pour l'aide qu'elle m'a apportée dans les tâches de réalisation ;

Jean-Michel Adam, Sophie Bordon, Carole Campani, Gloria Cortès, Agnès Front, Frédéric Heurtaux et *Pascal Stiévenard*, qui tous ont été impliqués dans le projet MELISA ;

Je tiens enfin et surtout à remercier ma compagne *Violaine,* et mes filles *Marion* et *Camille,* qui, depuis de nombreuses années, m'ont toujours encouragé et soutenu dans mes projets quelque peu farfelus...

TABLE DES MATIERES

PARTIE 2
LE PROCESSUS DE DEVELOPPEMENT EN MILIEU INDUSTRIEL

PARTIE 3

LE MODELE DE CONCEPTION .

PARTIE 4
MISE EN OEUVRE

INTRODUCTION

Nos travaux de recherche se situent dans le domaine des "Environnements Interactifs d'Apprentissage par Ordinateur".

Notre équipe de recherche s'est en premier lieu intéressée, à travers le laboratoire Arcade [CAG 90], à promouvoir un certain type de logiciel éducatif, basé sur l'expérimentation, la découverte, le jeu. Les résultats probants acquis à travers cette expérience [GUE 91], nous ont conduits à proposer de généraliser cette approche et de l'intégrer à la pratique quotidienne de l'enseignement. Nous voulons donner aux enseignants les moyens d'adapter, d'enrichir voire même de créer des applications basées sur la simulation, afin qu'ils puissent disposer de ressources complémentaires à leur enseignement traditionnel [GUE 93].

Cette volonté nous a conduits à réfléchir au mode de production de ce type de logiciels [BUR 93, DAV 94]. Pour que leur utilisation soit effective, ces applications doivent être produites ou modifiées rapidement, sûrement et à moindre coût. Les environnements de production existants ne permettent pas de répondre à ce besoin. Il faut donc améliorer les méthodes et les outils de production pour atteindre cet objectif [PER 95a, PER 95b].

Cette thèse se propose donc d'aborder la définition de nouveaux environnements de production d'un point de vue pragmatique. Plutôt que de raisonner de façon générale, nous avons préféré partir de cas concrets rencontrés dans la formation professionnelle technique où les besoins peuvent être clairement énoncés, où l'apport de la simulation peut être quantifié et où les impératifs économiques nécessitent d'étudier précisément les conditions de production des logiciels.

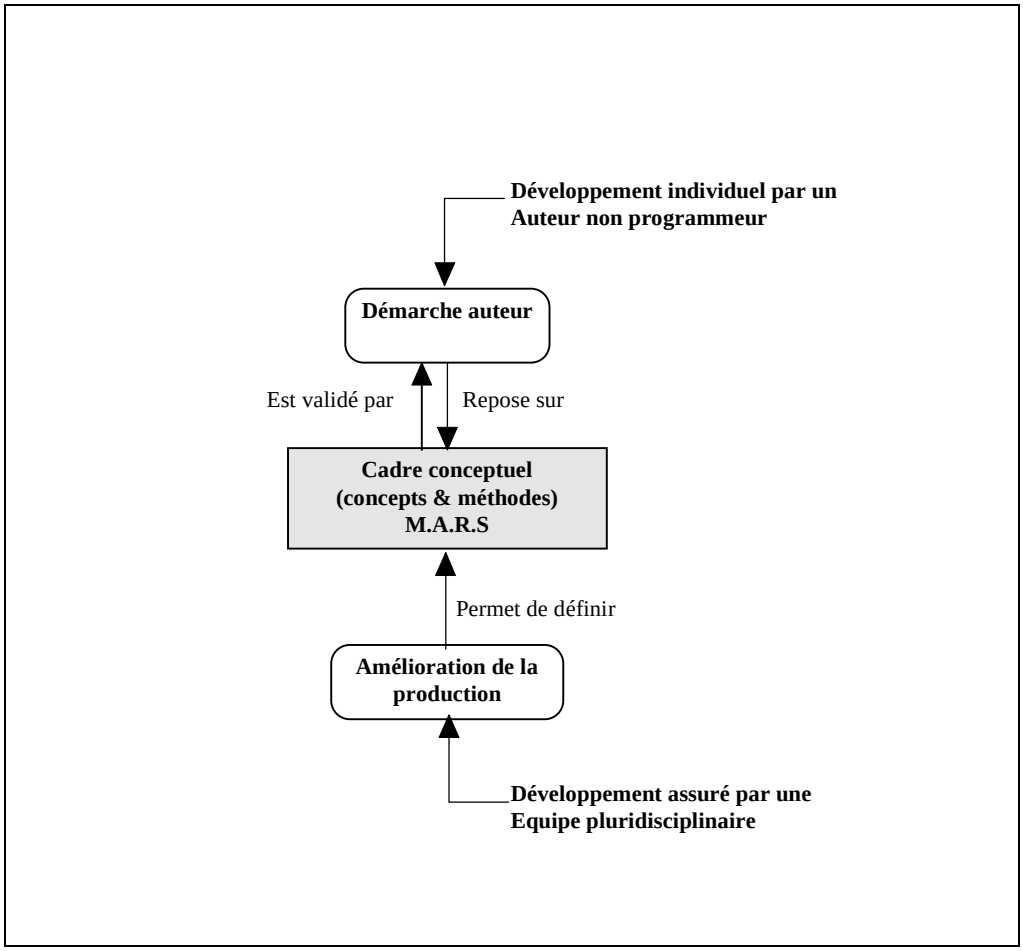
Nous avons pour cela étudié deux types de contextes industriels différents.

Le premier, représenté par la société CORYS, s'intéresse à la production de logiciels de haute qualité avec des équipes de développement regroupant des compétences très pointues et très différentes. Il s'agit, dans ce cas, de proposer un *nouveau cycle de développement* et de nouvelles méthodes permettant d'atteindre les objectifs de rapidité de développement et de rentabilisation des investissements.

Le second, représenté par la société Hewlett-Packard, s'intéresse au développement de simulations pédagogiques, moins ambitieuses que les précédentes mais dont la production est assurée par des équipes très restreintes, voire composées d'une seule personne. Bien évidemment, les différentes compétences requises pour ces "auteurs" ne peuvent être aussi pointues que celles de spécialistes ; en particulier, s'ils possèdent la maîtrise de l'outil informatique, ils ne sont souvent pas spécialistes en programmation. Il se pose alors, dans ce cas, le problème de fournir à ces auteurs un *environnement intégré* les assistant dans leur tâche de conception et de réalisation.

Nous proposons dans cette thèse un *modèle de conception de simulations pédagogiques* : le modèle M.A.R.S. Comme le montre la figure ci-dessous, la définition de ce modèle doit être issue d'efforts conjoints :

- La formalisation des concepts et méthodes permettant d'améliorer, dans un contexte de production industriel, le développement de simulations pédagogiques.
- L'utilisation du cadre conceptuel ainsi défini, comme base de la démarche à utiliser par un auteur dans un contexte de développement individuel.
- La confrontation des résultats obtenus à travers l'expérimentation.



Le modèle MARS, carrefour de deux approches

Le modèle proposé devra être *opérationnel*. Nous devrons donc nous poser la question de savoir si sa mise en œuvre avec l'aide d'un outil informatisé est aisée, et si l'utilisateur peut réellement maîtriser les formalismes utilisés.

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

La Partie 1 s'intéresse à décrire précisément le contexte de notre recherche, à en fixer les enjeux et les objectifs.

- Le chapitre 1 décrit les enjeux économiques liés à l'introduction de la simulation pédagogique dans le milieu industriel. Il décrit en détail les points de vue émis dans deux contextes industriels différents et s'attache à en dégager les points communs.
- Le chapitre 2 étudie le concept de simulation pédagogique de façon générale et tente de caractériser précisément le type de simulation pédagogique que nous voulons développer.
- Le chapitre 3 aborde les problèmes liés à la production de ce style de logiciel, en évaluant les différentes approches en fonction des objectifs que nous poursuivons.

La Partie 2 étudie un développement concret en milieu industriel, et fournit des réponses aux problèmes rencontrés.

- Le chapitre 4 étudie le processus de développement de simulation pédagogique à travers l'exemple de la production d'un logiciel que nous avons développé conjointement avec la société CORYS.
- Le chapitre 5 fournit des réponses aux problèmes spécifiques rencontrés dans la production du logiciel décrit au chapitre 4. En particulier, il propose un nouveau type de cycle de développement et définit un type d'environnement de production adapté.

La Partie 3 est consacrée à la définition précise de notre proposition, le modèle de conception M.A.R.S.

- Le chapitre 6 présente de façon globale le modèle M.A.R.S.
- Les chapitres 7 à 10 présentent de façon détaillée chacun des espaces de travail proposés dans le modèle M.A.R.S : Modèle, Scénario, Représentation et enfin Association.

La Partie 4 décrit les réalisations logicielles qui ont été effectuées dans le cadre de cette thèse.

- Le chapitre 11 présente l'environnement auteur MELISA développé sur la base du modèle M.A.R.S pour les besoins propres de la société Hewlett-Packard.

Enfin, dans la conclusion, nous énonçons les perspectives qui ont pu être ouvertes par ce travail de recherche.

PARTIE 1

CONSTATS, ENJEUX ET OBJECTIFS

Cette première partie est consacrée à la description précise de notre contexte de recherche.

Le chapitre 1 décrit les enjeux économiques liés à l'introduction de la simulation pédagogique dans le milieu industriel. Il décrit en détail les points de vue émis dans deux contextes industriels différents et s'attache à en dégager les points communs.

Le chapitre 2 étudie le concept de simulation pédagogique de façon générale et tente de caractériser précisément le type de simulation pédagogique que nous voulons développer.

Le chapitre 3 aborde les problèmes liés à la production de ce style de logiciel, en évaluant les différentes approches en fonction des objectifs que nous poursuivons.

1. LES ENJEUX ECONOMIQUES DANS LE MILIEU INDUSTRIEL

Aujourd'hui, l'une des raisons de la faible diffusion de l'EAO en milieu industriel est de nature économique. De façon générale, on peut considérer que les sommes investies dans les logiciels de formation sont trop élevées par rapport aux gains obtenus. Cette absence de rentabilité s'explique à notre avis par les facteurs suivants :

- *la surévaluation des possibilités de la formation par ordinateur*. Nombre d'industriels, préoccupés par leurs problèmes de formation interne, croient qu'il est possible de remplacer de façon massive les formations dispensées de façon traditionnelle par des logiciels d'apprentissage sophistiqués. Entretenus dans l'idée qu'avec les progrès incessants de l'informatique, tout devient possible, ils espèrent arriver à ce but en investissant d'importantes sommes. Le résultat obtenu est souvent largement décevant autant pour des raisons psychologiques (acceptation difficile de la formation par les apprenants) que par l'impossibilité de mesurer de façon objective le niveau des compétences atteint.
- *l'inadéquation des méthodes et la mauvaise intégration des outils existant sur le marché*. Il existe actuellement une grande variété de systèmes, langages ou environnements logiciels permettant chacun de couvrir correctement l'un des aspects du processus de développement (modélisation du domaine, modélisation pédagogique, conception d'interface, etc.). Par contre, l'intégration des résultats obtenus pose le plus souvent des problèmes importants liés à leur compatibilité. Ces problèmes peuvent être résolus par la réécriture de code ou par le choix de

solutions logicielles moins performantes.

- *le manque de maîtrise des coûts de développement*. Le développement d'un logiciel sophistiqué de formation implique le travail en coopération d'un nombre important d'intervenants de cultures différentes (experts du domaine, pédagogues, ergonomes, infographistes, spécialistes des interfaces, développeurs informatiques, etc.). Cette spécificité entraîne une gestion complexe de ce type de projet et peut aboutir à des dépassements très importants des budgets prévus.

Ces raisons ont récemment amené des entreprises à se pencher sur le problème de la rentabilité de telles applications. Notre équipe a en particulier travaillé avec deux partenaires industriels ayant des points de vue bien différents mais se recoupant sur de nombreux aspects. Nous présentons dans les paragraphes suivants chacun de ces points de vue.

1.1 LE POINT DE VUE D'UNE SOCIETE DE DEVELOPPEMENT DE LOGICIELS DE SIMULATION PEDAGOGIQUE

1.1.1 Contexte

La société CORYS est spécialisée dans la conception de systèmes de simulation adaptés aux métiers de l'énergie et du transport (par exemple, simulateurs de conduite de train, de centrale thermique ou nucléaire). Elle a, par la suite, diversifié ses activités dans le domaine de la formation aux métiers de l'énergie, pensant qu'elle pouvait proposer une démarche originale basée sur son expérience dans le domaine de la simulation. Son objectif est d'une part, de proposer un catalogue de logiciels de simulation pédagogique, et d'autre part, de réaliser à la demande des logiciels sur mesure. Chaque logiciel, conçu comme un complément à une formation dispensée de façon traditionnelle, est consacré à la formation sur un sujet spécifique (par exemple : les principes de fonctionnement d'un alternateur) et se présente sous la forme d'une base de connaissances et d'un ensemble d'exercices basés sur la simulation, classés selon un ordre de difficulté croissante. Les logiciels sont prévus pour être utilisés sous le contrôle de formateurs dans le cadre de sessions de plusieurs jours.

La nécessité de satisfaire les clients impose un très haut niveau de qualité des logiciels réalisés, tant pour l'expertise du domaine, que pour la pédagogie ou la présentation interactive.

1.1.2 Problématique

Le problème de la société est ici essentiellement lié au coût de revient des logiciels à produire. Les études de marché montrent qu'il existe bien une demande

pour ce genre de logiciels, sous réserve d'un prix relativement modique. Le développement des premières maquettes a démontré que la rentabilité de cette activité ne serait atteinte que par une modification importante des habitudes de production. Les principaux points-clés évoqués sont les suivants :

- *Rapidité de développement* : Des logiciels spécifiques peuvent être développés sur commande. La satisfaction du client impose des temps de développement très courts (ordre de grandeur : 2 à 3 mois). Cette contrainte impose une organisation spécifique : coopération étroite entre les divers intervenants, parallélisation de certaines tâches, réutilisation de composants déjà écrits, etc.
- *Contexte coopératif* : La coopération forcément très étroite entre certains acteurs (responsable de projet, expert du domaine, expert pédagogue) implique que chaque intervenant puisse comprendre de façon non ambiguë les spécifications des autres. Il en ressort donc une exigence de formalisation valable pour tous.
- *Formalisation précise des besoins* : Le besoin de formalisation ne doit pas être contradictoire avec la rapidité de développement. De façon idéale, l'application de simulation pédagogique devrait pouvoir être générée automatiquement à partir des documents de spécifications.
- *Démarche de type prototypage* : La plate-forme logicielle utilisée doit être assez souple pour permettre une démarche de type prototypage. Le résultat de l'application, même partiel, doit être très tôt visible afin d'en vérifier la validité.

1.1.3 Solutions envisagées par la société

Les solutions envisagées dans ce contexte se situent à deux niveaux. Le premier niveau consiste à introduire, dans le cadre actuel, un ensemble de méthodes et procédures permettant d'améliorer sensiblement la rentabilité. Le deuxième consiste à développer un environnement intégré de production basé sur les méthodes et procédures décrites au premier niveau.

- *Introduction de méthodes et procédures*. La première étape est la définition d'un processus de développement adapté précisant les rôles, tâches et responsabilités de chaque intervenant, la nature des informations échangées entre les intervenants, et les étapes à respecter. Sur la base de ce processus de développement, doit être précisé un ensemble de formalismes permettant de décrire de façon non ambiguë les besoins de chacun, la nature et la forme des différentes coopérations, etc. Les formalismes seront supportés dans un premier temps par une documentation papier. Cette introduction devrait améliorer la rapidité de développement et réduire le coût général du logiciel.
- *Développement d'un environnement intégré spécialisé dans la production de simulations pédagogiques*. Ce développement pourra être envisagé sur la base des

résultats obtenus dans la phase précédente. Il devra proposer une plate-forme logicielle intégrée offrant :

⇒ des outils spécialisés proposant à chacun des intervenants un espace de travail au sein duquel il pourra exprimer ses besoins selon des formalismes qui lui sont propres. En outre, ces outils devront permettre la validation des spécifications énoncées.

⇒ des mécanismes de génération automatique de code à partir des spécifications obtenues.

⇒ un mécanisme permettant la mise en correspondance des composants, obtenus de façon indépendante, en vue de leur intégration dans l'application finale.

En résumé, les solutions retenues par cette société relèvent d'une **démarche de génie logiciel** visant à rentabiliser les développements en exploitant au mieux les compétences techniques dont elle dispose.

1.2 LE POINT DE VUE D'UN GRAND CONSTRUCTEUR DE SYSTEMES INFORMATIQUES POUR SES BESOINS DE FORMATION INTERNE

1.2.1 Contexte

Le centre T.P.E.C. (Technical Planning and Education Center) situé à l'Isle d'Abeau en France est le principal centre européen de formation interne de l'entreprise Hewlett-Packard. Sa vocation est de:

- assurer la formation "centralisée" des personnels techniques de la compagnie. Le centre de l'Isle d'Abeau peut accueillir chaque semaine jusqu'à 500 stagiaires. Les cours, encadrés par des formateurs devant des classes allant de 16 à 24 personnes, portent sur des sujets techniques liés aux matériels et logiciels développés ou maintenus par Hewlett-Packard. Ces formations proposent de façon alternée, des cours traditionnels, et des séances de travaux pratiques ("labs") consacrés aux manipulations des équipements réels.
- mener une réflexion sur les "nouvelles technologies éducatives" et proposer de nouveaux types de formation adaptés aux besoins spécifiques de l’entreprise. Le centre de l'Isle d'Abeau dispose ainsi d'un studio de télévision permettant de réaliser en direct une émission "interactive" sur un sujet technique (par exemple, la commercialisation d'un nouveau produit). Cette émission est diffusée mondialement vers des centres de vidéoconférence situés dans les différents ensembles de production, devant un public qui pourra compter au même moment jusqu'à 500 participants. Cet exemple spectaculaire n'est pas unique et les préoccupations de la société s'orientent aujourd'hui principalement vers la décentralisation de la

formation.

- être un centre de conseil et d'expertise en matière d'Enseignement Assisté par Ordinateur auprès des autres divisions de l'entreprise. Hewlett-Packard dispose d'une structure fortement décentralisée où chaque "division" de production est responsable d'une gamme de produits matériels ou logiciels. A chaque fois qu'un nouveau produit va être commercialisé, le responsable de produit (Product Engineer) doit assurer la création de la formation correspondante. Il a le choix du ou des supports qu'il juge les mieux adaptés à cette formation : manuel papier, cours traditionnel, présentation interactive, simulation pédagogique, etc. Le produit ne pourra être commercialisé que si la formation est validée, c'est-à-dire si elle obéit aux critères de qualité et de complétude préconisés par la compagnie.

1.2.2 Problématique

Hewlett-Packard est confronté aux problèmes de formation suivants :

- *Mauvaise adéquation dans le temps de la formation centralisée*. Une importante part des formations s'adresse à un public constitué de techniciens de maintenance. Ces techniciens doivent être capables d'intervenir chez les clients sur un large éventail de matériels. Généralement, les formations correspondant à l'étude de chaque matériel sont organisées de façon centralisée durant des sessions planifiées à l'avance. L'organisation est soumise à des contraintes telles que : disponibilité d'un formateur, nombre suffisant de candidats pouvant se libérer de leurs tâches habituelles pendant une période bloquée, éloignement dans le temps entre deux sessions traitant du même thème, etc. Il arrive donc fréquemment que, dans une situation d'intervention de maintenance sur un matériel ou logiciel déterminé chez un client, la formation correspondante soit trop ancienne et donc inefficace.
- *Difficulté de transmettre certains savoir-faire*. Comme dans toute entreprise industrielle, les personnels chargés de l'installation, la configuration ou la maintenance des produits, doivent acquérir, pour être opérationnels, un certain nombre de savoir-faire, de type procédural (exemple: installer une machine) ou analytique (exemple: diagnostiquer les causes de dysfonctionnement d'une machine avant de la réparer). La transmission de ces savoir-faire est une composante importante de la formation chez Hewlett-Packard, et peut être abordée traditionnellement de diverses façons :

  ⇒ *formation théorique*. Des documents peuvent être consacrés à l'étude de ces problèmes (manuels d'installation, de maintenance, d'analyse de pannes, etc.). Ces documents sont généralement complexes à écrire, car ils cherchent par nature à être complets, c'est-à-dire à envisager de façon exhaustive les situations pouvant être rencontrées sur le terrain. Leur lecture ou consultation

en devient souvent malaisée.

⇒ *formation pratique*. Un proverbe hindou dit *"j'entends, j'oublie ? je vois, je me souviens; je fais, je comprends"*. Il est évident que pour former un technicien à la résolution des problèmes qu'il rencontrera dans la réalité, la manipulation directe de l'équipement réel est souhaitable. Mais elle présente plusieurs difficultés :

∗ *La disponibilité des équipements réels*. Il peut être très difficile, voire techniquement impossible de disposer des équipements réels pour des besoins de formation. Les raisons peuvent en être le coût (doit-on acquérir un super-calculateur pour pouvoir former des techniciens de maintenance ?), la faisabilité (comment réunir sur un même site des points de connexion d'un réseau sensés être distants de plusieurs milliers de kilomètres ?) ou la disponibilité d'un matériel qui n'est pas encore en production. Et de plus, pour des raisons d'efficacité pédagogique, il serait préférable d'avoir un équipement par étudiant.

∗ *La difficulté à reproduire des phénomènes réels*. Il peut être nécessaire de provoquer volontairement des anomalies ou des dysfonctionnements, pour placer le technicien dans des situations qu'il rencontrera dans la réalité. Cela doit-il amener à disposer de configurations réelles différentes avec toutes les combinaisons possibles de composants défectueux ? Il est évident qu'une telle solution n'est pas sérieusement envisageable, autant pour des raisons de coût (faut-il casser des machines pour savoir les réparer ?) que pour des raisons d'explosion combinatoire des cas à prendre en considération.

- *Difficulté d'évaluation des compétences effectives*. Un autre problème soulevé par TPEC est celui de l'évaluation effective des compétences des stagiaires à l'issue de leur formation. Cette évaluation est faite traditionnellement à l'aide de questionnaires visant à vérifier leurs connaissances et aptitudes. A l'expérience, on s'aperçoit qu'il existe une réelle distorsion entre les capacités et compétences supposées d'un technicien, par exemple, et son comportement effectif sur le terrain.

1.2.3 La simulation comme solution pédagogique

Par rapport aux points qui viennent d'être évoqués, TPEC pense que le développement d'applications de simulations pédagogiques peut apporter des solutions aux problèmes suivants :

- *Mauvaise adéquation dans le temps de la formation centralisée*. La solution est ici de privilégier la notion de formation décentralisée, c'est-à-dire de rendre disponible la formation auprès de l'utilisateur et au moment où il le juge nécessaire. En se basant sur l'infrastructure de réseaux existant au sein de l'entreprise, il est relativement aisé de prévoir la création de catalogues d'applications pédagogiques

spécialisées (basées ou non sur la simulation).

- *Difficulté à disposer des équipements réels*. Le concept même de simulation offre une approche particulièrement bien adaptée. Là où, pour diverses raisons, il n'est pas possible de disposer d'un matériel, la présence d'une application informatique simulant tout ou partie des fonctionnalités du produit permettra à son utilisateur d'en comprendre le fonctionnement, et ce, de façon active, par les manipulations qu'il opérera sur la simulation.
- *Difficulté à reproduire des phénomènes réels*. Dans ce cas également, la simulation prend tout son sens. Là où, par exemple, la reproduction de pannes sur un équipement nécessitait la dégradation volontaire d'un ou plusieurs composants, la simulation permettrait d'envisager un très grand nombre de combinaisons avec en plus la possibilité d'ajouter un aspect aléatoire.
- *Difficulté d'évaluation des compétences effectives*. Enfin pour ce dernier point, le recours à une simulation complétée par un contrôle du comportement de l'apprenant, permet d'évaluer de façon beaucoup plus réaliste ses compétences effectives qu'un simple questionnaire, diminuant ainsi les probabilités d'erreurs lors de ses interventions ultérieures sur le terrain.

Le recours à la simulation semble donc globalement positif, mais il se pose le problème de la mise en œuvre de cette solution dans le contexte de Hewlett-Packard. Rappelons une des contraintes essentielles énoncées plus haut, à savoir que la responsabilité du développement de ces applications est du ressort du responsable de produit (Product Engineer). Cet ingénieur possède une connaissance approfondie du produit, et des compétences pédagogiques lui permettant d'élaborer les supports de formation correspondants. Par contre, ni lui-même, ni les membres de son équipe, ne sont sensés être des spécialistes en développement informatique (maîtrise d'un langage de programmation, par exemple).

Ce contexte particulier impose que le développement d'applications pédagogiques ne pourra être réalisé qu'à l'aide de plates-formes logicielles ne nécessitant pas de compétences spécifiques en programmation. Ceci écarte donc le recours aux langages informatiques traditionnels et entraîne l'utilisation d'un environnement "auteur". Nous entendons ici par environnement auteur, une plate-forme logicielle, homogène ou éventuellement constituée de composants pouvant être liés entre eux, obéissant aux critères suivants :

- *Rapidité de développement*. Cette contrainte est essentielle dans le contexte de Hewlett-Packard. La formation relative à un produit doit être délivrée dans un temps réduit compris entre la décision de commercialiser un nouveau produit et sa mise sur le marché effective. Ce laps de temps est en moyenne de 3 mois.

- *Aptitude à créer des simulations "simples" à vocation pédagogique*. Etant données les contraintes de temps énoncées ci-dessus d'une part, et la volonté pédagogique de proposer des simulations basées sur le principe de résolution de problèmes d'autre part, il apparaît préférable de développer des simulations partielles, mais simples, dont l'objectif pédagogique est précis, plutôt que de simuler le plus complètement possible un système, sans se soucier de l'exploitation pédagogique ultérieure de cette simulation. L'environnement auteur retenu devra donc autoriser la création rapide de telles applications.
- *Aptitude à spécifier le contrôle pédagogique à effectuer sur les simulations produites*. Comme il est indiqué plus haut, l'objectif n'est pas de développer des simulations en tant que telles, mais de proposer un contrôle pédagogique de ces simulations. L'environnement proposé devra donc permettre de spécifier ce contrôle et de générer l'application résultante.
- *Aptitude au prototypage et à la réutilisation*. Les contraintes de temps énoncées plus haut imposent que les résultats du travail des auteurs soient visibles en permanence et que cette visibilité permette les modifications rapides des différents composants. Un corollaire à la rapidité de création ou de modification est la necessité de permettre à l'auteur de réutiliser ou mettre en catalogue des composants déjà écrits.
- *Aptitude à créer des applications réalistes*. La majeure partie des applications de simulations pédagogiques de Hewlett-Packard concernent la manipulation de composants matériels (calculateurs, appareils électroniques, connectique) ou logiciels (systèmes d'exploitation, logiciels applicatifs). Cette particularité conduit à insister sur le réalisme des simulations créées, afin de réduire au minimum la différence entre le système réel et le système simulé. Ceci implique que l'environnement utilisé permette d'intégrer facilement toutes les ressources du multimédia, c'est-à-dire le texte, le son, les graphiques, les images numériques (photographies notamment) ou la vidéo.
- *Simplicité et pertinence des interfaces proposées aux auteurs*. Le succès de la prise en main et de l'utilisation effective d'un environnement auteur réside essentiellement dans la simplicité de sa mise en œuvre, dans le côté intuitif de son abord. En conséquence, les interfaces proposées à l'auteur doivent le conduire à se concentrer sur la spécification de son application et non sur la mise en œuvre de cette spécification. On doit éviter au maximum à l'auteur les mécanismes de traduction entre ses différents objectifs et les formalismes utilisés par l'environnement. Ceci implique que l'environnement doit être réellement adapté aux objectifs décrits ci-dessus (spécifier des simulations, spécifier le contrôle pédagogique, créer des interfaces réalistes, réutiliser des composants, permettre une approche prototypage).

- *Sûreté des applications produites*. Un autre aspect important réside dans le fait que les applications étant produites par des non-spécialistes en programmation, la fiabilité du résultat doit être assurée. Actuellement, de nombreux systèmes-auteurs proposent, en plus de l'environnement de base, un langage de programmation (dit langage-auteur) sensé pouvoir être manipulé par des non-spécialistes en programmation en vue de développer des applications complexes. Cette approche s'avère souvent illusoire : non seulement l'écriture d'applications complexes nécessite un minimum de connaissances en programmation, mais de plus, il s'agit de langages interprétés, généralement faiblement typés, et syntaxiquement très permissifs. Les auteurs arrivent certes à écrire des programmes syntaxiquement justes, mais dont l'exécution aboutit parfois à une interruption de l'application qui peut leur paraître incompréhensible. Ce manque de "lisibilité" de l'environnement peut entraîner une perte de confiance très préjudiciable.

1.2.4 Solutions envisagées

Face aux exigences décrites ci-dessus, TPEC a envisagé plusieurs solutions :

- Utiliser des systèmes-auteurs existants. Il existe aujourd'hui un grande variété de logiciels de ce type sur le marché. Chacun des représentants les plus caractéristiques d'entre eux a été évalué par TPEC en fonction des critères énoncés précédemment (MacroMind Director™, Authorware™, IconAuthor™, Toolbook™, Visual Basic™). Aucun n'a pu être retenu, essentiellement à cause du manque de fonctionnalités préexistantes concernant la simulation et le contrôle pédagogique.
- Définir une plate-forme logicielle où cohabitent deux types d'environnement : l'un dédié à la conception de simulations et l'autre consacré à la présentation de cette simulation et à son contrôle pédagogique. Cette solution envisagée avec un générateur interactif de simulation (Rapid™ de la société Emultek) et un système auteur (Toolbook™), n'a pas été retenue à cause de la complexité des interfaces de spécification proposées, du manque de vision cohérente de l'ensemble dû à la duplication des environnements et du manque de performances dû aux communications entre logiciels.
- Développer un nouvel environnement. Cette démarche étant particulièrement coûteuse et comportant de nombreux risques, il a été décidé de proposer une solution construite sur la base d'un système auteur de création hypermédia existant. Le système Toolbook™ CBT a été retenu :

  ⇒ pour ses qualités propres concernant la création rapide d'applications interactives sophistiquées et dans une moindre mesure, sa prise en compte partielle du contrôle pédagogique.

  ⇒ pour sa capacité à être enrichi : les fonctionnalités et interfaces proposées à

l'auteur peuvent être totalement redéfinies.

⇒ pour sa compatibilité avec une démarche de réutilisation : l'orientation objets de l'environnement permet assez aisément de réutiliser des composants déjà développés.

En résumé, la solution préconisée par Hewlett-Packard est de développer un environnement permettant à des auteurs non spécialistes en programmation de créer rapidement des applications où la simulation est "réduite au contexte pédagogique pertinent", et où le contrôle pédagogique permet de fixer des objectifs à l'élève et d'évaluer son comportement. Cet environnement doit reposer sur une méthodologie implicite mais claire que l'auteur doit suivre. La réussite de ce projet dépend essentiellement de la **qualité des interfaces proposées à l'auteur** et de **la capacité du système à générer automatiquement** la majeure partie de l'application à partir des spécifications fournies par l'auteur.

Un projet de recherche et développement, établi entre TPEC et le laboratoire CLIPS-IMAG, se fixe cet objectif général. Ce projet s'intitule MELISA (**M**ethodology and **E**nvironment for developing **L**earning, **I**nstruction and **S**imulation **A**pplications). La première réalisation sera présentée au chapitre 11.

1.3 CARACTERISTIQUES COMMUNES A CES DEUX POINTS DE VUE

Nous venons dans les paragraphes précédents d'étudier deux points de vue différents, celui de CORYS et celui de Hewlett-Packard. Nous allons maintenant tenter de mettre en évidence leurs points communs et leurs différences.

1.3.1 Domaine d'application et nature des simulations

Dans les deux cas, le domaine d'application est scientifique et technique. Pour CORYS qui se consacre à la formation aux principes de base des métiers de l'énergie, l'objectif pédagogique est principalement d'ordre conceptuel. La nature des simulations à développer est le plus souvent liée à l'observation et au contrôle de phénomènes physiques continus évoluant dans le temps.

Les objectifs pédagogiques de Hewlett-Packard sont essentiellement pratiques et concernent l'acquisition de savoir-faire. La nature des simulations consiste le plus souvent à reproduire des systèmes réels (calculateurs, équipements électroniques, logiciels, systèmes connectés, ...). Dans la majorité des cas, ces systèmes peuvent être considérés comme des machines à états finis.

1.3.2 Temps de développement

Dans nos deux cas d'étude, la *réduction du temps de développement* est un besoin très important. Sa justification est bien évidemment financière (diminution des coûts de développement), mais aussi technique : le délai entre la décision de réaliser une application de simulation pédagogique et la disponibilité effective de l'application doit être très court (de quelques jours à 3 mois).

1.3.3 Taille des simulations

Les simulations qui intéressent CORYS et Hewlett-Packard ont pour but d'illustrer un propos pédagogique précis, non de reproduire dans son intégralité un phénomène ou un système existant. Etant données les contraintes de temps de développement, il sera souvent préférable de développer plusieurs simulations partielles faciles à modéliser et à exploiter d'un point de vue pédagogique. Nous retenons donc le concept de simulation partielle réduite au contexte pédagogique pertinent.

1.3.4 Qualité des interfaces proposées à l'apprenant.

Une autre caractéristique commune aux deux approches réside dans la qualité et dans la richesse des interfaces qui doivent être proposées à l'utilisateur final, c'est-à-dire à l'apprenant. Les raisons évoquées en sont :

- *l'attractivité des applications*. Les applications pédagogiques sont utilisées dans des contextes où l'apprenant se retrouve, à un moment donné, seul face à l'application. Sa motivation à utiliser le logiciel et à le manipuler est en grande partie fonction de l'aspect externe, du soin qui est apporté à la présentation graphique, du recours pertinent aux techniques multimédias.
- *le réalisme des simulations*. Dans certains cas, par exemple la manipulation d'équipements réels, une distorsion trop importante entre la présentation fournie dans la simulation et la réalité peut être une source ultérieure de confusion, voire d'erreur. Il est important de pouvoir définir une présentation réaliste notamment par le recours au son, au schéma fixe ou animé, à l'image photographique ou à la vidéo et d'offrir un éventail riche d'interactions à l'utilisateur.

1.3.5 Qualité des interfaces proposées aux auteurs.

Les interfaces proposées, basées sur des formalismes maîtrisés par chaque type d'auteur, doivent être pertinentes et suffisamment simples pour permettre à chaque intervenant de formuler ses spécifications sans être gêné par une mise en œuvre lourde ou une traduction de formalismes (par exemple recours à un langage de programmation).

1.3.6 Démarche de type prototypage - Réutilisation

Dans les deux cas étudiés, il ressort le besoin de pouvoir disposer en permanence d'un résultat visible, même partiel, de l'application en cours de développement. Ceci semble nécessaire si l'on veut éviter d'importants retours en arrière durant le développement de l'application. De telles remises en question peuvent s'avérer catastrophiques étant données les contraintes de temps. D'autre part, il est également important de pouvoir rassurer l'auteur en l'informant de l'avancement de son travail, surtout s'il n'est pas familier avec les méthodes classiques de développement informatique.

Un autre moyen de permettre un gain de temps et la capitalisation des expériences consiste à offrir des mécanismes de réutilisation.

1.3.7 Nature du contrôle pédagogique

Dans les deux cas, les besoins exprimés font état de simulation à but, de résolution de problèmes. Le contrôle pédagogique consiste donc le plus souvent à :

- Assigner un objectif à l'apprenant. Cet objectif peut se limiter au fait d'amener le système simulé dans un état ou un ensemble d'états déterminés.
- Vérifier éventuellement certaines contraintes (nature des opérations effectuées, dépassement de temps) pendant la recherche de l'objectif. Cette vérification peut donner lieu à une intervention "en direct" pour indiquer à l'apprenant d'éventuelles erreurs ou anomalies, pour l'aiguiller vers la bonne solution. Elle peut également donner lieu à une intervention "en différé" pour permettre à l'apprenant d'évaluer lui-même son comportement (auto-évaluation) ou pour permettre au formateur d'évaluer les compétences de l'apprenant (évaluation).

1.3.8 Contexte de développement. Nature des auteurs

Il s'agit ici de la plus grande différence entre nos deux cas d'étude. CORYS s'inscrit dans une démarche de développement informatique traditionnel : une équipe regroupant un ensemble de compétences techniques bien précises a pour objectif de développer une application dont le but a été au préalable clairement défini.

Il en va tout autrement pour Hewlett-Packard : c'est la même personne (éventuellement assistée de membres de son équipe ayant des compétences comparables) qui devra prendre en charge les tâches réparties dans le cas précédent sur une équipe pluridisciplinaire structurée. Elle aura à décider de l'opportunité du développement d'une application pédagogique, devra spécifier, concevoir, réaliser et tester la simulation, le contrôle pédagogique et la présentation interactive.

Cette importante différence de nature entre les deux approches doit-elle nous amener à les considérer comme inconciliables ? Non, il s'agit essentiellement d'une différence de niveau d'utilisation. Comme nous l'avons vu, il existe un grand nombre de caractéristiques communes entre les besoins exprimés concernant la nécessité d'un environnement de création de simulations pédagogiques mais deux points de vue différents s'affrontent :

- le premier point de vue est un point de vue *interne* et s'intéresse en premier lieu aux aspects méthodologiques. Le système proposé doit s'appuyer sur un processus de développement prenant en compte la diversité des intervenants et les contraintes de temps, et sur une méthodologie claire. Les aspects externes (disponibilité d'un outil basé sur cette méthodologie) ne viennent qu'ensuite pour automatiser et donc optimiser le processus de développement.
- Le second point de vue est un point de vue *externe*. Il est basé sur le fait que l'utilisation d'un environnement auteur dédié à la création d'applications de simulation pédagogique ne pourra être effective *que si* les interfaces proposées sont suffisamment simples et intuitives pour être acceptées par les concepteurs.

Nous considérons que ces deux points de vue peuvent être en fait *complémentaires*. D'un côté, les problèmes posés par CORYS nous aident à clarifier les concepts et les aspects méthodologiques liés à la production de simulations pédagogiques dans un contexte de développement industriel de logiciels ; de l'autre, la démarche de Hewlett-Packard nous permet de nous intéresser à une démarche de production automatisée et fortement assistée, par des concepteurs travaillant de façon individuelle et le plus souvent sans compétences informatiques. Notre objectif à long terme est d'encourager cette dernière approche, afin de la rendre accessible au plus grand nombre. Nous ne pourrons atteindre cet objectif que si les environnements auteurs proposés reposent sur un ensemble de concepts et de méthodes de conception clairement définis.

Souvent, les définitions d'ateliers de génie logiciel ne se préoccupent pas assez des interactions effectives qui existent entre le système et les auteurs lors de son utilisation. Pourtant, la qualité de ces interactions est extrêmement importante et est un argument décisif d'utilisation rationnelle de l'atelier. D'un autre côté, la démarche auteur insiste principalement sur la simplicité d'usage, sur l'abord intuitif de l'environnement. Malheureusement, cette simplicité apparente cache souvent une grande confusion ou une non-visibilité des concepts manipulés de façon sous-jacente, et rend impossible l'utilisation de l'environnement pour des usages un peu complexes.

Nous pensons quant à nous qu'il existe une relation étroite entre les aspects internes et les aspects externes. Notre approche est basée sur le fait qu'une démarche méthodologique appelée à être automatisée, doit être soumise à une expérimentation

auprès de ses utilisateurs au moyen d'outils interactifs simples. Si la conception de tels outils s'avère difficile, il doit être posé le problème de la pertinence des aspects méthodologiques proposés.

1.4 RESUME

Dans ce chapitre, nous avons examiné les problèmes actuels d'intégration de l'EAO en milieu industriel. Ces problèmes sont en majeure partie dus à :

- la surévaluation des possibilités de la formation par ordinateur.
- l'inadéquation des méthodes et la mauvaise intégration des outils existants.
- le manque de maîtrise des coûts de développement.

Certaines entreprises, comme Hewlett-Packard ou CORYS, se penchent actuellement sur le problème de l'amélioration de la rentabilité du développement de simulations pédagogiques, dans des contextes techniques bien délimités. Les caractéristiques communes à ces deux approches peuvent être résumées par le tableau suivant :

Caractéristiques des applications de simulation pédagogique considérées et de leur production	
Domaine d'application	Domaine scientifique et technique
Temps de développement	De quelques jours à 3 mois
Nature des simulations	Systèmes pouvant être considérés comme des machines à états finis Phénomènes physiques continus
Complexité des simulations	Simulations partielles réduites au contexte pédagogique pertinent
Type de simulation pédagogique	Simulations à but, résolution de problèmes Vérification
Contexte pédagogique	Apprentissage, Auto-apprentissage Evaluation, Certification, Auto-évaluation
Réalisme des simulations	Recours aux techniques multimédias Richesse des interactions
Interfaces auteurs	Simplicité de mise en œuvre Transparence des formalismes utilisés
Démarche de spécification	Démarche de type prototypage Réutilisation de composants déjà existants

2. SIMULATION ET PEDAGOGIE

Dans le domaine de l'informatique, le terme de simulation est employé très fréquemment et recouvre souvent des réalités fort différentes. L'objectif de ce chapitre est d'essayer d'éclaircir le concept de simulation pédagogique, tout d'abord en illustrant notre propos à l'aide d'exemples de logiciels existants, puis ensuite en l'abordant sous des angles d'appréciation différents.

2.1 LES OBJECTIFS DE LA SIMULATION

Selon la définition du dictionnaire, la simulation est considérée comme l'action *d'imiter, de copier, de reproduire*. Cette définition très générale ne permet pas de faire d'hypothèses sur la finalité de la simulation. Dans le domaine informatique, le terme de simulation remonte pratiquement aux origines de la discipline. Aujourd'hui, il recouvre un domaine important d'applications de types très variés. Pour des non-spécialistes, la simulation informatique sera associée par exemple, aux prévisions météorologiques où l'on sait que l'on utilise les calculateurs les plus puissants du monde, aux simulateurs de vols très répandus dans le domaine des jeux ou aux simulations d'emprunts dans le domaine financier. Autant d'exemples où la finalité de la simulation est à chaque fois différente.

Nous proposons donc de répartir les simulations informatiques en trois catégories selon l'objectif qu'elles poursuivent. On peut ainsi :

- Simuler pour comprendre.
- Simuler pour construire.
- Simuler pour apprendre.

2.1.1 Simuler pour comprendre

Dans les disciplines scientifiques fondamentales telles que les mathématiques ou la physique, l'ordinateur a été exploité très tôt en raison des capacités de calcul qu'il offrait. On s'est très vite rendu compte que grâce à cette rapidité, on disposait d'un moyen radicalement nouveau de vérifier des hypothèses de travail. En simulant les résultats d'une théorie proposée, on peut rapidement en vérifier la validité et progresser beaucoup plus vite dans le processus de mise au point de cette théorie.

Cette démarche est utilisée aujourd'hui à grande échelle dans tous les domaines scientifiques ou techniques tels que l'étude de phénomènes météorologiques, astronomiques, nucléaires, épidémiologiques, etc. L'approche consiste ici à progresser dans la *compréhension d'un phénomène réel* en proposant un modèle (le plus souvent numérique), et en comparant d'une part, les résultats produits par une simulation de ce modèle et d'autre part, les phénomènes constatés dans le monde réel.

2.1.2 Simuler pour construire

Un autre type d'application très courant consiste à utiliser l'outil informatique pour simuler un nouvel objet avant de le créer véritablement. Cet usage sert le plus souvent à proposer et valider des solutions variées pour un problème donné, en évitant de toutes les réaliser concrètement. On peut citer comme exemples :

- les logiciels d'assistance à l'architecture.
- les logiciels de simulation de prêts financiers.
- les logiciels de conception assistée permettant de créer des maquettes d'appareils ou de dispositifs techniques avant de décider leur développement concret.

2.1.3 Simuler pour apprendre

La simulation est aujourd'hui de plus en plus utilisée dans le processus d'apprentissage. Cet apprentissage peut se faire, par exemple, dans des domaines aussi variés que :

- L'apprentissage de pratiques médicales. Par exemple, un projet de la "New York University School of Medicine" a, entre autres, pour but l'apprentissage de la manipulation d'outils chirurgicaux en faisant appel à des techniques de réalité virtuelle.
- L'apprentissage de la conduite d'engins. Par exemple, le simulateur SIMULY de la société CORYS [NOR 94], dédié à l'apprentissage de la conduite de trains, insiste sur les notions de généricité et de configurabilité, et sur la nécessité pédagogique du réalisme (utilisation de son, d'images de synthèse).

- L'apprentissage de l'algorithmique. Par exemple, ARCADE [CAG 90] permet la simulation des phénomènes dynamiques liés à l'exécution des programmes, pour l'apprentissage de l'algorithmique.
- L'apprentissage de techniques agricoles. Par exemple, le logiciel Blé 2000 [GIB 94] permet l'apprentissage, à travers un micro-monde, des techniques et des contraintes liées à la culture céréalière.

Ces exemples montrent bien la variété d'utilisation de la simulation dans le processus d'apprentissage. On peut remarquer que selon les contextes, et en fonction des objectifs pédagogiques, il peut être fait appel à des techniques plus ou moins sophistiquées telles que la réalité virtuelle, la manipulation directe, le recours au multimédia.

Pourquoi utiliser la simulation dans le processus d'apprentissage? De Jong [DEJ 91] énonce un certain nombre de raisons qu'il qualifie pour certaines *d'affectives* et pour d'autres de *pratiques*. Il cite parmi les raisons affectives, l'attrait de la simulation pour l'apprenant, l'augmentation de sa motivation, une meilleure compréhension des phénomènes, une plus grande aptitude à l'adaptation pour des problèmes similaires dans d'autres contextes, etc. Parmi les raisons pratiques, le recours à la simulation est dû au fait que [DEJ 91, HER 94] :

- travailler sur le système réel peut être trop coûteux ou trop long.
- travailler sur le système réel peut être dangereux pour l'homme, l'environnement ou le matériel.
- travailler sur le système réel peut être source d'angoisse pour un débutant.
- dans une simulation, on peut introduire des situations d'extrême gravité pour entraîner l'apprenant à réagir.
- dans une simulation, on peut changer l'échelle de temps pour améliorer la compréhension.
- dans les simulations on peut simplifier ou altérer une réalité pour mieux l'étudier.

2.1.4 Et les autres simulations ?

Bien évidemment, il existe des applications logicielles basées sur la simulation qui n'entrent pas de façon claire dans une des catégories évoquées ci-dessus. On citera en particulier tous les logiciels de jeux qui prolifèrent à l'heure actuelle. Ces applications n'ont, le plus souvent, pas d'autre but avoué que celui de divertir. Il n'empêche que leur pratique peut entraîner de façon indirecte une modification de la connaissance, de la capacité de raisonnement ou du savoir-faire des utilisateurs. Par exemple, des jeux très répandus comme *Flight Simulator* ou *Tetris* améliorent

probablement les capacités d'appréhension de l'espace en 3D ou en 2D.

Dans les paragraphes qui suivent, nous entendrons par *simulations pédagogiques, les simulations dont le but avoué est l'apprentissage*.

2.2 OU CLASSER LA SIMULATION DANS L'EAO ?

Afin de situer les logiciels de simulation pédagogiques parmi les autres logiciels d'apprentissage, nous allons tout d'abord rappeler rapidement quelques classifications proposées.

2.2.1 Approche classique et approche cognitive

Une classification courante consiste à distinguer parmi les logiciels éducatifs ceux qui se réclament de l'intelligence artificielle et les autres. S'appuyant sur les techniques mises en oeuvre pour la réalisation des logiciels concernés, cette classification a initialement donné naissance à deux appellations différentes :

- *l'Enseignement Assisté par Ordinateur* (EAO) caractérise les logiciels pédagogiques ne mettant pas en œuvre des techniques d'intelligence artificielle

- *l'Enseignement Intelligemment Assisté par Ordinateur* (EIAO) caractérise les logiciels pédagogiques mettant en œuvre des techniques d'intelligence artificielle.

Le terme "intelligemment" du sigle E.I.A.O.[1] atteste de la volonté d'intégrer suffisamment de capacités de *raisonnement* dans le système pour que celui-ci soit à même de résoudre les problèmes posés, de s'adapter au comportement de l'apprenant et d'expliquer la démarche qu'il suit.

On assimile fréquemment le terme d'intelligence artificielle à deux notions qui nous semblent différentes :

- Premièrement, une volonté de séparer, d'une part, les connaissances relatives au système à modéliser, et d'autre part, le mécanisme d'exploitation de ces connaissances fondé sur une analogie avec le raisonnement humain. On parle généralement dans ce cas de *systèmes à base de connaissances*.
- Deuxièmement, les moyens techniques utilisés pour représenter les connaissances

[1] *Après cette première acception, le sigle EIAO a évolué en Environnements Interactifs d'Apprentissage par Ordinateur, définition à laquelle nous adhérons pleinement.*

et les mécanismes d'exploitation. Parmi ces moyens, on peut citer : les *systèmes à règles de production, les réseaux sémantiques, les objets structurés*.

Cette remarque nous semble importante parce qu'elle implique, par exemple, qu'un système à règles de production n'est pas toujours nécessairement utilisé en tant que moyen de modélisation du raisonnement. Il peut par exemple être utilisé comme mécanisme d'exécution interne du logiciel. A l'inverse, il peut exister d'autres moyens de représenter l'intelligence d'un système qu'en la basant sur le raisonnement. Par exemple, il peut émerger un comportement intelligent d'une société d'agents, où chaque agent a une connaissance très limitée et une capacité de raisonnement quasiment nulle.

Dans le domaine précis de l'Enseignement Assisté par Ordinateur, Nodenot [NOD 92] définit donc l'approche *cognitive* en opposition à une approche *classique*.

L'approche cognitive propose de se baser sur le raisonnement pour représenter [NIC 88] :

- l'expertise du domaine.
- l'expertise pédagogique.
- le comportement de l'élève.

Le développement d'un logiciel éducatif à partir d'une approche cognitive pose de nombreux problèmes. Nodenot [NOD 92] pose la question suivante :

"Comment demander à des pédagogues d'être à la fois capables :

- *de définir les connaissances nécessaires au logiciel éducatif pour qu'il soit "intelligent" (expertise du domaine, des stratégies pédagogiques et du comportement),*
- *de choisir le formalisme de représentation des connaissances le mieux adapté pour implémenter une telle expertise, de coder dans ce formalisme des connaissances qui ne s'y prêtent pas facilement, et de gérer la méta-connaissance qui constitue le cœur d'un tel système ?".*

Pour Nodenot, l'usage de générateurs de systèmes experts permet de répondre partiellement à ces questions. Des outils sont en effet fournis aux auteurs pour exprimer et manipuler un ensemble de règles. Cependant, il souligne les limites de cette approche, que nous résumons ci-dessous :

- L'expression de la connaissance à travers une juxtaposition de nombreuses règles rend complexe le maintien de la cohérence de la connaissance exprimée. L'évolution de cette connaissance est donc difficile à maîtriser. Cette tâche reste à

la charge de l'auteur.

- Les mécanismes d'exploitation de cette connaissance (chaînage avant, arrière ou mixte) intégrés dans ces générateurs ne sont pas forcément adaptés aux besoins de l'auteur.

Pour toutes ces raisons, le développement de logiciels pédagogiques dans une approche cognitive suppose à l'heure actuelle une collaboration étroite entre pédagogues et informaticiens. De nombreuses recherches se situent dans ce cadre, en particulier au sein du projet d'Atelier de Génie Didacticiel Intégré (AGDI) développé à Toulouse [NOD 92, MEN 95].

Dans notre contexte industriel où des outils de production opérationnels doivent être fournis, nous ne nous sommes pas orientés vers l'approche cognitive pour les raisons principales suivantes :

- L'approche cognitive impose un cadre conceptuel aux différents acteurs du développement qui ne correspond pas toujours à leur propre démarche de modélisation.
- Il existe une véritable difficulté à maîtriser la complexité de systèmes construits selon cette approche.
- La difficulté pour ces systèmes d'être explicables. *".. la question de savoir si un système relève de l'intelligence artificielle n'est pas tant qu'il modèle l'intelligence humaine, mais qu'une intelligence humaine puisse comprendre son comportement en terme de concepts clairement définis. L'approche I.A. est caractérisée par la prééminence de l'explicabilité sur l'efficacité" [KOD 86].*
- Il n'existe pas encore aujourd'hui d'environnements génériques véritablement opérationnels basés sur cette approche.

Pour nous, ce choix entraîne donc le recours à une approche dite "classique". C'est-à-dire que, de façon générale, nous ne tenterons de modéliser le raisonnement que lorsqu'il correspond à une démarche habituelle de modélisation du domaine pour une certaine catégorie de problèmes (par exemple, aide au diagnostic). Nous n'étendrons pas cette approche à la modélisation de l'expertise du pédagogue ou du comportement de l'apprenant.

2.2.2 Classer selon les approches d'apprentissage

Une seconde classification nous parait également intéressante ; elle consiste à prendre en compte la logique d'apprentissage proposée par les logiciels pédagogiques.

Nous nous référerons à la classification proposée par Thierry Mengelle [MEN 95] qui distingue deux approches : l'approche *transmission de connaissances* et l'approche *découverte / construction de connaissances*[2].

2.2.2.1 Approche transmission de connaissance

Les logiciels issus de l'approche transmission de connaissances regroupent les *tuteurs de l'EAO classique* ou *didacticiels* et les *tuteurs intelligents*. Les premiers sont basés sur une décomposition de la matière en unités élémentaires dont l'enchaînement est préétabli, sur une évaluation de l'acquisition après la présentation de chaque unité, et sur une progression conditionnée par la réponse fournie. Les seconds, les tuteurs intelligents, cherchent à adapter au mieux la transmission de connaissances aux caractéristiques de l'apprenant. Cette adaptation est réalisée en intégrant explicitement dans le logiciel, différentes expertises concernant respectivement le domaine, les stratégies pédagogiques, l'apprenant, les interfaces homme-machine [NIC 88].

2.2.2.2 Approche découverte-construction de connaissances

Les logiciels issus de l'approche découverte-construction de connaissances regroupent les *hyperdocuments*, les *environnements d'apprentissage* et les *micro-mondes*. Présentons rapidement chacun de ces types de logiciels pédagogiques, afin de mieux définir la place des simulations pédagogiques. Nous étudierons plus tard le contexte pédagogique dans lequel ils pourront être insérés.

Les hyperdocuments

Le concept *d'hyperdocument* (ou de façon plus précise *d'hypertexte* ou *hypermédia*) permet "l'immersion" dans une mer "d'information" où le lecteur "navigue" selon un processus associatif [MEN 95]. Ce type de logiciel connaît actuellement un engouement remarquable en particulier grâce à l'essor du World Wide Web. Un hyperdocument peut être considéré comme un réseau de noeuds et de liens, où les noeuds représentent les contenants d'information, et les liens, des relations entre les noeuds ou portions de noeuds [BEL 91].

L'utilisation pédagogique des hyperdocuments induit en général un *apprentissage par la découverte* [VIV 91] : l'apprenant a alors la tâche de donner du sens au chemin qu'il construit à travers la connaissance proposée.

Les micro-mondes et les environnements d'apprentissage

[2] *Le mot "connaissances" est pris ici au sens large, et inclut donc aussi bien, les connaissances conceptuelles que les connaissances opérationnelles.*

Quand il a introduit le concept de *micro-monde*, Papert [PAP 80] a en premier lieu basé son approche sur une démarche d'innovation pédagogique en reprenant les théories de J.Piaget. Papert propose d'offrir un environnement ouvert d'apprentissage dans lequel l'apprenant peut développer ses propres modèles du monde réel afin de les expérimenter par la simulation. Les micro-mondes permettent ainsi aux apprenants de développer et de corriger leurs propres théories, comme dans le cas de LOGO. Citons dans ce domaine, les exemples suivants :

- MEMOLAB [DIL 93] destiné à l'apprentissage de la psychologie expérimentale : au sein des micro-mondes fournis, l'apprenant construit une expérience, la simule et analyse les résultats obtenus.
- BLE 2000, micro-monde consacré au raisonnement et à la prise de décision en fonction d'impératifs techniques pour la culture du blé [GIB 93].
- Les travaux menés dans notre équipe sur les méthodes de définition et de conception de micro-mondes [DAV 94].

La notion *d'environnement d'apprentissage* [HER 94, MEN 95] est davantage basée sur la simulation d'un modèle que sur sa construction. L'élève apprend en modifiant les paramètres et en observant les conséquences de ses actions dans l'environnement simulé.

Comme le souligne De Jong [DEJ 91], la frontière entre environnements d'apprentissage basés sur la simulation et micro-mondes n'est pas toujours simple à tracer. Par exemple, le simple fait d'ajouter un composant dans un circuit électrique relève-t-il de l'un ou de l'autre concept ? Certaines caractéristiques permettent toutefois de distinguer chaque type d'environnement :

- Un micro-monde est un système isolé qui évolue selon ses propres lois internes, et les capacités d'apprentissage qu'il propose évoluent aussi avec le temps.
- Dans un micro-monde, l'objectif pédagogique externe peut être variable et est sous-tendu par un objectif ludique interne [GIB 93].
- La finalité d'un environnement d'apprentissage doit être explicite [HER 94].

Soulignons que ces différents types d'environnements (hyperdocuments, environnements d'apprentissage et micro-mondes) peuvent être proposés de façon conjointe [VIV 91]. Il est en effet intéressant d'imaginer que les connaissances proposées dans un hyperdocument soient renforcées par la mise à disposition de simulations lorsque cela est pertinent, comme c'est le cas par exemple dans l'application *Alternateur* que nous avons développée pour CORYS, présentée au chapitre 4. Inversement, un apprenant en train de manipuler une simulation et s'apercevant qu'il ne maîtrise peut-être pas les connaissances nécessaires pourra

avantageusement rechercher les connaissances qui lui font défaut à l'intérieur d'un hypertexte [HER 94].

La figure 2-1 montre où se situe l'utilisation de la simulation pédagogique dans les différentes approches d'apprentissage. Notre démarche personnelle se place plus spécialement dans le domaine des Environnements d'Apprentissage à buts. Cet aspect sera étudié plus en détail au §2.3.3.

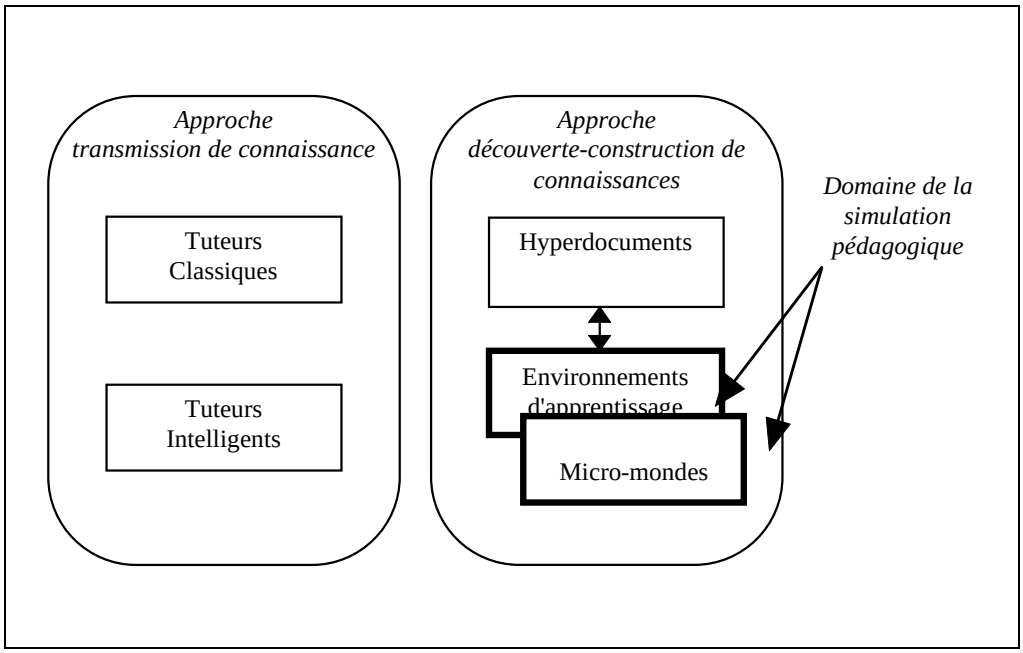


Figure 2-1 : Approches d'apprentissage et simulation pédagogique

2.2.3 La situation de notre domaine d'étude

Nous venons d'envisager une classification des logiciels pédagogiques selon deux axes :

- un premier axe qui tient compte de l'importance donnée à la modélisation du raisonnement.
- un second axe qui tient compte de l'approche d'apprentissage retenue.

Comme le montre la figure 2-2, la simulation pédagogique relève d'une approche d'apprentissage de type découverte / construction de connaissances, et est utilisée comme support des micro-mondes et des environnements d'apprentissage. La simulation peut être exploitée au niveau pédagogique en faisant appel à des degrés plus ou moins importants de modélisation du raisonnement. Pour les raisons que nous avons citées plus haut, nous limiterons notre champ d'étude à des solutions faisant peu appel à ces techniques de modélisation.

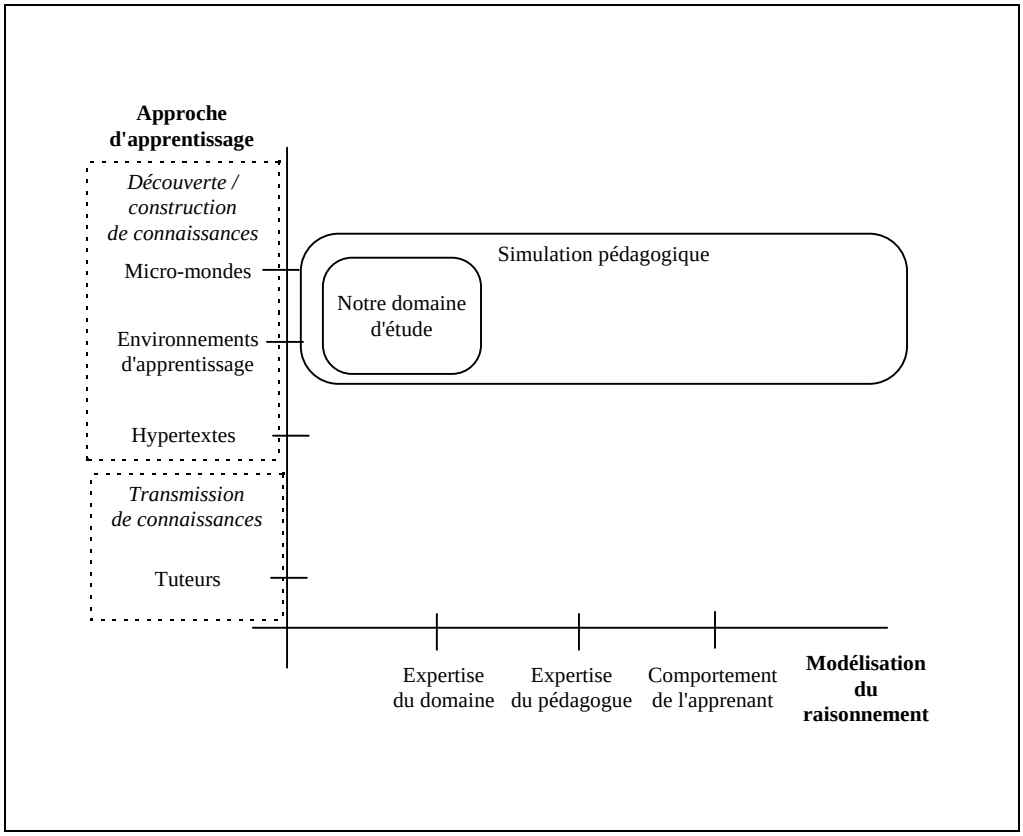


Figure 2-2 : Approches d'apprentissage et simulation pédagogique

2.3 LES SIMULATIONS PEDAGOGIQUES

Nous nous proposons maintenant de définir de façon plus précise le concept de simulation pédagogique que nous voulons proposer, en l'abordant sous les angles d'appréciation suivants :

- Quelles sont les caractéristiques des connaissances que l'apprenant doit acquérir grâce à la simulation ?
- Dans quels contextes pédagogiques les simulations peuvent-elles être utilisées ?
- Quel est le but poursuivi par l'apprenant quand il utilise une simulation pédagogique ?
- Quel type de contrôle pédagogique l'ordinateur permet-il d'assurer ?

2.3.1 Simulations et caractéristiques des connaissances acquises

Comme nous venons de le voir, les simulations pédagogiques relèvent d'une approche découverte-construction de connaissances. Essayons de préciser quels types de connaissances sont visés dans notre contexte.

De façon traditionnelle, on fait une différence entre les connaissances conceptuelles (le savoir), et les connaissances procédurales (le savoir-faire). Dans le domaine précis de la simulation pédagogique, les connaissances que devra acquérir l'apprenant pourront être de l'une ou l'autre catégorie [DEJ 91]. Certains logiciels se reposeront davantage sur la démarche de résolution de problèmes, alors que d'autres s'intéresseront davantage aux aspects purement opératoires, voire psychomoteurs.

A l'intérieur de la classe de problèmes que nous nous sommes proposés de résoudre, nous aurons pour objectif tantôt l'apprentissage de connaissances de type résolution de problèmes (ex : savoir dépanner un imprimante) et tantôt de connaissances opératoires (ex : démonter un ordinateur pour accéder à la carte mère). L'acquisition de connaissances purement conceptuelles est un cas que nous aurons rarement à traiter.

2.3.2 Simulations et contextes d'utilisation pédagogique

De façon générale, dans le domaine de l'Enseignement Assisté par Ordinateur, il est nécessaire de réaliser une analyse fine des usages pédagogiques possibles des logiciels produits. Expliciter ces usages, à savoir la présence ou non du formateur, des co-apprenants, les interactions possibles (maître-élève ou élève-élève) en fonction des problèmes, des disponibilités de chacun, des niveaux respectifs des étudiants, etc., peut permettre d'augmenter l'efficacité pédagogique de ces logiciels. Vivet préconise d'intégrer à terme cette modélisation du contexte pédagogique dans les logiciels eux-mêmes, en particulier dans le domaine des tuteurs intelligents [VIV 91].

Les simulations pédagogiques peuvent être utilisées dans plusieurs contextes :

- *Simulations pédagogiques intégrées au sein d'un enseignement traditionnel avec un pédagogue.*

  Par exemple, les logiciels de simulation dédiés aux métiers de l'énergie développés par CORYS (Cf. §1.1.1), ont pour objet d'être utilisés pendant des sessions de formation continue réservées aux personnels techniques de grandes sociétés. Ils viennent en tant qu'illustration du cours dispensé de façon traditionnelle, et peuvent être utilisés en présence du pédagogue. Dans ce cas précis, son rôle sera souvent mineur et consistera à orienter si nécessaire les apprenants.

- *Simulations pédagogiques utilisées de façon autonome.*

  Par exemple, Hewlett-Packard T.P.E.C envisage de développer des applications de simulation afin de les décentraliser et de rapprocher la formation des besoins spécifiques de l'utilisateur. Dans ce cas, l'usage du logiciel est individuel et autonome. Pour T.P.E.C, ce type d'utilisation peut avoir deux types d'objectifs :

  $\Rightarrow$ *l'apprentissage* par l'apprenant d'un domaine technique particulier sur les

matériels ou logiciels dont il doit assurer l'installation ou la maintenance.

⇒ *la certification* qui recouvre, pour Hewlett-Packard, la vérification des aptitudes techniques d'un élève à résoudre un problème. Dans ce cas, l'élève n'est pas en situation d'apprentissage mais d'évaluation.

- *Simulations pédagogiques utilisées en présence d'autres apprenants*.

  L'utilisation pédagogique peut également s'appuyer sur la présence de co-apprenants, présence effective ou virtuelle par l'intermédiaire d'un réseau. On parle alors *d'apprentissage coopératif* [VIV 91]. Citons par exemple le cas du logiciel "Toi, Moi et Lui" offert dans le cadre du laboratoire Arcade et qui propose une activité par essence coopérative [GUE 89]. Ce logiciel vise un apprentissage de certaines méthodologies de programmation. Il repose sur une analogie entre la création d'un programme informatique (activités de spécification, d'analyse et de programmation), et la production d'une bande dessinée (définition du synopsis, du scénario, et réalisation de la bande dessinée). Il prend la forme d'un jeu de rôles où les apprenants coopèrent à travers un réseau pour mener à bien le travail fixé. Le client exprime un besoin, le scénariste (alias l'analyste) spécifie le travail à faire tandis que le dessinateur (alias le programmeur) produit une réalisation conforme aux spécifications. La communication entre les différents rôles est formalisée. L'analyse a posteriori, par les étudiants et les enseignants impliqués, de ce qui s'est bien ou mal passé pendant ce travail coopératif est riche d'enseignements [GUE 90].

En résumé, les problèmes spécifiques que nous avons à résoudre relèvent le plus souvent d'une utilisation autonome des simulations, plus rarement d'une utilisation avec une assistance réduite de la part du pédagogue, et ne s'inscrivent pas dans une démarche d'apprentissage coopératif.

2.3.3 Simulations et objectifs de l'apprenant

Un certain consensus émerge actuellement pour affirmer que “*la logique de transmission des connaissances utilisée seule ne fonctionne pas, pas plus d’ailleurs que la logique de construction personnelle de connaissances pour un élève largué seul dans un micro-monde* ” [VIV 91]. De même Mengelle fait remarquer que "*pour être efficace, l'utilisation d'un micro-monde, d'un environnement d'apprentissage ou d'un hypertexte doit être finalisée*" [MEN 95]. L'objectif peut idéalement être un projet personnel de l'apprenant comme dans le cas des micro-mondes, mais il est le plus souvent fixé par l'enseignant.

En ce qui concerne précisément les simulations, le fait de fixer un objectif à l'apprenant présente de nombreux avantages par rapport à lui laisser toute liberté d'utilisation. Par exemple, Herzog et Forte [HER 94] comparent simulation à but et

simulation libre dans les termes suivants :

- la simulation à but offre un défi (un *challenge*) à l'apprenant et renforce ainsi sa motivation.
- la simulation à but donne un sens aux actions de l'apprenant, une orientation à son comportement.
- la simulation à but évite à l'apprenant de modifier de façon aléatoire ou incohérente les paramètres de la simulation.
- la simulation à but donne à l'apprenant la possibilité d'examiner certains aspects à côté desquels il serait sans doute passé s'il avait utilisé librement la simulation.

L'objectif (ou un ensemble d'objectifs) peut être fixé par le formateur (personne physique) si le logiciel de simulation pédagogique est intégré à une session de formation pilotée par un enseignant. Dans le contexte du laboratoire Arcade développé par notre équipe, c'est le cas des objectifs explicités dans les scénarios pédagogiques où l'enseignant, en fonction de l'avancement de son cours, dirige les étudiants vers des logiciels précis avec des objectifs bien définis (exemple : découvrir les méthodes de tri utilisant telles actions ou fonctions élémentaires, réaliser telle méthode de tri étudiée, comparer l'efficacité de différents algorithmes de tri,...) [GUE 91].

L'objectif peut également être intégré aux logiciels de simulation pédagogique. Il peut être simplement suggéré à l'étudiant comme c'est le cas dans les logiciels du laboratoire Arcade : une liste d'idées d'exercices est intégrée à chaque logiciel et peut être consultée par l'étudiant [CAG 90]. L'objectif peut également être explicitement choisi par l'apprenant dans une liste d'objectifs possibles [HER 94] ou donné de façon plus directive.

Vouloir finaliser une simulation pédagogique implique de placer l'apprenant dans une situation donnée et de fixer alors (éventuellement avec sa participation) l'objectif à atteindre. Il peut être nécessaire de préciser un certain nombre de contraintes que l'apprenant devra respecter durant sa progression vers l'objectif final. Des exemples de telles contraintes sont donnés dans [HER 94] pour des simulations de chimie ; citons également, pour des exemples qui nous touchent de plus près, des objectifs assortis de contraintes : couplage d'un alternateur au réseau EDF sans détérioration majeure (CORYS), calibration d'un appareil sans dommage ni pour l'appareil ni pour l'intervenant, réparation d'un appareil en prenant en compte une limitation des coûts de dépannage (H.P.). Nous nous plaçons donc résolument dans une démarche de simulation à buts (i.e. simulation à objectifs).

Sur un autre plan, le fait de fournir à l'apprenant un objectif clairement défini permet d'envisager un suivi de ses activités, une assistance, un guidage ou une

évaluation. Ces différentes activités peuvent être réalisées soit par l'enseignant dans un contexte d'utilisation assistée, soit par le système dans un contexte d'utilisation autonome. Nous posons ici le problème du contrôle pédagogique automatisé qui est abordé au paragraphe suivant.

2.3.4 Simulations à buts et contrôle pédagogique

Nous proposons d'évaluer le contrôle pédagogique automatisé d'une simulation à buts selon deux dimensions :

- La première dimension concerne la nature et le degré de finesse du contrôle effectué par le système. Les possibilités suivantes peuvent se rencontrer :
  - ⇒ pas de contrôle de la simulation. On fixe uniquement un objectif à l'apprenant et c'est à lui d'estimer s'il a ou non atteint l'objectif qui lui était assigné.
  - ⇒ un premier niveau de contrôle consiste à garder une trace des manipulations de l'apprenant.
  - ⇒ un second niveau de contrôle consiste à déterminer si l'objectif imparti a été ou non atteint pendant la session de travail.
  - ⇒ un troisième niveau de contrôle consiste à s'intéresser de façon plus fine à la façon dont l'apprenant parvient au résultat, à vérifier sa progression vers l'objectif fixé.
- La deuxième dimension concerne le degré d'interaction entre le contrôle pédagogique et l'apprenant durant la manipulation de la simulation. Cette réactivité du système peut s'exprimer de plusieurs façons :
  - ⇒ de façon asynchrone. Dans ce cas, les résultats du contrôle sont traités après la manipulation. Ils peuvent être exploités après coup par l'apprenant lui-même, par un enseignant ou par le système. C'est en particulier le cas pour le problème de la certification (l'évaluation) préconisée par H.P. où la nature du contrôle consiste à déterminer après coup si l'utilisateur a correctement exécuté la simulation.
  - ⇒ de façon synchrone pendant l'exécution de la simulation. Dans ce cas, on peut proposer des démarches plus ou moins ambitieuses :
    - ◊ un premier type de démarche consiste à fournir des indications à l'apprenant pour l'aiguiller, le conseiller, l'encourager, lui fournir de l'aide.
    - ◊ Une autre démarche consiste à établir un dialogue entre l'apprenant et le système afin que ce dernier puisse comprendre les intentions de l'élève. C'est par exemple la démarche suivie par [AZZ 95] qui se place dans une

problématique de diagnostic et réparation du dysfonctionnement d'un système de production. Afin de mieux cerner la démarche de l'apprenant et de situer ses lacunes, l'auteur propose à l'apprenant de procéder en trois étapes. La première étape est celle où l'apprenant établit le diagnostic. Celui-ci étant validé, l'apprenant peut ensuite élaborer le plan d'action. Ce n'est qu'après validation de ce plan, que l'apprenant est autorisé à le mettre en exécution.

◊ Une troisième démarche plus sophistiquée consiste à essayer de doter le système d'une capacité d'interpréter les intentions de l'apprenant à partir de ses actions [NIC 88]. Le contrôle ainsi que la réactivité peuvent alors théoriquement s'adapter au fur et à mesure de la progression de l'apprenant dans sa démarche de résolution de problèmes.

Notre démarche personnelle est précisée de façon détaillée dans le chapitre 8 où nous donnons notre définition de la notion de scénario pédagogique. Cette démarche peut se résumer de la façon suivante :

- Le type de contrôle essentiel que nous voulons assurer est la vérification de l'objectif principal de la simulation.
- Nous ne nous intéressons pas à essayer d'interpréter le raisonnement de l'élève dans sa démarche de résolution de problèmes.
- Nous nous intéressons à la façon dont un apprenant résout un problème, mais uniquement en termes de franchissement de sous-étapes intermédiaires.
- Durant l'exécution d'une simulation, l'interaction pédagogique que nous préconisons se compose d'informations délivrées par le composant pédagogique à l'usage de l'apprenant, ou de demandes d'aide de la part de l'apprenant. Elle ne prend pas en compte la possibilité de négociations entre les deux parties.

2.4 RESUME

Dans ce chapitre, nous avons défini le type de simulation que nous voulons proposer. Après avoir rappelé qu'en informatique, la simulation peut être utilisée pour comprendre, pour créer ou pour apprendre, nous avons ciblé notre attention sur les simulations dont le but avoué est l'apprentissage, dans un contexte d'utilisation autonome ou d'utilisation assistée par un pédagogue.

Nous nous orientons vers des solutions opérationnelles pouvant être mises en œuvre rapidement dans des contextes techniques précis. De ce fait, notre démarche,

qui se place dans le domaine des Environnements d'Apprentissage à buts, ne tentera de modéliser le raisonnement que lorsqu'il correspond à une démarche habituelle de résolution de problème.

Les simulations que nous voulons proposer concernent l'apprentissage de connaissances de type résolution de problèmes ou de type opératoire, et beaucoup plus exceptionnellement la transmission de connaissances purement conceptuelles.

Les simulations doivent fixer à l'apprenant un objectif pédagogique clairement défini. Nous préconisons un contrôle pédagogique de type discret, privilégiant l'explicabilité sur la sophistication : nous vérifierons uniquement le franchissement de sous-étapes intermédiaires, sans tenter d'interpréter le raisonnement de l'élève.

3. LES APPROCHES DE PRODUCTION DE SIMULATIONS PEDAGOGIQUES

3.1 LES ENJEUX DE LA PRODUCTION

Nous avons expliqué au chapitre 1 quels pouvaient être les enjeux dans les contextes industriels étudiés (CORYS et H.P.). Dans les deux cas, et pour des raisons différentes, nous avons remarqué que la maîtrise de la production représentait un point critique. La société Hewlett-Packard doit impérativement maîtriser *les délais de développement* pour être capable de livrer au moment de l'industrialisation d'un nouveau produit, les supports de formation correspondants. La société CORYS doit impérativement maîtriser *les coûts de production* (délai et ressources) si elle veut proposer des logiciels de formation à des prix raisonnables.

Ces enjeux mis en évidence dans ces contextes de formation continue en milieu industriel, se retrouvent également dans le domaine de la formation initiale. B. Cornu [COR 92] dans son introduction au livre "*L'ordinateur pour enseigner les maths*" constate qu'en matière d'informatique pédagogique, le décalage est très grand entre ce qui se fait de façon expérimentale et ce qui se pratique de façon quotidienne dans les classes. Face à ce constat, Nodenot [NOD 92] évoque des raisons psychologiques (peur, manque de formation des pédagogues,...) mais aussi des raisons liées aux contraintes de production. En effet, nous pouvons constater qu'il existe aujourd'hui des logiciels de grande qualité pédagogique, en général issus de travaux d'enseignants et/ou de chercheurs. Mais nous pouvons également remarquer que ces logiciels sont produits à des coûts déraisonnables, que ce soit en termes d'effort financier, ou, ce qui

est plus difficilement mesurable, en termes d'investissement personnel des enseignants. Cette raison, ainsi qu'une reconnaissance très relative du travail des enseignants impliqués dans l'essor des "Nouvelles Technologies Educatives", représente sans aucun doute le frein le plus important au développement à grande échelle de logiciels pédagogiques de qualité.

La maîtrise des coûts de production doit permettre de développer des logiciels de qualité à moindre effort, et de favoriser ainsi l'apparition sur le marché commercial d'un nombre nettement plus important de produits intéressants à des coûts nettement moins élevés. Cette augmentation du volume et cette diminution des prix devraient faciliter l'usage de ce type de logiciels par les enseignants en milieu scolaire.

Dans le champ d'étude que nous nous sommes fixés, la maîtrise des coûts de production passe aujourd'hui, à notre avis, par l'utilisation ou la mise au point de méthodologies et d'outils de production réellement adaptés à chaque type de contexte de développement. Plusieurs approches de production sont envisageables. Elles se distinguent par le type de logiciel visé, par la nature des personnes assurant le développement (informaticiens ou non), ou par les environnements de production utilisés. Ces différents critères vont nous servir de base pour évaluer :

- La production d'applications de simulation (hors du contexte pédagogique).
- La production de logiciels pédagogiques.
- Enfin, la production de simulations pédagogiques.

3.2 LA PRODUCTION D'APPLICATIONS DE SIMULATION

La simulation est un domaine à part entière de l'informatique. Elle peut se scinder au moins en deux grandes sous-disciplines : la simulation de phénomènes continus et la simulation à événements discrets. Chacune de ces sous-disciplines propose ses propres règles d'écriture, ses propres modes de production. De façon plus générale, on peut indiquer les approches suivantes [AZZ 95] :

L'approche Langage

Les langages utilisés dans le développement des simulations ont pour but de réduire les efforts de conception des modèles, de diminuer les coûts de programmation et d'exploitation. Ces langages ont suivi les évolutions récentes de l'informatique. On peut ainsi distinguer :

- *les langages classiques* de type Fortran, Pascal, C ,...
- *les langages orientés objets* comme SIMULA, Smalltalk, Eiffel,...

- *les langages de programmation généraux auxquels sont associées des bibliothèques dédiées au développement de simulation*. C'est le cas de GASP, FORTSIM,...
- *les langages génériques de simulation*. Ces langages reposent sur des fonctions abstraites du type file d'attente, ressource, entité, processus, événement, etc. C'est le cas de SLAM, GPSS, SIMSCRIPT, SIMAN, ...

L'approche Outils

Dans cette catégorie on peut distinguer :

- *les outils de simulation dédiés à un domaine particulier*. Dans le domaine de la simulation, il existe des classes de problèmes très spécifiques à telle ou telle activité. Il existe donc des outils spécialisés "préprogrammés" permettant au concepteur de modéliser à moindre effort ces classes de problème. On peut citer, par exemple, des outils tels que SAME, SIMFACTORY, dédiés au domaine de fabrication discontinue en milieu industriel. Azzi souligne comme avantages de ces outils la simplification de la conception, la réduction des efforts de modélisation, l'amélioration de la lisibilité et de la maintenance apportées par ces langages mais signale en contrepartie une certaine rigidité de ce type d'outil.
- *Les outils de génération de simulation.* Leur objectif est de rendre accessible la création de simulations par des non-programmeurs. Ils se basent de façon générale sur un ensemble d'éditeurs spécialisés permettant de décrire d'une part, l'interface de la simulation, et d'autre part, la logique fonctionnelle de la simulation. Des techniques de programmation visuelle et de génération automatique de code rendent effectivement ce type d'outils très attrayant. On peut citer dans ce domaine :

  ⇒ *LabVIEW* de la société *National Instruments*. Ce logiciel permet de définir sans programmation textuelle, des systèmes d'instrumentation, reliés à des équipements réels ou totalement simulés. Le concepteur dispose d'objets appelés *Instruments virtuels* qui proposent un aspect graphique et un aspect logique (basé sur un ensemble de connecteurs d'entrée et de sortie). Il peut à partir de ces objets (qu'il a lui même créés ou qu'il a importés de bibliothèques) créer son système grâce à la programmation graphique d'un diagramme de flots de données reliant les divers instruments virtuels.

  ⇒ *Rapid* de la société *EMULTEK*. De la même façon, Rapid a pour objectif la création rapide de simulations d'instruments, d'appareillage ou de systèmes. Il propose également deux espaces de travail. Le premier (*Objects*) permet de définir l'interface graphique de l'application grâce à des bibliothèques d'objets au comportement prédéfini. Le second espace (*Logic)* permet de définir de façon graphique (mais aussi souvent textuelle) les relations qui lient ces objets. Cette fois-ci, la programmation du fonctionnement du système est basé sur des

diagrammes d'états concurrents (formalisme de Harel).

Les avantages de ce type d'outils sont :

⇒ l'accessibilité aux non-programmeurs.

⇒ une réelle efficacité pour réaliser des simulations simples.

Par contre, les inconvénients que nous pouvons relever sont les suivants :

⇒ la trop grande interdépendance des aspects qui relèvent de l'interface et de ceux qui relèvent du modèle de simulation. Le plus souvent, une modification mineure apportée à l'interface entraîne une reprogrammation significative de la partie fonctionnelle.

⇒ Une certaine limitation des problèmes pouvant être modélisés due au type de programmation sous-jacent. Par exemple, le flot de données se prête très mal à l'expression de la concurrence et de la synchronisation.

L'approche Environnement de développement

Il s'agit ici de gérer la complexité de certains projets de simulation à grande échelle. La problématique soulevée par ce genre d'environnement repose souvent sur l'intégration de composants hétérogènes (codes sources de simulation, composants d'interface, etc.) ou leur répartition. On peut citer par exemple NANCE [NAN 89], STRICOM [JIN 94] ou ESCAPE [FLE 95]. Nous n'approfondirons pas cet aspect, car nous ne nous situons pas dans une problématique de gestion de projet complexe. On pourra trouver une description plus détaillée de cette classe d'environnements dans [AZZ 95].

3.3 LA PRODUCTION DE LOGICIELS PEDAGOGIQUES

Le deuxième domaine que nous nous sommes proposés d'étudier concerne les techniques de production des logiciels pédagogiques de façon générale, qu'ils soient ou non basés sur la simulation. Nous avons proposé au chapitre 2 une classification des applications pédagogiques selon l'approche d'apprentissage sur laquelle ils s'appuyaient et leur degré de prise en compte des aspects cognitifs. A chacun des différents types d'applications peuvent correspondre une approche et des techniques différentes de production. Nous nous proposons donc d'étudier :

- Les différentes approches de production en termes de techniques employées.
- Les démarches de production spécifiques et génériques.

3.3.1 Les techniques employées

Depuis les débuts de l'EAO, on a pu constater une évolution progressive des environnements de production vers des solutions techniques de plus en plus accessibles à des non-programmeurs. Ainsi, on peut isoler quatre grandes classes d'environnements qui se sont succédés :

- Les langages de programmation traditionnels.
- Les langages auteurs.
- Les systèmes auteurs.
- Les générateurs d'applications hypermédias.

3.3.1.1 Les langages de programmation traditionnels

S'ils permettent bien entendu de s'adapter à une grande variété de logiciels pédagogiques, les langages de programmation traditionnels sont inutilisables sans une compétence réelle en programmation. Leur utilisation a donc été possible selon deux approches : l'une imposait à l'enseignant le recours à une équipe d'informaticiens, l'autre impliquait la formation de l'enseignant aux concepts et aux langages de programmation. L'insuffisance de ces approches a été rapidement perçue, et a motivé l'apparition des premiers langages auteurs.

3.3.1.2 Les langages auteurs

Les langages auteurs sont apparus à partir des années 75. Ils offrent à des auteurs, a priori non-spécialistes en programmation, des fonctionnalités spécifiques à l'écriture de logiciels pédagogiques, plus précisément de tuteurs de l'EAO classique. Ils s'appuient généralement sur un langage textuel offrant des primitives permettant la mise en page des écrans, des manipulations graphiques simples, l'analyse de réponse, le contrôle du cheminement grâce à des branchements conditionnels. Citons par exemple des langages tels que TUTOR, COURSE-WRITER, ARLEQUIN...

De tels langages, dédiés à la production de tuteurs classiques, restent néanmoins difficiles d'accès pour des auteurs non programmeurs et leur emploi s'avère délicat lorsque l'auteur ne se satisfait pas des fonctionnalités de base prévues.

3.3.1.3 Les systèmes auteurs classiques

En réponse aux difficultés inhérentes aux langages précédents, les systèmes auteurs ont pour objectif principal d'éviter la phase de programmation des logiciels pédagogiques. Ils sont, comme les langages auteurs, dédiés à la production de tuteurs de l'EAO classique et proposent un ensemble d'éditeurs pédagogiques permettant à l'auteur de construire et d'agencer le contenu de son logiciel éducatif. Ces éditeurs permettent de présenter à l'apprenant des informations (textuelles, graphiques,

multimédias pour les systèmes les plus récents), de formaliser l'analyse des réponses et les orientations associées, de définir les différents modules éducatifs qui constituent le cours. Citons des systèmes auteurs tels que DUO, EGO, PROF, AUTHORWARE, ICONAUTHOR...

Les systèmes auteurs ont pour principaux avantages :

- Leur facilité d'emploi qui les rend nettement plus accessibles à des auteurs non programmeurs que les langages auteurs.
- Une certaine possibilité d'extensibilité pour certains d'entre eux qui proposent un langage d'expression associé aux fonctions préprogrammées.

Il est à noter que les plus récents de ces logiciels, comme Authorware, intègrent des fonctionnalités permettant la création d'hypertextes ou plus généralement d'hypermédias. Par contre, ils restent limités le plus souvent à une interaction de type navigation.

3.3.1.4 Les générateurs d'applications hypermédias

Dédiés à la création d'hypertextes (ou plus généralement d'hypermédias), ces logiciels permettent la création, par manipulation directe, d'unités d'informations (des pages écran) et de leur contenu (zones de texte, boutons, graphismes, etc.). Ils reposent sur un langage de script de niveau "auteur" permettant d'une part, la définition de liens entre les unités d'informations, et d'autre part, la définition pour les objets de certains comportements associés aux actions de l'utilisateur. Le premier logiciel à avoir adopté cette approche est Hypercard, fonctionnant sur Macintosh. Nous pouvons également citer SuperCard (Macintosh), MetaCard et Multicard (Unix) ou Toolbook (MS-Windows). Certains environnements comme Toolbook de la société Asymetrix ont considérablement enrichi cette approche en portant leur effort dans plusieurs directions :

- La mise à disposition d'outils de haut niveau permettant de s'affranchir de l'écriture de scripts. Par exemple, Toolbook propose un ensemble de fonctions basées sur la programmation par démonstration, pour définir les liens, l'animations d'objets graphiques, la gestion de Q.C.M., l'importation d'objets interactifs préprogrammés, etc.
- L'enrichissement du langage de script. Cet enrichissement peut concerner :
  - ⇒ l'ouverture de l'application par rapport à d'autres environnements (communication entre applications, liens avec d'autres documents).
  - ⇒ la mise à disposition de certains concepts propres aux langages de programmation traditionnels (typage de données, structures de contrôle variées,

etc.) ou de certains concepts informatiques avancés (gestion de délais de garde, synchronisation, etc.).

⇒ une certaine orientation par objets facilitant la réutilisation de composants déjà définis. Par exemple, Toolbook offre la possibilité de définir, pour un objet donné, des propriétés ou des méthodes, et d'assurer une communication par messages entre les objets.

- La redéfinition possible des environnements de manipulation, que ce soit :

  ⇒ au niveau utilisateur (mise à disposition de menus, gestion de fenêtres, etc.).

  ⇒ au niveau auteur, par exemple, en inhibant ou modifiant certaines fonctionnalités préexistantes, ou en créant de nouvelles fonctionnalités.

Actuellement, ce style de logiciel connaît un engouement remarquable dans le domaine de l'Enseignement Assisté par Ordinateur. On peut d'ailleurs souligner que Toolbok a été retenu comme plate-forme de développement EAO, dans des contextes différents, que ce soit :

- dans le cadre de l'Education Nationale en France.
- au sein d'entreprises généralistes comme Hewlett-Packard au niveau mondial.
- au sein d'entreprises plus spécialisées comme CORYS pour le développement de ses simulateurs pédagogiques.

A notre avis, les avantages de ce type d'environnement sont :

- une évidente simplicité d'utilisation pour des non-programmeurs quand il s'agit de créer des classes d'applications bien répertoriées. Le recours intensif à la manipulation directe et à la programmation par démonstration permettent la création rapide d'hypermédias à usage éducatif, de simulations graphiques élémentaires ou de tuteurs simples.
- la puissance des fonctionnalités proposées qui permettent de se dispenser dans de nombreux cas du recours à d'autres environnements de programmation. Pour un programmeur expérimenté, le langage fourni (par exemple OpenScript pour Toolbook) permet une expression algorithmique d'un niveau comparable à celui de langages traditionnels. Cet aspect permet d'utiliser le même environnement pour plusieurs facettes d'une application complète (interface, noyau fonctionnel, contrôle pédagogique sophistiqué). La contrepartie de cet avantage réside dans le faible niveau de contrôles d'écriture des programmes (langage interprété à typage faible), et la lenteur à l'exécution. On peut tout de même noter que dans l'exemple de l'alternateur chez CORYS, le choix a été fait de coder les modèles de simulations (équations différentielles) dans le langage OpenScript, et que les performances obtenues sont acceptables.

D'un autre côté, les inconvénients de ce type d'environnement se font sentir essentiellement en ce qui concerne les auteurs non-programmeurs. On peut citer parmi ces inconvénients :

- un manque de vue d'ensemble de l'application développée, se prêtant mal à la maintenance, à la gestion des modifications.
- une simplicité toute relative du langage de script. Cette simplicité qui se réclame d'une soi-disant approche du langage naturel, se traduit en fait par une certaine verbosité, et implique quand même une maîtrise par l'auteur des concepts fondamentaux de la programmation, s'il veut effectivement retoucher les scripts. L'expérience que nous avons pu acquérir de l'enseignement de ce type de langage, nous a permis de constater souvent un sentiment d'impuissance et de frustration de la part de non-programmeurs face à cette complexité.
- la fragilité des codes générés automatiquement. Nous nous situons dans un type d'environnement interprété, et le code généré peut être modifié à tout moment par l'auteur. En particulier une manipulation erronée ou involontaire peut "casser" un script sans que l'auteur ne s'en aperçoive avant l'exécution de son application. S'il ne maîtrise pas le langage, il n'aura pas moyen de comprendre les raisons de son erreur, et en reportera naturellement la responsabilité sur le système.

3.3.2 Production spécifique et environnements génériques

On peut également aborder le problème de la production d'applications en termes d'approche spécifique ou d'approche générique. On distingue ainsi deux niveaux :

Le premier niveau consiste à s'intéresser à un *domaine particulier*, à réfléchir avec les pédagogues aux activités à proposer à l'apprenant, et à élaborer un logiciel concrétisant ces réflexions. Cette démarche permet de partir des besoins pédagogiques du domaine, d'imaginer comment l'ordinateur peut fournir une réponse pertinente à ces besoins, puis de s'attaquer aux problèmes de mise en œuvre et d'apporter une réponse spécifique à ces problèmes. C'est par exemple ce genre de démarche qui a été utilisé par notre équipe lors de la définition du laboratoire [GUE 89, LIE 89] pour l'apprentissage de l'algorithmique. Dans d'autres domaines, citons également les travaux de C. Desmoulins [DES 94] pour l'apprentissage de la géométrie, de J.F. Brette [BRE 94] pour l'apprentissage de la programmation, P. Teutsch [TEU 94] en ce qui concerne l'apprentissage des langues étrangères, ...

Le second niveau vise à adopter une *approche plus générique* visant soit la *production d'une catégorie particulière* de logiciels pédagogiques, soit la *prise en compte d'aspects particuliers* dans la production de logiciels pédagogiques. Dans les

deux cas, il s'agit de proposer des méthodes et des outils adaptés aux objectifs. Ces méthodes et ces outils peuvent selon les cas être destinés directement aux pédagogues, ou bien s'inscrire dans une démarche nécessitant une collaboration étroite entre pédagogues et informaticiens.

On peut citer comme exemples concernant la production générique de catégories particulières de logiciels pédagogiques :

- la production de tuteurs intelligents : COCA [MAJ 93] ou EDDI [MAR 90].
- les travaux de J.P. David concernant les micro-mondes d'apprentissage basés sur la métaphore du laboratoire d'expérimentation [DAV 94].
- les environnements d'apprentissage à initiative mixte intégrant les approches tutorielles et micro-mondes : ETOILE [DIL 93].

En ce qui concerne la prise en compte d'aspects particuliers dans la production de logiciels, certaines études s'intéressent à offrir des fonctionnalités pour gérer, par exemple :

- l'assistance dans les logiciels pédagogiques : proposition d'une structure d'accueil permettant à des pédagogues de produire des environnements d'apprentissage où l'ordinateur joue le rôle d'assistant. Cette structure d'accueil peut être instanciée avec la connaissance spécifique pour produire un environnement particulier [MEN 95].
- l'aide à la résolution de problèmes : proposition d'une architecture distribuée, SCENT3, permettant l'intégration de fonctionnalités pédagogiques adaptées à la résolution de problèmes dans des domaines variés [McC 88].
- les interactions avec l'apprenant : proposition d'un système, GEODE, permettant de générer les interactions avec l'apprenant en autorisant d'une part une spécification graphique des éléments d'interaction, et d'autre part une spécification déclarative des stratégies pédagogiques ; la conduite du dialogue est indépendante de la représentation de la matière à enseigner [BOU 93].

Il peut s'agir également de propositions visant à résoudre des problèmes génériques de production comme :

- l'intégration des différentes expertises dans les tuteurs intelligents : proposition d'un modèle d'Agents Centrés Tâches, modèle MACT [PEN 93].
- la coopération entre informaticiens et pédagogues : proposition d'un méta atelier de génie éducatif MAGE offrant une cadre méthodologique et des services [NOD 92].
- l'intégration de méthodes hybrides de production au sein d'ateliers de génie didacticiels : propositions au sein de l'atelier de Génie Didacticiel Intégré AGDI

[GOU 90, NOD 92].

Toutes ces recherches prenant en compte des facteurs génériques de la production, établissent des propositions intéressantes et prometteuses. Il convient de noter que leurs auteurs s'accordent en général dans leurs conclusions : les propositions faites par chacun dans son propre axe d'étude constituent des points de départ mais débouchent rarement sur des environnements opérationnels ou commercialisables.

3.4 LA PRODUCTION DE SIMULATIONS PEDAGOGIQUES

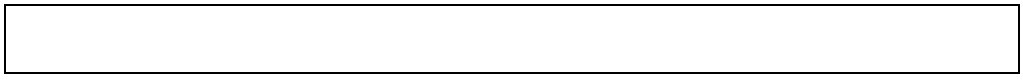
Après avoir étudié les diverses approches concernant la production de simulations isolées et de logiciels pédagogiques, intéressons-nous au problème spécifique de la production de simulations pédagogiques. Là encore, nous avons deux approches :

- *L'approche couplée* qui consiste à utiliser deux environnements différents de production, spécialisés chacun dans un des deux aspects, et à assurer leur communication.
- *L'approche intégrée* qui consiste à traiter dans le même environnement les aspects relatifs à l'expertise du domaine et ceux relatifs au contrôle pédagogique.

3.4.1 L'approche couplée

Certains environnements de construction de simulateurs s'intéressent aujourd'hui à proposer une ouverture vers le contrôle pédagogique. C'est le cas par exemple de l'environnement RAPID, qui s'est d'abord spécialisé dans le prototypage rapide d'applications de simulation (*Prototyping and Design*) pour s'orienter de plus en plus vers l'exploitation pédagogique des simulations (*Simulation and Training*). RAPID propose un module d'interface pédagogique (*CBT interface*) permettant de communiquer (Cf. figure 3-1) avec une application de contrôle pédagogique développée avec un environnement auteur (tel que Authorware, Visual Basic, Toolbook, ...).

Une démarche comparable a été adoptée dans le projet européen SAM (Simulation and Multimédia) [ROS 94] qui préconise une architecture à trois composants : un composant interface réalisé à l'aide du logiciel Hypercard, un composant simulation réalisé avec LabVIEW et un composant pédagogique indépendant permettant de spécifier le contrôle pédagogique et d'assurer les communications entre les deux autres composants.



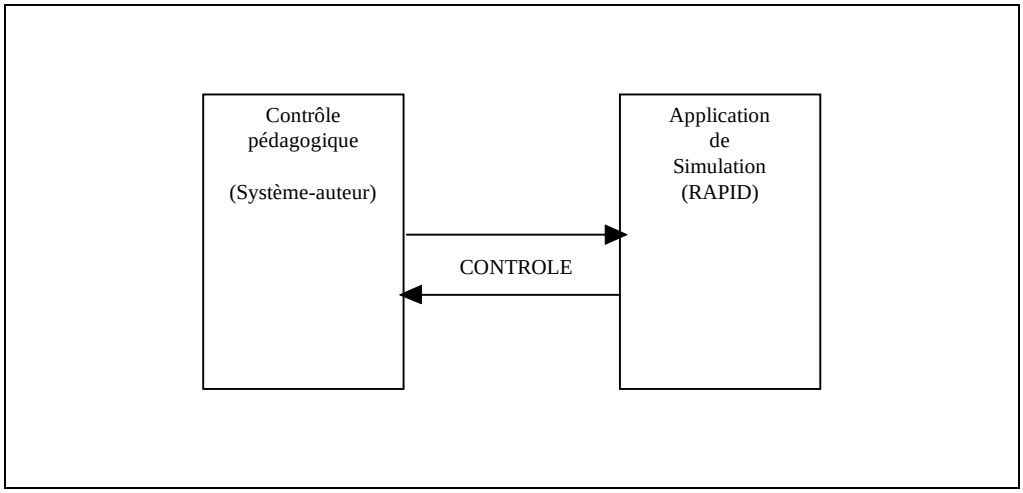


Figure 3-1 : Approche couplée : communication d'une application de simulation

Nous avons envisagé ce type d'approche, mais nous ne l'avons pas retenu pour les raisons suivantes :

- La communication entre deux logiciels s'exécutant de façon parallèle réduit considérablement les performances. Ceci peut être particulièrement pénalisant pour des applications où le facteur temps est important.
- La présence de deux environnements différents complique notablement la tâche du ou des concepteurs (manque d'homogénéité entre les interfaces utilisateurs).
- Les interfaces de contrôle proposées par l'outil de simulation influencent nécessairement le type de contrôle possible de la part de l'outil pédagogique. Par exemple, l'interface CBT fournie par RAPID propose essentiellement le contrôle des événements utilisateurs (redirection, duplication, activation). La possibilité de vérifier un état du système n'est pas prévue dans l'interface.
- Le produit initial, conçu pour réaliser des simulations, se prête mal à la conception d'une simulation pédagogique. Par exemple, les outils de simulations n'associent qu'une seule vue (une page écran) comme moyen de visualisation de l'état du système à simuler. A l'inverse, la simulation pédagogique nécessite fréquemment de recourir au concept de vues multiples d'un même modèle de simulation.

3.4.2 L'approche intégrée

Certains auteurs se sont penchés sur la définition d'environnements intégrés dédiés à la conception de simulations pédagogiques. En particulier, les projets SIMULATE (*SIMULation Authoring Tools Environment*) et SMISLE (*System for Multimedia Integrated Simulation Learning Environments)* ont pour objectif de proposer un environnement auteur permettant à des non-programmeurs de développer

des simulations [NJO 91]. SMISLE est conçu autour de cinq modèles, remplissant chacun une fonction spécifique : le modèle exécutable, le modèle cognitif, le modèle pédagogique, le modèle de l'élève et le modèle de l'interface. Cette approche nous semble particulièrement intéressante, mais sans doute trop complexe pour les raisons que nous avons citées plus haut (les limites dans notre contexte de l'approche cognitive).

Une autre solution consiste à proposer une solution intégrée à partir d'un environnement existant. Les environnements de conception de simulation étant par nature très peu ouverts, nous nous sommes orientés vers les générateurs de logiciels hypermédias que nous avons présentés plus haut. Les avantages de ces environnements (définition aisée d'interfaces, langage relativement puissant, redéfinition possible des outils auteurs, ouverture vers l'extérieur) nous permettent d'envisager de les utiliser comme couche de base pour un environnement auteur plus riche, proposant les fonctionnalités désirées (conception des modèles de simulation, des scénarios pédagogiques, de l'interface, et intégration des résultats). L'avantage de cette solution réside essentiellement dans l'homogénéité des interfaces, dans la réduction des coûts de communication entre composants, et dans sa flexibilité.

En particulier, pour le projet MELISA que nous présentons au chapitre 11, nous proposons un environnement composé de surcouches construites autour du système Toolbook, comme le montre la figure 3-2.

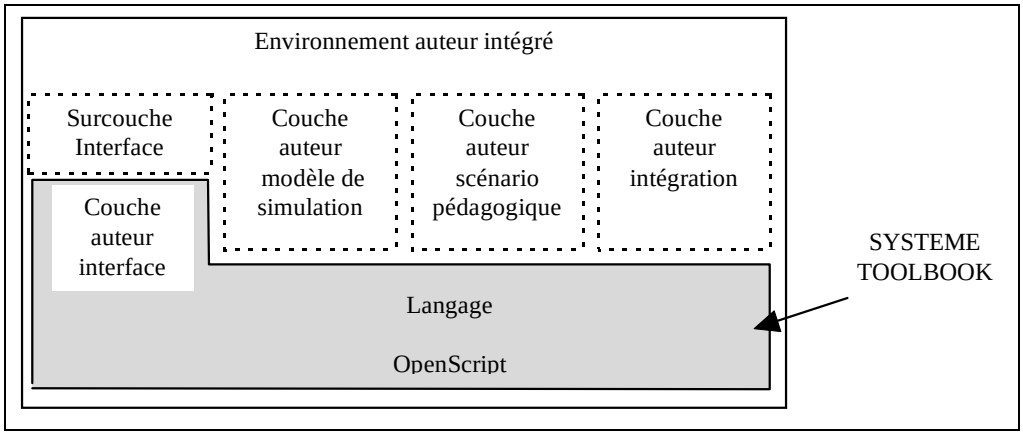


Figure 3-2 : L'approche intégrée :la solution MELISA

3.5 RESUME

Nous nous sommes intéressés, dans ce chapitre, aux problèmes liés à la production de logiciels d'apprentissage et plus particulièrement de simulations pédagogiques.

Les environnements actuels de production permettent de développer efficacement certains composants spécifiques des simulations pédagogiques (modèle de simulation, application interactive multimédia, composant pédagogique). L'intégration de ces composants s'avère toutefois souvent malaisée (difficulté de communication, performances médiocres).

Dans l'état actuel des produits disponibles, et compte tenu de notre souci de fournir une solution opérationnelle, nous préconisons donc une approche totalement intégrée, pour des raisons d'homogénéité, d'efficacité et de flexibilité. Cette solution consiste à développer des surcouches logicielles sur des environnements existants ouverts et adaptables, tels que certains systèmes auteurs hypermédias.

PARTIE 2

LE PROCESSUS DE DEVELOPPEMENT EN MILIEU INDUSTRIEL

Dans cette partie, nous allons illustrer notre propos à l'aide d'un cas réel. Notre équipe de recherche a participé avec la société CORYS, à l'élaboration et au développement d'un logiciel pédagogique expérimental, que nous appellerons "Alternateur". Le but de notre intervention se situait à deux niveaux. Le premier aspect consistait à formaliser et à évaluer les méthodes existantes de développement dans la société et fait l'objet du chapitre 4. La seconde partie de notre tâche consistait à expérimenter une nouvelle démarche permettant de mener à bien un développement dans un laps de temps très court (10 semaines). Elle est décrite dans le chapitre 5.

4. ETUDE D'UN PROCESSUS CONCRET DE DEVELOPPEMENT INDUSTRIEL

4.1 UN CAS REEL DE DEVELOPPEMENT D'UNE SIMULATION PEDAGOGIQUE

L'application " Alternateur " a pour objectif l'apprentissage des concepts de base du fonctionnement d'un alternateur, et est destinée à un public de techniciens travaillant dans des entreprises liées aux métiers de l'énergie. Elle propose à l'apprenant une base de connaissances, ainsi qu'une série de 15 simulations pédagogiques classées par ordre de difficulté croissante. Chaque exercice, basé sur la simulation, est consacré à l'étude d'un phénomène précis du domaine. A titre d'exemple, nous décrivons ci-dessous un exercice consacré au couplage d'un alternateur.

Exercice : Couplage d'un alternateur

Il s'agit d'amener un alternateur dans des conditions telles (vitesse de rotation, courant produit) qu'il puisse être raccordé sans dommage au réseau électrique national. En règle générale, cette opération est faite automatiquement, mais en cas de défaillance exceptionnelle, elle peut être réalisée de façon manuelle. Dans cette éventualité, l'opérateur devra agir sur différents actionneurs (potentiomètres, curseurs, ...) afin d'atteindre un état stable du système. Une fois qu'il estime avoir atteint cet état stable, ce qu'il peut vérifier à l'aide de certains dispositifs de visualisation (cadrans, synchronoscope, ...), il peut décider de raccorder l'alternateur au réseau à l'aide d'un interrupteur. Le raccordement effectif ne se produira qu'un

certain laps de temps après cette action. Si au moment du raccordement, les conditions ne sont pas favorables (caractéristiques du courant produit, déphasage), il peut s'ensuivre une détérioration plus ou moins grave de l'alternateur.

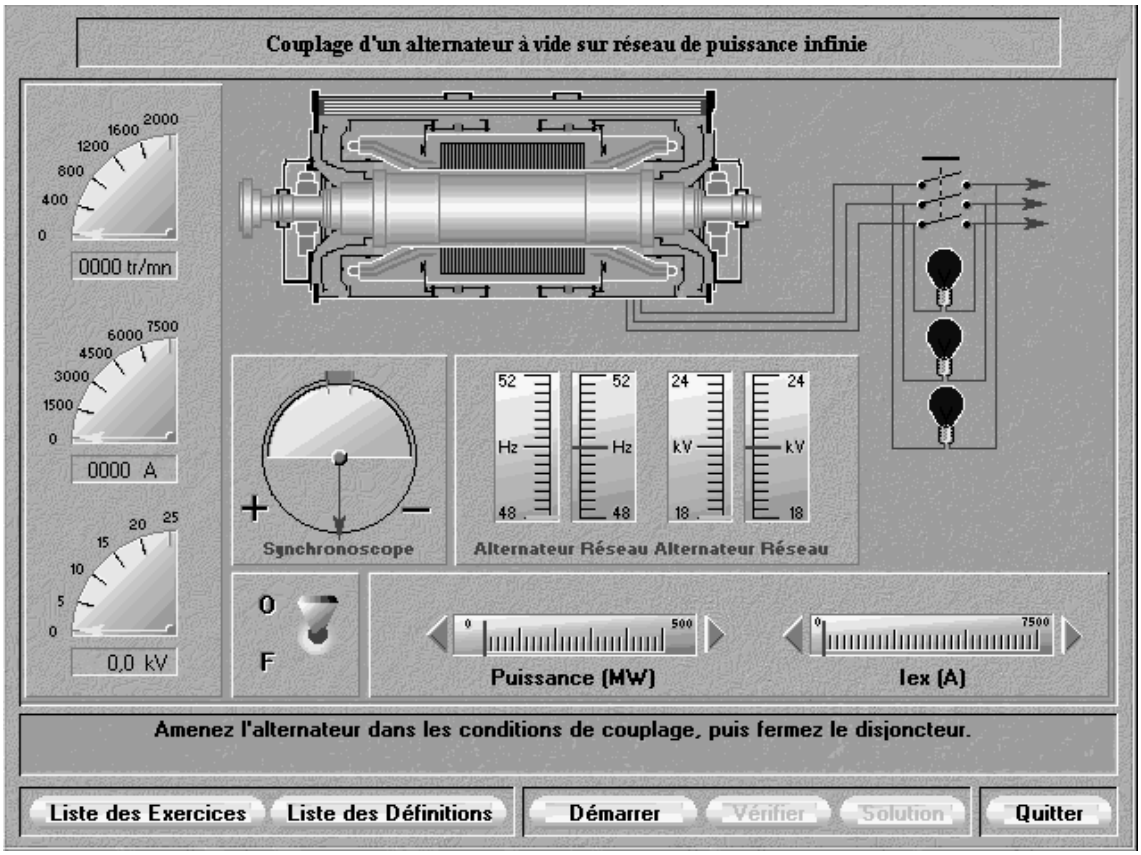


Figure 4-1 : Ecran couplage d'un alternateur

4.2 DEFINITION DES ROLES

La première tâche dans la définition d'un cycle de développement consiste à identifier les différentes personnes qui vont intervenir au cours du processus de développement. Nous distinguons les notions de rôle et d'acteur :

- Un rôle peut être considéré comme un ensemble cohérent de tâches pouvant être accomplies par une même personne physique aux compétences clairement définies (ou même éventuellement accomplies de façon automatique), à un moment déterminé du processus de développement ou d'utilisation de l'application.
- Un acteur représente une personne physique qui joue un rôle (ou éventuellement plusieurs rôles) au cours du processus de développement ou de l'utilisation de

l'application. Le cumul des rôles par un même acteur peut être dû soit à la nécessité de son intervention à plusieurs phases du processus, soit à un contexte particulier de développement nécessitant le regroupement de compétences variées par une même personne.

Dans le contexte spécifique de la société CORYS, nous avons identifié un ensemble de sept rôles intervenant dans le processus de conception de l'application "alternateur". Nous allons décrire chacun de ces rôles et les compétences requises, et nous préciserons les caractéristiques des acteurs correspondants.

4.2.1 La responsabilité du projet de développement

Dans un contexte industriel, la première décision porte sur l'intérêt de développer une application de simulation pédagogique, et l'évaluation du coût et du temps nécessaire à ce développement. Une fois la décision prise, il convient de s'assurer de la disponibilité des compétences nécessaires et d'organiser le développement. Tout au long de celui-ci, le responsable de projet doit assurer la synchronisation des différentes tâches et l'échange efficace d'informations entre les différents intervenants. C'est en fait le personnage central, le seul possédant une vue d'ensemble du projet. Il doit avoir les compétences usuelles d'un chef de projet ainsi qu'une bonne connaissance du domaine spécifique de la simulation pédagogique et des environnements de production.

> Dans notre exemple "alternateur", ce rôle a été tenu par un chef de projet de la société habitué à gérer le développement d'applications de simulation.

4.2.2 L'expertise technique du domaine.

Une application pédagogique a pour but la transmission de connaissances ou de savoir-faire concernant un domaine particulier. Il est donc le plus souvent nécessaire d'avoir recours, dès l'étape d'étude des besoins, à un expert dominant parfaitement les aspects techniques de son activité.

> Dans notre exemple, ce rôle a été occupé par un "modéliseur" de la société, expérimenté dans le domaine de la modélisation mathématique de phénomènes physiques liés aux métiers de l'énergie.

4.2.3 L'expertise en pédagogie du domaine

L’expert technique ne connaît pas forcément les meilleures approches pédagogiques. Ce rôle est donc celui d'un spécialiste de la pédagogie, possédant une maîtrise minimale du domaine abordé.

> Dans notre exemple, il a été fait appel à un professeur d'Université, spécialisé en électrotechnique, comme expert en pédagogie du domaine.

4.2.4 La spécification des aspects externes de l'application

Une des motivations de l'utilisation de simulations pédagogiques est le fait qu'elles offrent à l'apprenant des activités qui ne pouvaient pas être proposées dans le cadre de l'enseignement habituel. Le pédagogue "traditionnel" est une personne expérimentée dans le processus de transmission des connaissances et de savoir-faire. Mais cette seule qualité n'est pas suffisante pour la production d'une simulation pédagogique de qualité. La spécification des aspects externes nécessite donc d'exploiter au mieux, en accord avec le pédagogue, les possibilités des nouvelles technologies éducatives, et en particulier l'introduction de l'interactivité dans les applications produites.

> Pour l'application "alternateur", c'est le responsable de projet et moi-même qui avons spécifié les aspects externes du produit à développer et défini les différents types d'interaction proposés à l'élève.

4.2.5 La conception de l'architecture interne de l'application

Une fois la spécification des différents composants effectuée, il faut planifier les moyens informatiques à utiliser pour mettre en œuvre les solutions choisies. Ce rôle consiste à décrire l'architecture interne de l'application à développer, sa répartition en modules, et les interfaces logicielles permettant l'intégration ultérieure des différents composants.

> Pour l'application "alternateur", c'est moi-même qui ai tenu ce rôle de définition de l'architecture logicielle interne de l'application à produire.

4.2.6 Le développement informatique des applications

Une fois l'architecture globale conçue, l'application pédagogique doit être développée. Plusieurs types de compétences informatiques peuvent être mises à contribution : développeurs utilisant des langages de programmation classiques ou spécialisés, spécialistes du développement d'applications hypermédias. Cette diversité est dépendante de la complexité des techniques informatiques mises en jeu dans l'application.

> Dans notre exemple, l'équipe de développement était composée de deux programmeurs connaissant de façon approfondie le système auteur hypermédia "Toolbook" et de moi-même.

4.2.7 La médiatisation des applications

Nous avons vu que le développement d'une application pouvait nécessiter le recours aux techniques multimédias. Une utilisation rationnelle de ces techniques impose la participation de véritables spécialistes tels que : infographistes, spécialistes en création de documents sonores, vidéo,...

> Pour l'application "Alternateur", une infographiste est intervenue pour concevoir et réaliser aussi bien l'aspect général de l'application (choix des fonds, des caractères,...) que les graphismes techniques représentant les composants de l'application (alternateur, cadrans, etc.). D'autre part, la production et le traitement des documents sonores ont été effectués par l'équipe d'informaticiens.

4.3 ACCUMULATION DE ROLES PAR UN MEME ACTEUR

Dans l'exemple précis que nous venons de donner, chacun des rôles que nous avons présenté était tenu par une personne physique différente. Dans d'autres cas rencontrés, en fonction de la complexité et de la taille des applications à développer, certains de ces rôles peuvent être joués par la même personne. Nous présentons dans le tableau qui suit, les possibilités de cumul de rôles dans ce contexte industriel précis (cette présentation ne tient pas compte des cas exceptionnels de certains individus possédant des compétences a priori très différentes) :

	Cumul 1	Cumul 2	Cumul 3
Responsabilité de Projet	**OUI**		
Expertise du Domaine		**OUI**	
Expertise Pédagogique		**OUI**	
Spécification Aspects Externes	**OUI**		
Conception Architecture	**OUI**		
Développement informatique			**OUI**
Médiatisation des applications			**OUI**

Figure 4-2 : Cumul des rôles

Nous isolons donc trois éventualités de cumul de rôles par une même personne :

1. Cumul de la responsabilité du projet, de la définition des aspects externes et de la conception de l'architecture informatique de l'application. Il est fréquent dans les entreprises que les responsables de projet aient une formation antérieure de chef de projet informatique. Cette possibilité les pousse naturellement à cumuler ces fonctions, quand les conditions le permettent : projet de taille moyenne, bonne connaissance des environnements de production.
2. Cumul de l'expertise du domaine étudié et de l'expertise pédagogique. A priori, la connaissance approfondie d'un domaine technique ou scientifique par un expert ne le prédispose pas naturellement à enseigner ce domaine, et le recours à deux personnes différentes est alors nécessaire. Pour éviter cette lourdeur, on essaiera dans certains cas, de faire appel à un pédagogue suffisamment spécialisé pour maîtriser la complexité technique du domaine.
3. Cumul du développement informatique et de la médiatisation des applications. Le recours à des spécialistes en création ou traitement de graphismes, vidéo ou sons, n'est pas toujours indispensable. Selon la complexité de mise en œuvre de ces techniques dans les applications, et le degré de finition esthétique ou technique désiré, cette tâche de médiatisation des applications pourra être confiée à des informaticiens connaissant les environnements de production ou de traitement correspondant.

4.4 DESCRIPTION DU CYCLE EFFECTIF DE DEVELOPPEMENT

Une fois les rôles définis, il convient de décrire les tâches du processus du développement. Nous pouvons regrouper ces tâches en trois grandes phases : la phase d'étude des besoins, la phase de spécification et la phase de conception-réalisation.

L'actuel processus de développement adopté par la société CORYS est un cycle en V [BOE 76] allégé, où sont enchaînées séquentiellement les phases d'étude des besoins, de spécification et de conception-réalisation. A l'issue de chaque phase, un ensemble de documents est produit, définissant les résultats de cette phase:

- Les Dossiers de Définition des Besoins, établis à l'issue de la phase d'étude des besoins, décrivent les caractéristiques essentielles de l'application à développer.
- Les Dossiers de Spécifications, établis à l'issue de la phase de spécification de l'application, définissent dans des formalismes appropriés, les résultats obtenus lors de la spécification détaillée du modèle de simulation, des aspects pédagogiques et de l'interaction homme-machine de l'application.
- Les Dossiers de Réalisation et de Maintenance, établis à l'issue de la phase de conception-réalisation, documentent les choix de réalisation effectués : environnements de production utilisés, architecture détaillée des applications créées.

D'autre part, des mécanismes de test sont mis en place pour effectuer une validation à chacun des niveaux. On distingue ainsi :

- les tests unitaires, qui permettent de valider individuellement les composants créés séparément (parties du logiciel, documents multimédias).
- les tests d'intégration qui permettent de valider l'assemblage des différents composants.
- les tests fonctionnels qui permettent de valider les spécifications détaillées de l'application.
- les tests d'acceptance qui permettent de valider l'application dans son ensemble.

La figure 4-3 décrit le processus de façon globale.

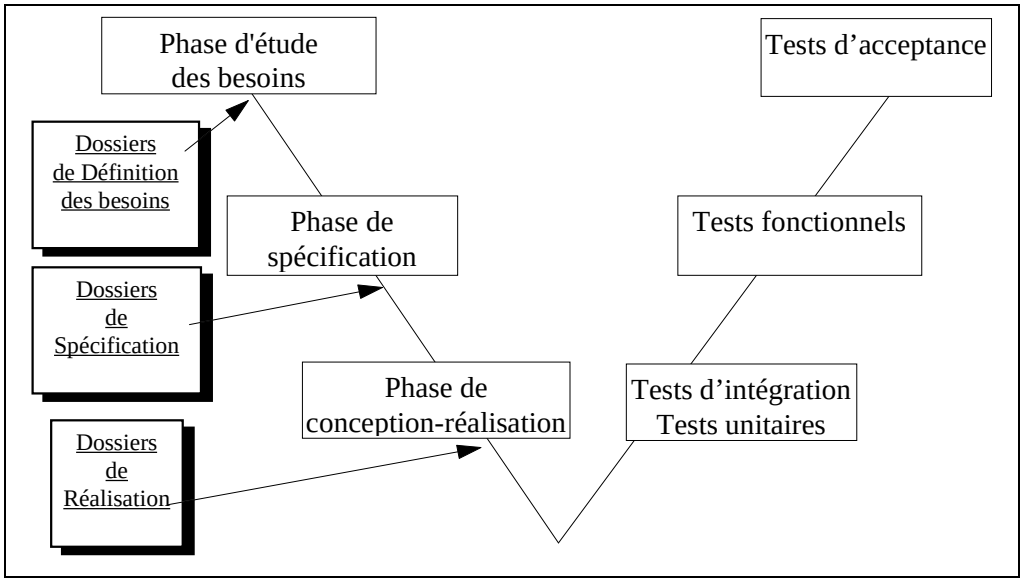


Figure 4-3 : Le processus global de développement actuel

Dans les paragraphes suivants, nous allons détailler les différentes tâches nécessaires à l'aboutissement d'une phase en précisant pour chacune d'elles : (1) une description de la tâche, (2) les différents rôles concernés et la nature de la coopération entre ces rôles et (3) les productions attendues en fin de tâche. Insistons sur le fait que notre présentation n'est pas normative, mais descriptive : elle synthétise le processus observé dans la Société CORYS.

Les conventions graphiques suivantes seront utilisées :

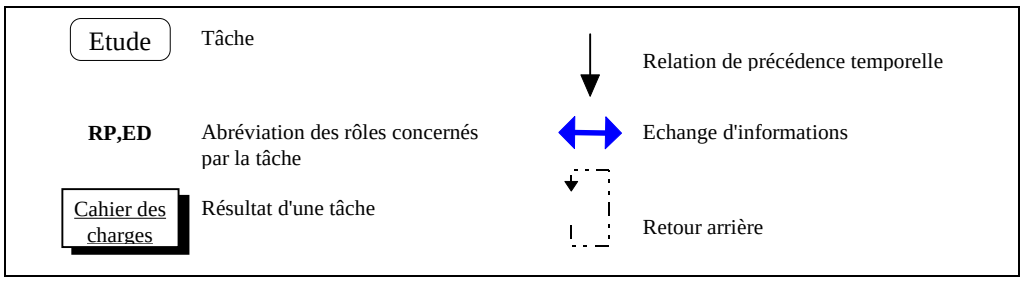


Figure 4-4 : Conventions de représentation graphique

4.4.1 La phase d'étude des besoins

4.4.1.1 L'étude d'opportunité

Description de la tâche

Comme avant tout développement informatique d'importance, la première tâche consiste à réaliser une étude d'opportunité. Cette étude a pour but d'apprécier de façon globale, la faisabilité technique du projet, les coûts à engager et les bénéfices escomptés. Les coûts peuvent être évalués approximativement en fonction de certains

critères tels que la complexité du domaine, la richesse des interactions proposées, la taille du logiciel, l'expérience acquise dans le développement de logiciels comparables etc. D'un autre côté, les bénéfices escomptés peuvent être estimés en fonction du prix de vente envisagé, du marché potentiel existant, etc.

Rôles concernés et nature de la coopération

L'étude d'opportunité est principalement du ressort du ***responsable de projet***. Il peut éventuellement prendre l'avis d'experts (expert du domaine, pédagogue, informaticien) pour préciser les caractéristiques de l'application à développer.

La production attendue en fin de tâche

Le document *"Etude d'opportunité"* rédigé par le responsable de projet définit de façon très globale les caractéristiques de l'application à développer, sa faisabilité technique, les coûts à engager et les bénéfices escomptés. C'est sur la base de ce document que sera prise la décision de développement effectif du logiciel.

4.4.1.2 L'élaboration du cahier des charges et du plan de développement

Description de la tâche

Cette tâche a pour but d'établir de façon précise les caractéristiques de l'application à développer et d'en prévoir le plus exactement possible le développement.

Rôles concernés et nature de la coopération

L'élaboration du cahier des charges est du ressort du ***responsable de projet***. L'exemple précis que nous traitons ici concerne le développement d'un produit proposé par l'entreprise. Le cas du développement d'une application sur commande requiert, bien entendu, la participation du *client* à cette phase d'étude du besoin. L'élaboration du cahier des charges peut également nécessiter le recours aux compétences de ***l'expert du domaine***, de ***l'expert en pédagogie*** ou du ***concepteur d'applications pédagogiques*** en vue d'évaluer précisément certains des aspects du logiciel à produire.

La production attendue en fin de tâche

Le document "*Cahier des charges"* établit les caractéristiques suivantes :

- définition précise du domaine ou sous-domaine étudié.
- définition des objectifs pédagogiques (public ciblé, niveau prérequis et niveau à atteindre) ainsi que les contraintes associées (contexte pédagogique d'utilisation du logiciel, type d'interactions proposées, etc.).
- définition globale de l'application : taille du logiciel, nombre de simulations proposées, choix et quantification du recours à certaines techniques multimédia (vidéo, infographie, enregistrements sonores, etc.).
- contraintes techniques (plate-forme d'utilisation, etc.).

Un second document appelé *"Plan de développement"* permet de décrire de façon précise :

- Le déroulement du projet dans le temps. Il fixe les dates-clés du projet : début et fin des phases de spécification, de conception-réalisation et de tests, mise sur le marché (ou livraison) du logiciel.
- Les ressources humaines à affecter sur le projet. L'équipe de conception et réalisation sera plus ou moins importante, selon le degré de complexité du logiciel à développer, le budget prévu, l'existence ou la disponibilité de compétences adaptées. Du fait de la forte spécialisation des tâches décrites, cette affectation est le plus souvent nominative. Quand c'est possible, les compétences sont trouvées à l'intérieur même de la société de développement. Sinon, il peut être fait appel à des personnes extérieures ayant les compétences requises.
- Les ressources matérielles à affecter sur le projet : machines, environnement de production logicielle, etc.
- Les aspects financiers relatifs au développement du logiciel.

4.4.1.3 L'organisation des tâches d'étude des besoins

Dans la phase d'étude des besoins, les tâches d'étude d'opportunité et d'élaboration du Cahier des Charges sont effectuées de façon séquentielle, le principal rôle concerné étant celui de responsable de projet.

La figure ci-dessous résume le déroulement de cette phase.

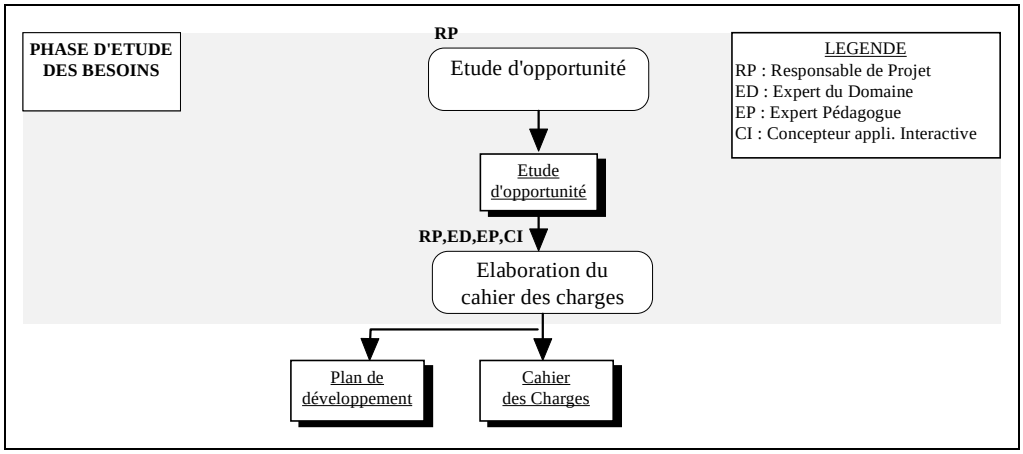


Figure 4-5 : La phase d'étude des besoins

4.4.2 La phase de spécification

Nous avons vu que de façon globale, le développement d'une application pédagogique nécessite trois types de compétences : les connaissances techniques propres au domaine, les compétences pédagogiques et les compétences informatiques. Cette dissociation se retrouve au niveau de la phase de spécification où nous pouvons distinguer quatre tâches :

- La spécification globale du scénario d'enchaînement pédagogique.
- La spécification détaillée des scénarios de contrôle pédagogique.
- La spécification détaillée des modèles de simulation.
- La spécification détaillée de l'interaction homme-machine.

Pour nous, la notion de scénario pédagogique se situe à deux niveaux. A un premier niveau, plus global, le *scénario d'enchaînement pédagogique* s'intéresse à définir les différentes unités pédagogiques ainsi que les relations qui les unissent. Au deuxième niveau plus détaillé, le *scénario de contrôle pédagogique de simulation* concerne la définition précise des interactions pédagogiques pour un exercice de simulation donné. Dans la suite de ce document, nous parlerons souvent plus simplement de *scénario d'enchaînement* et de *scénario de contrôle*.

Après avoir décrit chacune des tâches de spécification, nous examinerons les relations de précédence ou de dépendance qui existent entre elles.

4.4.2.1 La spécification globale du scénario d'enchaînement pédagogique

Description de la tâche

La spécification globale du scénario pédagogique consiste à définir précisément la démarche et les moyens à adopter pour atteindre les objectifs pédagogiques énoncés dans le cahier des charges. Dans notre contexte, la démarche pédagogique est fondée sur la présentation d'exercices interactifs basés sur la simulation. Il s'agira donc ici de définir un ensemble d'exercices permettant au public ciblé d'appréhender les concepts du domaine, et d'organiser ces exercices selon une progression pédagogique cohérente. Pour chaque exercice, on précisera de façon globale la nature du phénomène à simuler et l'interface qui en permet la manipulation et la visualisation.

Rôles concernés et nature de la coopération

La spécification globale du scénario est du ressort de *l'expert en pédagogie*. Dans certains cas, il pourra faire appel à *l'expert du domaine* afin de préciser la nature exacte de certains concepts techniques complexes, et de déterminer la validité technique des objectifs pédagogiques.

La production attendue en fin de tâche

- Le *dossier de spécification du scénario d'enchaînement pédagogique* définit de

façon globale l'enchaînement et la nature des exercices de simulation pédagogique. Ce dossier fournit à chacun des concepteurs un cadre général pour les activités de spécification détaillée de chacun des composants.

4.4.2.2 La spécification détaillée des scénarios de contrôle pédagogique

Description de la tâche

La spécification détaillée des scénarios de contrôle pédagogique consiste à définir les caractéristiques détaillées de chaque exercice de simulation. Cette définition tient compte de l'objectif à atteindre par l'élève, de la nature du contrôle ou du guidage à effectuer pendant le déroulement de l'exercice, des contraintes de temps associées, etc.

Rôles concernés et nature de la coopération

La spécification du scénario est du ressort de *l'expert en pédagogie*. Comme pour le scénario global, il pourra faire appel à *l'expert du domaine* afin de préciser la nature exacte de certains concepts techniques complexes, et de déterminer la validité technique des objectifs pédagogiques.

La production attendue en fin de tâche

- Pour chaque exercice, le *dossier de spécification détaillée d'un exercice* décrit le scénario de contrôle, c'est-à-dire les prérequis, les concepts abordés, l'objectif à atteindre et la description précise de la réactivité pédagogique du système au comportement de l'élève.

4.4.2.3 La spécification détaillée des modèles de simulation

Description de la tâche

La notion même d'exercice est basée sur la simulation. Comme nous l'avons vu dans le chapitre précédent, la simulation pédagogique consiste pour nous à permettre à un utilisateur d'observer et de manipuler une représentation partielle ou altérée d'un phénomène réel ou d'un concept abstrait. On appelle donc, dans notre contexte, *modèle de simulation* l'expression *abstraite* de cette représentation partielle à *l'aide de formalismes mathématiques, logiques ou temporels*. Il est à noter que pour un même domaine abordé dans une application pédagogique, il peut exister plusieurs modèles de simulation exprimant chacun des représentations plus ou moins partielles, plus ou moins altérées du domaine en fonction des objectifs pédagogiques poursuivis. Réciproquement, un même modèle de simulation complexe pourra être exploité au travers de représentations plus ou moins simplifiées.

Rôles concernés et nature de la coopération

La spécification des modèles de simulation est du ressort de *l'expert du domaine*. Sa

tâche est étroitement liée à celle de *l'expert en pédagogie* qui lui communique ses exigences pédagogiques : tel concept devant être abordé avant tel autre, tel phénomène doit être simplifié dans un premier temps avant d'aborder ses aspects plus complexes.

La production attendue en fin de tâche

Le *dossier de spécification des modèles de simulation* définit :

- L'ensemble des formalismes utilisés dans l'expression des modèles. Ces formalismes peuvent être, par exemple, la caractérisation d'un processus continu par ses variables d'entrée et de sortie et par les équations permettant de calculer l'évolution du processus dans le temps. Un autre exemple de formalisme peut être la représentation de processus dynamiques par des diagrammes états/transitions.
- Pour chaque exercice de simulation (ou éventuellement groupe d'exercices) : l'expression détaillée et non ambiguë d'un modèle de simulation en utilisant les formalismes décrits ci-dessus.

4.4.2.4 La spécification détaillée de l'interaction homme-machine

Description de la tâche

La spécification détaillée de l'interaction homme-machine de l'application de simulation nécessite une bonne connaissance des techniques de l'informatique interactive et du multimédia. La tâche consiste à établir précisément les interactions qui seront proposées à l'élève : navigation, manipulation directe, consultation ou contrôle de documents multimédia. Nous nous situons ici dans le domaine informatique classique de la conception d'interfaces.

Rôles concernés et nature de la coopération

La spécification informatique de l'application de simulation est du ressort du *concepteur d'applications interactives*. Celui-ci travaille en collaboration avec *l'expert pédagogue* qui établit la spécification du scénario pédagogique ; il indique à ce dernier les possibilités et limites techniques offertes par l'outil informatique (navigation, manipulation directe, interactivité, gestion de données multimédia) ; il prend en compte ses exigences pédagogiques dans la conception des interfaces.

D'autre part, le concepteur de l'application échange également des informations avec *l'expert du domaine* afin de valider l'adéquation entre les phénomènes réels ou les concepts abordés et la représentation qu'il compte proposer à l'utilisateur de la simulation.

La production attendue en fin de tâche

Le *dossier de spécifications externes de l'application* fournit :

- Une description très précise de tous les composants externes de l'application aussi bien en termes de fonctionnalités qu'en termes de présentation (définition des écrans, des graphismes utilisés, des interactions, de la nature et du contrôle des documents multimédias).

4.4.2.5 L'organisation des tâches de spécification

Une des principales caractéristiques de la phase de spécification réside dans la diversité des compétences requises et dans les relations étroites qui lient les différentes tâches identifiées. Nous pouvons faire les remarques suivantes :

- La tâche de spécification globale du scénario pédagogique est la tâche centrale : c'est elle qui conditionne les tâches de spécification détaillée.
- On peut prévoir une parallélisation des tâches de spécification détaillée. Les intervenants sont différents, et le cadre général a été fixé lors de la spécification globale du scénario pédagogique. Néanmoins, ce travail en parallèle demande un effort de communication d'informations entre les différents intervenants.
- Les résultats obtenus à la fin de chaque phase de spécification sont des documents formels. En particulier, le dossier de spécification d'un modèle de simulation, qui propose une description abstraite basée sur des formalismes précis, est prévu pour être exploité par l'équipe de réalisation. Il ne doit donc susciter aucune ambiguïté d'interprétation. D'un autre côté, rien ne garantit qu'une description formelle syntaxiquement juste, corresponde au modèle souhaité par l'expert du domaine, les compétences de l'équipe de réalisation ne lui permettant pas d'en vérifier la validité.

La figure 4-6 résume l'organisation des différentes tâches pendant la phase de spécification.

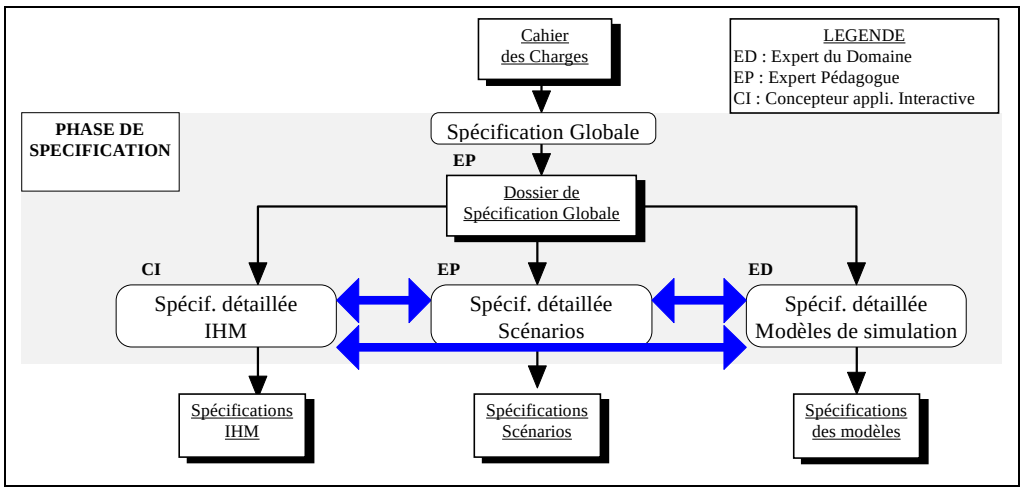


Figure 4-6 : La phase de spécification

4.4.3 La phase de conception-réalisation

La phase de conception-réalisation démarre une fois que les dossiers de spécification sont achevés. On entend ici par réalisation, la conception détaillée et le codage d'un des composants du logiciel. On peut ainsi distinguer :

- La conception globale de l'architecture de l'application de simulation.
- La réalisation des modèles de simulation.
- La réalisation de la partie interactive de l'application.
- La médiatisation de l'application.

Nous allons maintenant préciser chacune de ces tâches.

4.4.3.1 La conception globale de l'architecture de l'application de simulation

Description de la tâche

Une fois les spécifications terminées, il faut définir précisément les moyens informatiques qui seront utilisés lors de la production du logiciel, découper l'application en modules, en spécifier les interfaces logicielles et prévoir les modes de production de chacun d'eux. Cette tâche permet, en particulier, de fixer l'ensemble des règles qui permettront l'intégration ultérieure des réalisations partielles.

Rôles concernés et nature de la coopération

La conception globale de l'architecture de l'application de simulation est du ressort du *concepteur d'architecture*. Celui-ci se base sur les documents de spécification qui lui sont fournis.

La production attendue en fin de tâche

- Le *dossier d'architecture de l'application* définit l'architecture du logiciel à développer. Cette architecture prévoit le découpage en modules, la création des documents multimédias, ainsi que les modes de production et d'intégration. Ce document sert de cadre général aux activités de réalisation.

4.4.3.2 La réalisation des modèles de simulation

Description de la tâche

Parmi les tâches de réalisation, la réalisation d'un modèle de simulation a un statut particulier : il s’agit de traduire en termes informatiques des formalismes mathématiques, logiques ou temporels définis dans le dossier de spécification détaillée. Pour des raisons d'efficacité, cette traduction nécessite des langages ou

environnements supportant aisément ce type d'expression (bibliothèque de fonctions mathématiques, gestion des événements, gestion du temps).

Rôles concernés et nature de la coopération

Le codage d'un modèle de simulation est réalisé par un développeur informatique.

La production attendue en fin de tâche

- Le code informatique correspondant au modèle de simulation.

4.4.3.3 La réalisation de la partie interactive de l'application

Description de la tâche

Le seconde tâche de réalisation consiste à concevoir et coder la partie interactive de l'application, sur la base des spécifications détaillées de l'IHM et des scénarios de contrôle. Pour des raisons d'efficacité, ce développement nécessite l'utilisation d'environnements proposant des fonctionnalités avancées concernant la définition rapide d'interfaces graphiques, la navigation, la gestion des événements utilisateurs, la gestion de documents multimédias, etc.

Rôles concernés et nature de la coopération

Le codage de la partie interactive de l'application est réalisé par un développeur informatique maîtrisant un environnement de création d'applications interactives hypermédias.

La production attendue en fin de tâche

A l'issue de cette tâche, on dispose de l'ossature de l'application finale. On entend par ossature le fait qu'à ce stade ne sont intégrés ni les modèles des simulations, ni les documents multimédias. Par contre, sont développés les interfaces graphiques, les écrans, la navigation, les objets d'interaction.

4.4.3.4 La médiatisation de l'application

Description de la tâche

La médiatisation de l'application consiste à introduire des composants multimédias au sein de l’application de simulation. Ces documents (graphismes techniques ou artistiques, documents sonores, animations, documents vidéos, enregistrements de manipulation de logiciels, etc.) sont en général produits à l’aide d’environnements spécialisés.

Rôles concernés et nature de la coopération

Les documents multimédias sont réalisés par des spécialistes maîtrisant ces environnements spécialisés : infographistes, spécialistes du traitement du son, de la création et du montage vidéo, etc.

La production attendue en fin de tâche

Un ensemble de documents multimédias fournis sous forme de fichiers indépendants.

4.4.3.5 L'intégration de l'application

Description de la tâche

Les tâches de conception-réalisation décrites ci-dessus sont prises en charge par des personnes de compétences différentes et peuvent de ce fait être menées en parallèle. Une fois les résultats partiels produits de façon indépendante, il faut les mettre en correspondance, les intégrer pour constituer l'application finale, selon les règles établies dans le Dossier d'Architecture.

Rôles concernés et nature de la coopération

L'intégration des résultats partiels est du ressort d'un développeur informatique. Dans la majorité des cas, il s'agit de celui qui a développé la partie interactive de l'application de simulation.

La production attendue en fin de tâche

Le résultat de cette dernière tâche est en fait l'application finale.

4.4.3.6 L'organisation des tâches de conception-réalisation

Comme nous l'avons indiqué plus haut, les tâches de réalisation et de médiatisation peuvent être menées de façon parallèle. Une fois les résultats partiels obtenus, ils pourront être intégrés pour constituer l'application finale.

La figure ci-dessous résume l'organisation des différentes tâches pendant la phase de conception-réalisation :

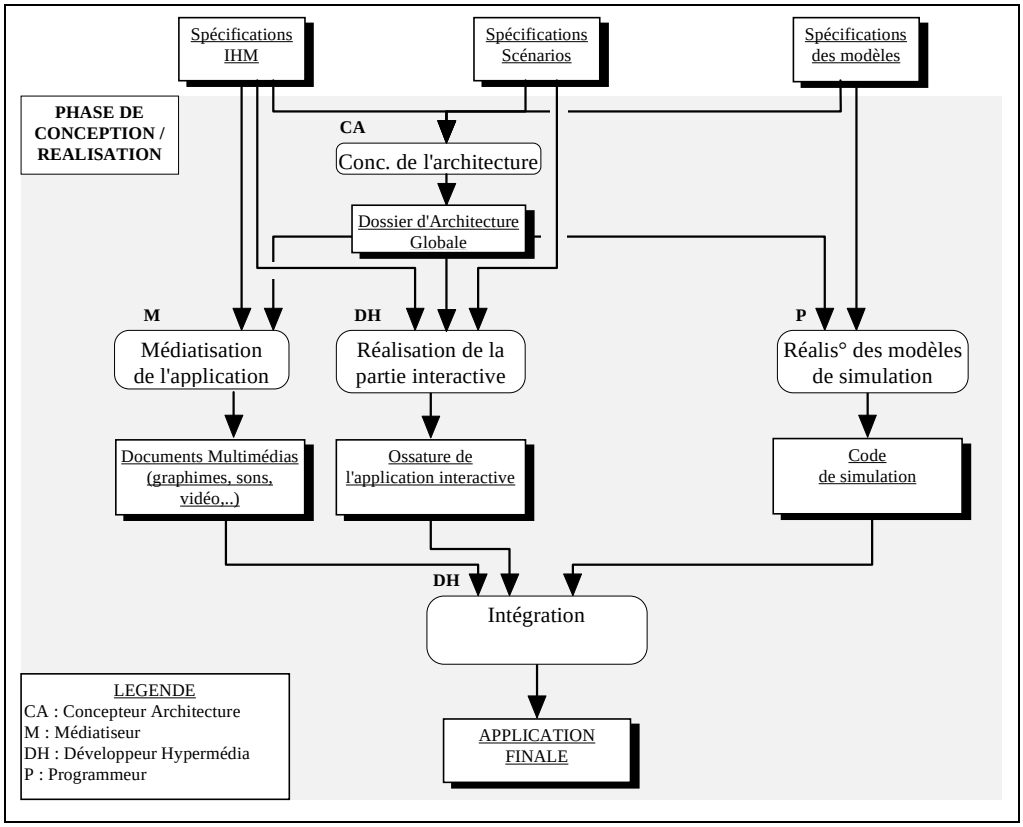


Figure 4-7 : La phase de conception-réalisation

4.5 LES PROBLEMES RENCONTRES

Le processus que nous venons de décrire, et qui reflète la démarche effectivement appliquée dans la société CORYS, est résumé par la figure 4-8.

La seconde partie de notre collaboration avec la société CORYS a consisté à évaluer et à optimiser ce processus dans le but de développer une application précise dans un temps très court.

Un certain nombre de problèmes sont apparus à divers moments du développement. Nous les détaillons dans les paragraphes suivants.

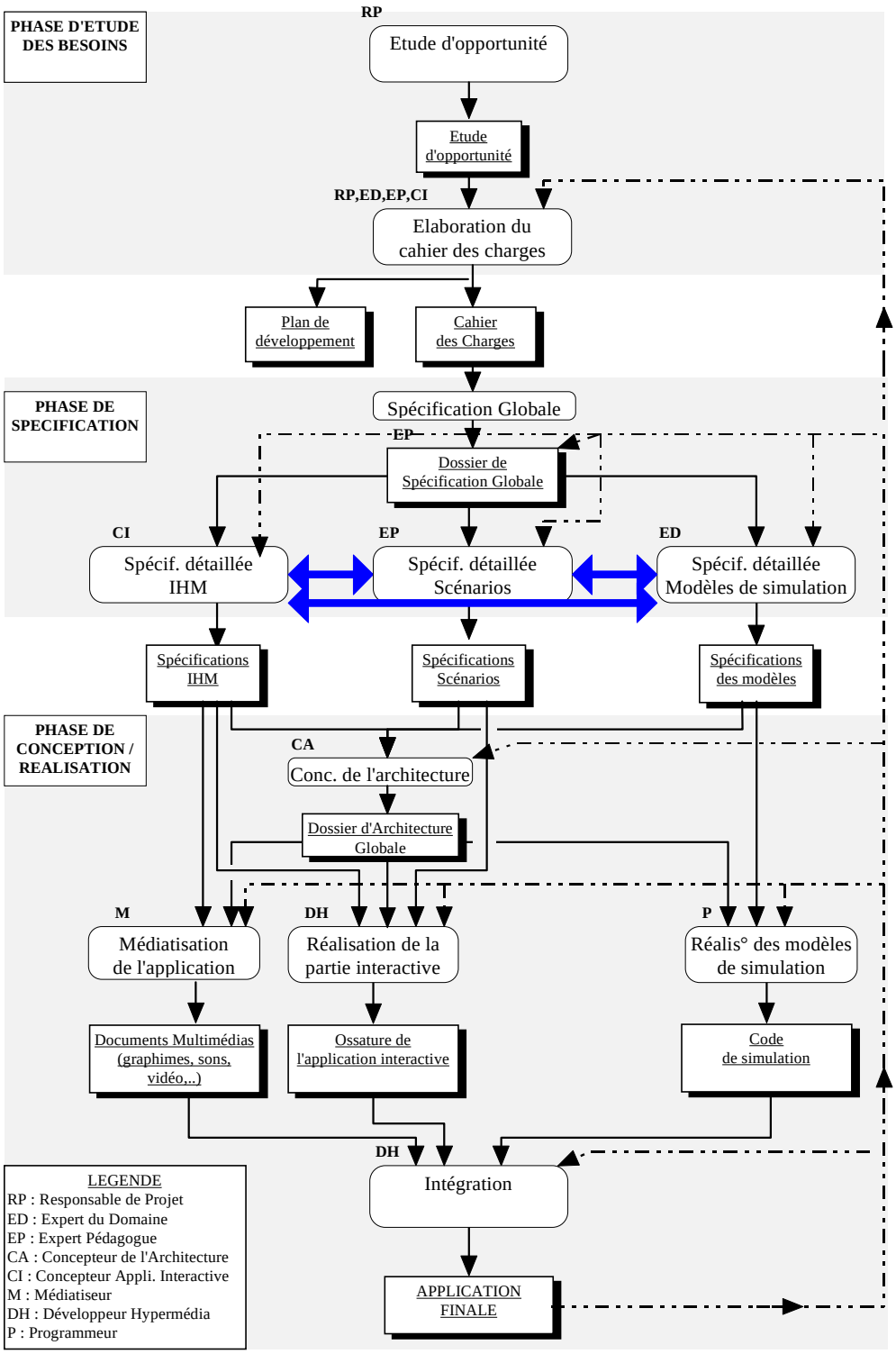


Figure 4-8 : Modélisation du processus actuel de développement

4.5.1 La formalisation des concepts

Un des premiers problèmes que nous avons rencontrés a été celui de la formalisation des concepts, en particulier ceux relatifs à l'expertise du domaine. Le modèle de simulation correspondant au couplage d'un alternateur est basé sur un

ensemble d'équations différentielles représentant l'évolution temporelle du système. L'expert du domaine a rédigé un document décrivant les variables utilisées et les équations permettant de calculer l'évolution du système. Mais ce document n'a pas pu être exploité directement car il subsistait de nombreuses ambiguïtés ou erreurs. Les ambiguïtés ou omissions étaient d'ordre syntaxique ou sémantique (symboles inconnus des non-spécialistes, absence de parenthésage, non affectation de constantes de temps, utilisation de noms identiques pour des variables différentes, etc.). Les erreurs concernaient des formalisations syntaxiquement justes mais ne correspondant pas à la solution souhaitée. De ce fait, un nombre important de réunions de travail a été nécessaire pour définir cette formalisation.

Une solution à ce type de problème consiste à définir des règles de formalisation permettant d'éviter les erreurs d'ordre syntaxique. Cette solution, si elle n'est pas automatisée, nécessite une validation manuelle des documents produits par l'expert du domaine. Mais ce genre de solution ne garantit en rien l'adéquation entre le modèle souhaité par l'expert et la formalisation produite. Cette adéquation ne pourra être vérifiée que plus tard, une fois l'application intégrée.

Ce problème de formalisation des concepts n'est pas propre à l'expertise du domaine, et se pose également dans l'expression des scénarios pédagogiques. Le pédagogue doit pouvoir décrire de façon non ambiguë l'enchaînement, le contenu et l'objectif des exercices.

D’une façon générale, on doit proposer, pour chaque type d’expertise, des formalismes d'expression appropriés pour décrire les concepts mis en œuvre.

4.5.2 La communication des informations entre acteurs

Un autre problème, corollaire au précédent, tient à la collaboration nécessaire entre divers spécialistes pendant la phase de spécification. Les différents formalismes doivent, dans certains cas, être mis en correspondance. Par exemple, l'objectif pédagogique "Placer l'alternateur dans des conditions acceptables de couplage" correspond à un prédicat logique portant sur les propriétés du modèle de simulation. Se pose donc ici le problème de la formalisation des différents types de coopération existant entre les différents espaces de conception.

4.5.3 Séquentialité des phases, parallélisme des tâches et retours arrière

Nous avons constaté que le processus utilisé par CORYS était basé sur un cycle séquentiel enchaînant les phases de spécification, de conception et de réalisation. A l'intérieur de chaque phase, on prévoit une certaine parallélisation des tâches. Dans notre contexte précis, cette approche amène les remarques suivantes :

- La séquentialité des phases d'étude des besoins et de spécification est nécessaire et offre des garanties de qualité du logiciel produit.
- La séquentialité des tâches de spécification et de conception-réalisation entraîne le fait qu'on ne dispose de résultats visibles, et donc testables, qu'après l'intégration finale des différents composants. Cette constatation, classique, est particulièrement pénalisante dans notre contexte. D'une part, nous nous situons dans une problématique de production rapide, et cette approche alourdit et ralentit sensiblement le processus de développement. D'autre part, nous avons isolé un ensemble de tâches de spécification bien différentes mais très interdépendantes. Une erreur de définition du modèle de simulation ne pourra être constatée qu'après l'intégration des différents composants et nécessitera un retour à la phase précédente (Cf. Figure 4-8). La modification à effectuer pourra avoir des répercussions sur le scénario pédagogique ou la spécification de l'interface, et entraînera une nouvelle mobilisation de compétences non obligatoirement disponibles et un nouveau parcours du cycle de développement. L'approche séquentielle ne garantit donc pas une gestion cohérente des relations d'interdépendance des tâches.

4.5.4 Dépendance des composants "scénario de contrôle pédagogique" et "partie interactive de l'application "

Il existe dans le processus décrit une relation forte de dépendance entre la spécification du scénario de contrôle pédagogique et celle de la partie interactive de l'application. En effet, dans notre type d'application, une part importante des interactions entre l'apprenant et le système sont de nature pédagogique : par exemple, la présentation d'une consigne, le contrôle par le système du comportement de l'apprenant face à l'objectif à atteindre, l'émission de messages en cas de réussite, d'échec ou de dépassement de temps. De ce fait, toute modification du scénario de contrôle pédagogique, même mineure, nécessite un nouveau parcours partiel du cycle de spécification et de conception-réalisation.

Cette dépendance existant dans le processus de développement masque la dissociation implicite qui existe entre la description abstraite du comportement attendu de la part de l'élève et la représentation fournie à l'élève lors de l'exécution de ce scénario. En particulier, deux interfaces identiques présentées à l'élève peuvent supporter deux scénarios d'exécution différents. Comme le montre la Figure 4-9, la conception d'un scénario de contrôle pédagogique d'une simulation donnée comporte deux aspects : la spécification fonctionnelle du déroulement du scénario d'un côté, et la définition de la représentation concrète de l'autre. Le découplage explicite de ces deux activités permet d'isoler le composant exécution de scénario qui pourra ainsi être testé de façon indépendante sans pour autant nécessiter la disponibilité de

l'application interactive.

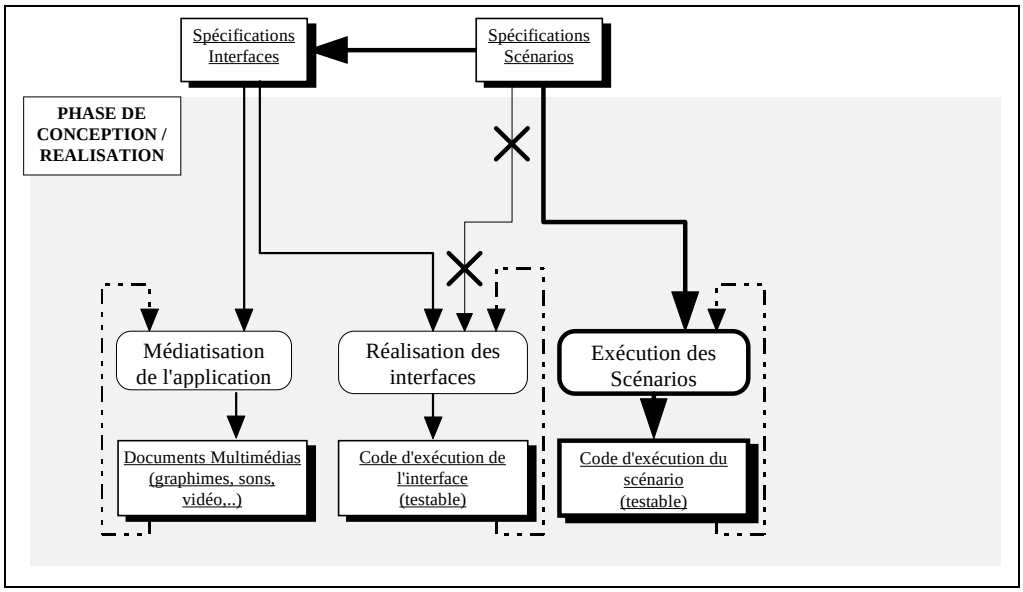


Figure 4-9 :Découplage des aspects fonctionnels et de représentation du scénario

4.6 BILAN DU DEVELOPPEMENT DE L'APPLICATION "ALTERNATEUR"

Dans le cadre du développement de l'application "Alternateur", nous avons apporté des solutions ponctuelles permettant de remédier aux problèmes évoqués ci-dessus.

- Nos efforts ont tout d'abord porté sur la parallélisation des tâches de spécification et de conception-réalisation. Nous avons introduit la possibilité pour chaque concepteur de disposer d'un résultat partiel visible de sa tâche en cours avant l'intégration finale des résultats. Par exemple, pour la tâche de spécification des modèles de simulation, nous avons formé une équipe constituée d'un expert du domaine et d'un programmeur pour pouvoir vérifier au plus tôt la validité des modèles spécifiés. Les résultats étaient visibles sur de petites maquettes dont les présentations n'avaient aucun rapport avec celles de l'application à développer. Cette démarche a permis une mise au point plus rapide des modèles et des retours-arrière de portée limitée.
- Un deuxième aspect concerne la formalisation des spécifications de chaque composant. Nous avons introduit un ensemble de règles d'écriture des documents de spécifications permettant une communication non ambiguë des informations entre les différents acteurs.
- Nous avons introduit le concept de module indépendant de spécification fonctionnelle du scénario pédagogique, afin de bien distinguer cette exécution, des

modalités techniques d'activation ou de visualisation de ce scénario.

La figure qui suit résume la nouvelle approche que nous avons utilisée. Dans ce schéma, ne figure pas la phase d'étude des besoins qui reste inchangée.

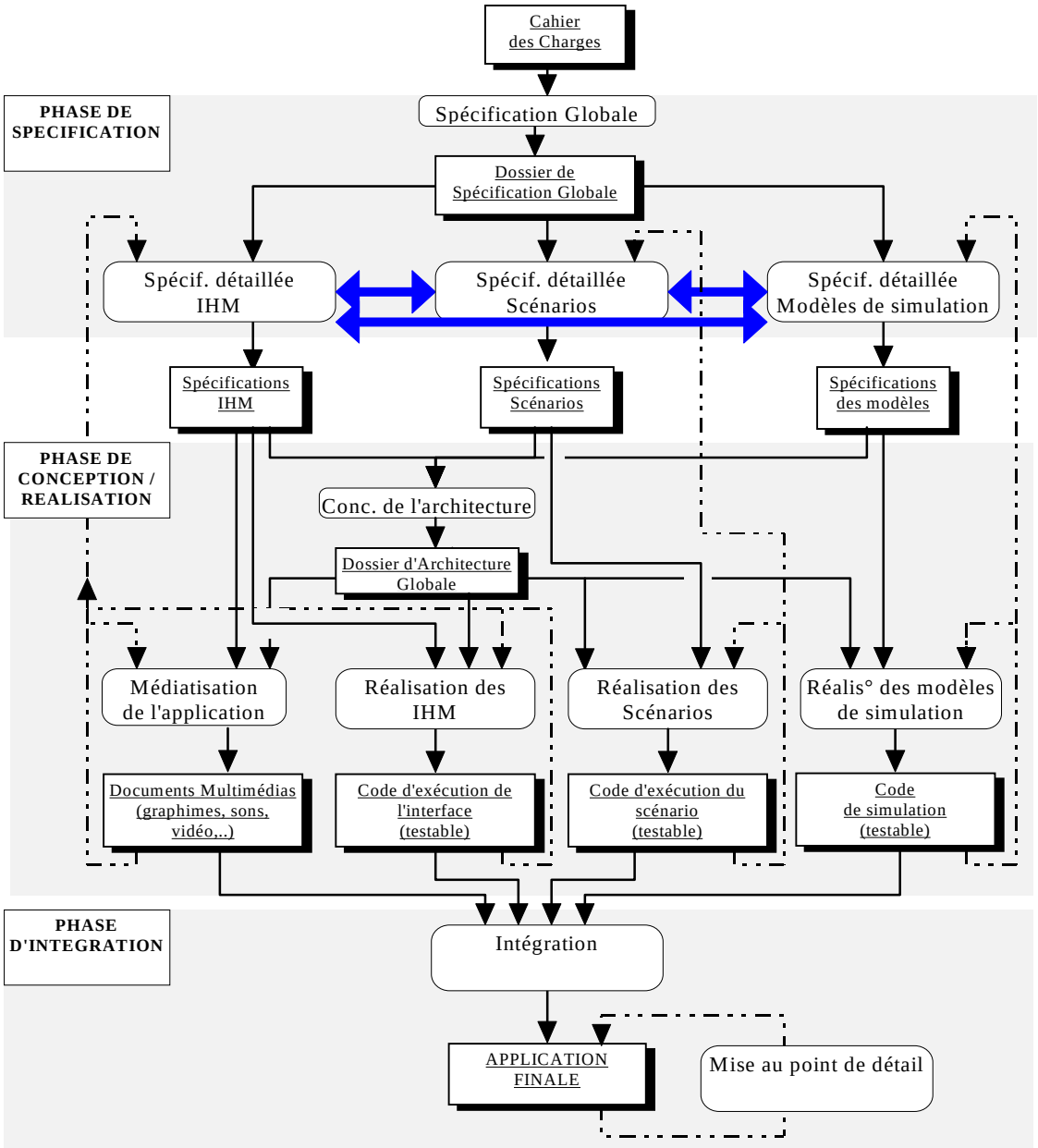


Figure 4-10: Une optimisation provisoire du cycle de développement

Les méthodes et formalismes que nous avons introduits dans le processus de développement ont permis d'atteindre l'objectif que nous nous étions fixés, c'est-à-dire la production de l'application "Alternateur" dans le temps prévu (10 semaines). Mais ce résultat n'a pu être obtenu que grâce à un investissement élevé. En particulier, la possibilité d'assurer une disponibilité quasi-permanente de certains intervenants (responsable de projet, informaticiens, expert du domaine, expert pédagogue) a

constitué un élément déterminant, mais non réaliste dans un contexte de développement visant la rentabilité. Cette disponibilité a permis notamment la collaboration directe des acteurs par le biais de réunions de travail.

4.7 RESUME

Nous avons décrit dans ce chapitre un exemple réel de développement d'application de simulation pédagogique. Il ressort de cette expérience les points suivants :

- Le développement de simulations pédagogiques nécessite la définition d'un processus de développement spécifique prenant en compte la parallélisation des diverses tâches de spécification.
- Les différentes tâches de spécification peuvent être en partie réalisées de façon indépendante, avant d'être intégrées dans l'application finale. Il faut en particulier proposer à chaque type de concepteur (scénario pédagogique, modèle de simulation, partie interactive de l'application) un moyen de tester isolément la validité de ses spécifications.
- La coopération indispensable entre les différents intervenants nécessite une formalisation précise des spécifications de chaque composant et la compatibilité des divers formalismes retenus.
- Des méthodes basées sur la communication directe entre les intervenants et la communication par documents formalisés permettent un gain de temps appréciable. Par contre, ces méthodes entraînent un surcoût important et une difficile mise en place dans un contexte industriel, dus en particulier à la mobilisation simultanée d'experts de compétences diverses pendant un laps de temps non négligeable.
- L'utilisation d'un environnement de production proposant l'automatisation partielle des tâches de conception-réalisation et offrant des mécanismes de coopération apparaît comme une solution intéressante en vue de la réduction de ces coûts.

Nous allons donc étudier dans le chapitre suivant la notion d'environnement intégré adapté à la production d'applications de simulations pédagogiques.

5. PROPOSITIONS D'UN CYCLE ADAPTE ET D'UN ENVIRONNEMENT INTEGRE

Les difficultés que nous avons rencontrées dans le cadre d'un développement expérimental et que nous avons présentées au chapitre précédent, nous ont permis de soulever un certain nombre de problèmes. Ces problèmes peuvent être résolus, partiellement et à un coût élevé, par l'introduction de méthodes plus ou moins empiriques. Une autre approche consiste à fournir des solutions plus globales par :

- la définition d'un cycle de développement réellement adapté au contexte que nous avons décrit.
- la mise à la disposition de l'équipe de développement d'un environnement intégré de production de logiciel, non pas générique mais adapté à une classe de problèmes bien définie : la production de simulations pédagogiques.

5.1 ETUDE DE CYCLES DE DEVELOPPEMENT

Nous avons décrit dans le chapitre précédent un processus réel de développement. Ce processus répartit les différentes tâches selon un découpage séquentiel en phases de développement (Etude des Besoins, Spécification, Conception-Réalisation). La solution provisoire que nous proposons afin d'améliorer ce processus, tend à organiser les tâches selon un autre critère : celui du domaine de compétence auquel elles se rapportent. On peut, par exemple, regrouper toutes les tâches liées au modèle de simulation : spécification, conception, réalisation, validation. La figure 5-1 illustre de façon générale les différences entre ces deux

modes de répartition.

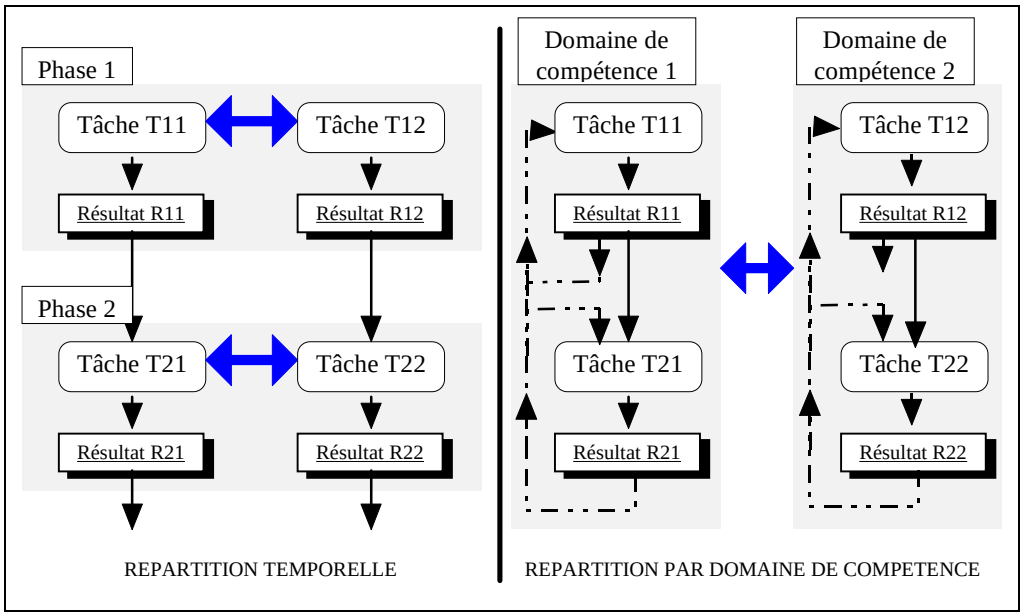


Figure 5-1 : Répartition temporelle et répartition par domaine de compétence

Examinons maintenant comment ce type de répartition des tâches par domaine de compétence peut être pris en compte par les modèles de cycles de développement.

5.1.1 Les différents types de cycles de développement

Les modèles de cycles de développement peuvent être répartis en deux grandes classes : l'approche séquentielle, et l'approche évolutive ou incrémentale.

5.1.1.1 Approche séquentielle

L'approche séquentielle est la plus ancienne, et a constitué une tentative de réponse à la "crise du logiciel" dénoncée dans les années 70 [ROY 70, BOE 76]. Le paradigme séquentiel suppose que la construction et le développement d'un logiciel se résument à une succession de tâches disjointes dans le temps : Analyse des Besoins, Spécification du Logiciel, Conception Globale, Conception Détaillée, Codage, Test et Maintenance. Le modèle en cascade de Boehm [BOE 76] admet uniquement le retour d'une phase à la phase immédiatement précédente (Cf. Fig. 5-2). A partir de ce modèle de base, toute une gamme d'autres modèles utilisant le même paradigme a été déclinée : modèle en V, avec ou sans retour, possibilité de parallélisation de sous-tâches, etc.

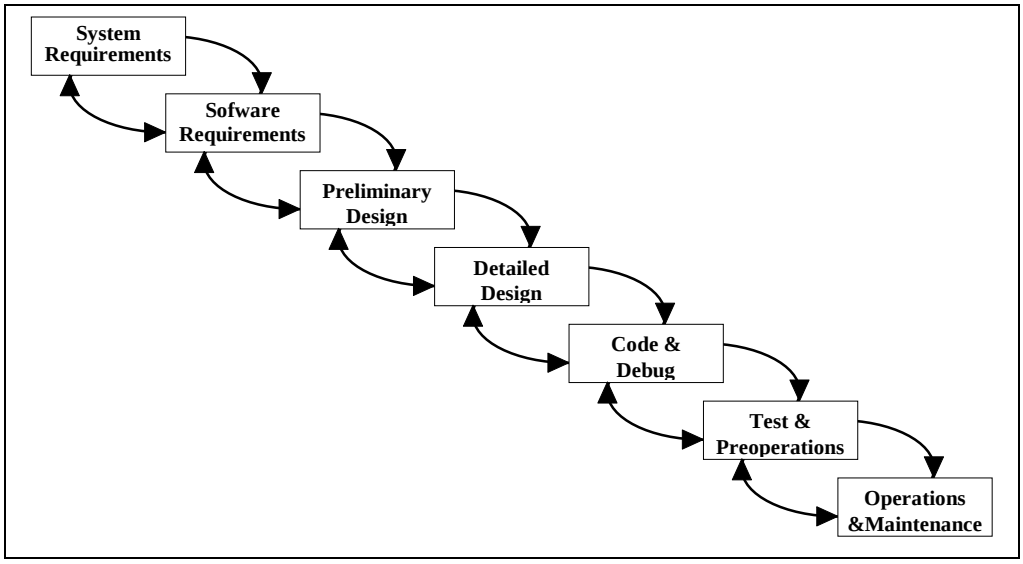


Figure 5-2 : Le modèle en cascade de Boehm

Cette approche séquentielle a pour avantage de fournir un cadre général précis qui délimite clairement les activités principales du développement logiciel. Elle peut être mise en œuvre sans se soucier ni du domaine d'application, ni de la taille ou de la complexité des logiciels, et incite à une structuration certaine de l'application.

En revanche, le modèle de base suggère que l'idée même d'itération dans le processus de développement est exceptionnelle, ce qui est faux dans la réalité. D'autre part, la délimitation très stricte entre les phases du processus (en théorie, aucune phase ne peut en couvrir une autre) est une vue de l'esprit. Dans la pratique, les interactions entre les différentes phases sont beaucoup trop complexes pour être exprimées par un simple modèle séquentiel d'entrée-sortie.

5.1.1.2 Approche évolutive ou incrémentale

Une réponse aux faiblesses des modèles séquentiels a consisté dans l'approche dite *évolutive*. Un constat fréquent a été qu'avec l'application stricte d'un modèle séquentiel, on devait être prêt en cas d'échec, à redévelopper l'application. "Do it twice" est un principe évoqué par Brooks [BRO 75].

Boehm [BOE 86] définit le modèle de processus évolutif comme un "*modèle dont les étapes consistent à enrichir progressivement un produit logiciel opérationnel, en fonction de l'évolution déduite par des expérimentations opérationnelles*". Les résultats des étapes intermédiaires du logiciel, ou *incréments,* peuvent être livrés au client. La stratégie du développement évolutif peut être résumée par les principes suivants [GIL 88] :

1. *Fournir* quelque chose à l'utilisateur.

2. *Evaluer* le bénéfice qu'en retire l'utilisateur.
3. *Ajuster* la conception et les objectifs à partir des réalités observées.

Sur la base de cette approche évolutive, Boehm a proposé un cadre de travail avec le modèle en spirale [BOE 86] qui se concentre essentiellement sur la gestion des risques avec un modèle cyclique composé de 4 *étapes* récurrentes, chaque itération constituant une *phase* d'évolution du logiciel. On distingue ainsi les phases de Conception de l'opération, de Spécifications externes, d'Architecture logicielle et de Conception détaillée et réalisation. Chaque *phase* comprend les *étapes* suivantes :

- Etape 1 : Identification des objectifs relatifs à la phase de développement concernée, étude des alternatives et de leur répercussion sur l'application.
- Etape 2 : Identification des risques entraînés par chaque alternative et évaluation de ces risques grâce à l'élaboration de prototypes.
- Etape 3 : Développement et vérification de la phase en cours.
- Etape 4 : Evaluation des phases déjà parcourues et planification de la prochaine phase, c'est-à-dire de la prochaine itération du cycle.

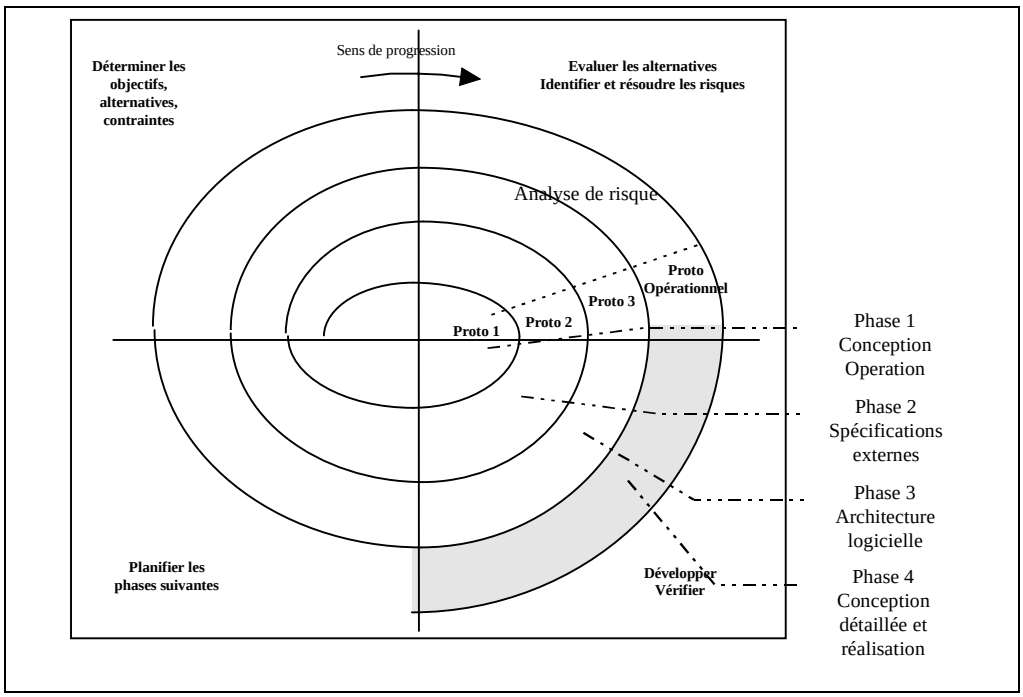


Figure 5-3 : Le modèle en spirale de Boehm [BOE 86]

Le modèle en spirale n'est pas à proprement parler un modèle de cycle de vie mais plutôt un méta-modèle, dans la mesure où il peut s'accommoder de différents cycles concrets de développement.

5.1.2 L'approche prototypage

5.1.2.1 Définitions : prototypage et prototype

Nous avons décrit avec l'approche incrémentale, la possibilité de valider certaines étapes à l'aide de prototypes. L'approche par prototypage en tant que telle ne peut pas être définie comme un troisième type de cycle de développement ; selon les auteurs, son utilisation peut être faite ou dans un contexte relativement séquentiel, ou dans un contexte incrémental. Tout d'abord, cette notion paraissant vague, Bischofberger [BIS 92] se propose de définir clairement les différentes acceptions des termes "*prototypage*" (la tâche qui consiste à créer un prototype) et "*prototype"* (le résultat du prototypage).

Pour "*prototypage*", il reprend la classification proposée par [FLO 84] et distingue :

- *le prototypage exploratoire*. Le but de ce type de prototypage est de préciser le plus complètement possible les *besoins de l'utilisateur* en lui offrant des exemples réalistes du produit final qu'il espère obtenir. Les facteurs les plus importants dans ce cas ne sont pas la robustesse du prototype produit, mais les fonctions qu'il remplit, et sa capacité à être modifié très rapidement.
- *le prototypage expérimental*. Relatif à *l'architecture du système* à créer, son but est de valider expérimentalement la pertinence des choix de conception globale, des modèles d'architecture et d'interactions entre composants logiciels. Basé sur des exemples concrets, le prototypage expérimental permet de vérifier les interfaces de composants individuels, la souplesse et l'évolutivité de l'architecture générale. De même que pour le prototypage exploratoire, la qualité du prototype n'est pas essentielle.
- *le prototypage évolutif*. Son but est le *développement incrémental d'un système*. Il propose l'approche suivante : un premier prototype est développé pour répondre aux besoins initiaux de l'utilisateur. Le résultat est utilisé comme système de base pour l'élaboration de l'étape suivante durant laquelle de nouveaux besoins sont pris en compte. Cette approche ne fait pas de différence formelle de nature entre les prototypes successifs et le produit final. Pourtant, la pratique démontre qu'il se produit souvent une détérioration, progressive mais certaine, de l'architecture interne du logiciel au fur et à mesure de son développement. Mais le choix de la redéfinition totale de l'architecture logicielle en vue d'assurer la robustesse du produit final, est souvent rejeté pour des raisons financières.

Le second terme à définir est celui de "*prototype"*. Boar [BOA 84] donne la définition suivante : "*Un prototype est un modèle opérationnel, modifiable et extensible d'un système donné, non nécessairement représentatif du système complet, et qui fournit aux utilisateurs de l'application finale une représentation physique des*

principaux aspects du système avant son implémentation.". Parmi les différents prototypes, Bischofberger distingue:

- les prototypes complets ou incomplets. Un prototype est *complet* lorsque toutes les fonctions significatives du système à développer sont présentes et complètes. Un prototype *incomplet* ne permet d'étudier qu'un des aspects du logiciel à créer (par exemple, l'interface utilisateur, l'architecture du système, etc.).
- les prototypes jetables ou réutilisables. Un prototype est *jetable* quand les composants développés lors de sa création ne sont pas réutilisés lors de l'implémentation du produit final, mais servent uniquement de modèles de validation. Un prototype est *réutilisable* lorsque la majeure partie de ses composants ont été développés en respectant des règles cohérentes de qualité logicielle et qu'ils peuvent être intégrés dans l'application finale.

5.1.2.2 Approche par prototypage et cycle de vie du logiciel

Une fois ces définitions données, qu'en est-il du cycle de vie prenant en compte l'approche par prototypage ? Sur ce point, deux écoles paraissent s'affronter : l'une prétend comme Bishofberger [BIS 92] que "*la stratégie de développement orienté-prototype d'un logiciel ne diffère pas radicalement de la stratégie séquentielle orientée-cycle"*, et l'autre plus radicale espère comme Balzer [BAL 83] que cette approche peut profondément modifier les habitudes de développement de logiciel en "*reportant les aspects non créatifs de modification et de maintenance de l'homme vers la machine*".

La première tendance défend donc l'idée que l'approche séquentielle et l'approche par prototypage sont davantage complémentaires qu'alternatives. La principale différence réside dans le fait que les phases sont parcourues plutôt itérativement que séquentiellement et que les endroits où sont effectuées ces itérations sont précisément localisés comme le montre la figure 5-4.

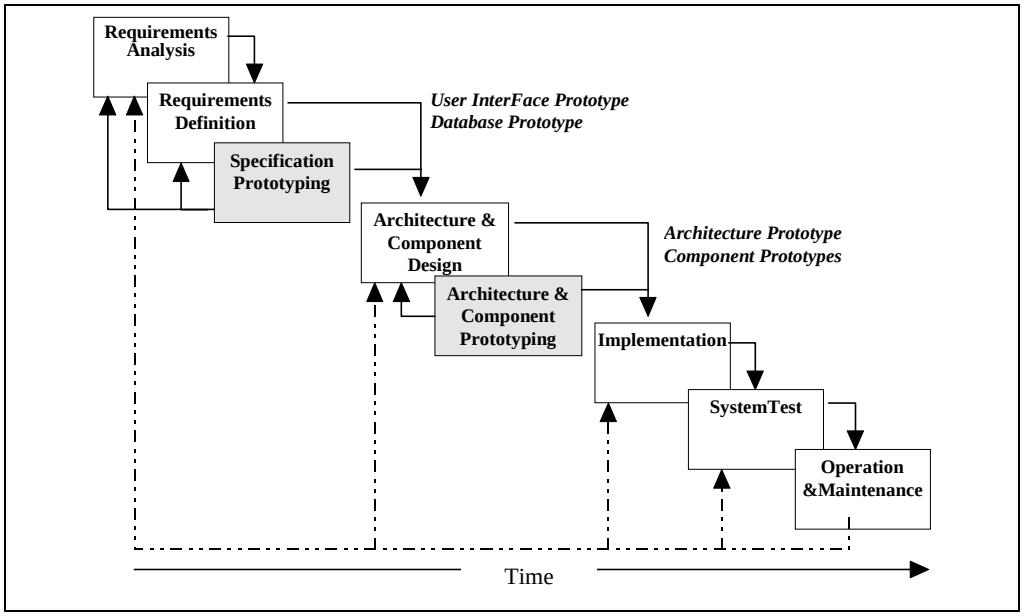


Figure 5-4 : Le cycle de vie "orienté-prototype"

Le second point de vue se démarque du précédent en défendant l'idée que le prototypage ne peut prendre tout son sens qu'en lui adjoignant le concept de génération automatique de code. Balzer préconise même l'idée d'un nouveau *paradigme "basé sur l'automatisation"*.

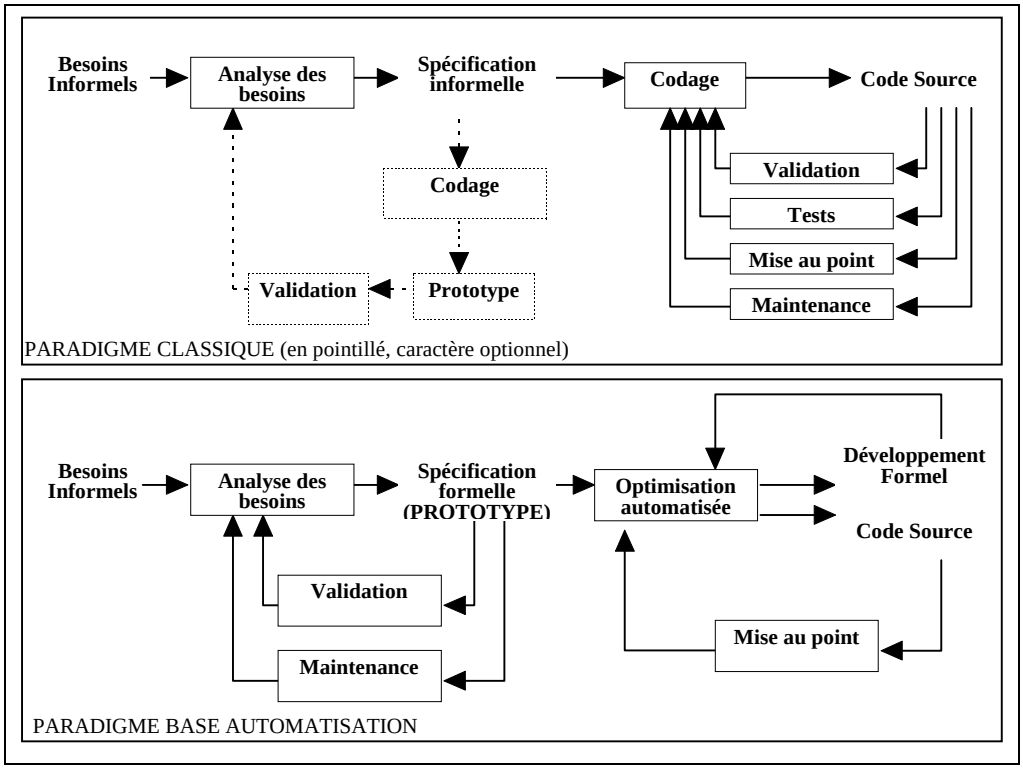


Figure 5-5 : Comparaison des paradigmes "classique" et "basé sur l'automatisation", proposée par Balzer dans [BAL 83]

Balzer [BAL 83] part de la constatation que les tâches d'implémentation, de modification et de maintenance de code sont des tâches répétitives, non-créatives et sources d'erreurs dans le logiciel. Il propose que les modifications d'un logiciel ne s'opèrent plus sur l'implémentation (le code du programme) mais sur les spécifications, le code étant généré automatiquement à partir de ces spécifications basées sur des formalismes clairement définis. Il préconise donc que les spécifications soient "opérationnelles", qu'elles puissent être "exécutées" directement par l'utilisateur à travers un prototype, et qu'elles permettent ainsi une validation au plus tôt. A partir de ce prototype ayant servi à la spécification, l'implémentation, qui doit être rapide, fiable et de faible coût, est réalisée de façon automatique. Les avantages avancés de cette démarche sont donc les suivants :

- La spécification est réalisée de façon formelle
- La spécification est le prototype
- Le prototype est validé de fait
- Le prototype devient l'implémentation
- L'implémentation est automatisée
- Une partie des tests est éliminée
- La maintenance s'opère sur la spécification
- La documentation est produite automatiquement
- La maintenance est assurée par un nouveau cycle automatisé

Nous allons maintenant nous intéresser aux environnements permettant de mettre en œuvre une telle approche.

5.2 LE CONCEPT D'ENVIRONNEMENT INTEGRE DE PRODUCTION DE LOGICIEL

Depuis une vingtaine d'années, d'innombrables méthodes, formalismes ou outils se préoccupent d'améliorer la tâche des producteurs de logiciel. Ce foisonnement a débouché sur une certaine atomisation de l'offre en de multiples secteurs spécialisés, où chacun s'efforce de proposer une solution basée sur son propre point de vue.

Le concept *d'environnement* se réfère à une collection d'outils matériels ou logiciels qu'un développeur, ou une équipe de développeurs, utilise pour produire des systèmes logiciels. La prolifération des outils s'est accompagnée d'une multiplication des incompatibilités de toutes natures, de recouvrement ou de duplication des fonctionnalités, de différences de style de manipulation. Le problème de *l'intégration* de différents outils au sein d'un même environnement s'impose donc aujourd'hui comme une question majeure.

Pour [LON 92], un Environnement Intégré de Production de Logiciel (EIPL) doit offrir à ses utilisateurs :

- une grande *diversité* des services, touchant aux différentes *phases* du cycle de fabrication des logiciels et aux différents *rôles* à assurer.
- une réelle *homogénéité* de ces services, par exemple en termes d'uniformité des interactions homme/machine et d'uniformité de l'assistance apportée.
- un grand nombre de règles visant à assurer un *niveau élevé de qualité*.

L'état actuel de l'offre en moyens de production de logiciel laisse apparaître un éventail très diversifié de solutions, souvent basées sur des méthodes empiriques, ne couvrant que tel ou tel aspect du développement, ou n'offrant qu'un ensemble restreint de fonctionnalités.

Le déséquilibre indiscutable qui existe entre les besoins en logiciel, et les moyens disponibles pour satisfaire ces besoins est une cause souvent évoquée dans la situation de crise du logiciel [BOE 81, DEM 82]. Il est admis aujourd'hui que cette crise ne se résoudra pas par l'effet d'une solution miracle mais grâce à une *convergence d'efforts très divers* parmi lesquels [LON 92] :

- *l'analyse des processus de développement*, suivant les domaines d'applications, pour en acquérir une meilleure maîtrise.
- *l'automatisation de la production* autant qu'il est possible.
- la *réutilisation* à grande échelle des composants existants.
- la *délégation* d'une partie de la production du logiciel *aux utilisateurs finaux*, grâce à des moyens adaptés.
- *l'amélioration de la communication homme-machine*.

Les Environnements Intégrés de Production de Logiciel doivent apparaître comme les fédérateurs de ces solutions partielles. Le problème de la définition de l'intégration a été abordé par certains auteurs. Retenons l'approche de Waserman [WAS 89] qui propose cinq niveaux d'intégration pour le développement de projets de taille importante : plate-forme, présentation, données, contrôle et procédés. Parmi ces niveaux, deux nous intéressent particulièrement :

- intégration au niveau de la présentation : l'interface utilisateur doit être uniforme pour tous les outils de l'environnement aussi bien en ce qui concerne les mécanismes que les règles de manipulation.
- intégration au niveau des données : les outils doivent pouvoir se partager des objets communs et doivent pouvoir s'échanger les objets qu'ils génèrent.

Si la mise en place du premier aspect ne présente pas de difficulté majeure, l'intégration au niveau des données a donné naissance à différentes approches.

L'approche dite "*fortement couplée*" permet les communications entre les outils de l'environnement par le partage d'une base de données, accessible dynamiquement. Cette solution est utilisée dans des environnements interprétatifs plus adaptés à un usage individuel par des développeurs de haut niveau qu'à une production industrielle. Une autre approche dite "*faiblement couplée*" préconise une systématisation des communications par la définition d'un protocole commun auquel doit souscrire chacun des outils de l'environnement [ISO 89, FER 89b].

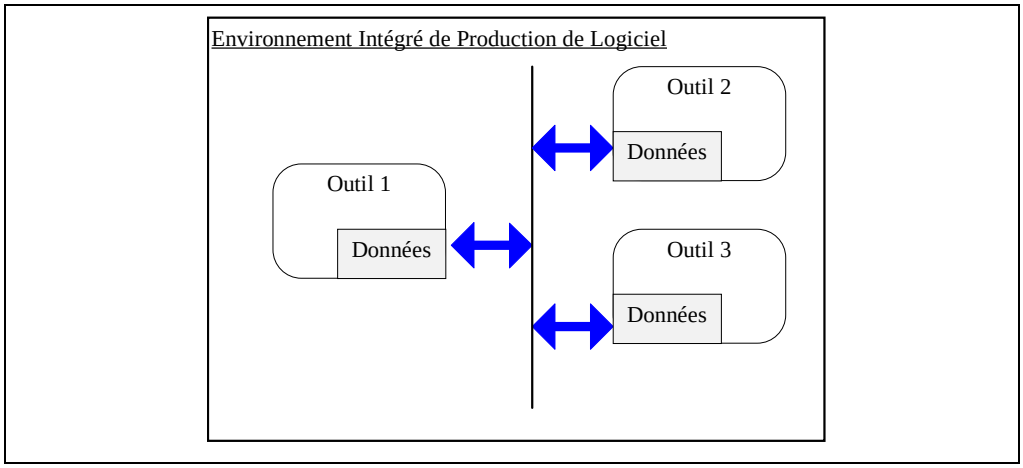


Figure 5-6 : La communication des données dans l'approche faiblement couplée

C'est cette dernière approche que nous retiendrons. Nous avons en effet répertorié dans notre problème un ensemble de rôles (spécification du scénario, spécification des modèles de simulation, spécification de l'IHM de l'application) qui sont :

- *indépendants*, dans la mesure où chaque concepteur doit résoudre une certaine partie du problème global en manipulant ses propres données ou formalismes.
- mais également *étroitement liés*, dans le sens où chaque concepteur a besoin d'avoir accès à certaines données ou objets définis par d'autres intervenants en vue d'intégrer ses résultats dans l'application finale.

Résumons maintenant nos objectifs. Les critères que nous exigeons pour la réelle efficacité d'un tel environnement sont :

- *l'adéquation réelle* entre les services proposés par les outils, d'une part, et les tâches, habitudes de travail, et compétences de chaque utilisateur.
- l'*homogénéité des services proposés*, en particulier en ce qui concerne les interactions homme-machine.
- La *prise en compte des exigences de coopération*. Cette prise en compte nécessite l'existence dans l'environnement de mécanismes et formalismes permettant

l'échange ou l'exploitation de données entre les différents outils.

Pour arriver à la définition d'un tel environnement, nous nous fixons pour objectifs :

- *la définition d'un processus de développement adapté à notre problème*, visant à déléguer à chaque utilisateur d'outil, partout où c'est possible, une partie de la production du logiciel grâce à l'automatisation et à la réutilisation à grande échelle des composants existants. Cet aspect est abordé de façon générale au paragraphe suivant, et de façon plus détaillée dans la partie 3 de ce document.
- *La visibilité du processus de développement*. Dans le développement d'un projet complexe, il est impératif que chacun des acteurs puisse situer la nature exacte de son rôle et de sa coopération avec les autres intervenants. Nous nous sommes donc attachés à formaliser ces aspects par un modèle. Le modèle M.A.R.S. (Modèle Association Représentation Scénario) que nous présenterons au chapitre 6, propose donc aux différents acteurs une vision abstraite du développement en cours par la définition d'un ensemble d'espaces de travail.
- *l'amélioration de la communication homme-machine*. Ce problème peut être abordé dans notre contexte sous deux aspects :
  ⇒ un *aspect externe* : prise en compte homogène de règles de conception et de manipulation dans les interfaces proposées aux utilisateurs des outils de conception.
  ⇒ un *aspect interne* : évaluation de la distance existant entre les opérations effectuées par un utilisateur sur l'interface et les concepts sous-jacents manipulés.
  Ces aspects seront abordés dans la partie 3 de ce document.

5.3 NOTRE PROPOSITION

5.3.1 Processus de développement et prototypage automatisé

Avant de décrire le processus de développement que nous voulons proposer, rappelons les contraintes propres à notre contexte spécifique :

- Rapidité de développement des applications.
- Existence de domaines de compétence bien délimités permettant d'envisager une parallélisation des différentes tâches de conception et de réalisation.
- Nécessité de formaliser la complémentarité des tâches de conception et la communication entre les concepteurs.

Dans ce contexte, l'approche séquentielle est exclue, parce qu'elle nécessite un

découpage en phases par type d'activité (spécification, conception, réalisation) et non par domaine de compétence. L'approche évolutive basée sur l'automatisation nous parait donc une solution satisfaisante en particulier parce que notre domaine d'application (le développement de simulations pédagogiques concernant les disciplines scientifiques et techniques) est relativement bien délimité et que l'on peut de ce fait envisager avec optimisme la définition de formalismes de spécification. Mais cette approche doit être adaptée. Nous proposons donc le processus de développement suivant :

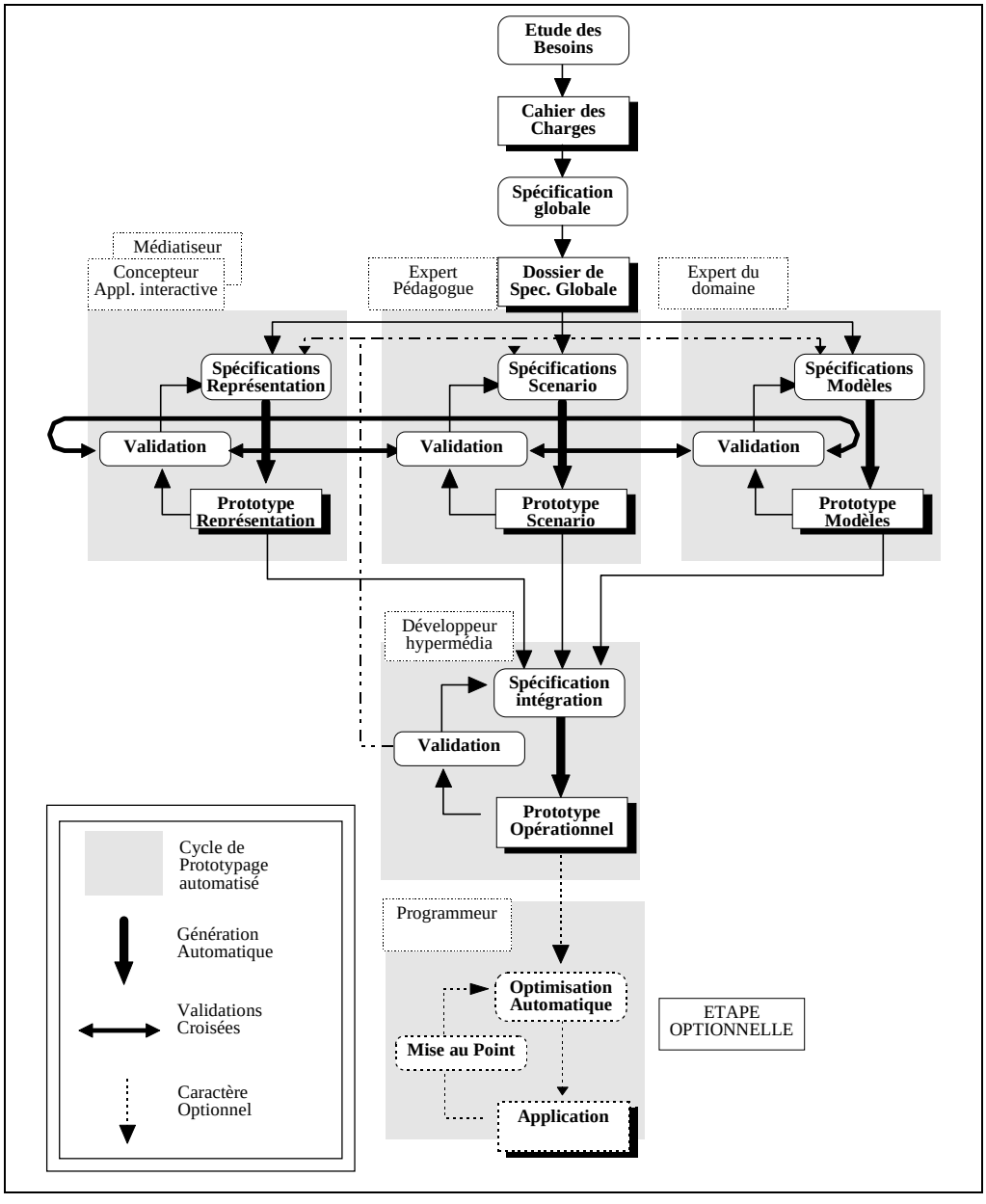


Figure 5-7 : Le processus de développement proposé

Ce processus est basé sur la définition de trois phases :

- Une phase d'étude des besoins telle que nous l'avons définie plus haut.

- Une phase de spécification-prototypage.
- Une phase optionnelle d'optimisation de l'application.

La phase de spécification-prototypage s'appuie, en premier lieu, sur la définition des spécifications globales qui permettent de fixer le cadre dans lequel seront effectuées les activités de prototypage. Trois de ces activités de prototypage peuvent se dérouler en parallèle, et correspondent chacune à un type de compétence spécifique : la spécification de la représentation, la spécification du scénario pédagogique, et la spécification des modèles de simulation.

Chacune de ces activités se déroule selon un cycle de prototypage automatisé, c'est-à-dire que chaque concepteur dispose d'un environnement lui permettant :

- *l'expression assistée de ses spécifications* selon un ensemble de formalismes préétablis propres à son domaine de compétence, grâce à un ensemble d'éditeurs spécialisés.
- la *production d'un prototype partiel* à partir des spécifications fournies. Nous entendons par prototype partiel, un sous-ensemble des données et du code nécessaire, concernant un aspect précis et délimité de l'application finale. La production de chaque prototype obéit à un ensemble de règles permettant son intégration ultérieure au sein de l'application finale, par la génération assistée des interfaces de communication.
- la *validation de ses spécifications* par des programmes utilitaires appropriés. Ces programmes doivent offrir non seulement une visualisation statique de ces spécifications mais encore une visualisation dynamique de leur exécution lorsque c'est nécessaire. Par exemple, la définition abstraite d'un mécanisme industriel à l'aide d'un diagramme états/transitions doit pouvoir être vérifiée par une animation temporelle d'une représentation graphique de ce diagramme. La visualisation peut donc être, dans certains cas, totalement différente de ce qui sera perçu ultérieurement par l'utilisateur final.

Nous distinguons ainsi :

- le *prototypage du **scénario** pédagogique*. Nous entendons par scénario une représentation abstraite de la démarche pédagogique décrite à deux niveaux. Le premier niveau, macroscopique, permet de définir l'enchaînement des unités pédagogiques de simulation. Le second permet de définir en détail les caractéristiques de chaque unité : objectif à atteindre, comportement de l'élève à prendre en compte, réactivité pédagogique, etc.
- le *prototypage des **modèles** de simulations*. Rappelons que pour nous, un modèle de simulation est une expression abstraite et partielle d'un phénomène réel ou d'un

concept exprimée à l'aide de formalismes mathématiques, logiques ou temporels.

- le *prototypage de la **représentation***. Nous regroupons sous le terme de *représentation*, l'ensemble des éléments perçus par l'utilisateur final (unités de présentation, objets interactifs ou non, documents multimédias). Le prototype partiel "représentation" obtenu en fin de phase correspond en fait à l'application finale privée de ses aspects fonctionnels pédagogiques et de simulation, mais pouvant contenir certains aspects dynamiques (navigation, animation, ...).

Comme nous l'avons indiqué précédemment, le rôle du concepteur pédagogique est central : c'est de ses choix que dépendront les spécifications de la représentation et des modèles de simulation. Au fur et à mesure de l'avancement de chacun des prototypes, un mécanisme de *validations croisées* permet à chaque concepteur de vérifier la cohérence de ses choix en regard des objectifs pédagogiques généraux de l'application.

Une fois les prototypes partiels réalisés, ils peuvent être intégrés pour constituer l'application finale. De même que pour les activités de conception, nous proposons l'idée que *l'intégration des composants peut, elle aussi, obéir à une démarche de type prototypage automatisé*. Un outil d'intégration assistée doit ainsi s'appuyer sur une expression formelle des ***associations*** pouvant être constituées entre les interfaces logicielles des différents prototypes partiels en vue de produire l'application finale. A partir de cette description formelle, il est donc possible de générer de façon automatique un *prototype opérationnel* comportant toutes les fonctionnalités de l'application finale. En cas de résultat non satisfaisant après la validation de ce prototype, il peut être décidé :

- de spécifier un nouveau type d'associations entre les composants et de générer, de ce fait, une nouvelle application.
- de spécifier de nouveau un des trois composants, et de modifier la spécification d'intégration si nécessaire.

Enfin, une phase d'optimisation est prévue : elle a pour objectif le renforcement de l'efficacité du logiciel à l'exécution ou bien la portabilité sur d'autres environnements que celui qui a servi à la production du prototype opérationnel (environnement souvent interprétatif). Cette phase se traduit par la génération d'un code exécutable.

5.3.2 Notion d'environnement intégré adaptatif

Nous avons dit plus haut que notre objectif était de fournir un environnement

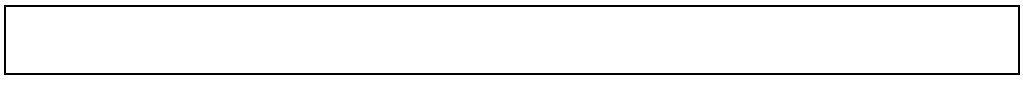
intégré de production de logiciel, non pas générique, mais adapté à une classe de problèmes bien définie. Pourtant, dans le domaine de la production d'applications de simulation pédagogique dédiées aux disciplines scientifiques et techniques, nous pouvons identifier un certain nombre de critères permettant de les distinguer. Parmi ces critères nous pouvons retenir :

- le *domaine scientifique ou technique abordé*. Selon les disciplines étudiées, les formalismes à utiliser pour exprimer les concepts ou connaissances peuvent être très différents. Par exemple, la formalisation du fonctionnement d'un alternateur est très différente de celle du fonctionnement d'une imprimante. De la même façon, suivant les domaines, les besoins de représentation, graphiques, sonores, etc., ne seront pas identiques.
- La *démarche pédagogique utilisée*. Selon la nature de l'apprentissage, la nature du contrôle pédagogique peut être différente. Tantôt on s'intéressera à savoir si l'apprenant résout correctement un problème, tantôt on vérifiera qu'il maîtrise correctement un savoir-faire opératoire.
- Le *contexte de conception*. La conception du logiciel nécessite des compétences diverses qui peuvent être réparties sur un nombre plus ou moins grand de personnes. On peut donc distinguer deux approches :

  ⇒ la *conception coopérative*. Dans ce cas, les rôles sont clairement répartis : chaque tâche de conception est prise en charge par un expert aux compétences bien définies. Cette répartition peut aller jusqu'à la non-présence des concepteurs dans le même lieu physique : la collaboration s'effectue alors par le média informatique et l'on peut parler d'approche *collecticielle*.

  ⇒ la *conception centralisée*. Dans ce cas, les rôles sont tous tenus par la même personne physique. La répartition physique des tâches doit ici faire place à une vue logique qui va permettre au concepteur d'organiser sa tâche, en situant en permanence la nature des actions qu'il effectue. D'autre part, la conception centralisée exige une pluri-compétence dans des domaines différents et on ne peut exiger de la part d'une seule personne le même niveau de spécialisation que des experts. L'environnement devra alors pallier ces déficiences en offrant au concepteur des outils de plus haut niveau et des interactions simplifiées.

Il ressort de cette analyse que l'environnement intégré ne doit pas offrir les mêmes services selon les cas rencontrés. Nous proposons donc d'introduire la notion *d'environnement intégré adaptatif* dans lequel peut être définie une configuration spécifique d'outils relative à un contexte précis. Ceci nous conduit à proposer l'architecture suivante :



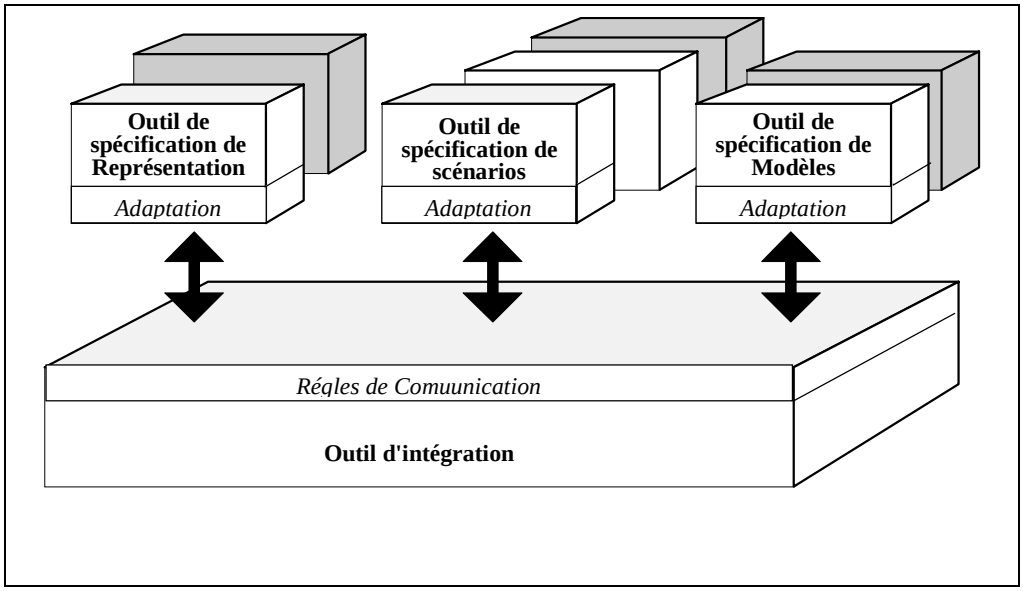


Figure 5-8 : Architecture de l'environnement intégré adaptatif

La figure 5-8 montre que, pour un contexte donné, l'environnement fournit un ensemble de quatre outils.

Chacun des trois outils de spécification propose au concepteur un ensemble de formalismes adaptés à la catégorie de problèmes visée. L'outil d'intégration est basé sur un ensemble de règles de communication à observer par chacun des composants à intégrer. Cette contrainte impose pour chaque outil de spécification la nécessité d'une couche d'adaptation permettant la correspondance entre les formalismes proposés au concepteur et ceux proposés par l'outil d'intégration.

5.4 RESUME

Dans ce chapitre, nous avons proposé un cycle de développement adapté à notre problématique.

Après avoir rappelé l'existence de deux grandes approches, l'approche séquentielle et l'approche évolutive, nous nous sommes concentrés sur cette dernière en insistant sur l'apport de la démarche de prototypage automatisé.

Nous avons ensuite énoncé un certain nombre de critères auxquels doivent obéir les Environnements Intégrés de Production de Logiciel mettant en œuvre cette démarche. Parmi ces critères, on relèvera :

- l'automatisation de la production.
- la réutilisation à grande échelle des composants existants.
- la délégation d'une partie de la production du logiciel aux utilisateurs finaux.

- l'amélioration de la communication homme-machine.

Nous avons ensuite proposé un processus de développement s'appuyant sur le déroulement parallèle de trois cycles de prototypage automatisé, dédiés chacun à la production d'un des composants du logiciel (le Modèle, le Scénario et la Représentation), puis sur un dernier cycle de prototypage automatisé dont le but est l'intégration des trois composants obtenus.

Nous avons enfin tenté de rendre notre démarche plus générique en proposant le concept d'environnement intégré adaptatif, qui permet de fournir différents outils de spécification de modèles, de scénarios ou de représentations, chacun d'entre eux étant adapté à une classe de problèmes déterminée.

PARTIE 3

LE MODELE DE CONCEPTION

Cette troisième partie est consacrée à la définition précise de notre proposition, le modèle de conception M.A.R.S.

- Le chapitre 6 présente de façon globale le modèle M.A.R.S.
- Les chapitres 7 à 10 présentent de façon détaillée chacun des espaces de travail proposés dans le modèle M.A.R.S : Modèle, Scénario, Représentation et enfin Association.

6. PRESENTATION DU MODELE M.A.R.S.

Nous présentons dans ce chapitre le modèle M.A.R.S. (Modèle, Association, Scénario, Représentation), modèle opérationnel d'analyse et de conception de simulations pédagogiques qui découle des propositions effectuées dans la Partie 2 de ce document. Cette présentation est volontairement succincte pour donner un point de vue global. Les différents composants du modèle seront présentés plus en détail dans les chapitres 7 à 10.

Le modèle MARS est avant tout un *cadre conceptuel général* qui va guider le concepteur ou l'équipe de conception dans sa démarche de définition d'une application de simulation pédagogique. Son objet est de rendre tangible la notion *d'espaces de travail distincts* dans lesquels intervient chaque acteur, muni de compétences bien répertoriées et chargé de tâches bien délimitées. Son but est donc en premier lieu de *servir de référence* pour l'organisation du développement logiciel d'une application.

6.1 ESPACES DE TRAVAIL

Un espace de travail peut être défini 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.

M.A.R.S. définit quatre espaces de travail comme le décrit la figure suivante :

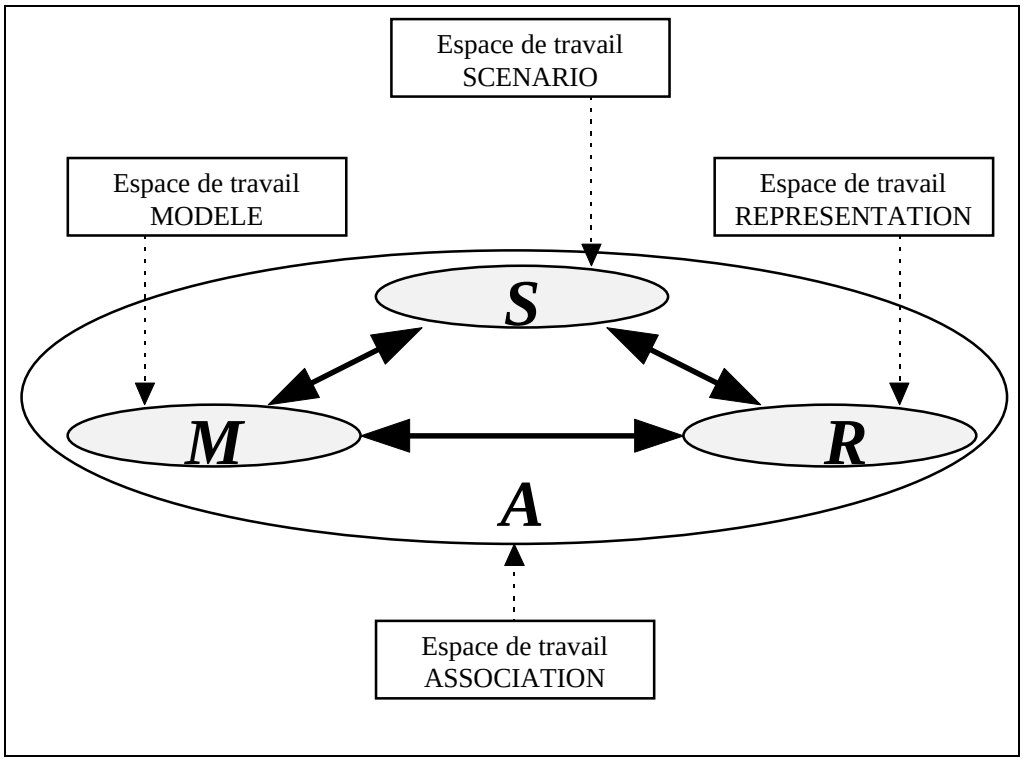


Figure 6-1 : La répartition des espaces de travail dans M.A.R.S.

- L'espace S est utilisé par les *experts en pédagogie* dans leur tâche de définition des *scénarios* pédagogiques.
- L'espace M est utilisé par les *experts techniques du domaine* dans leur tâche de définition des *modèles* de simulation.
- L'espace R est utilisé par les *concepteurs d'applications interactives* et les personnes responsables de la *médiatisation des applications* dans leur tâche de définition de la *représentation* interactive fournie aux utilisateurs de l'application pédagogique.
- Le dernier espace A est un espace de travail dans lequel va être réalisée l'intégration de l'application par la définition d'*associations* entre les résultats produits dans les espaces S, M et R. Il n'y a donc pas équivalence entre les quatre composants du modèle ; cette dissymétrie tient au fait que nous voulons exprimer d'une part l'existence de *domaines de compétence indépendants* et d'autre part la nature des *interrelations* qui les unissent.

6.2 LE CYCLE DE PROTOTYPAGE AUTOMATISE

Pour nous, la notion d'espace de travail recouvre la notion d'environnement de production autonome tel que nous l'avons défini au Chapitre 5. Il a pour objet la *production d'un résultat* en se basant sur *un processus* et un ensemble de rôles bien définis. Le processus que nous avons retenu pour chacune des tâches de conception repose sur la notion de cycle de prototypage automatisé composé de trois phases :

1. Spécifications formelles.
2. Génération automatique d'un prototype à partir des spécifications.
3. Validation des résultats par évaluation du prototype.

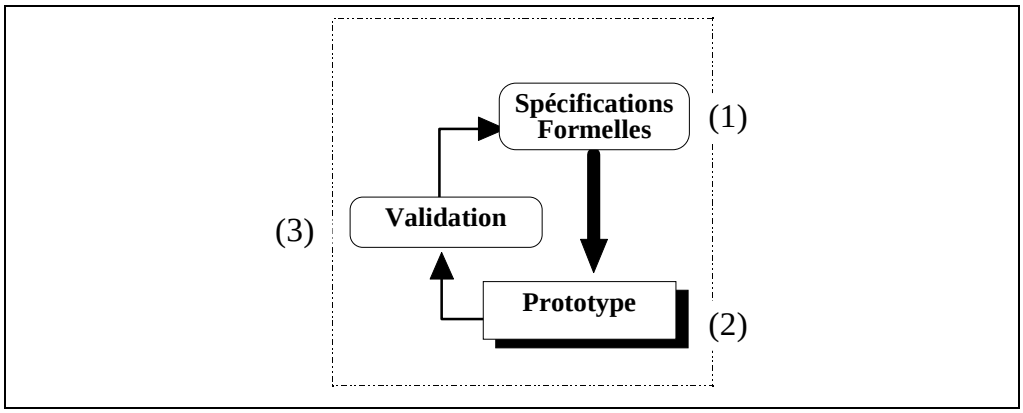


Figure 6-2 : Le cycle de prototypage automatisé

Comment cette solution peut-elle être mise en œuvre en respectant les contraintes que nous nous sommes fixées quant à la définition d'un environnement de production ? Rappelons quelques-unes de ces contraintes :

- la *délégation* de la production du logiciel aux utilisateurs, c'est-à-dire, dans notre cas d'espèce, à des auteurs n'ayant pas forcément de compétences de formalisation informatique.
- *l'amélioration de la communication homme-machine.*
- *l'automatisation de la production* autant qu'il est possible.
- la *réutilisation* à grande échelle des composants existants.

La prise en compte de ces contraintes nous demande en premier lieu de caractériser le processus de formalisation.

6.3 LE PROCESSUS DE FORMALISATION

Le processus de formalisation est un domaine abordé dans la littérature, en particulier dans le domaine des spécifications formelles [HEN 81, WAS 80, PAR 90]. Partsch [PAR 90] définit le processus de formalisation d'un problème spécifique en proposant les étapes suivantes :

- *Identifier le problème*

  ⇒ Identifier le domaine dans lequel se place le problème spécifique.

  ⇒ Dégager le concept ou les concepts du domaine.

  ⇒ Choisir une représentation pour chaque concept.
- *Décrire le problème* spécifique dans les termes des représentations choisies.
- *Valider la spécification* en la comparant avec le problème d'origine.

Nous proposons d'analyser cette démarche à partir d'un exemple. Nous avons testé un logiciel permettant la création automatique d'un plan de maison, ***3D Home Architect*** de la société ***Broderbund Software***. Ce logiciel propose à des auteurs (en général, pour un usage non-professionnel) deux modes d'utilisation :

- un mode de création en deux dimensions
- un mode de visualisation en trois dimensions

En *mode création*, le concepteur dispose de commandes et d'une palette d'outils lui permettant :

- de définir plusieurs niveaux de bâtiment
- pour chaque niveau, de définir un plan ayant pour caractéristiques :

⇒ une hauteur commune à tout le niveau

⇒ un ensemble de murs ou cloisons qui lorsqu'ils constituent un périmètre fermé définissent une pièce

⇒ un ensemble d'ouvertures (portes ou fenêtres) qui peuvent être placées sur les murs ou cloisons existants, en respectant certaines contraintes topologiques.

En *mode visualisation*, le concepteur ou le client dispose de commandes qui lui permettent :

⇒ de ne visualiser qu'un seul niveau à la fois

⇒ de visualiser ce niveau en trois dimensions en plaçant et orientant une caméra à un endroit quelconque du plan.

La figure 6-3 décrit le cycle d'utilisation de cet outil par un concepteur. Il s'agit

d'un cycle itératif basé sur le prototypage automatisé où sont enchaînées les étapes suivantes :

- (1) spécification graphique en deux dimensions
- (2) génération automatique de la représentation interne
- (3) sur demande de l'utilisateur, génération d'une visualisation en trois dimensions
- (4) validation du résultat obtenu, puis retour à (1)

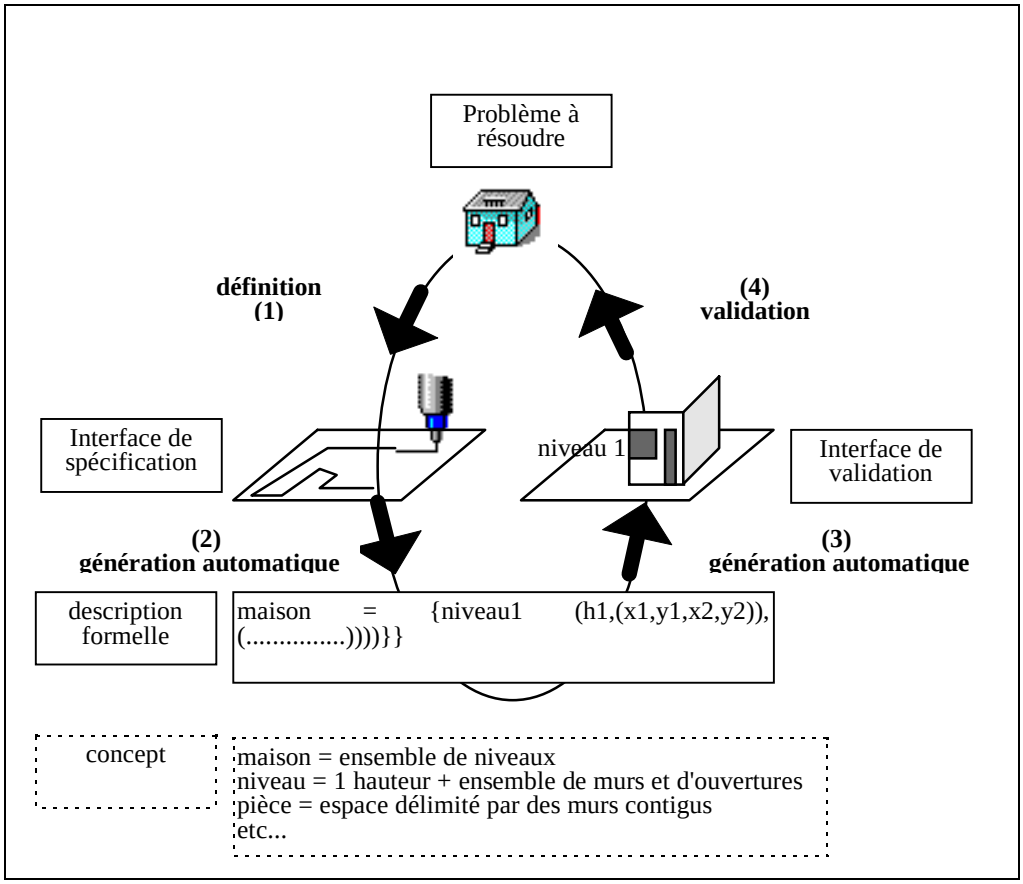


Figure 6-3 : Processus de conception automatisé

De façon globale, l'utilisation de ce logiciel est rendu malaisée par la quasi-absence d'un concept essentiel : celui de pièce d'habitation. Ce concept est en fait dérivé de celui de mur, au point même que l'on ne peut créer de pièce qui ne soit pas totalement fermée. Dans la présentation générale du produit, l'accent est mis sur le fait que l'utilisateur peut créer interactivement des murs, des portes, des éléments de mobilier pour un niveau donné, mais il n'est pas fait allusion à la possibilité de créer des pièces. On peut pourtant faire remarquer qu'il n'est courant ni pour un architecte, ni pour un maçon, ni pour un client potentiel, de se représenter un logement comme un ensemble de murs qui incidemment forment des pièces. La démarche naturelle est plutôt inverse, et pourtant la formalisation utilisée est juste.

L'origine de cette faiblesse de conception de l'outil provient sans aucun doute de la complexité à représenter formellement le concept de pièce sans le lier à celui de mur.

A partir de cet exemple, nous pouvons faire les remarques suivantes :

- La présence de deux interfaces différentes illustre bien la nécessité de séparer spécification et validation dans le cycle de prototypage.
- L'activité de spécification est rendue dans ce cas plus ardue, par le fait que les concepts proposés par le logiciel ne correspondent pas à ceux manipulés de façon naturelle par l'auteur. L'efficacité d'une spécification peut donc se mesurer par son degré *d'accessibilité* reflétant la distance sémantique [NOR 86] existant entre les concepts manipulés par l'auteur et ceux proposés par le système. Une distance trop importante impose à l'auteur un effort de traduction de formalismes.
- La validation n'est pas une vérification de la spécification : elle a pour objet de vérifier l'adéquation du résultat produit avec le problème initial. L'efficacité de la validation pourra donc se mesurer par la *compréhensibilité*, c'est-à-dire l'écart existant entre la perception du résultat obtenu et les propres concepts de l'auteur. Dans notre cas précis, lors de la validation du résultat, l'auteur peut appréhender sans aucune difficulté le concept de pièce d'habitation par la vue en trois dimensions : ceci ne nécessite donc pas de sa part d'effort de traduction.
- C'est essentiellement de la qualité des interfaces que va dépendre la *visibilité* des concepts sous-jacents proposés par le système. Dans le logiciel étudié, les possibilités offertes par la manipulation directe (palette d'outils, mur, fenêtre, porte, mobilier, etc.) ou la visualisation en trois dimensions assurent pratiquement un accès direct aux concepts proposés par le système.

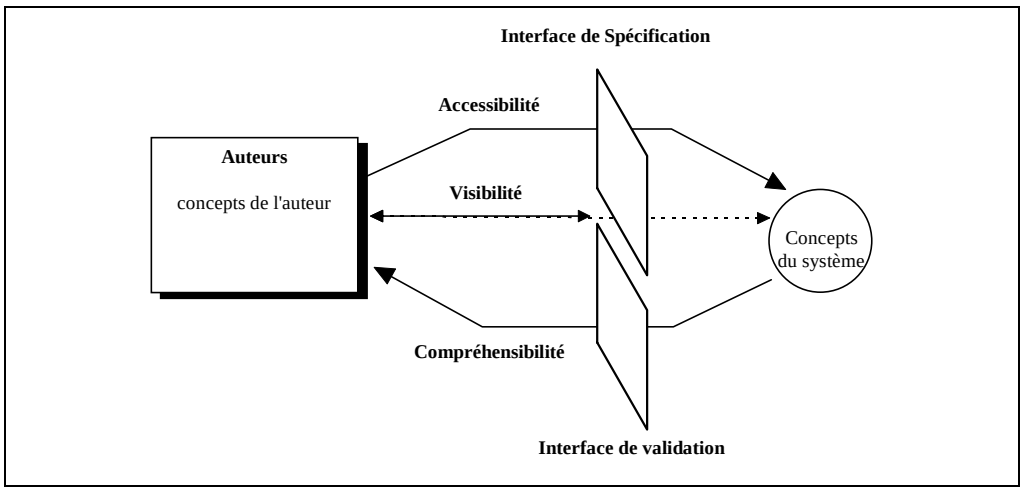


Figure 6-4 :Distance entre concepts de l'auteur et concepts du système

6.4 CHOISIR LES BONS CONCEPTS ET FOURNIR LES BONNES INTERFACES

Nous nous proposons de systématiser le processus décrit par Partsch [PAR 90] en visant le respect des critères que nous avons introduits.

Accessibilité, compréhensibilité et visibilité seront améliorées par :

- le choix dans le domaine concerné de concepts correspondant de façon la plus proche à la démarche la plus fréquente de l'auteur.
- un choix de représentation interne de ces concepts permettant de les rendre accessibles par l'interface.
- le choix d'une interface favorisant la manipulation et la visualisation la plus directe des concepts retenus.

La validation de ces choix ne pourra se faire que de façon expérimentale auprès des catégories d'auteurs concernées. Il s'agira de répondre aux questions suivantes :

- L'interface développée permet-elle la manipulation aisée ou la visualisation des concepts ?
- Les représentations des concepts sont-elles les plus efficaces ?
- Les concepts retenus sont-ils ceux adoptés de façon majoritaire par les auteurs ?

Nous voulons appliquer cette démarche systématique dans chacun des espaces de travail que nous avons définis (Cf. Fig. 6-1). Comme le montre la figure 6-5, cette démarche consiste à :

- Clairement identifier le domaine des problèmes abordés dans l'espace de travail.
- Définir le profil des auteurs qui vont intervenir dans l'espace de travail.
- Dégager les concepts du domaine.
- Identifier les représentations possibles de ces concepts et évaluer leur accessibilité.
- Définir des interfaces de manipulation et de validation.
- Enfin, valider les diverses solutions proposées.

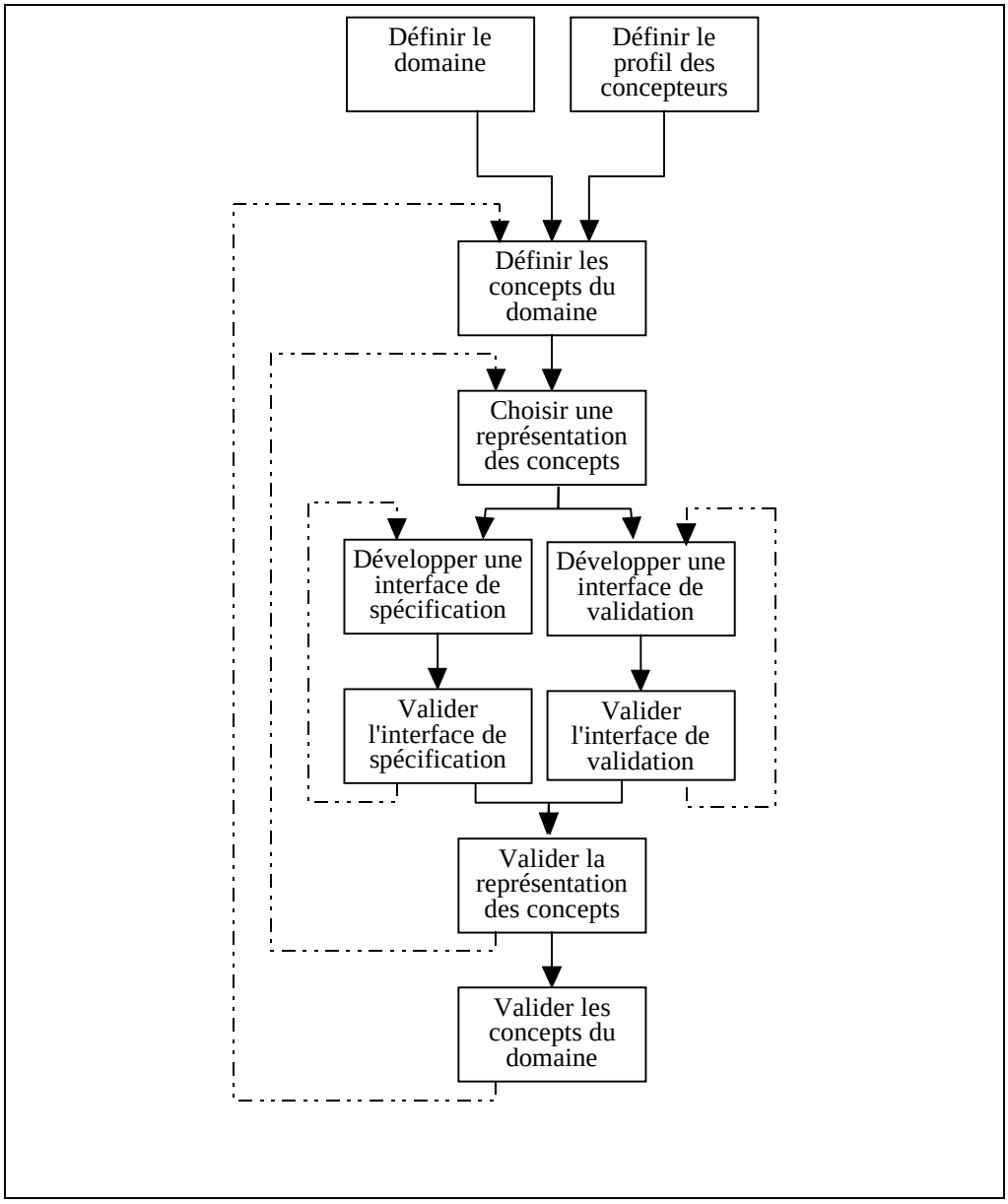


Figure 6-5 : La démarche de validation des concepts dans chaque espace de travail

Les chapitres 7 à 10 abordent chaque espace de travail (respectivement Modèle, Scénario, Représentation, Association) sous cet angle d'appréciation. Nous ne prétendons pas dans cette thèse apporter une solution définitive pour chacun des domaines abordés. Pour certains d'entre eux, nous menons encore aujourd'hui au sein de notre équipe des recherches complémentaires.

Toujours est-il que dans chacun des espaces, nous avons pu proposer des solutions opérationnelles qui ont été mises en œuvre dans le projet MELISA. Ces solutions seront exposées au chapitre 11.

6.5 RESUME

Dans ce chapitre, nous avons succinctement présenté le modèle MARS qui découle des propositions effectuées dans la Partie 2 de ce document.

Le modèle MARS est un cadre conceptuel général basé sur la notion d'espaces de travail distincts dans lesquels chaque acteur ou groupe d'acteurs va pouvoir intervenir. Le modèle MARS propose quatre espaces de travail :

- l'espace de travail Modèle.
- l'espace de travail Association.
- l'espace de travail Représentation.
- l'espace de travail Scénario.

Dans ce chapitre, nous avons également défini un certain nombre de critères que doivent respecter les outils proposés dans les différents espaces de travail : accessibilité, compréhensibilité et visibilité. Ces critères ont pour objectif de réduire la distance sémantique entre les concepts manipulés par les auteurs et ceux utilisés de façon interne par le logiciel.

Sur la base de cette proposition, nous préconisons une démarche de validation des concepts proposés aux auteurs, démarche que nous allons appliquer à chacun des espaces de travail dans les chapitres suivants.

7. LA SPECIFICATION DES MODELES DE SIMULATION

Dans notre contexte, la spécification d'un modèle de simulation est l'activité qui consiste à établir un *modèle*, une *vue abstraite* d'un système réel ou fictif faisant l'objet d'une simulation à vocation pédagogique.

La notion même de modèle se rapporte au principe d'abstraction. Coad [COA 91] adopte la définition suivante du Dictionnaire d'Informatique [OXF 86] :

> ***Abstraction*** *: principe qui consiste à ignorer les aspects d'un sujet qui ne se rapportent pas au propos courant, dans le but de se concentrer pleinement sur ceux qui s'y rapportent.*

Cette définition permet d'aborder deux questions fondamentales du processus de modélisation : (1) *Comment définir le propos courant* ? et (2) *Quelles sont les méthodes qui permettent d'analyser le sujet afin d'en formaliser les aspects pertinents ?*

Notre objectif propre est assez précisément définissable : nous voulons permettre à un utilisateur (un apprenant) d'observer ou de manipuler une représentation, partielle ou altérée, d'un système ou d'un phénomène, *dans le but d'enrichir ou d'évaluer ses connaissances*. De même, les domaines d'étude se situent dans des disciplines bien répertoriées : les *disciplines scientifiques* (mathématiques, physique, chimie, etc.) *ou techniques* (maîtrise de processus, maîtrise du fonctionnement de machines ou de systèmes). Cette spécificité a pour nous deux

conséquences :

- Notre approche est avant tout *pédagogique*. Ce point de vue doit servir de filtre dans le processus de modélisation et dans les choix des aspects pertinents.
- Les problèmes auxquels nous nous intéressons appartiennent par nature à des disciplines où existent déjà des *formalismes d'expression non ambigus et universellement reconnus*. Nous pourrons admettre une extension de la notion de simulation pédagogique à d'autres types de problèmes (sciences "molles", disciplines littéraires, simulation de concepts abstraits), à la condition que leur expression à l'aide de formalismes mathématiques, logiques ou temporels soit d'un intérêt pédagogique indiscutable.

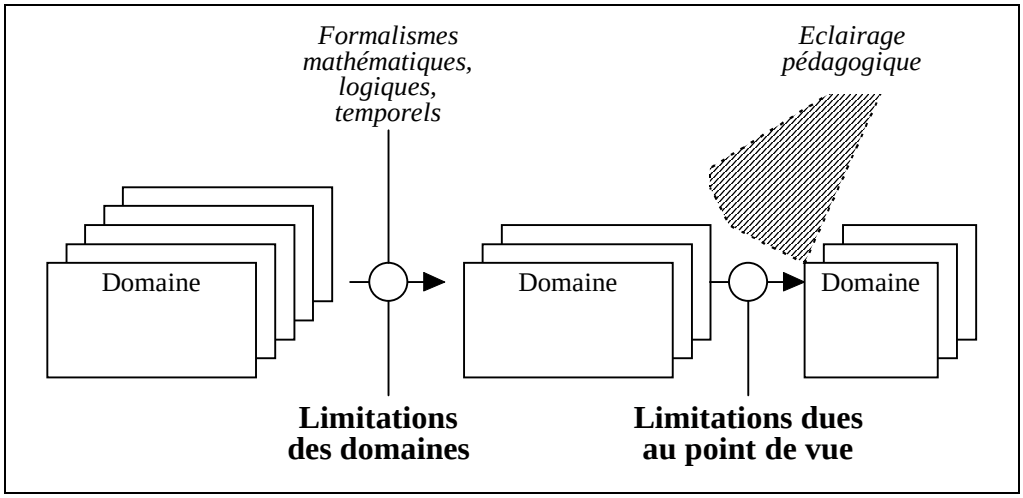


Figure 7-1 : Limitations

Notre propos consiste donc à modéliser des problèmes relevant de domaines précis, sous l'éclairage d'une démarche pédagogique définie (cf. Fig. 7-1). C'est ce que nous avons déjà qualifié de simulation réduite au contexte pédagogique pertinent.

7.1 LE PROFIL DES CONCEPTEURS DE MODELES DE SIMULATION

Lors de la définition du processus de développement, nous avons déterminé que la tâche de spécification des modèles de simulation était du ressort de *l'expert du domaine*. Cet expert du domaine n'a pas, a priori, de connaissances informatiques.

Il est toutefois fréquent que dans l'exercice normal de son activité, l'expert ait recours à un certain nombre de formalismes qu'il maîtrise, pour décrire les problèmes à résoudre. Prenons deux exemples. Dans le cas de l'alternateur décrit au chapitre 4, l'expert du domaine, ou modéliseur, va utiliser une série d'équations différentielles pour décrire l'évolution du système. Dans l'exemple de la manipulation d'un ordinateur lors de son démarrage (Cf. Chapitre 11), le concepteur de la simulation tentera de

décrire l'évolution du système à l'aide d'un diagramme états/transitions.

Parmi les différents acteurs que nous avons définis (concepteur de modèle, concepteur de scénario, concepteur de représentation interactive), le concepteur de modèle est certainement le plus familiarisé avec le processus de formalisation. Mais cette formalisation est souvent basée sur des règles spécifiques ou sur des habitudes particulières. Nous devons de ce fait fournir aux concepteurs de modèles un cadre général dans lequel ils pourront exprimer la majorité des problèmes qu'ils ont à résoudre.

7.2 LES CONCEPTS DU DOMAINE

La simulation est un domaine à part entière de l'informatique, avec sa propre communauté scientifique, son propre langage. Dans cette communauté, le terme de simulation est très souvent associé de façon implicite à la notion d'expérimentation. Reprenons quelques définitions générales citées par [CEL 91] :

> *Un système est une source potentielle de données*
> *Une expérience est le processus consistant à extraire les données d'un système en lui appliquant des entrées*
> *Un modèle M pour un système S et une expérience E donnés est quelque chose sur lequel E peut être mise en œuvre afin de répondre à des questions portant sur S*
> *Une simulation est une expérience réalisée sur un modèle*

Il est évident que ces définitions portent en elles-mêmes des hypothèses quant à la finalité et à la caractérisation des simulations : une simulation est réalisée pour pratiquer des expériences en recueillant les données calculées par un modèle, ce modèle étant caractérisé par des données d'entrée, des règles d'évolution temporelles et des données de sortie. Ce point de vue induit une vision souvent très mathématique de la discipline qui peut se vérifier dans l'activité scientifique de la communauté "simulation".

D'un autre côté, la majeure partie des domaines d'application couverts par cette discipline (il s'agit très souvent de physique), recouvrent une partie des nôtres et, de ce fait, la typologie des problèmes rencontrés peut nous être très utile. Dans la discipline "simulation", le principal critère de distinction des applications est celui du mode de description de l'évolution temporelle du système à simuler : cette description peut être basée sur les événements (*event-oriented*) ou bien basée sur le processus (*process-oriented*). On distingue donc deux approches principales [BRA 83] :

- La *simulation des systèmes continus* (*Continuous Simulations*). Les modèles à temps continu sont caractérisés par le fait que, pour un laps de temps fini, les

variables d'état changent de valeur un nombre infini de fois [CEL 91]. Les modèles à temps continu sont représentés par un ensemble d'équations différentielles qui représentent l'évolution temporelle du système. Pour des raisons de représentation informatique, ces modèles sont discrétisés.

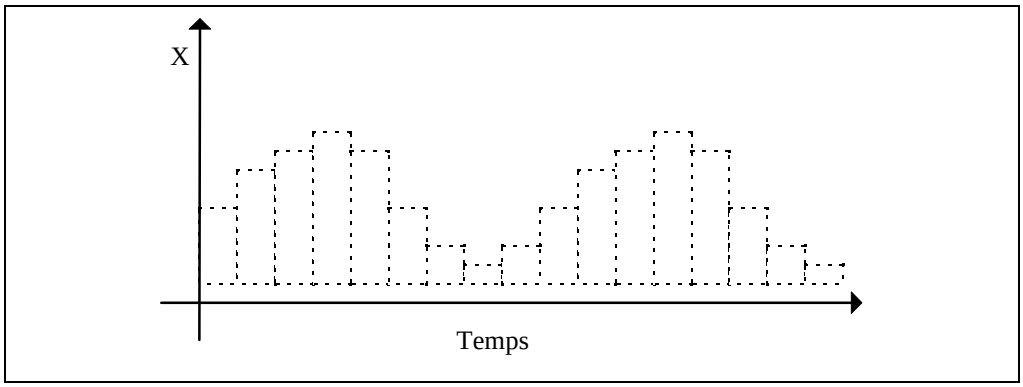


Figure 7-2 : Modèle continu discrétisé

- La *simulation des systèmes à événements discrets* (*Discrete-Event Simulations*). Contrairement à l'approche précédente, on admet ici que, pour un laps de temps déterminé, seul un nombre fini de changements d'état peut survenir. Chaque changement est provoqué par l'occurrence d'un événement qui par définition n'a pas de durée.

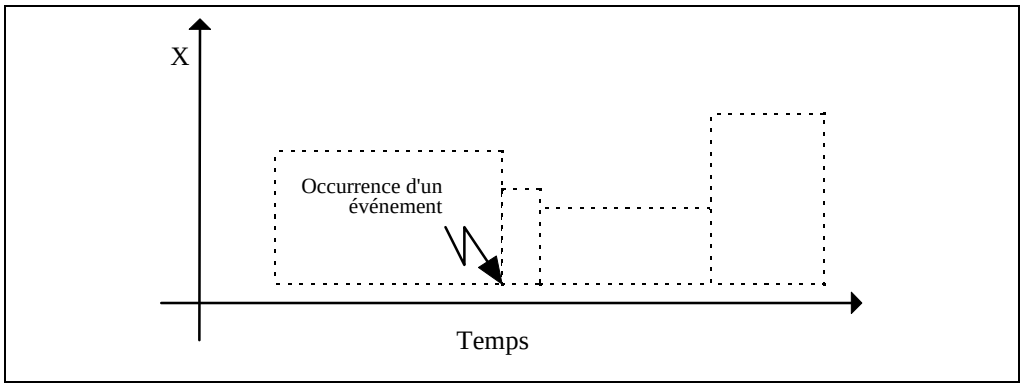


Figure 7-3 : Modèle à événement discret

Chacune de ces deux approches a donné naissance à une sous-discipline bien délimitée, proposant ses propres formalismes mathématiques ou logiques, et des langages de spécification ou de programmation spécialisés. La typologie générale proposée dans le domaine de la simulation peut être transposée dans notre domaine spécifique de simulation à vocation pédagogique, et recouvre effectivement l'ensemble des problèmes que nous nous sommes proposés de résoudre.

Pourtant la démarche proposée ci-dessus ne nous convient pas, parce qu'elle s'intéresse peu à la description structurelle du système à simuler, mais beaucoup plus à

sa description fonctionnelle en termes d'entrées, de sorties et d'évolution temporelle. Examinons de plus près les définitions suivantes de la notion de système :

> *Système : Appareil, dispositif formé par une réunion d'organes, d'éléments analogues constituant un ensemble cohérent [ROB 92].*
> *Système : Un ensemble ou un arrangement de choses reliées ou connectées de telle manière qu'elles forment un tout unique et organique [WEB 77]*

Ces définitions qui peuvent être appliquées aux problèmes que nous voulons modéliser, suggèrent deux idées. La première concerne la structuration : il doit être avantageux d'analyser un système complexe en s'attachant à décrire les composants qui le constituent et à identifier les relations qui les unissent. La deuxième concerne l'évolution temporelle d'un système : chacun des composants du système évolue dans le temps selon des règles qui lui sont propres ou bien dépendantes d'autres composants (concurrence et synchronisation).

Pour nous résumer, nous suggérons de proposer au concepteur les concepts suivants :

Un système est un ensemble de composants connectés constituant un ensemble cohérent. Dans la majorité des applications qui nous intéressent, le nombre de ces composants est connu, et n'évolue pas pendant la durée de vie de l'application.
Un composant est une entité logique du système dont la structure, le comportement et l'évolution temporelle peuvent être décrits de façon indépendante des autres composants. En particulier, l'évolution temporelle d'un composant peut être décrite grâce à des formalismes continus ou à événements. Dans certains cas, ces composants peuvent être réutilisés pour être intégrés dans d'autres systèmes.
Un modèle de simulation est une description abstraite et partielle, d'une part, de chacun des composants d'un système, et d'autre part, des relations de structuration, de concurrence et de synchronisation qui unissent ces composants.

L'activité du concepteur de modèles de simulation consiste à produire une description abstraite d'un système qui peut être complexe. Il doit donc en premier lieu disposer d'un cadre formel qui lui permet de gérer cette complexité, d'organiser et de structurer l'information qu'il veut manipuler. Il doit également s'appuyer sur des formalismes précis pour décrire la dynamique de son système. Pour ces raisons, nous

allons étudier, d'une part les principales approches de modélisation des systèmes, et d'autre part les formalismes les plus courants d'expression de la dynamique des systèmes.

7.3 LES DIFFERENTES APPROCHES DE MODELISATION

Coad et Yourdon [COA 91] répertorient quatre approches principales utilisées dans la démarche de modélisation d'un système existant. Chacune de ces approches privilégie tel ou tel aspect de la modélisation de l'information : les fonctions ou opérations du système, les données manipulées, la structuration.

7.3.1 L'approche fonctionnelle

L'approche fonctionnelle donne la priorité aux *traitements*. La démarche de base consiste à déterminer quelles sont les fonctions à appliquer au système et propose de hiérarchiser ces fonctions en partant des plus globales vers les plus précises. La décomposition fonctionnelle et la spécification des interfaces fonctionnelles et des traitements sont donc les formalismes majeurs mis à la disposition du concepteur.

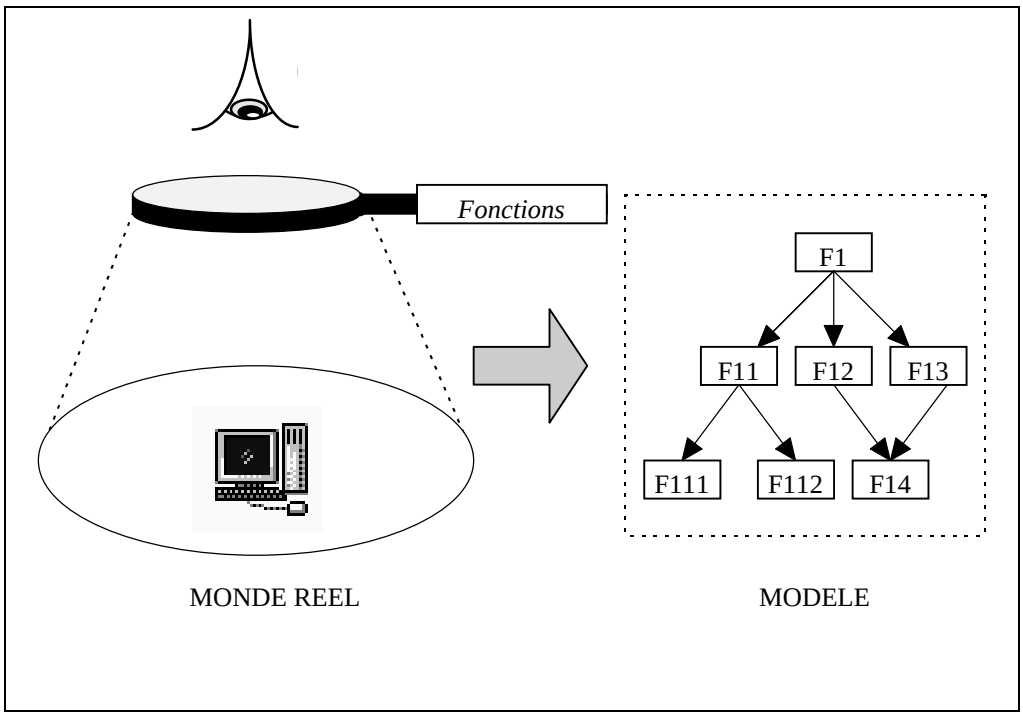


Figure 7-4 : L'approche fonctionnelle

Comme exemples de méthodes de cette catégorie, on peut citer les méthodes de programmation structurée [DAH 72], Jackson [JAC 75], SADT [MAR 88].

Les points forts de cette approche sont [BOU 94] : une relative simplicité et la

capacité de proposer plusieurs niveaux d'abstraction. Les points faibles concernent principalement : (1) une concentration trop importante de l'effort d'analyse sur les fonctions, négligeant la structuration et la cohérence des données ; (2) des règles de décomposition non explicites produisant des hiérarchies de décomposition différentes selon les analystes ; (3) l'inaptitude à traduire certaines interactions non-hiérarchiques dans les systèmes complexes et (4) l'inaptitude aux exigences de réutilisation.

7.3.2 L'approche par flots de données

L'approche par flots de données donne la priorité aux *données.* Elle prend à contre-pied la démarche précédente et privilégie une décomposition du problème basée sur la circulation des informations manipulées par le système. Les traitements sont ensuite définis et associés aux données. Ces méthodes sont plutôt adaptées aux problèmes de type système d'information où le nombre de données est très important et les traitements relativement élémentaires. Des représentants de cette tendance sont les méthodes MERISE [TAR 82], JACKSON [JAC 83] et Yourdon [YOU 89].

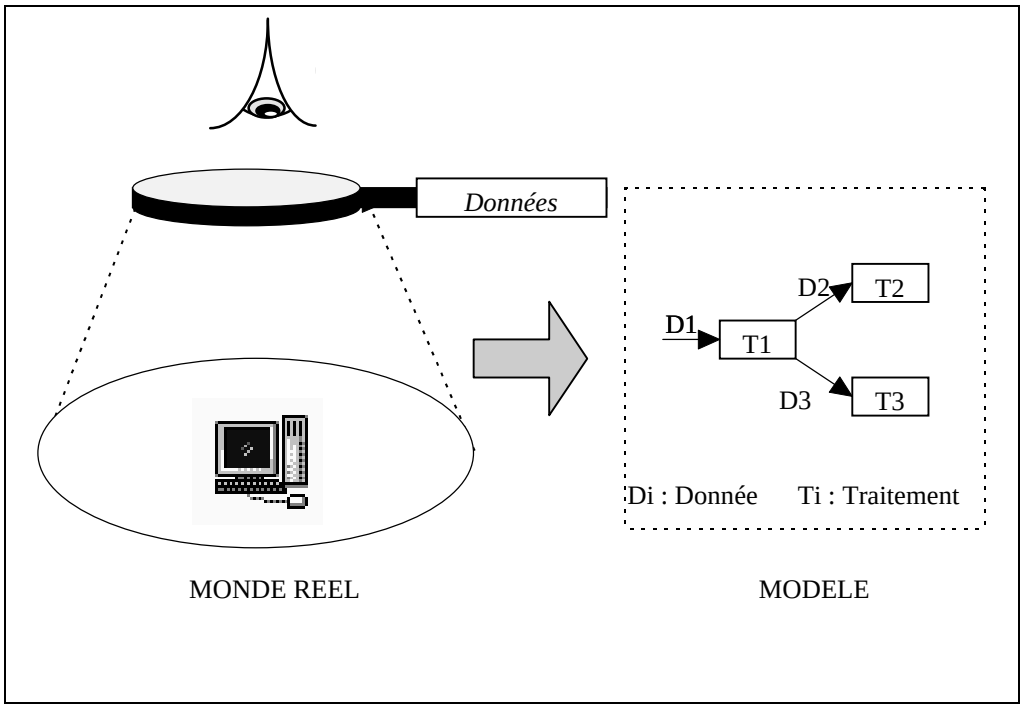


Figure 7-5 : L'approche par flots de données

Le principal point fort de ces méthodes est une plus grande cohérence des données. Les points faibles sont [BOU 94] : (1) un manque de cohérence entre modèles de données et modèles de traitement ; (2) une distinction pas très nette des niveaux de conception ; (3) la faiblesse de la modélisation des traitements qui mélange connaissance et contrôle, ce qui ne favorise pas l'évolution et la réutilisation.

7.3.3 L'approche "Modélisation de l'information"

Une autre approche, complémentaire à la précédente, a consisté à proposer un cadre formel dans le but de structurer les données. Le point de vue de départ consiste donc à définir les entités pertinentes du domaine à étudier et à définir les associations qui les relient. Les modèles entité-association [CHE 76, FLA 81], et plus récemment les modèles sémantiques de données [SCH 88], sont les représentants de cette tendance. L'originalité de cette approche réside dans le fait que pour la première fois, le processus de modélisation s'intéresse à représenter directement les objets du domaine du problème par des objets du modèle.

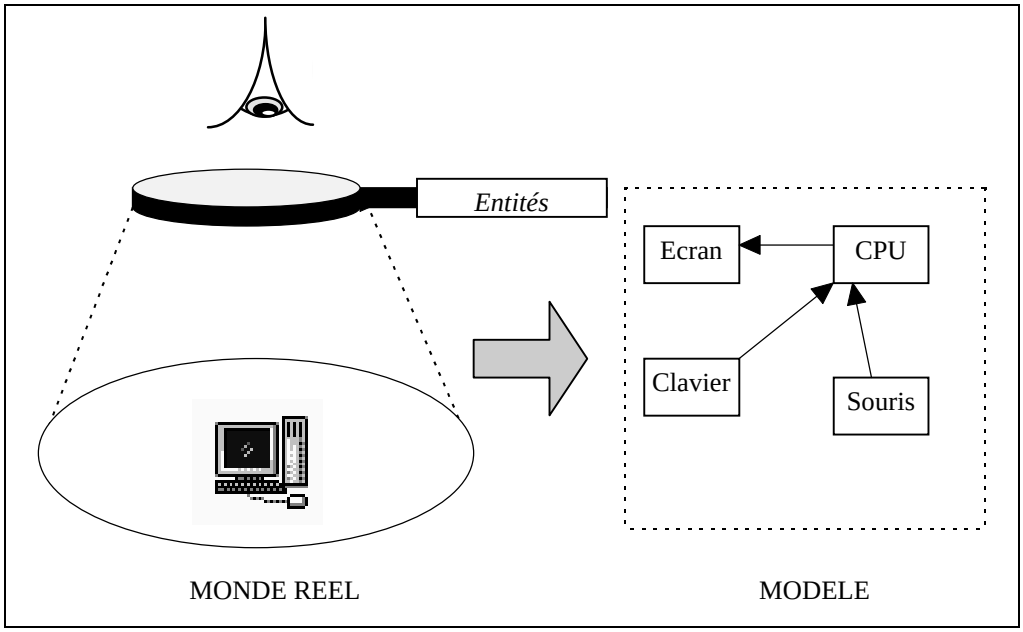


Figure 7-6 : Exemple de modèle entité-association

En revanche, on peut reprocher à cette approche [COA 91] qu'elle ne traite pas certains concepts tels que : (1) l'encapsulation du comportement (notion de service ou méthode) ; (2) la définition minutieuse des interfaces entre entités (grâce à la notion de messages), représentant l'interdépendance des traitements ; (3) une représentation explicite des éléments communs pour les attributs et les services (grâce à la notion d'héritage) ; (4) l'appel privilégié aux modes de structuration les plus utilisés par l'être humain (les liens de généralisation ou spécialisation, et ceux de composition).

7.3.4 L'approche par objets

Pour Coad et Yourdon [COA 91], la démarche de modélisation par objets est en fait une synthèse de différentes disciplines :

- la modélisation de l'information que nous venons de présenter. La préoccupation principale est ici de proposer au concepteur de construire une *représentation*

directe de la réalité qu'il veut modéliser, au lieu d'en fournir une représentation indirecte par le biais de fonctions, flots de données ou procédures.

- les langages de programmation par objets qui sont bien antérieurs à l'apparition des méthodes de conception et d'analyse par objets. La problématique de ces langages est de structurer plus efficacement les programmes, de renforcer la *modularité* et la *réutilisation* en proposant en particulier les concepts de classe et d'objet.

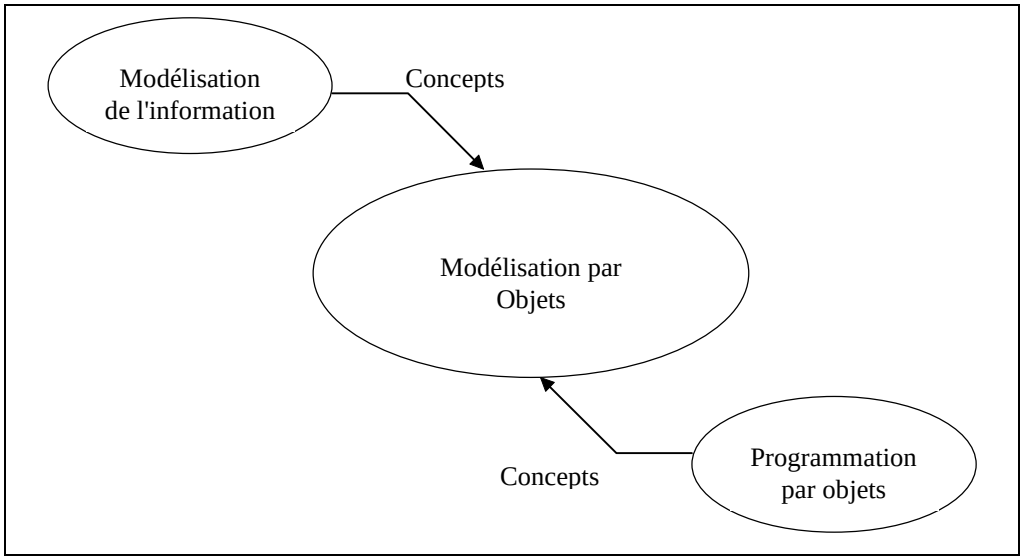


Figure 7-7 : La modélisation par objets, synthèse de deux points de vue

La synthèse de ces différentes disciplines a permis d'unifier des concepts d'origines diverses afin de proposer un cadre formel cohérent basé sur les principes fondamentaux de la gestion de la complexité. Aujourd'hui, le champ d'action de l'approche par objets est extrêmement vaste ; il touche à toutes les phases du cycle de vie du logiciel (analyse, conception, programmation), et à tous les domaines de l'informatique. Il n'est pas beaucoup de produits qui aujourd'hui ne se réclament de l'"orienté-objets", et malheureusement ce terme est souvent employé à mauvais escient.

Rappelons donc brièvement quels sont les principes fondamentaux de cette approche, en nous basant sur les travaux de référence de Booch [BOO 91], Coad et Yourdon [COA 90, COA 91], Meyer [MEY 90], Rumbaugh [RUM 91], Schlaer et Mellor [SCH 88, SCH 91]. Nous pouvons définir quatre grands principes de l'approche par objets :

- *Le principe d'abstraction*. Nous avons vu plus haut que le principe d'abstraction consistait à ignorer certains aspects d'un sujet pour ne retenir que les aspects se rapportant au propos courant. En particulier, dans l'approche par objets, "le mécanisme d'abstraction de données permet de définir un type de données en termes d'opérations qui s'appliquent sur les objets du type, avec la contrainte que

les valeurs de tels objets ne peuvent être modifiées et observées qu'en utilisant des opérations" [OXF 86]. Ce principe permet de définir pour chaque *objet* un ensemble *d'attributs* (ou propriétés) et *d'opérations* manipulant ces attributs. Accéder à un attribut d'un objet ne peut se faire que par l'intermédiaire d'une opération.

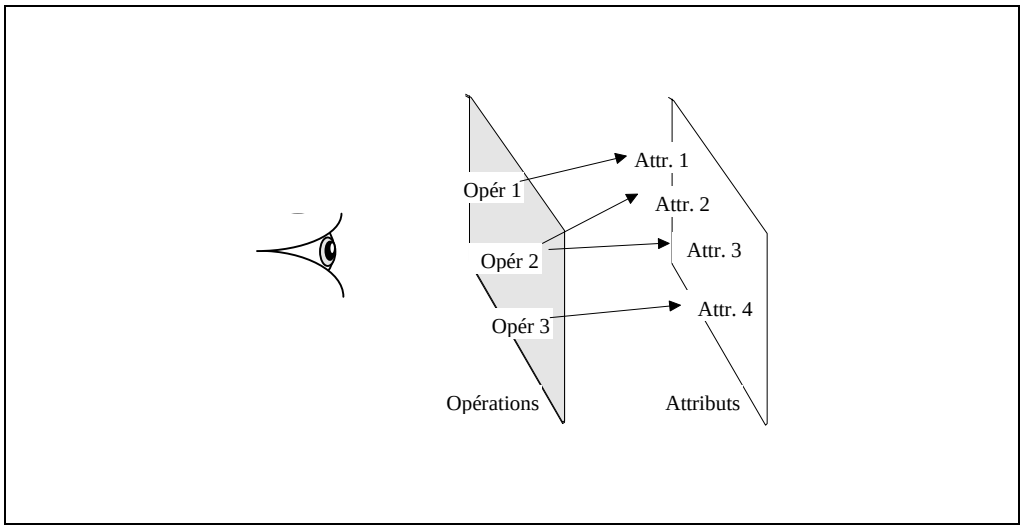


Figure 7-8 : Le principe d'abstraction des données

- *Le principe d'encapsulation*. C'est le principe qui consiste à séparer les aspects externes d'un objet, définis dans *l'interface* et accessibles depuis l'extérieur, des aspects internes de l'objet, invisibles depuis l'extérieur. L'encapsulation permet d'éviter les répercussions des modifications partielles en les circonscrivant au niveau d'une entité bien déterminée. L'encapsulation n'est pas propre au modèle par objets, mais la faculté de combiner les données (les attributs) et le comportement (les opérations) au sein d'une entité unique la rend plus propre et plus puissante que dans les formalismes qui séparent données et comportement [RUM 91]. Le principe d'encapsulation est associé à celui de *communication par message* qui permet l'activation des opérations définies dans l'interface.

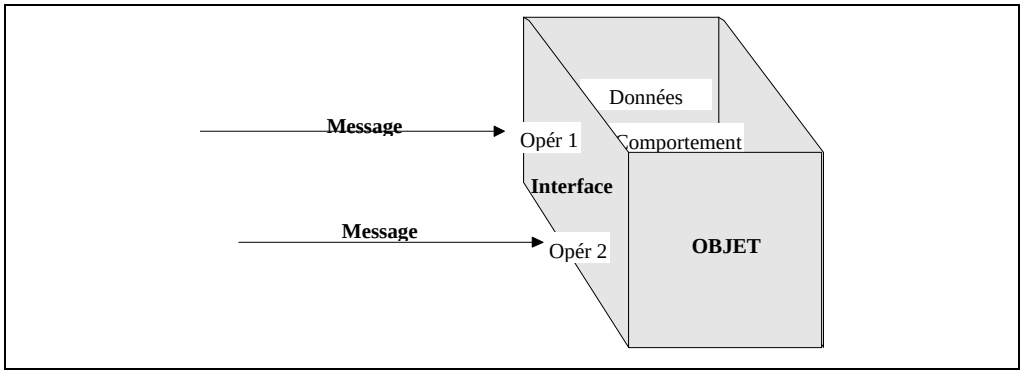


Figure 7-9 : Le principe d'encapsulation et de communication par messages

- *La notion de classe*. C'est un principe essentiel de la modélisation par objets qui part de la constatation qu'un des modes prépondérants d'organisation des êtres humains dans leur appréhension du monde réel est *la formation de classes et d'objets et la distinction entre classes et objets différents [BRI 86]*. Une classe représente donc un ensemble potentiel d'objets ayant des propriétés et un comportement identiques. Cette notion permet de décrire de façon générique un ensemble d'objets sans se soucier ni de leur nombre ni de la valeur spécifique de leurs propriétés. Une classe est en quelque sorte un moule à partir duquel peuvent être créés des *objets*, grâce au mécanisme *d'instanciation*.

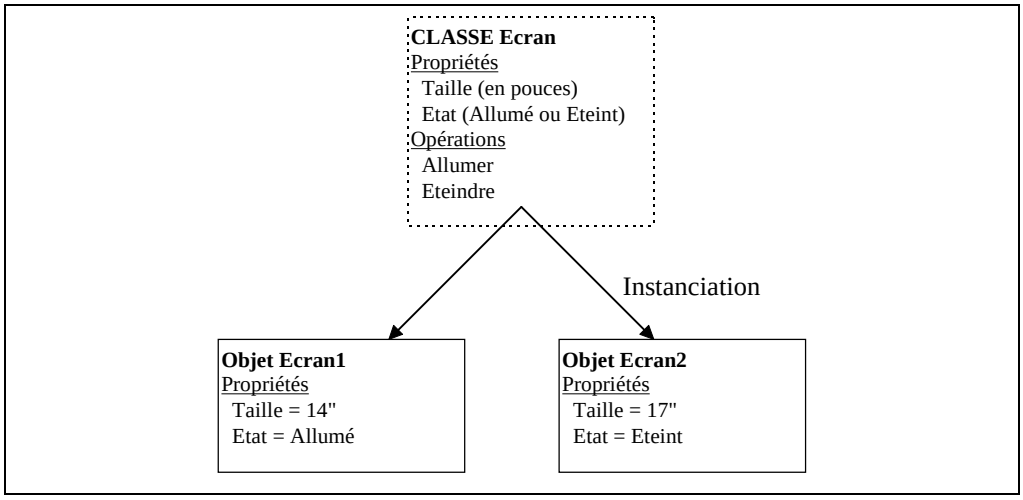


Figure 7-10 : Les notions de classe et d'instances

- *L'héritage*. C'est un mécanisme qui permet d'exprimer les similitudes entre classes, pour simplifier la définition de classes en partie similaires à celles qui ont été définies précédemment. Il décrit la généralisation et la spécialisation, en mettant explicitement à l'intérieur d'une hiérarchie ou d'un treillis de classes, les attributs et les services communs [COA 91]. L'héritage permet à un concepteur de spécifier les attributs et les opérations communs à plusieurs classes une seule fois, puis de décrire de façon séparée les caractères véritablement spécifiques de chaque classe.

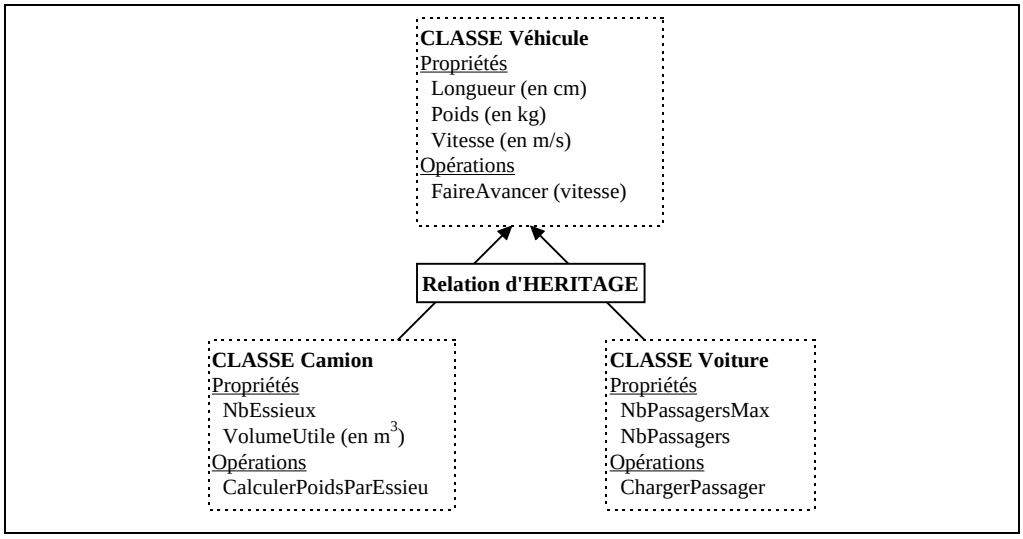


Figure 7-11 : La relation d'héritage

Pour Booch [BOO 91], les quatre grands principes que nous venons de décrire (le concept d'objet, l'encapsulation, la notion de classe et l'héritage) sont indispensables pour caractériser une véritable démarche par objets.

7.4 EVALUATION ET PROPOSITIONS

7.4.1 Evaluation des différentes approches pour nos besoins

Nous venons d'examiner plusieurs approches de modélisation. Nous devons maintenant évaluer chacune de ces approches en fonction des critères suivants :

- La prise en compte des concepts que nous avons retenus et que nous pouvons résumer par les termes suivants : (1) décomposition modulaire du système en un nombre fini de composants, (2) description indépendante de la structure, du comportement et de l'évolution temporelle de chacun des composants, (3) expression de la coopération, de la concurrence et de la synchronisation des différents composants, (4) réutilisation des composants.
- La facilité d'appréhension des formalismes pour les concepteurs dont nous avons défini le profil.
- La capacité des formalismes à être automatisés, en particulier par la puissance des représentations graphiques qu'ils proposent.
- La possibilité de valider les modèles produits.

Parmi les approches de modélisation que nous avons présentées, l'approche fonctionnelle et l'approche par flots de données n'obéissent pas à ces critères, en particulier par les difficultés d'expression de l'évolution temporelle des systèmes et

leur inaptitude à la réutilisation. Il existe des environnements de conception de simulation basés sur ces approches, tels par exemple LabView™ de la société National Instruments.

LabView est bien adapté au développement de simulations dans des domaines qui reflètent effectivement des phénomènes physiques de type flots de données, tels que l'électricité ou l'hydraulique. En revanche, son application à des systèmes à événements discrets et la prise en compte de l'évolution temporelle, de la concurrence et de la synchronisation sont complexes. D'autre part, la réutilisation de composants existants dans de nouveaux systèmes s'avère malaisée.

L'approche "modélisation de l'information" apporte une amélioration par rapport aux approches précédentes parce qu'elle s'intéresse à représenter directement le domaine du problème en objets du modèle. Par contre, cette approche est essentiellement un outil d'analyse de haut niveau permettant de décrire de façon globale d'importants systèmes d'informations ; elle n'est pas adaptée à une représentation détaillée d'un système [COA 91].

L'approche de modélisation par objets nous semble beaucoup plus intéressante parce qu'elle répond effectivement à nos exigences de structuration des systèmes, d'encapsulation des composants, de prise en compte des aspects dynamiques et de réutilisation. Toutefois, certains des principes de base de l'approche par objets semblent difficilement manipulables par les concepteurs non spécialisés en informatique.

7.4.2 Une expérience d'enseignement de l'approche par objets

La constatation qui vient d'être faite provient en partie d'une expérience que j'ai acquise pendant des sessions de formation dispensées auprès de publics non informaticiens. J'ai animé durant ces trois dernières années des formations destinées à deux types de public bien différents :

- d'une part, des professeurs de l'enseignement secondaire ou supérieur, rassemblés par l'intérêt qu'ils portent à l'introduction des nouvelles technologies dans leur enseignement. Ces enseignants exercent leur activité dans des disciplines en général scientifiques (mathématiques, physique, chimie, électrotechnique, sciences de la vie) et de façon plus exceptionnelle littéraires (français, langues). Leur objectif le plus fréquent est de maîtriser suffisamment les concepts informatiques pour être en mesure de développer seuls de petites applications pédagogiques (d'ailleurs, de plus en plus souvent basées sur la simulation).
- d'autre part, des étudiants de formation continue préparant un diplôme d'ingénierie en technologie multimédia. La formation initiale ou la profession exercée par ces

étudiants concernait des métiers liés au domaine de la communication ou à celui de l'informatique.

Le contenu des enseignements que j'ai assurés peut se diviser en deux grandes parties : (1) les principes de modélisation par objets et (2) la mise en œuvre de ces principes à travers l'utilisation d'un système auteur hypermédia à faible orientation objets : le logiciel Toolbook de la société Asymetrix.

L'expérience dont je voudrais faire part ici concerne l'assimilation de l'approche par objets par ces différents publics. On peut faire ressortir les points suivants :

- L'assimilation des concepts de l'approche par objets est beaucoup plus facile pour des publics "vierges", n'ayant pas été influencés auparavant par d'autres techniques de modélisation informatique. Les plus grandes difficultés ont été rencontrées avec certains enseignants en option informatique du secondaire ayant vécu l'évolution des langages de programmation depuis vingt ans (Basic, puis Pascal). La transition d'un mode de pensée impératif ou fonctionnel à une approche par objets s'avère souvent très douloureuse. Donnons un autre exemple : en ce qui concerne la formation d'ingénierie multimédia, le public était séparé en deux sous-groupes : celui des "communicants" qui avaient une pratique de l'outil informatique pratiquement nulle, et celui des "informaticiens" dont les connaissances informatiques se limitaient en général à des notions plus ou moins anciennes de programmation et à l'utilisation de logiciels spécialisés. Les modélisations des mêmes problèmes produites par les groupes de communicants se sont pratiquement toujours avérées être de meilleure qualité que celles des groupes d'informaticiens.
- Certains concepts de l'approche par objets apparaissent comme intuitifs et relevant du bon sens. Les notions *d'objets, d'attributs* et *d'opérations* ne sont en général pas complexes à appréhender, surtout si elles se rapportent à des objets concrets du monde réel.
- D'autres notions apparaissent moins naturelles, bien que nécessaires pour des raisons de formalisation. Par exemple, la nécessité *d'encapsuler* les objets et d'offrir un mécanisme de communication (les *messages*) est en général bien perçue : très vite, la mise en œuvre d'un exemple permettra de saisir toute la richesse du concept. Dans le même ordre d'idée, on peut citer les formalismes permettant l'expression de la *dynamique*, tels les diagrammes états/transitions.
- Souvent, la notion de *classe instanciable* apparaît, quant à elle, plus artificielle. La nécessité d'identifier les composants d'un système ayant même structure et même comportement n'est pas remise en cause et se révèle être un facteur efficace de maîtrise de la complexité. En revanche, le fait de raisonner à deux niveaux, un niveau abstrait, celui de la classe, et un niveau concret, celui des instances, semble souvent factice quand on manipule des exemples de petite taille où le nombre

d'instances de la même classe est très restreint, voire réduit à l'unité. Les mêmes remarques peuvent être faites pour *l'héritage*. La nécessité d'établir une classification au moyen d'un graphe d'héritage semble souvent un peu disproportionnée en regard du nombre d'objets manipulés.

7.4.3 La notion de classe est-elle adaptée ?

Ces différentes constatations proviennent d'expériences acquises avec des publics non-informaticiens. Elles nous amènent à nous poser la question suivante : *Devons-nous offrir tous les concepts de la modélisation par objets à nos concepteurs de modèles de simulation ? Certains concepts ne sont-ils pas trop complexes en regard des applications que nous avons à développer.*

N'oublions pas en effet que nous nous situons dans un contexte de développement d'applications relativement restreintes où les systèmes à simuler comprennent un nombre fini de composants. Si par exemple, nous devons créer une application de simulation pédagogique pour enseigner le fonctionnement d'un réseau informatique, il nous suffira de modéliser quelques stations connectées au réseau pour couvrir tous les problèmes liés à la gestion de ce réseau. La démarche pédagogique est par nature simplificatrice, et le problème du traitement d'un grand nombre d'informations n'est pas pertinent dans notre contexte spécifique.

7.4.4 L'approche par objets prototypes

Ceci nous amène donc à réfléchir aux concepts que nous pouvons proposer au concepteur dans sa démarche de structuration des composants du système, s'il ne dispose pas de la notion de classe.

Certains auteurs posent la question de la nécessité du concept de classes dans la construction de programmes "orientés-objets". Par exemple, Blaschek [BLA 94] constate que, pour résoudre un problème complexe, un grand nombre d'objets appartenant à des classes différentes doivent coopérer. Un réseau compliqué d'objets doit être construit avant qu'un quelconque programme puisse être exécuté. Un effort important de la part du concepteur doit être accompli pour spécifier la nature des objets, leur mode de création, et comment ils doivent être connectés les uns avec les autres. Pour Blaschek, une bonne part d'expérience, d'imagination et de réflexion préalable est nécessaire pour décrire de façon statique à quoi doit ressembler le comportement dynamique des objets d'un système au moment de l'exécution.

La programmation "par objets prototypes" [UNG 91, TAI 92, BLA 94] se propose donc de pallier ces inconvénients en fournissant le concept d'*objet prototype*.

*Un objet **prototype** est un objet préfabriqué muni d'une structure, d'un*

contenu et d'un comportement prédéfinis, à partir duquel de nouveaux objets peuvent être créés par copie (ou clonage).

Cette approche tente de rendre la démarche du programmeur la plus directe possible. Au lieu de décrire un objet de façon statique, l'objet est d'abord créé. Dès qu'il existe, ses propriétés peuvent être initialisées et le réseau d'objets représentant le système peut être enrichi par de nouvelles connexions. La composition des objets est typiquement réalisée de façon interactive à travers un éditeur spécialisé ou grâce à un langage de commande.

Au lieu de définir le comportement d'un objet dans la description de la classe auquel il appartient, la programmation par objets prototypes permet d'associer les opérations à l'objet lui-même. Ce mode de construction se rapproche beaucoup plus du modèle mental du programmeur utilisant une démarche par objets [BLA 94]. Quand il reçoit un message donné, c'est l'objet lui-même qui "sait" ce qui doit être fait, alors que dans les langages de classes, c'est la classe auquel il appartient qui lui "dicte" ce comportement.

Si la suppression du concept de classe permet une manipulation plus directe des objets par le programmeur, elle entraîne la disparition d'un puissant moyen d'expression des relations structurelles entre les objets : comment traduire le partage (total ou partiel) d'une structure ou d'un comportement identique par plusieurs objets ? L'approche par objets prototypes propose différents mécanismes pour résoudre ce problème.

Le langage de programmation SELF [CHA 89, UNG 91] est basé sur le principe de *délégation de messages* pour traduire le partage de propriétés ou de comportement par plusieurs objets. Un nouvel objet peut être obtenu, soit par clonage, soit par construction à l'aide d'un langage de spécification. Il est possible de créer un objet avec certaines propriétés, d'en faire plusieurs copies, d'ajouter une opération à un des objets obtenus, et d'utiliser ce dernier objet en tant que prototype pour la construction d'autres objets bénéficiant du nouveau comportement. Une fois créés, les objets perdent la référence de leur origine, et deviennent donc indépendants. SELF propose la possibilité de définir pour chaque objet un *parent*, qui représente la référence d'un autre objet du système. Quand un objet ne sait pas traiter un message qu'il reçoit, le message est *délégué* à son objet parent. La délégation de messages peut donc être utilisée pour établir un *mécanisme d'héritage dynamique* où les traitements génériques sont spécifiés dans des objets "abstraits" situés au sommet de la hiérarchie, et où les objets "instances" constituent les feuilles de l'arbre d'héritage. La délégation peut être également utilisée pour traduire le concept de *collection* d'objets au comportement identique. Il est à noter que certains langages utilisés dans les systèmes-auteurs hypermédias, tels que *Hypertalk* pour Hypercard ou *Openscript* pour Toolbook,

adoptent une approche similaire.

Le langage de programmation Kevo [TAI 92] a été conçu pour lever certaines restrictions inhérentes au modèle précédent, telles que, par exemple, la difficulté d'ajouter une propriété à un ensemble d'objets appartenant à une même collection. Le concept *d'opération modulaire* de Kevo permet de définir une opération (ajout, renommage, suppression ou masquage d'une propriété) qui peut s'appliquer à un ensemble d'objets plus ou moins important (un objet seul, l'ensemble de tous les clones d'un objet, l'ensemble des clones d'un objet proposant strictement la même interface, l'ensemble des clones d'un objet proposant une interface commune). Le partage de comportement est donc proposé de façon implicite dans Kevo par le biais du clonage d'objets. Chaque objet créé par clonage se situe dans une hiérarchie d'objets dont la racine (*Root*) est prédéfinie par le système.

Le langage de programmation Omega [BLA 91] se rattache, quant à lui, à un modèle plus traditionnel d'héritage. Le clonage d'un objet est une opération interactive ne nécessitant pas de manipulation du langage. Pour un objet donné, les liens de clonage de tous ses *descendants*, directs ou indirects, sont conservés. Cette structure permet la *propagation* de toute modification effectuée sur un objet (ajout, suppression, modification d'une propriété ou d'une opération) vers tous les objets qui en sont dérivés. Par exemple, l'ajout d'une propriété à un objet nécessite la saisie par le programmeur d'une valeur initiale. De façon automatique, la propriété initialisée avec cette valeur sera ajoutée à tous les descendants issus de l'objet initial. Omega repose donc sur une hiérarchie d'objets dont la sémantique s'apparente à celle d'un arbre d'héritage de classes.

Les différentes approches que nous venons de présenter se situent dans le domaine des langages de programmation. Notre problématique personnelle se place dans une démarche de type auteur et tente d'affranchir au maximum le concepteur de la tâche d'écriture textuelle de programmes. Nous nous sommes en particulier intéressés aux concepts de clonage, de délégation par messages et de collection d'objets ; nous avons déjà développé des maquettes permettant de les mettre en œuvre de façon interactive [PER 93a].

Ces propositions font actuellement l'objet d'expérimentations auprès d'auteurs dans le cadre du projet MELISA décrit au chapitre 11.

7.5 LES FORMALISMES D'EXPRESSION DE LA DYNAMIQUE

Il existe une assez grande diversité de formalismes d'expression du comportement dynamique. Davis [DAV 88a], dans une présentation de ces techniques,

regroupe ces formalismes dans quatre grandes classes[3] : les automates à états finis, les tables de décision, les Statecharts de Harel et les réseaux de Petri. Notre objectif étant de fournir une interface graphique permettant la manipulation directe de ces formalismes, nous ne nous intéresserons pas aux tables de décision.

7.5.1 Les automates à états finis

Les automates à états finis, ou diagrammes états-transitions sont les formalismes les plus anciens pour décrire la dynamique d'un système, et constituent une des bases de la discipline informatique. La représentation graphique d'un diagramme états-transitions est un graphe orienté où les noeuds représentent les états, et les arcs les transitions entre les états.

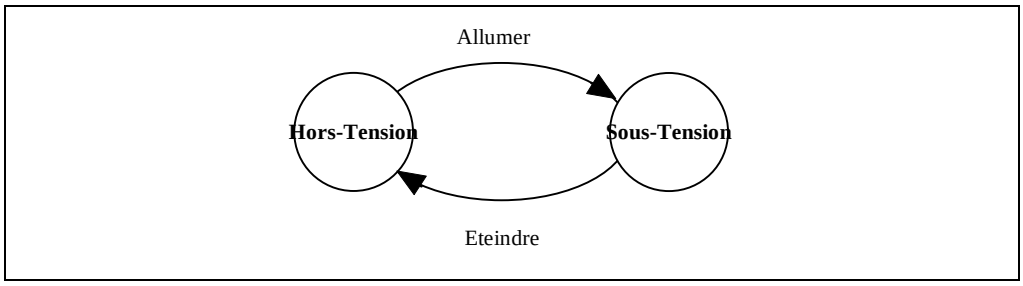


Figure 7-12 : Un exemple d'automates à états finis

Les automates à états finis ont un pouvoir d'expression limité. En particulier, ils ne permettent pas de représenter de façon compréhensible des systèmes possédant un grand nombre d'états. Un certain nombre d'extensions ont été proposées pour pallier ces insuffisances. Parmi ces extensions, on peut citer :

- les *Recursive Transition Networks* ou RTN [WOO 70], qui ont pour objectif la structuration d'un ensemble d'automates en proposant une hiérarchisation grâce au concept de macro-état (qui peut lui-même être un automate), et en permettant la communication entre automates.
- les *Augmented Transition Networks* ou ATN [WOO 70], qui introduisent la notion de *transition gardée* qui conditionne le franchissement d'une transition à la vérification d'un prédicat portant sur un ensemble de variables globales.
- les *Generalized Transition Networks* ou GTN [KIE 83], qui veulent généraliser le degré de récursivité en proposant que les préconditions, actions ou états puissent eux-mêmes être des GTN.

---

[3] Les modèles décrits ici s'intéressent principalement à la modélisation de phénomènes non continus. Nous montrons plus loin que de tels formalismes peuvent également être adaptés à l'expression de phénomènes continus, en particulier grâce au concept d'activité.

Ces différentes extensions des automates sont largement utilisées dans les différentes méthodes de conception [COA 91, BOO 91, SCH 91, RUM 91]. De façon générale, les automates à états finis ont l'avantage de se baser sur une représentation du monde réel (les états du système), de proposer une représentation graphique claire (bien que pouvant rapidement devenir lourde) et de pouvoir être exécutables (permettant ainsi leur validation). En revanche, même munis des extensions décrites ci-dessus, ils sont insuffisants pour remédier à l'explosion combinatoire des transitions, et surtout ils ne permettent pas de tenir compte de la concurrence et de la synchronisation.

7.5.2 Les Statecharts de Harel

Les Statecharts sont un formalisme proposé par Harel [HAR 87, 88, 90] pour structurer les machines à états finis, afin d'éviter l'explosion combinatoire des transitions et de pouvoir modéliser des systèmes complexes dans lesquels la concurrence joue un rôle important. Les Statecharts proposent deux modes de structuration : une structuration en profondeur permettant la modélisation hiérarchique des états d'un système et une structuration orthogonale permettant de représenter la concurrence et la synchronisation de graphes indépendants.

Un des avantages majeurs des Statecharts réside dans la puissance et la richesse de leurs représentations graphiques qui permettent l'expression de la concurrence, de la synchronisation et de la structuration des modèles (Cf. Fig. 7-13). Les fondements mathématiques des Statecharts permettent d'envisager la validation des modèles, bien que cette solution ne soit pas pratiquement mise en œuvre, tout du moins à notre connaissance. Les Statecharts de Harel sont la base de l'expression de la dynamique dans la méthode de conception par objet OMT (Object Modelling Technique) [RUM 91].

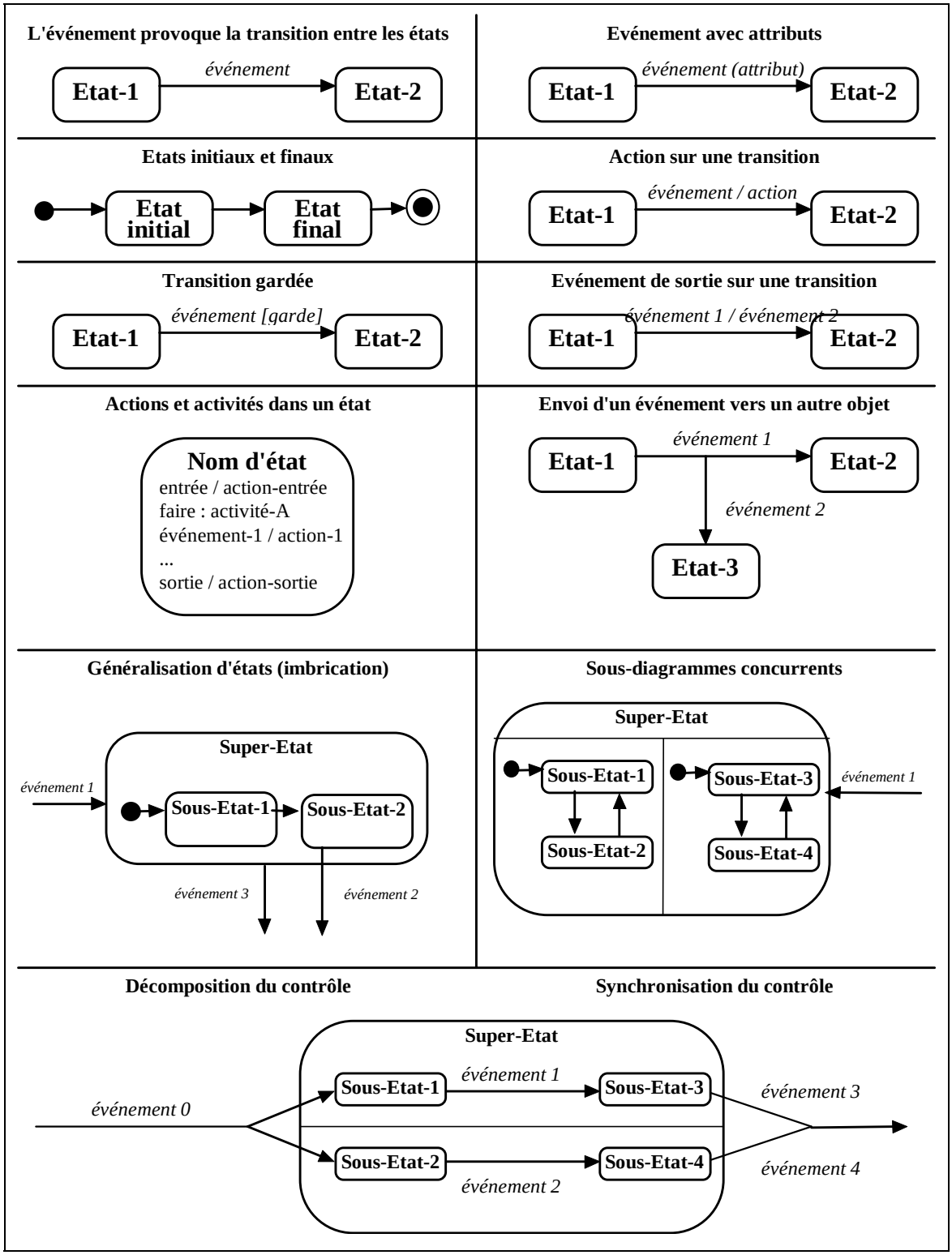


Figure 7-13 : Les représentations graphiques des Statecharts de Harel

7.5.3 Les réseaux de Pétri

Les réseaux de Petri [PET 62] sont un formalisme dédié à la modélisation du parallélisme dans les systèmes discrets. Un réseau de Petri est un graphe composé de deux types d'objets : les places et les transitions. Les places jouent le rôle de variables d'état et les transitions celui d'opérateurs de changement d'état. L'état du réseau, défini par les valeurs des variables d'état, est modélisé par un ensemble de marques indifférenciées (ou jetons) réparties sur les places du réseau. Des arcs peuvent relier des places aux transitions (on parle alors de places d'entrée) ou des transitions aux places (on parle alors de places de sortie). Un transition est franchissable si et seulement si toutes ses places d'entrée contiennent au moins une marque. Ce franchissement se traduira par une modification de l'état du réseau : on retirera une marque à chacune des places d'entrée et on déposera une marque dans chacune des places de sortie.

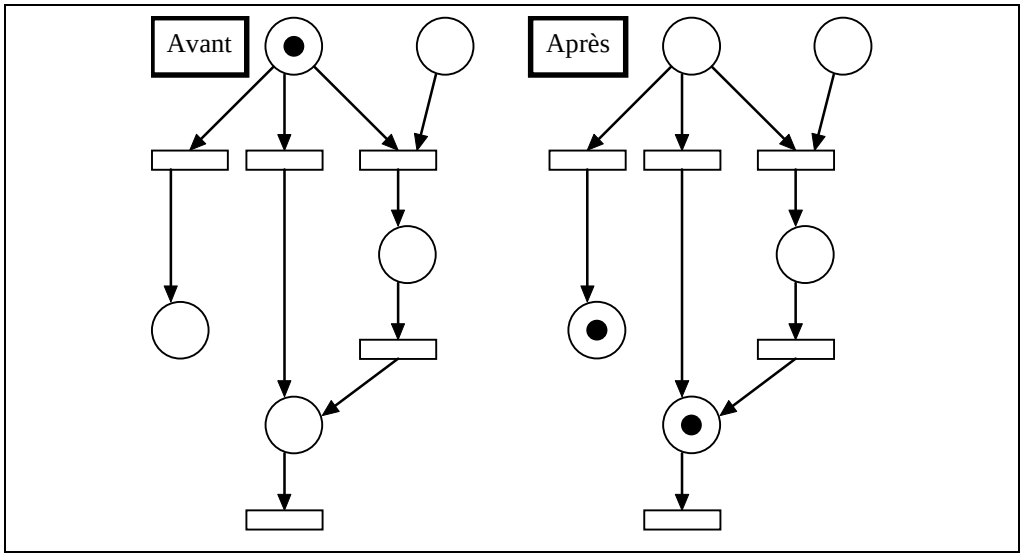


Figure 7-14 : Exemple de franchissement dans un réseau de Petri

Les réseaux de Petri ont fait l'objet de nombreuses études depuis une quinzaine d'années, et le modèle de base a été enrichi par des formalismes permettant d'accroître leur pouvoir d'expression. On notera par exemple, les Réseaux Prédicats/Transitions [GEN 86], les réseaux colorés [JEN 83] et les Réseaux de Petri à Objets [SIB 85].

Conçus pour modéliser des systèmes présentant un fort parallélisme, les Réseaux de Petri sont un outil puissant de formalisation de la concurrence et de la synchronisation. Cette puissance d'expression a favorisé leur utilisation dans des domaines variés tels que les protocoles de communication, l'architecture des ordinateurs, les interfaces homme-machine, etc. L'approfondissement théorique de leurs fondements mathématiques permet d'envisager de façon pratique la validation a

priori des modèles construits, ce qui présente un avantage certain par rapport aux autres formalismes décrits auparavant. On peut par exemple s'assurer de l'atteignabilité de certains états, du bon parallélisme de l'algorithme général, etc.

Les principales objections faites aux réseaux de Petri concernent la complexité de mise en œuvre du modèle pour des néophytes et les difficultés à structurer des modèles importants. La complexité des réseaux de grande taille arrive à dépasser les capacités de compréhension des concepteurs, et les rend difficiles à analyser et à modifier.

Les réseaux de Petri sont rarement utilisés dans les méthodes de conception de logiciels. On peut citer la méthode OOM (Object Oriented Merise) qui utilise les réseaux de Petri pour la description de la dynamique d'un système [BOU 94].

7.5.4 Notre choix

Parmi les trois classes de formalismes que nous venons d'étudier, notre choix s'est porté sur les Statecharts de Harel pour les raisons suivantes :

- Les Statecharts sont un outil puissant d'expression ; ils prennent en compte les exigences de structuration et de concurrence dans les systèmes.
- Les représentations graphiques proposées sont claires et assez aisément manipulables par un non-spécialiste en informatique.
- Les Statecharts peuvent être exécutés afin de vérifier si la modélisation correspond bien aux spécifications désirées.
- Outre les systèmes à événements, il est possible de modéliser un système à temps continu avec un Statechart, grâce au concept d'activité propre à un état : tant que l'on est dans cet état, l'activité se résume alors au calcul périodique des équations du modèle de simulation.

En revanche, on peut faire quelques remarques sur les représentations graphiques concernant la notion de super-état. Cette notion regroupe à la fois la composition structurelle d'états (visant à éviter l'explosion combinatoire) et la concurrence d'états. A notre avis, la représentation graphique d'états concurrents reflète mal le fait que ces états représentent le plus souvent la dynamique d'objets du monde réel véritablement indépendants. Harel a d'ailleurs envisagé d'avoir recours à des représentations en trois dimensions pour traduire cette structuration en profondeur et orthogonale.

7.6 LA MISE EN OEUVRE DES CONCEPTS A TRAVERS L'INTERFACE

Nous allons maintenant nous intéresser à la façon dont les formalismes retenus peuvent être mis en œuvre de façon interactive par un concepteur, et comment les résultats de ses spécifications peuvent être en partie validés.

Les différentes tâches à effectuer par le concepteur de modèles de simulation peuvent être résumées par le schéma suivant :

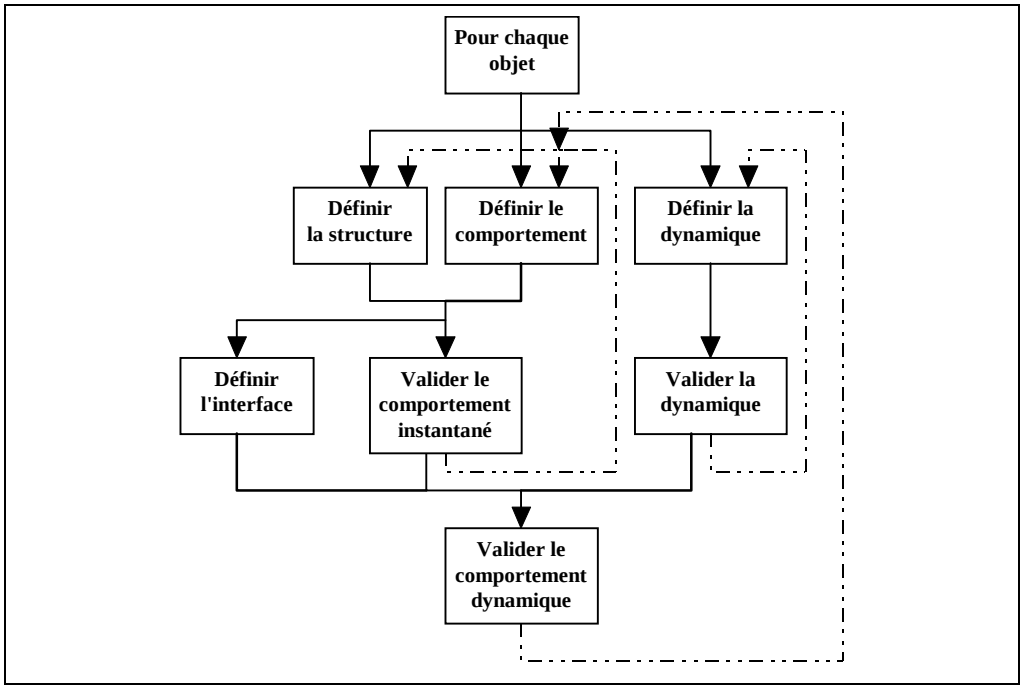


Figure 7-15 : Le processus de prototypage du modèle de simulation

On isole donc dans cette figure des tâches de définition et des tâches de validation :

- *Tâches de spécification* :

  ⇒ *Spécification de la structure* : cette tâche consiste à définir les propriétés de l'objet et à leur assigner un type.

  ⇒ *Spécification du comportement* : cette tâche consiste à définir les opérations qui peuvent être effectuées sur l'objet, qui permettront de modifier les valeurs des propriétés.

  ⇒ *Spécification de la dynamique* : cette tâche consiste à définir l'évolution temporelle de l'objet, au moyen des formalismes offerts par les Statecharts de Harel (états, événements, transitions, actions, activités, conditions, etc.).

  ⇒ *Spécification de l'interface* : cette tâche consiste à définir les opérations qui pourront être manipulées depuis l'extérieur, à préciser la visibilité externe de

l'objet, en vue de sa coopération au sein d'un système.

- *Tâches de validation*. Nous isolons plusieurs tâches de validation :

  ⇒ *Validation du comportement instantané*. Cette tâche consiste à valider les opérations qui ont été définies lors de la spécification du comportement, en examinant les modifications de l'état de l'objet lors de l'activation de ces opérations.

  ⇒ *Validation de la dynamique*. Cette tâche consiste à valider les diagrammes d'états produits grâce à des scénarios d'événements datés.

  ⇒ *Validation du comportement dynamique*. Cette tâche consiste en l'intégration des deux tâches précédentes, c'est à dire à valider de façon globale le comportement du modèle en l'activant grâce aux primitives définies dans l'interface.

7.6.1 Interfaces de spécification

L'interface de spécification se compose d'éditeurs spécialisés chacun dans la réalisation d'une tâche de spécification et la génération d'une partie du code. Ces éditeurs font au maximum appel à la manipulation directe. On distinguera ainsi :

- l'éditeur d'objet qui permet de définir les propriétés et les opérations du modèle.
- l'éditeur dynamique qui permet la création interactive des états, événements et transitions par manipulation directe.
- l'éditeur d'interface d'objet qui permet de définir la visibilité externe de l'objet.

Des exemples d'éditeurs de ce type ont été expérimentés dans le cadre du projet MELISA (Cf. Chapitre 11).

7.6.2 Interfaces de validation

Chacun des éditeurs permet la génération d'une partie du code du modèle. Il est important que le concepteur dispose d'un outil interactif lui permettant de vérifier le plus exactement possible le résultat de ses spécifications, comme le montre la figure 7-16.

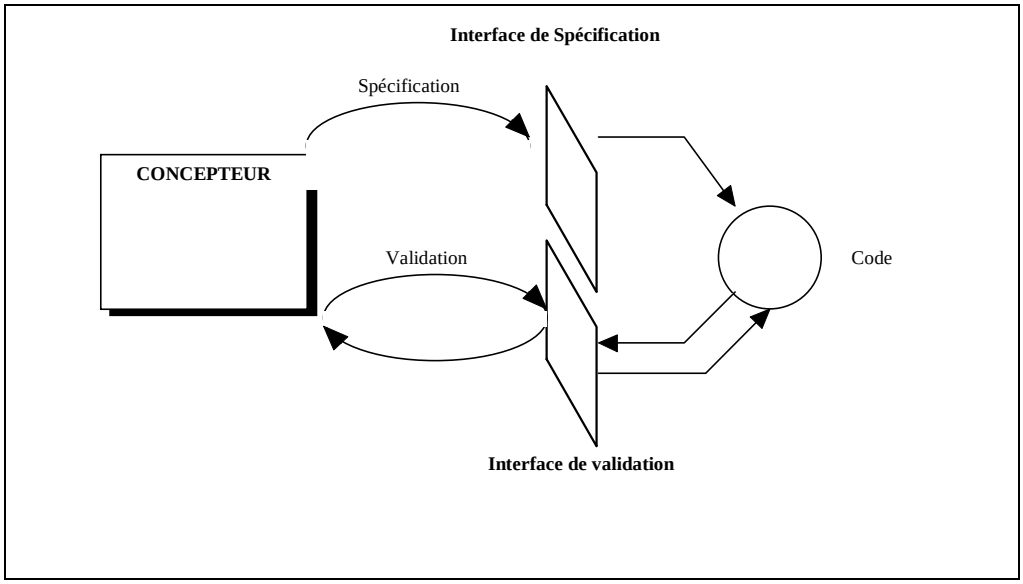


Figure 7-16 : Interfaces de spécification et de validation

Nous distinguons donc :

- *L'outil de validation de comportement instantané* qui permet de tester chacune des opérations définies dans l'éditeur d'objet.

  On peut grâce à cet outil activer une opération en fixant interactivement la valeur des paramètres de chaque opération. Cet outil permet au concepteur d'évaluer directement les incidences de cette activation sur les valeurs des propriétés du modèle. On retrouve ici certains concepts développés dans les outils de déboguage de code informatique. En règle générale, les interfaces de ces "debuggers" sont très textuelles et ne permettent pas une validation "visuelle" des résultats.

  Dans certains cas, et ceci s'est produit lors de la mise au point effective de modèles de simulation avec CORYS, il est nécessaire de développer une interface dont le seul but est d'interagir avec le modèle et de visualiser les résultats. La présentation sur un même écran de multiples données numériques ou textuelles devient rapidement inexploitable pour le concepteur ; on a alors recours à des interfaces plus visuelles (cadrans, curseurs, etc.) permettant d'appréhender de façon plus globale la complexité du système.

  Cette approche pose le problème de la génération automatique d'interface. On peut en effet imaginer que pour la nécessité de la mise au point du modèle de simulation, l'environnement recrée automatiquement, après chaque modification, une interface permettant la manipulation du modèle et la visualisation des résultats. Cette interface pourrait être abandonnée à la fin de la période de mise au point, ou bien être réutilisée pour servir de base à la présentation finale à fournir à l'apprenant.

- *L'outil de validation de la dynamique* qui permet de vérifier l'exécution des diagrammes d'état produits. Cette vérification est effectuée par la définition de scénarios d'événements (ici, le terme de scénario est employé dans le sens utilisé par Rumbaugh dans la méthode OMT [RUM 91]). Un scénario d'événements est une suite d'événements datés du type :

Scénario_Evénements = { (t_0,e_0) , (t_1,e_1) , ... } avec t_i = date et e_i = événement

Une fois les scénarios d'événements définis, un outil peut permettre de visualiser l'exécution de la dynamique par une animation des graphes d'états.

7.6.3 La réutilisation

Une fois qu'un modèle a été mis au point, il peut être considéré comme un objet encapsulé. En tant que tel, il peut être sauvegardé afin d'envisager une réutilisation ultérieure. L'interface doit donc proposer : (1) la possibilité de mettre en catalogue les modèles ou des composants du modèle, (2) la possibilité de rechercher dans les catalogues les composants afin de les réutiliser tels quels ou de les modifier.

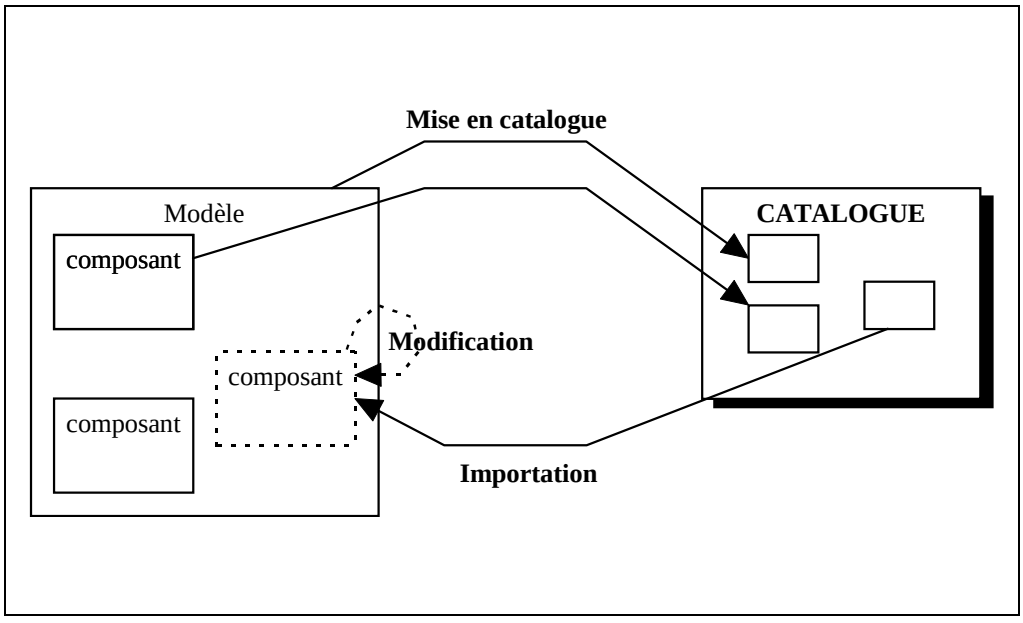
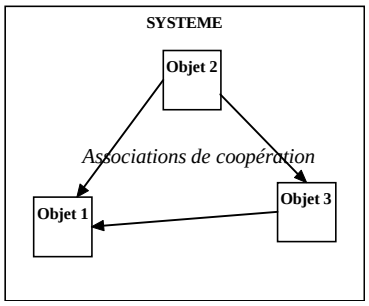


Figure 7-17 : Réutilisation de modèles et de composants

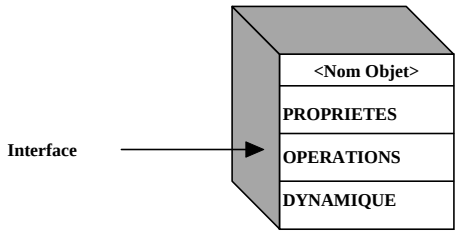
7.7 FORMALISMES RETENUS

Nous pouvons donc résumer les choix de formalismes que nous avons retenus dans ce chapitre par l'encadré suivant :

Un **système** est composé d'un nombre fini **d'objets** connectés entre eux au moyen **d'associations de coopération** [4]



Un **objet** a plusieurs facettes ; il est défini par une structure, un comportement et une dynamique. Il est accessible depuis l'extérieur grâce à une interface.
La **structure** d'un objet est définie par un ensemble **d'attributs** ayant chacun un type
Le **comportement** d'un objet est défini par un ensemble **d'opérations** applicables sur cet objet. La manipulation d'un attribut depuis l'extérieur ne peut se faire que par le biais d'une opération.
La **dynamique** d'un objet est défini par un **Statechart** de Harel, formalisme à base d'états et d'événements permettant en particulier la hiérarchisation des états et l'expression de la concurrence et de la synchronisation.
L'interface d'un objet précise le sous-ensemble des opérations de l'objet activables depuis l'extérieur.



[4] *pour plus de précisions sur le concept d'association de coopération, consulter le chapitre 10*

7.8 RESUME

Dans ce chapitre, nous avons abordé l'espace de travail Modèle.

Nous avons donné, en premier lieu, une définition des termes de système, de composant et de modèle de simulation.

Ensuite, après avoir décrit les différentes approches utilisées dans le processus de modélisation, nous nous sommes concentrés plus particulièrement sur l'approche par objets. Sur la base d'expériences d'enseignements effectués auprès de publics non-informaticiens, nous avons proposé de ne pas fournir explicitement certains concepts de l'approche par objets (classe, héritage), ces concepts étant difficilement assimilables ou manipulables par des auteurs non expérimentés en programmation. Nous proposons, en revanche, de fournir les concepts d'objets-prototypes et de clonage. Certains mécanismes plus sophistiqués, tels que la délégation ou la propagation de messages, la création de collections d'objets, permettent de compenser la diminution de puissance d'expression entraînée par l'abandon de la notion de classe et d'héritage. Ces mécanismes sont actuellement expérimentés auprès d'auteurs afin d'être validés.

Nous avons ensuite examiné les différents formalismes d'expression de la dynamique des systèmes à simuler. Nous avons retenu les StateCharts de Harel, parce qu'ils permettent un bon compromis entre puissance d'expression et aptitude à l'automatisation et à la vérification.

Enfin, nous avons décrit un processus de prototypage des modèles de simulation en précisant les contraintes auxquelles doivent obéir les interfaces de spécification et de validation.

8. LA SPECIFICATION DES SCENARIOS PEDAGOGIQUES

8.1 DEFINITIONS PRECISES DE LA NOTION DE SCENARIO PEDAGOGIQUE

Le deuxième espace de travail que nous allons étudier est réservé à la spécification des scénarios pédagogiques. La notion de scénario est fréquemment employée en informatique, souvent dans des acceptions bien différentes. Pour cette raison, il convient de lever les ambiguïtés en donnant une définition précise du terme.

8.1.1 La notion de scénario cinématographique

Le sens du terme scénario dans la langue courante est souvent lié à la notion de plan, de planification. Selon le dictionnaire, un scénario est un "*processus qui se déroule selon un plan préétabli*" [ROB 92]. L'usage sans doute le plus courant en est dans le sens "cinématographique". Un scénario cinématographique décrit, par avance, le découpage séquentiel d'un film en scènes, puis pour une scène donnée, le découpage en plans et enfin pour un plan donné, la description du décor, les attitudes et répliques de chacun des personnages. Cette acception du terme est fortement liée à la notion de structure séquentielle : les scènes se déroulent les unes après les autres, les plans se succèdent à l'intérieur d'une scène. Au niveau de la description d'un plan, le scénariste tente dans certains cas de décrire le déroulement en parallèle de l'action :

> *Au premier plan, X discute avec Y sur une table de café. Pendant ce temps, à l'arrière plan, une dispute oppose le garçon de café à un client irascible.*

Cette définition cinématographique du terme scénario fait ressortir deux problèmes. Le premier concerne la différence entre la réalité que l'on veut décrire (ici, la simultanéité de deux événements) et les formalismes d'expression (ici, deux phrases enchaînées séquentiellement essayant de traduire la simultanéité) comme le montre la figure suivante.

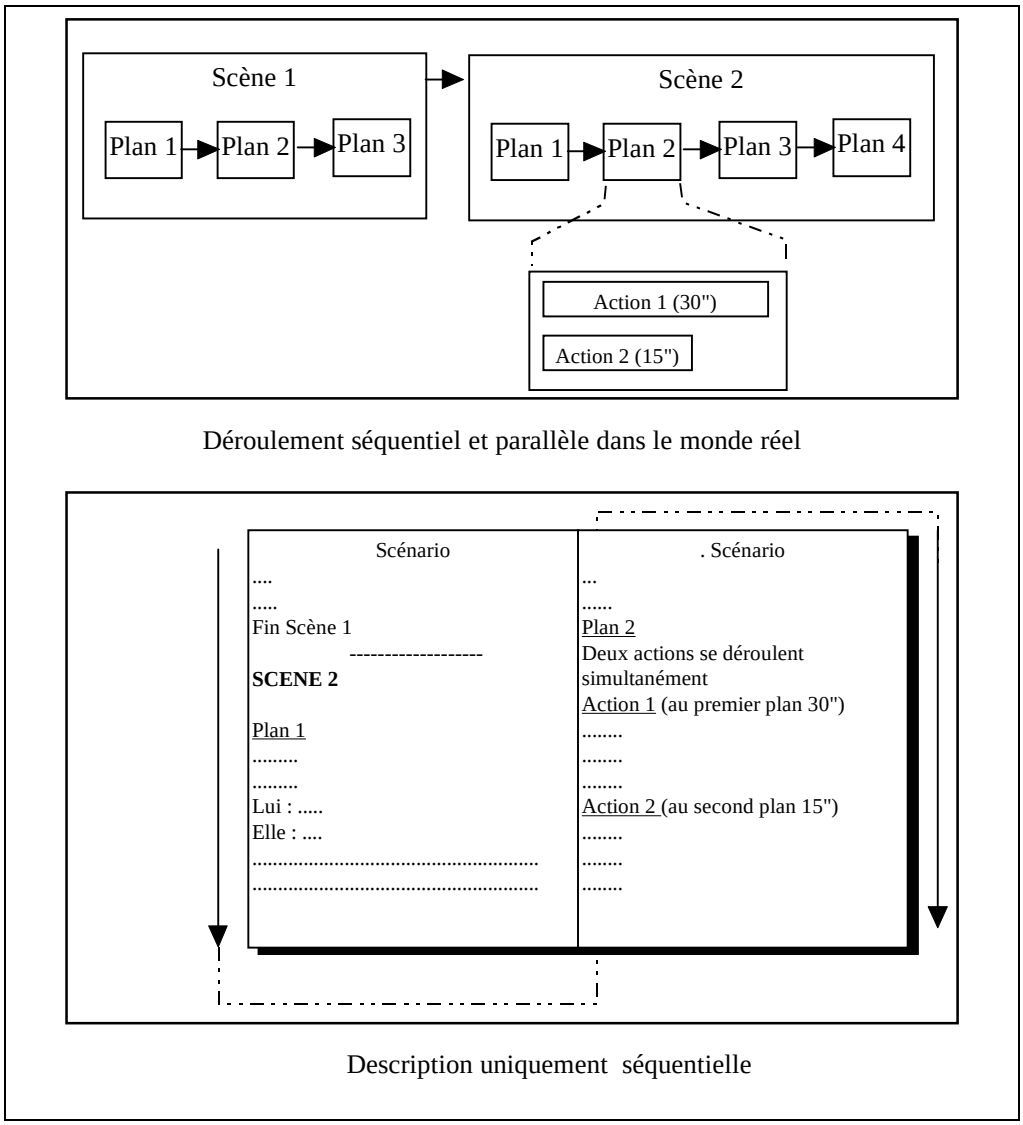


Figure 8-1 : Les limites de l'expression séquentielle

Ce problème de formalisation est important à la fois en ce qui concerne *l'expression du problème* et *l'interprétation du résultat :* le choix du scénariste de décrire l'action 1 avant l'action 2 signifie-t-il qu'il y accorde plus d'importance ? La difficulté de la langue orale ou écrite à traduire certaines contraintes temporelles doit être palliée par le recours à d'autres formalismes, graphiques le plus souvent. Dans le domaine cinématographique, le concept de *storyboard* est d'ailleurs directement issu de cette préoccupation. Cette volonté d'utiliser des représentations graphiques,

davantage lisibles, se retrouvera lors de nos choix de formalisation des scénarios.

La seconde remarque concerne la finalité du scénario. Il a pour but de fournir des consignes aux personnes chargées de l'exécuter (réalisateur, scripte, comédiens, ...). Il peut donc être considéré comme un document *prescriptif*, son rôle étant de décrire ce qui doit se dérouler. Le non-respect de ces prescriptions est rare et concerne en général des détails, tels que les répliques prononcées par les comédiens. Nous débouchons donc sur une première définition :

> *Scénario = document prescriptif décrivant un enchaînement temporel de tâches qui devront être exécutées selon un plan préétabli.*

8.1.2 La notion de scénario d'enchaînement pédagogique

La définition précédente peut être directement appliquée dans le domaine de la formation où le pédagogue établit une progression qui définit l'enchaînement séquentiel des unités d'apprentissage (modèle skinnerien [SKI 76]). Un enseignant dans une discipline déterminée répartit son cours en modules successifs, chaque module étant lui-même découpé en un certain nombre de séances. Il organise les modules et les séances afin d'assurer la meilleure progression à ses élèves. Notons que le terme de *scénario pédagogique* recouvre souvent cette définition que nous qualifierons de *scénario global* ou *scénario d'enchaînement*.

L'introduction de l'ordinateur dans la formalisation du processus de formation apporte une nouvelle dimension à cette conception du scénario pédagogique. En effet, les capacités de calcul offertes par la machine permettent d'automatiser l'exécution d'un scénario, vu non plus comme une simple séquence, mais comme un réseau d'unités d'apprentissage (modèle crowderien [CRO 76]). Dans ce réseau, l'apprenant navigue, soit de façon libre, soit de façon guidée, le guidage étant assuré par un module "intelligent" capable d'interpréter le comportement de l'apprenant et de prendre les décisions adaptées.

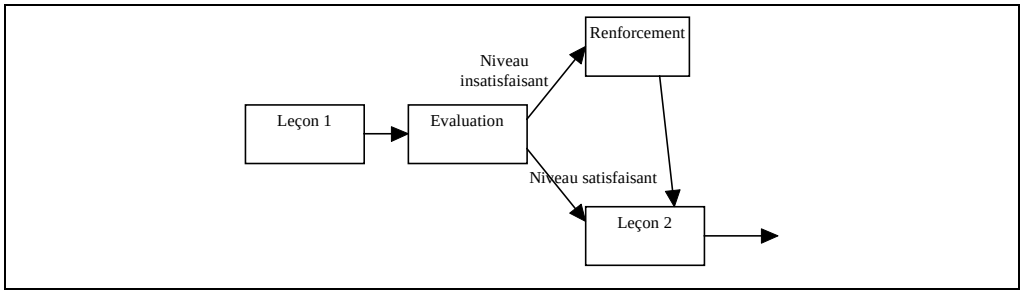


Figure 8-2: Un scénario pédagogique adaptable

Nous pouvons noter, d'ailleurs, que ce type de stratégie existe, mais de façon intuitive et très rarement formalisée, dans l'enseignement traditionnel.

Avec cette démarche, nous nous écartons de la définition précédente dans le sens où le scénario précise, non plus ce qui va être fait, mais ce qui peut être fait. C'est en fonction des choix et/ou du comportement de l'apprenant que telle ou telle séquence sera exécutée. Nous arrivons donc à une deuxième définition :

> *Scénario = document prescriptif décrivant l'enchaînement temporel de tâches qui pourraient être exécutées, en respectant un plan préétabli.*

8.1.3 La notion de scénario de résolution pédagogique

Jusqu'à ce point, nous avons considéré le scénario comme une base servant à décrire par avance les comportements possibles de l'utilisateur. Qu'en est-il de ce principe quand nous nous intéressons, non plus à l'enchaînement des unités d'apprentissage, mais au niveau inférieur, au déroulement d'une unité d'apprentissage déterminée, en particulier s'il s'agit d'une simulation pédagogique ?

A ce niveau, l'apprenant interagit avec un modèle de simulation avec des degrés de liberté plus ou moins importants. Il dispose, à travers l'interface, d'objets lui permettant de modifier l'état du modèle et d'autres lui permettant de percevoir l'effet de ces modifications. Dans ce cas, comme nous l'avons dit au chapitre 2, plusieurs stratégies peuvent être appliquées :

- l'apprenant a toute liberté d'interagir avec la simulation, et aucun contrôle n'est effectué par le système : on parle alors de simulation libre.
- L'apprenant a toute liberté d'interagir avec la simulation, mais le système contrôle son comportement. Ce contrôle peut être effectué soit à des fins d'analyse ultérieure par le pédagogue, soit à des fins d'autoévaluation sur sollicitation explicite de l'apprenant.
- L'apprenant est guidé par le système dans ses interactions avec la simulation. Le système contrôle et interprète son comportement pendant la session. Les stratégies de guidage peuvent être plus ou moins directives [NIC 88].

Dans les deux derniers cas, le contrôle du comportement de l'apprenant suppose l'existence préalable d'un plan précisant le ou les comportements corrects de l'élève. C'est sur la distance existant entre ce plan théorique et la réalité du comportement de l'apprenant que va reposer le contrôle du système.

Ce problème de planification du comportement de l'élève a fait l'objet de

nombreuses études, en particulier dans le domaine de l'Enseignement Intelligemment Assisté par Ordinateur. Un plan d'apprentissage est *un plan structuré défini par un acquis supposé de l'apprenant et par un objectif d'apprentissage* [CHO 93]. Cette définition fait référence au concept classique de plan d'action dans la démarche de résolution de problèmes en intelligence artificielle :

> *... résoudre un problème consiste à développer et à ordonner avant sa mise en œuvre un ensemble d'actions qui permettent depuis une situation initiale ("connue") d'atteindre une situation finale ("désirée") : le but, correspondant à la solution du problème. Si l'on considère que cet ensemble d'actions est totalement ordonné dès son élaboration et qu'il forme ainsi toujours une séquence d'actions, une telle séquence est appelée plan d'action ....[POR 91]*

On retrouve, dans le domaine des tuteurs intelligents, les concepts de *plan prédicatif* et de *plan effectif* [CAN 90] qui décrivent respectivement la stratégie de résolution souhaitée d'un problème et la stratégie effective adoptée par l'utilisateur. Dans ce contexte, un *scénario pédagogique* est considéré comme l'exécutant d'un plan et son rôle est de [CHO 93] : (1) activer un plan prédictif précis, (2) déterminer la surveillance associée ; (3) vérifier le bon fonctionnement du plan en fonction des informations mises à sa disposition ; (4) avertir le module "tuteur" des éventuels dysfonctionnements dans l'exécution du plan, tuteur qui peut décider d'activer d'autres scénarios pédagogiques.

Nous aboutissons donc maintenant à une nouvelle définition du terme de scénario pédagogique plus complexe que les précédentes :

> *Pour un problème déterminé, un scénario pédagogique décrit : (1) un plan de résolution de ce problème et (2) la stratégie de guidage pédagogique associée en cas de non-respect de ce plan par l'élève*

8.1.4 Notre définition du scénario de contrôle pédagogique

Nous avons annoncé dans le chapitre 2 que nous ne nous inscrivions pas dans une démarche de type cognitif et que nous ne nous préoccupions pas de modéliser les raisonnements du pédagogue et de l'élève. Nous partons d'une approche beaucoup plus pragmatique et nous voulons proposer des solutions opérationnelles aux concepteurs de scénarios. Rappelons les spécificités de notre approche :

- Nous ne nous situons pas dans un contexte de production lourde de logiciels. La liste des rôles que nous avons définis précédemment inclut le pédagogue, mais pas

de didacticien ni d'expert en modélisation du raisonnement humain. Le pédagogue doit être capable de modéliser de façon simple le contrôle pédagogique à exercer sur la simulation.

- Un des critères que nous jugeons le plus important est *l'acceptabilité*, c'est-à-dire la capacité du logiciel de simulation pédagogique à être jugé pertinent par son utilisateur, l'apprenant. En particulier, l'interprétation du raisonnement de l'élève par le logiciel est souvent perçue comme un abus de pouvoir de la machine, surtout si cette interprétation est erronée. Cette contrainte nous impose que toute intervention pédagogique doit, d'une part pouvoir être expliquée (par le pédagogue "automatisé"), et d'autre part, être comprise et acceptée par l'apprenant. Ceci nous oriente naturellement vers un guidage de type "discret".

Nous nous sommes donc orientés vers une définition de scénario pédagogique moins ambitieuse que la précédente, basée sur la notion d'étapes intermédiaires. Pour expliquer cette définition, nous allons utiliser un exemple concret, celui d'une manipulation en classe de chimie.

Métaphore de la manipulation en classe de chimie

Chacun de nous se souvient, parfois avec émotion, des travaux pratiques de chimie effectués au lycée. Il arrive que le sujet de ces T.P. concerne la manipulation de produits dangereux en vue d'étudier une réaction chimique.

L'enseignant fournit à chaque groupe un ensemble de matériel (éprouvettes, récipients, ...) et de produits. Il rappelle à la classe le contexte de l'expérience, détermine l'objectif à atteindre et le temps accordé, et fixe les consignes de sécurité en précisant éventuellement les opérations qu'il faut absolument éviter de faire.

Réfléchissons maintenant à l'attitude de l'enseignant durant une telle manipulation.

Tout au long du déroulement de l'expérimentation, il passe de groupe en groupe pour évaluer l'avancement des travaux. Devant chaque groupe, il pourra avoir l'une des attitudes suivantes :

(1) il ne dit rien, et c'est sans doute le cas le plus fréquent. Les opérations ont l'air de se dérouler à peu près normalement.

(2) il constate qu'au moment où il passe, l'état de l'expérience ne correspond pas à un état auquel aurait conduit une démarche normale, mais que cet état n'est pas dangereux. Il peut questionner les élèves pour leur demander quelle est leur démarche, et en fonction des réponses, donner une indication ou se taire.

(3) il constate qu'au moment où il passe, l'état d'avancement de l'expérience est tel qu'elle présente un danger. Il prend alors les mesures qui s'imposent pour stopper cette évolution et explique aux élèves la nature de leur erreur.

(4) il constate *de visu* une opération dangereuse ou interdite, volontaire ou non de la part d'un élève. Il prend immédiatement les mesures nécessaires, et explique, plus ou moins poliment, l'erreur à son auteur.

Enfin, à la fin de la séance, l'enseignant détermine avec chacun des groupes le succès ou l'échec de l'expérimentation, en apportant si nécessaire des précisions complémentaires.

Cet exemple présente pour nous de nombreux intérêts. Il permet en particulier d'identifier un certain nombre de stratégies de contrôle que le pédagogue est habitué à employer dans la vie réelle. Nous pouvons donc faire les remarques suivantes :

- Une première distinction peut être faite entre *l'évaluation du résultat final* et la *prise en compte de résultats intermédiaires*.
- Quand il se déplace d'un groupe d'élève à un autre, l'enseignant change de contexte. Face au nouveau groupe, il tente en premier lieu d'évaluer l'avancement des travaux en *surveillant l'état* de la manipulation en cours. Il ne sait en rien, à ce stade là, comment les élèves sont parvenus à ce résultat, quel a pu être leur raisonnement, ni même quelles opérations ils ont effectuées. Ce n'est que lors d'une éventuelle discussion avec le groupe qu'il peut tenter de préciser ces éléments.
- Dans le cas (4), l'enseignant ne constate pas cette fois-ci un état, mais une *opération effectuée* par un élève. Cette seule constatation lui suffit pour réagir, l'opération effectuée n'étant pas valide dans ce contexte particulier. On entend ici par contexte, l'état précédent mémorisé par l'enseignant (ce peut être l'état initial) et le but final à atteindre.
- A chaque fois qu'il réalise une évaluation de l'état de l'expérimentation, et selon la gravité des situations, l'enseignant peut avoir une *réaction différente* : ne rien dire, conseiller, expliquer, prendre les mesures techniques qui s'imposent.

Nous voyons bien, dans cet exemple, que 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.

Qu'advient-il de cette approche si l'on remplace une véritable expérimentation

par une simulation sur ordinateur ? Le cas de la simulation libre correspond à l'absence du professeur de la salle de T.P. Le cas de la simulation contrôlée correspond au remplacement du professeur par un composant informatique susceptible de se comporter de façon analogue. Nous situant dans une démarche de guidage discret, et ne désirant pas interpréter les intentions de l'apprenant, nous abandonnons l'idée d'automatiser la phase d'interrogation des élèves. En définissant des étapes intermédiaires d'analyse, nous répondons ainsi partiellement à un question soulevée dans le domaine des tuteurs intelligents :

> *Le problème de la compréhension des intentions de l'élève peut être approché avec moins de difficultés si l'on se limite à l'analyse d'étapes intermédiaires, analyse dans laquelle il suffit de faire une conjecture sur une sous-étape que l'élève a réalisée mentalement, mais qu'il n'a pas exprimée au tuteur [NIC 88].*

Pour une simulation donnée, nous appelons donc *scénario de contrôle pédagogique de simulation* la définition abstraite du contrôle. Cette définition comprend :

- l'état initial dans lequel est placé l'apprenant au début de la simulation.
- l'objectif à atteindre.
- une description structurée des états intermédiaires et des opérations que le pédagogue veut analyser.
- la réactivité lors de l'analyse de chaque sous-étape ou opération en fonction des résultats obtenus.

Nous arrivons donc à nos propres définitions du terme de scénario pédagogique : nous identifions deux notions différentes, celle de *scénario d'enchaînement pédagogique* et celle de *scénario de contrôle pédagogique de simulation.*

Scénario d'enchaînement pédagogique
Description abstraite de l'enchaînement des unités d'apprentissage au sein d'un logiciel pédagogique
Scénario de contrôle pédagogique de simulation
S'applique à une unité d'apprentissage basée sur la simulation
Description abstraite du contrôle pédagogique d'une simulation et de la réactivité associée, en termes d'états intermédiaires ou d'actions à analyser lors de l'exécution de cette simulation.

8.2 LE PROFIL DES CONCEPTEURS

Comme nous l'avons vu dans l'exemple présenté au chapitre 4, les experts pédagogues seront chargés de la conception des scénarios pédagogiques. Leurs compétences concernent le domaine de l'enseignement et de la formation : ils sont capables de décrire une ou plusieurs approches pédagogiques pour aborder l'apprentissage d'un concept, d'un savoir ou d'un savoir-faire. En revanche, ils ne sont pas sensés posséder de compétences spécifiques en modélisation informatique ou en modélisation du raisonnement humain.

8.3 LES CONCEPTS DU DOMAINE

La notion de *scénario de contrôle pédagogique de simulation* présentée ci-dessus permet de préciser notre point de vue sur une certaine philosophie de contrôle pédagogique que nous préconisons. Il nous faut maintenant davantage préciser les concepts à exprimer dans les cas les plus fréquents de simulation que nous avons à traiter.

Nous avons mené un travail dans ce sens avec la division T.P.E.C. de Hewlett-Packard. Il ressort de cette étude les éléments suivants :

- Une simulation est caractérisée par un état initial et un objectif à atteindre.
- Analyse des états intermédiaires du système simulé. Cette analyse peut être effectuée pour répondre à deux types d'exigences :

  ⇒ La vérification que l'apprenant franchit de façon correcte les étapes intermédiaires correspondant à la démarche souhaitée de résolution de problèmes. Ces étapes peuvent être par exemple : (1) le diagnostic d'un dysfonctionnement, (2) la réparation du système et (3) le test du système après réparation.

  ⇒ La vérification que l'apprenant ne place pas le système dans un état dangereux ou tel qu'il signifie qu'il n'a pas compris la tâche à résoudre. Ainsi, dans l'exemple du démarrage de l'ordinateur décrit au chapitre 11, le fait que la machine se retrouve déconnectée démontre que l'apprenant n'a pas du tout saisi l'objectif de l'exercice.
- Analyse des actions de l'apprenant. Cette analyse peut être faite pour :

  ⇒ Affiner l'évaluation du comportement de l'apprenant entre deux sous-étapes. Par exemple, dans le cas d'un défaut du transformateur à l'intérieur d'une machine, le fait de tester avant tout la présence du courant à la prise est un élément démontrant une démarche systématique et doit de ce fait être reconnu.

⇒ Détecter des opérations non pertinentes, voire dangereuses. Par exemple, le fait de déconnecter un P.C. alors qu'une application tourne sous Windows comporte de gros risques de perte d'informations.

- Respect de certaines contraintes :

  ⇒ Contraintes d'enchaînement. L'enchaînement séquentiel des états intermédiaires n'est pas le seul envisagé. Le pédagogue peut être amené à exprimer par exemple, la vérification simultanée de deux sous-états.

  ⇒ Contraintes temporelles. Il est fréquent que le concepteur de scénario veuille fixer un temps maximum imparti pour l'exécution totale du scénario ou pour le franchissement d'une sous-étape.

  ⇒ Contraintes d'exclusivité sur une action donnée. Il peut arriver que dans certains contextes, il soit nécessaire à un moment donné de ne réaliser qu'une et une seule opération. C'est le cas par exemple, si l'on veut s'assurer de la maîtrise par un opérateur d'une connaissance purement procédurale de type vérification de checklist en aéronautique.

- Réactivité du système. Chaque réaction est consécutive à l'atteinte d'un sous-état, à l'occurrence d'une action, ou au dépassement d'un temps limite.

  ⇒ la cause de la réaction peut être soit :

    ◊ un succès : le sous-état a été atteint ou l'action attendue a été réalisée.

    ◊ un échec : le sous-état n'a pas été atteint ou l'action n'a pas été réalisée dans le temps imparti.

  ⇒ la nature de la réaction peut être :

    ◊ non perçue par l'apprenant. C'est le cas lorsque l'application pédagogique a pour but d'évaluer de façon interne un comportement et conserve un historique du travail effectué (en vue, par exemple, de l'attribution d'une note).

    ◊ Un message (textuel, sonore, graphique, ...) délivré à l'apprenant.

    ◊ Une opération sur la simulation elle-même. Par exemple, il peut être décidé, en cas d'échec de l'apprenant, de lui expliquer la nature de son erreur puis de placer la simulation dans un nouvel état.

    ◊ L'activation d'une autre unité pédagogique, afin par exemple, de fournir une démonstration automatique du comportement qui était attendu de la part de l'apprenant (par document expositif, vidéo, etc.).

Nous pouvons donc résumer par le schéma suivant les concepts ainsi dégagés :

Caractérisation d'une simulation	- Etat initial dans lequel est placé l'apprenant - Objectif à atteindre par l'apprenant
Analyse des sous-états	- Franchissement d'étapes intermédiaires - Détection des erreurs
Analyse des actions	- Affinement de l'évaluation du comportement - Détection des actions non pertinentes
Contraintes	- Enchaînement - Temporelles - Exclusivité
Réactivité	- Historisation - Message - Action

Figure 8-3 : Concepts liés au scénario

Une fois ces concepts énoncés, nous avons recherché des formalismes susceptibles de les traduire. Nos préoccupations sont en fait de deux ordres :

- Formaliser la structuration du contrôle défini dans un scénario de simulation pédagogique.
- Décrire, sur la base de cette structure, la réactivité associée au scénario.

Pour le premier aspect, nous avons orienté nos recherches vers les formalismes d'analyse de la tâche utilisés en Interaction Homme Machine (IHM). La problématique posée par ces formalismes recouvre partiellement la nôtre. Sur la base de cette étude, nous avons donc défini une première proposition de formalismes.

Pour le second point, notre démarche consiste à enrichir la structure de contrôle définie par des éléments relatifs à l'exécution de cette structure (la réactivité).

8.4 LES FORMALISMES D'ANALYSE DE LA TACHE EN IHM

Rappelons brièvement les objectifs de l'analyse de la tâche en IHM. D'après [BAL 94], l'utilisation de modèles de tâche peut avoir diverses finalités :

- Le modèle de tâche peut être vu comme un outil de conception d'un système interactif. Dans ce cas, il s'intéresse à une ou plusieurs phases de développement du logiciel : Définition des besoins, Spécifications externes ou Spécifications détaillées.

- Le modèle de tâche peut être considéré comme un outil permettant l'analyse prédictive de l'utilisabilité d'une interface. Il s'intéresse alors à éviter les erreurs de conception les plus grossières et donc à guider des choix de conception avant la mise en œuvre.
- Le modèle de tâche peut être un composant logiciel intégré dans le logiciel final. Il peut alors être commodément utilisé pour proposer une aide dynamique à la résolution de tâches.

La majorité des modèles existants s'intéresse aux deux premiers types d'utilisation et s'inscrit donc dans une perspective de définition "a priori" d'interfaces. La figure 8-4 résume cette classification :

Finalité	**Modèles**
Analyse de Besoins	DIANE, JSD*, MAD, CLG
Spécifications externes	ETAG, UAN
Conception logicielle	SIROCO, ADEPT
Analyse prédictive	GOMS, CCT, ETIT

Figure 8-4 : Une classification des principaux modèles de la tâche [BAL 94]

Tous ces modèles utilisent un vocabulaire plus ou moins analogue que nous allons tout d'abord préciser à travers les définitions proposées par S. Balbo [BAL 94] :

- **Une tâche** est un but que l'utilisateur cherche à atteindre. Ce but est assorti d'une procédure ou d'un plan qui décrit les moyens pour atteindre ce but [NOR 92].
- **Un but** est un état du système que l'utilisateur souhaite atteindre.
- **Une procédure** (ou composant exécutif d'une tâche) est un ensemble d'opérations organisées par des relations structurelles et temporelles.
- **Une opération** est soit une action soit une tâche.
- **Une action** désigne une opération terminale. Elle intervient dans l'accomplissement d'un but terminal.
- Un but terminal et les actions qui permettent de l'atteindre définissent une **tâche élémentaire.**
- **Les relations temporelles** entre les opérations d'une procédure traduisent la séquentialité, l'interruptibilité et le parallélisme.
- **Les relations structurelles** servent à exprimer la composition logique des

opérations ou la possibilité de choix.

- **Une tâche** est dite **composée** si sa description inclut des sous-tâches.

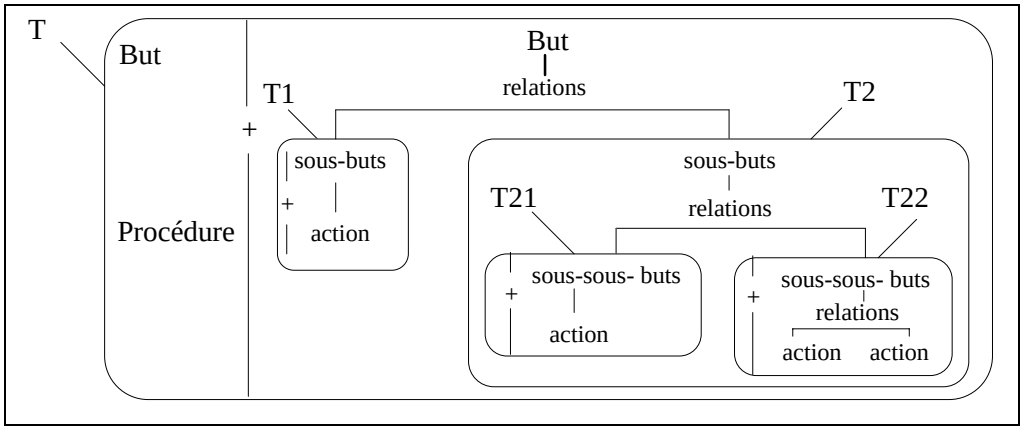


Figure 8-5 : Une définition de tâche selon S. Balbo, adaptée de [NOR 92].

Par rapport à ce canevas général, les modèles de tâches se distinguent principalement sur les points suivants :

- la définition de la granularité d'une tâche élémentaire ou action. Cette granularité peut se situer à des niveaux différents depuis une description abstraite des fonctions du système jusqu'à la description d'une action physique (une action physique est une opération indivisible que l'utilisateur applique à un dispositif d'entrée physique).
- La richesse des relations temporelles ou structurelles proposées. Chaque modèle propose un éventail plus ou moins large de relations parmi lesquelles on relève :

  ⇒ la séquence : A puis B

  ⇒ la composition ou séquence non ordonnée : A et B dans un ordre quelconque

  ⇒ l'alternative : A ou B

  ⇒ l'itération : n fois A

  ⇒ le parallélisme vrai : A en même temps que B

  ⇒ l'entrelacement : A en même temps que B, de manière entrelacée

  ⇒ l'existence de délais (minimums ou maximums) entre l'exécution de plusieurs tâches

  ⇒ le caractère obligatoire ou optionnel d'une tâche
- les formalismes utilisés. Les différents modèles utilisent des formalismes qui dépendent fortement de l'objectif visé [SEB 91], et qui vont des grammaires aux réseaux de Petri, en passant par des représentations graphiques ainsi que des langages.

- la lisibilité. La lisibilité d'un formalisme vient souvent en contradiction avec la rigueur de son pouvoir d'expression [BAL 94]. Parmi les modèles cités, ceux qui reposent sur une représentation graphique de type graphe orienté ont l'avantage de la clarté et de la concision. Les modèles textuels, basés sur un raisonnement de type algorithmique, sont plus puissants que les modèles graphiques et sont mieux adaptés à l'expression de relations temporelles complexes.
- la capacité à l'automatisation.
- La facilité d'utilisation. Pour un utilisateur "non-averti", l'utilisation de méthodes telles que UAN [HAR 92] et XUAN [GRA 94] semble très difficile. En effet, ces deux méthodes nécessitent l'apprentissage d'un langage complexe. A contrario, les méthodes graphiques ont l'avantage d'être claires et peu chargées en formalismes différents.

8.5 UNE PROPOSITION

En ce qui concerne la formalisation de la structuration du contrôle pédagogique, notre première approche a consisté à nous inspirer des modèles de tâches utilisés en IHM

Il existe des différences notables entre ces modèles de tâches et nos objectifs :

- une *différence de finalité*. La structuration des tâches de l'utilisateur est utilisée le plus souvent en IHM pour décrire ou évaluer de façon anticipée les interfaces à développer. Notre objectif n'est pas le même et consiste à donner au pédagogue les moyens d'exprimer ce qu'il veut observer du comportement de l'élève au moment de l'utilisation de la simulation.
- une *différence de niveau d'abstraction*. Les modèles IHM s'intéressent le plus souvent au niveau lexical, celui de la manipulation directe des éléments de l'interface. Nous nous situons à un niveau beaucoup plus abstrait, et nous nous intéressons principalement à l'aspect sémantique de l'interaction entre l'apprenant et la simulation.

Nous avons donc dû adapter ces modèles en proposant notamment au pédagogue :

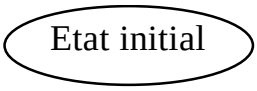
- nos propres définitions des termes utilisés (§8.5.1).
- la possibilité de surcharger les structures définies par l'expression de comportements erronés (§8.5.2).
- la possibilité d'exprimer la réactivité du système lors de l'exécution (§8.5.3).

8.5.1 Définitions générales et notations

Le formalisme décrit ici est issu d'une étude présentée dans le mémoire de DEA de Sophie Bordon [BOR 95].

Un scénario pédagogique est caractérisé par :

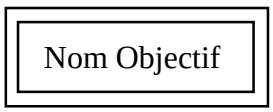
- un **état initial** de la simulation dans lequel doit se trouver l'élève au début de l'exercice.



- un **objectif général** fixant le but à atteindre par l'élève. Il correspond à un état du système.

Les étapes ou actions intermédiaires à analyser pour le suivi pédagogique de l'élève s'expriment par des objectifs :

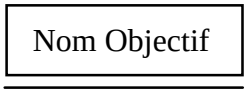
- un **objectif élémentaire,** qui est associé à une opération physique.



- Un **objectif composé** (non élémentaire), qui peut être :

  ⇒ **décrit** lorsque l'on s'intéresse à la façon d'atteindre cet objectif. Il est alors associé à une procédure décrivant les sous-objectifs à atteindre pour que l'objectif composé soit atteint.

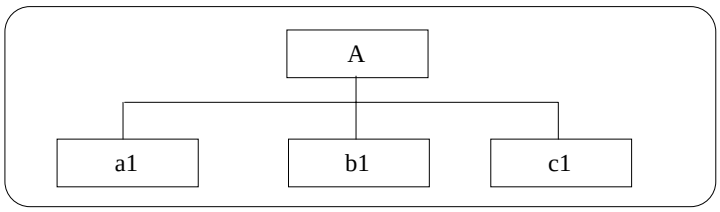
  ⇒ **non décrit** lorsque l'on ne s'intéresse qu'au résultat, et que l'on souhaite ignorer la façon dont l'objectif a été atteint. Aucune procédure n'est alors associée à l'objectif.



Les relations structurelles ou temporelles qui relient les différents objectifs s'expriment à l'aide **d'opérateurs**. Parmi ces opérateurs, on distingue des opérateurs n_aires et de opérateurs unaires.

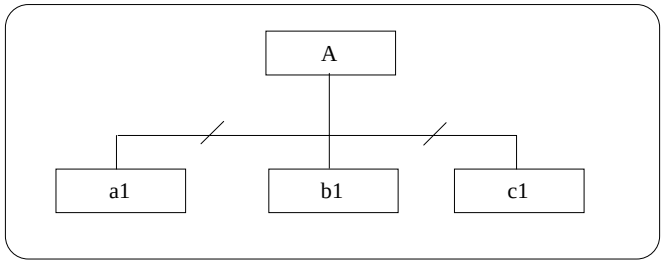
OPERATEURS N-AIRES

séquence ordonnée :



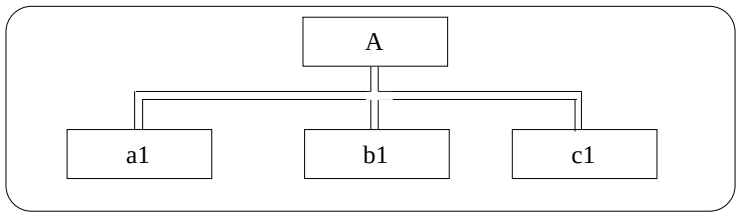
L'utilisateur doit atteindre l'objectif a1, puis l'objectif b1 et enfin l'objectif c1.

séquence non ordonnée :



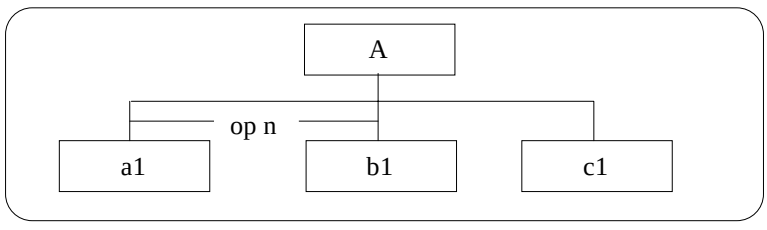
L'utilisateur peut atteindre les objectifs a1, b1 et c1 dans n'importe quel ordre mais ces trois objectifs sont obligatoires.

l'alternative :



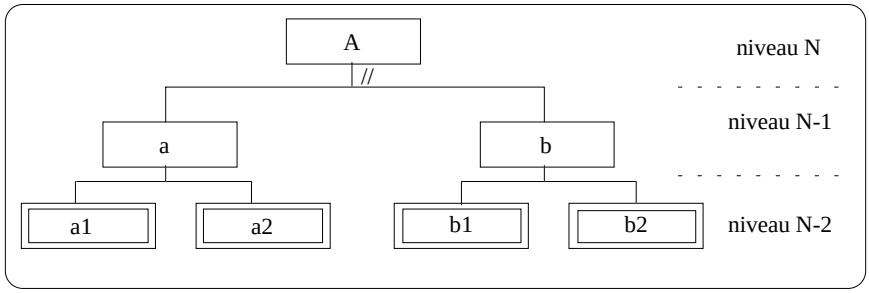
L'utilisateur peut atteindre soit l'objectif a1, soit l'objectif b1, soit l'objectif c1 (disjonction).

temporalité :



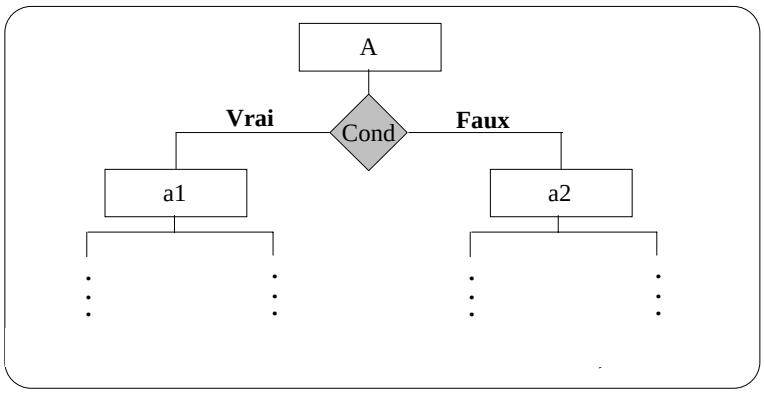
L'opérateur op peut être : ≤, < , >, ≥ et n exprime le nombre d'unités de temps (minimum ou maximum) qui peut s'écouler entre la réalisation de l'objectif a1 et la réalisation de l'objectif b1

l'entrelacement bidirectionnel :



// : signifie que les procédures permettant d'atteindre les objectifs a1 et b1 peuvent être entrelacées tout en respectant la structure qui leur est propre.

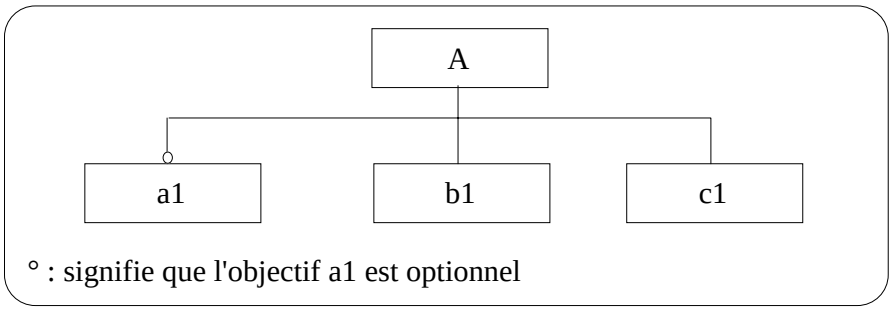
le choix



Si Cond est évaluée à vraie, alors l'objectif a1 doit être atteint, a2 sinon

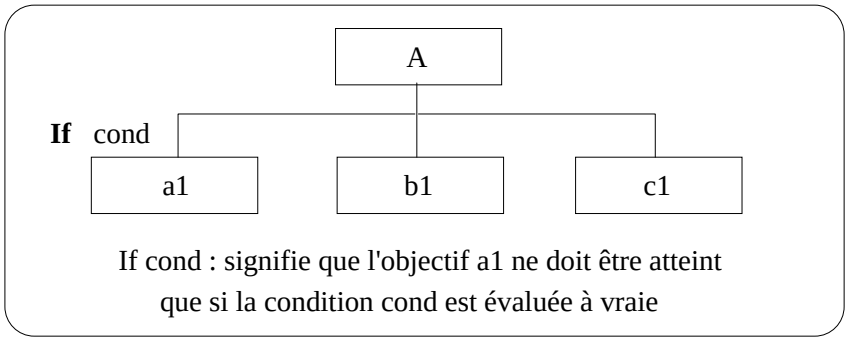
OPERATEURS UNAIRES

l'option :



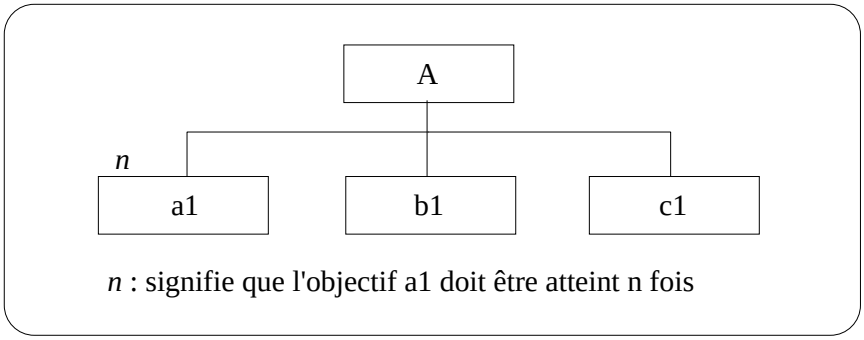
L'utilisateur peut atteindre ou non l'objectif a1.

la condition:



La condition porte sur un élément de l'environnement.

l'itération :



Le n peut être fixé comme inférieur à un entier donné

UN EXEMPLE

Nous montrons ici un exemple simple de scénario adapté à un problème de réparation d'un pneu sur un véhicule.

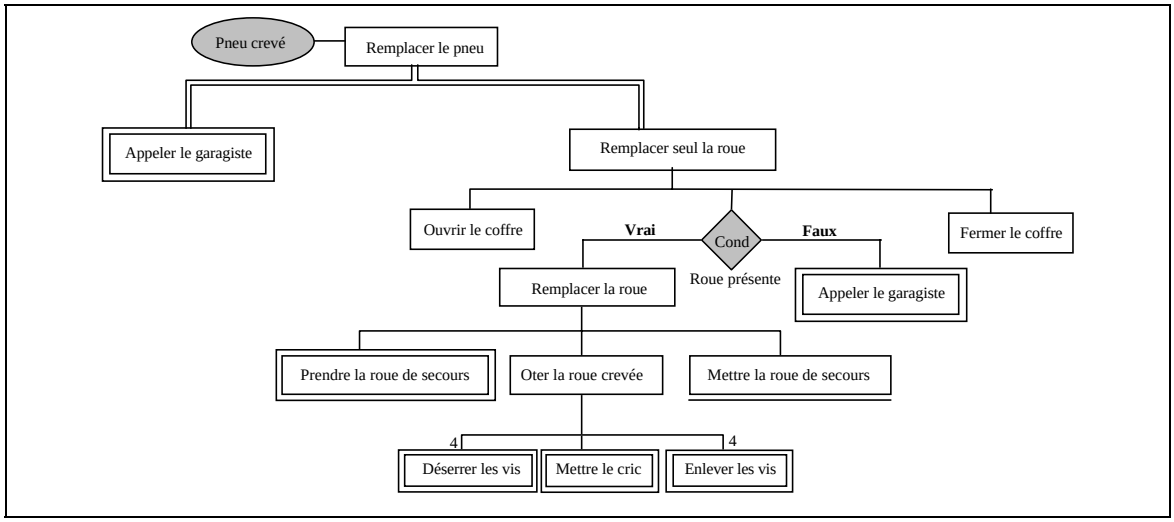
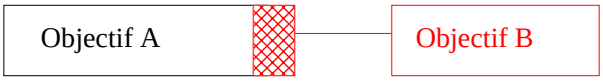


Figure 8-6 : Exemple de scénario de réparation de pneu crevé

8.5.2 L'expression des comportements erronés

L'expression des comportements erronés vient en surcharge de celle des comportements attendus. Un comportement erroné consiste en une dérivation du comportement attendu.



L'objectif A doit être atteint et l'objectif B ne doit pas être atteint

Dans le scénario précédent, un comportement erroné correspond, par exemple, au fait de monter dans la voiture pendant que celle-ci est sur cric. On exprimera alors cette contrainte de la façon suivante :

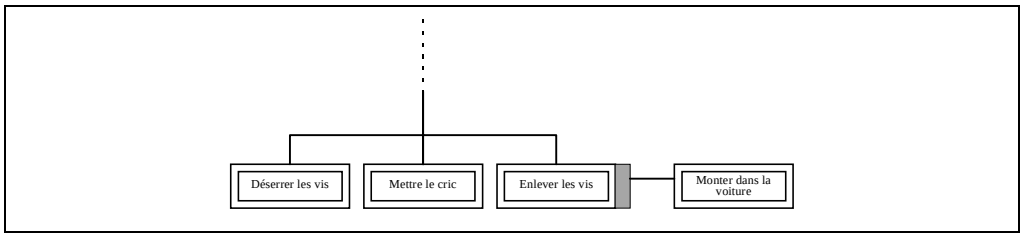


Figure 8-7 : Un exemple de comportement erroné

8.5.3 L'expression de la réactivité

L'expression de la réactivité vient en surcharge sur l'arbre de comportement. Chaque objectif est enrichi d'une description de la réaction du système au moment où l'on cherche à atteindre l'objectif, en cas de réussite ou en cas d'échec. Cette réaction peut être plus ou moins complexe :

- *historisation*. Le concepteur indique si le système doit enregistrer le succès ou l'échec de l'apprenant face à l'objectif.
- *message*. Le concepteur indique quel est le message que le système doit délivrer à l'apprenant, dans trois éventualités : (1) au moment ou le système commence à attendre un certain comportement de la part de l'apprenant, (2) en cas de succès ou (3) en cas d'échec.
- *action*. De la même façon, le concepteur peut indiquer l'action que le système doit réaliser, dans chacune des trois éventualités précédemment citées. Il peut s'agir de proposer une démonstration, de réaliser une opération à la place de l'apprenant, etc.

Par exemple, on peut associer la réactivité suivante à l'objectif "*Ouvrir le coffre*" du scénario précédent :

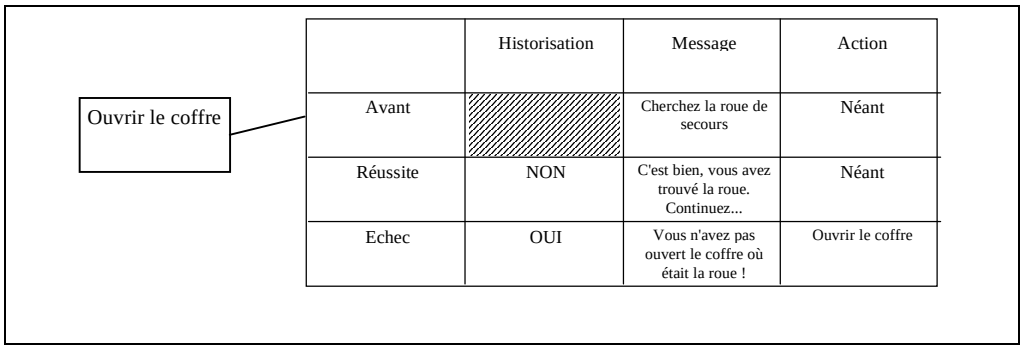


Figure 8-8 : Un exemple de réactivité

8.6 LA MISE EN OEUVRE DES CONCEPTS A TRAVERS L'INTERFACE

Comme dans chacun des espaces de travail, nous voulons proposer :

- des éditeurs spécialisés permettant de créer et modifier des scénarios. Ces éditeurs doivent permettre :

  ⇒ la manipulation directe de l'arbre représentant le scénario, grâce à des palettes d'outils permettant l'ajout, la suppression des noeuds, le déplacement de sous-arbres, etc...

  ⇒ l'enrichissement de la structure créée par les informations relatives à la réactivité

- un outil permettant de valider les scénarios. N'oublions pas que nous nous sommes intéressés à définir une description *abstraite* du scénario. C'est-à-dire que les objectifs sont décrits de façon abstraite à l'aide d'une expression textuelle. Le scénario n'est donc pas encore, à ce stade, lié au modèle de simulation défini dans un autre espace de travail. Il est donc possible d'envisager une validation partielle et locale de ce scénario abstrait. Cette validation concerne deux aspects et peut se réaliser en deux étapes :

  ⇒ la validation des relations structurelles liant les objectifs. De la même façon que nous avons prévu des outils pour la vérification de la dynamique dans le modèle par une animation graphique des Statecharts, nous pouvons prévoir de tester l'exécution des scénarios abstraits par l'animation temporisée des arbres de scénario. La séquence d'événements, précisée de façon explicite par le concepteur ou bien générée de façon automatique, correspond alors à une suite :

  Séquence_Evénements = { (t_0,Obj_0) , (t_1,Obj_1) , ... }
  avec t_i = date et Obj_i = Objectif

  ⇒ La validation de la réactivité associée aux objectifs. Une fois validée la structure de l'arbre, les mêmes séquences d'événements peuvent être testées, en

s'intéressant cette fois-ci au point de vue de l'utilisateur final. S'il réalise une certaine séquence spécifique, quelle sera la perception qu'il aura de la réactivité du système ? Cet aspect nous semble très intéressant, car il permet de s'assurer beaucoup plus tôt de l'acceptabilité de l'application finale.

- des mécanismes de sauvegarde et de réutilisation des scénarios. De la même façon que pour les modèles de simulation, les scénarios pédagogiques doivent pouvoir être mis en catalogue afin de les réutiliser ou de les modifier ultérieurement.

8.7 UNE SOLUTION INTERMEDIAIRE PLUS OPERATIONNELLE

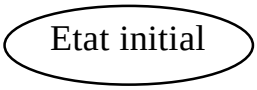
L'approche de formalisation que nous avons proposée ci-dessus est issue des modèles de la tâche proposés en IHM. Nous avons soumis le principe de cette approche à des "auteurs pédagogues" issus du contexte Hewlett-Packard, et il en ressort les impressions suivantes :

- Premièrement, l'écriture d'un scénario de contrôle pédagogique peut rapidement s'avérer difficile. On retrouve ici une complexité de niveau "écriture de programme", avec toutes les conséquences que cela entraîne : vérification de la syntaxe, détection des ambiguïtés, non prédictibilité des résultats à l'exécution, etc. Fournir des outils basés sur un tel formalisme à des non-informaticiens peut s'avérer un choix délicat.
- Nous pouvons ensuite nous interroger sur la nécessité de proposer un tel niveau de complexité en regard des scénarios réels proposés par les auteurs.

Ces constats nous ont amenés à proposer plus concrètement une seconde approche, où le niveau de complexité parait plus adapté aux besoins des concepteurs et qui semble moins ambitieuse dans la perspective de l'automatisation.

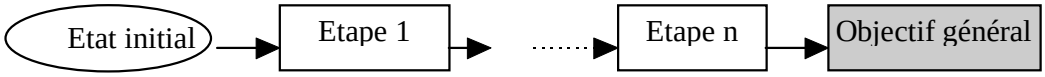
Cette seconde proposition est basée sur la métaphore de la classe de chimie que nous avons décrite plus haut. Un scénario de contrôle est alors caractérisé par :

- un **état initial** de la simulation dans lequel doit se trouver l'élève au début de l'exercice

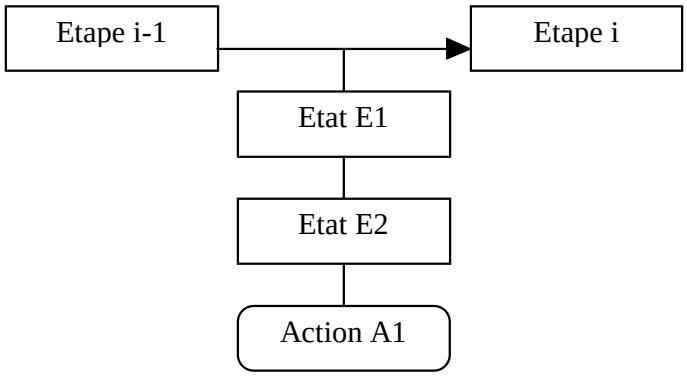


- un **objectif général** fixant le but à atteindre par l'élève. Il correspond à un état du système.

- Un ensemble **d'étapes intermédiaires** correspondant également à des états du système. Un scénario est donc constitué par un enchaînement séquentiel d'étapes intermédiaires à franchir avant d'atteindre l'objectif général.



On propose au concepteur la possibilité d'associer un certain nombre **d'états** du système à vérifier ou **d'actions** à contrôler, avant le passage à l'étape suivante :



Concernant l'exemple de la réparation d'une roue, on peut, par exemple, prévoir le scénario indiqué sur la figure 8.9.

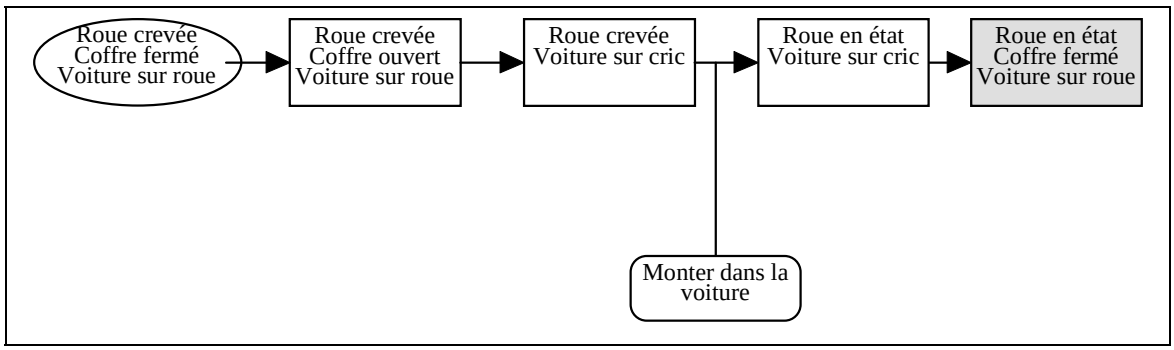


Figure 8-9 : Un nouvel exemple de scénario

Cette proposition, qui allie les avantages d'une plus grande simplicité et de l'aptitude à l'automatisation, est actuellement soumise à la validation des utilisateurs à travers le prototype de l'environnement MELISA. L'objectif de cette validation est de déterminer si ce formalisme permet d'exprimer la majorité des besoins exprimés. En fonction des résultats obtenus, il pourra être enrichi de fonctionnalités supplémentaires, décrites dans la première solution.

8.8 RESUME

Dans ce chapitre, nous avons abordé l'espace de travail Scénario.

Dans un premier temps, nous avons donné plusieurs définitions du terme "Scénario" avant de proposer les nôtres :

- Scénario d'enchaînement pédagogique : description abstraite de l'enchaînement des unités d'apprentissage au sein d'un logiciel pédagogique.
- Scénario de contrôle pédagogique de simulation : description abstraite du contrôle pédagogique d'une simulation en termes d'états intermédiaires ou d'actions à analyser lors de l'exécution de cette simulation, puis de la réactivité associée.

Ensuite, nous avons dégagé un ensemble de concepts liés à la définition du scénario, et examiné comment ils pouvaient être formalisés. Sur la base des formalismes d'analyse de la tâche proposés dans le domaine de l'IHM, nous avons établi une première proposition décrivant un scénario comme un arbre d'enchaînement d'objectifs liés par des relations structurelles et temporelles. Une telle solution parait riche mais difficile à mettre en œuvre par des concepteurs non-informaticiens.

Nous avons donc fait une seconde proposition, qui allie les avantages d'une plus grande simplicité et d'une meilleure l'aptitude à l'automatisation. Cette dernière solution pourra être progressivement enrichie de certaines fonctionnalités proposées dans la première solution, sur la base d'expérimentations en cours actuellement.

9. LA SPECIFICATION DE LA REPRESENTATION

Nous nous intéressons dans ce chapitre à l'espace de conception "Représentation", dont le but est définir l'interface entre l'utilisateur final (l'apprenant) et la simulation pédagogique.

Nous allons en premier lieu présenter une classification des systèmes de gestion d'interface afin de situer l'approche utilisée dans MARS. Nous montrerons ensuite comment réutiliser des objets préexistants et divers moyens de créer de nouveaux objets.

9.1 LES SYSTEMES DE GESTION D'INTERFACES

Les systèmes informatiques qui se préoccupent de la gestion des interfaces utilisateurs souhaitent prendre en charge de façon automatisée certaines tâches jusque là réservées aux informaticiens afin de les rendre accessibles à des non-programmeurs. Cette volonté s'explique par le fait que la spécification d'une interface peut être utilisée pour aider à l'expression des besoins d'un client, et qu'elle constitue un excellent moyen de réaliser une maquette d'un produit à créer.

D. Hix [HIX 90] propose de classer les UIMS[1] en quatre générations. Pour elle, ce découpage n'est pas un strict découpage chronologique, mais correspond davantage à un niveau de fonctionnalités et d'accessibilité. On peut donc distinguer :

[1] *UIMS : User-Interface Management Systems = Systèmes de gestion d'interface utilisateur*

- *la première génération* : cette première classe de systèmes, apparue au début des années 1980, avait pour objectif de rendre plus aisée la conception statique d'interface et la gestion des affichages dans une démarche de prototypage. Ces environnements sont souvent spécifiés à l'aide de grammaires BNF (Backus-Naur Form) et de langages traditionnels. Ils présentent des fonctionnalités limitées et, par nature, ne sont accessibles qu'aux programmeurs. On citera comme exemple de cette génération un système comme Act/1 [MAS 81].

- *La deuxième génération* : les UIMS de la seconde génération ont davantage porté leur attention sur les problèmes d'exécution de l'interface. On a constaté une évolution depuis les grammaires BNF vers des diagrammes états/transitions pour représenter l'interface et séparer de façon formelle l'interface du reste de l'application. De façon générale, les fonctionnalités restent limitées et l'accès est le plus souvent réservé aux seuls programmeurs. Nous pouvons citer comme représentants de cette génération :

  ⇒ Le système Rapid/USE de Wasserman [WAS 85]. Cette méthode préconise une conception séparée de l'interface du système avec une participation effective de l'utilisateur dès les premières étapes du développement. Orientée vers le prototypage rapide d'interface, elle permet la spécification du système à l'aide de diagrammes états-transitions, puis l'exécution de ces diagrammes.

  ⇒ Le système COUSIN de Hayes [HAY 85] permet de créer une interface basée sur des formulaires. Il permet également d'établir des associations entre les valeurs manipulables ou affichables dans le formulaire et des valeurs de l'application. Les interfaces générées restent limitées à un style "formulaire".

- *La troisième génération* : le principal effort de cette classe d'environnements a porté sur la migration de la programmation textuelle vers des outils interactifs, proposant souvent une manipulation directe. Elle permet de spécifier des dialogues utilisateur complexes avec gestion du multifenêtrage, prise en compte de la souris, etc. Souvent basés sur une approche par objets, ces systèmes proposent des bibliothèques d'objets d'interaction réutilisables (les *widgets*). Ils permettent donc une augmentation sensible des fonctionnalités et de la flexibilité, mais exigent encore souvent une mise en œuvre complexe. On peut citer comme représentants de cette classe :

  ⇒ GWUIMS [SIB 88] qui représente une tentative de produire une architecture générique d'UIMS. Basé sur l'approche par objets, ce système permet de créer des interfaces par manipulation directe d'objets d'interfaces à partir de bibliothèques. Cette démarche facilite grandement le prototypage rapide d'une grande diversité de dialogues.

⇒ PERIDOT de Myers [MYE 87] qui utilise un mécanisme de programmation par démonstration pour développer interactivement l'interface. Le développeur indique comment les dispositifs d'entrée doivent être mis en œuvre en montrant des exemples d'utilisation. Le système, par un processus d'inférence, déduit de façon automatique le code à partir des valeurs des paramètres utilisées et des actions effectuées.

- *La quatrième génération* : elle correspond à la tendance actuelle et s'intéresse en particulier à résoudre les difficultés de spécification d'interface dynamique. Elle s'occupe davantage de l'intégration de l'interface au sein de l'application finale. L'objectif est de faire accepter les UIMS comme un composant à part entière des ateliers de Génie logiciel.

  ⇒ Un des meilleurs exemples de cette classe est *Interface Builder* de Next Step [NEX 91]. Il permet d'une part, la création d'une interface par manipulation directe des objets souhaités, et d'autre part, la connexion des objets créés dans le but de produire des interfaces d'applications utilisant la manipulation directe.

Dans le cadre de M.A.R.S. nous nous situons naturellement dans la dernière approche. Nous allons maintenant examiner comment ce type d'approche peut être mis en œuvre.

9.2 REUTILISATION D'OBJETS DE REPRESENTATION

9.2.1 Le concept d'objet de représentation

La notion d'objet d'interaction (ou widget) est un concept central de l'approche des UIMS les plus récents. Les outils de spécification interactive d'interfaces ont pour objectif la construction de prototypes d'interface sans programmation et doivent permettre au concepteur [COU 90] :

- de créer des "*tableaux de bord*" par assemblage d'objets de présentation déjà définis.
- de décrire les enchaînements entre les "*tableaux de bord*".
- d'associer les éléments de l'interface aux concepts de l'application.

Cet ensemble de tâches nous amène à réfléchir aux relations qui doivent exister, d'une part entre les objets d'interactions eux-mêmes, et d'autre part entre ces objets et le noyau fonctionnel de l'application. Nous allons maintenant examiner cet aspect à travers le modèle pipeline.

9.2.2 Niveaux de retour d'information

Adapté par L. Nigay [NIG 94], le modèle *pipeline* a pour but de rendre explicites, pour les systèmes interactifs, les traitements effectués par l'utilisateur et par le système. Il propose trois niveaux : les actions physiques, les unités informationnelles (entités conceptuelles manipulables à travers l'interface) et les effets (modifications de l'état interne du système). Il distingue, en particulier quatre catégories de retours d'information : sensori-moteurs, niveau syntaxique, niveau unité informationnelle et niveau fonctionnel.

Nous reprenons cette classification pour l'adapter au problème des objets d'interaction. Nous ne nous intéressons pas aujourd'hui dans notre contexte aux aspects sensori-moteurs liés aux dispositifs physiques (réactivité d'une souris, bruit d'un clavier, etc.). Nous pouvons donc distinguer :

- les retours d'information de niveau syntaxique. Il s'agit par exemple d'un bouton poussoir à deux états visibles. Tant que le bouton de la souris est appuyé, l'objet graphique bouton est lui aussi enfoncé. Dans ce cas, le retour d'information sert à guider l'utilisateur dans sa tâche syntaxique élémentaire avant une éventuelle validation : *est-ce que je suis en train d'appuyer sur le bouton* ?

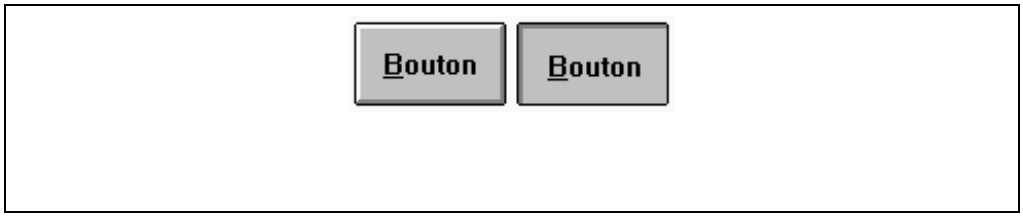


Figure 9-1 : Les deux aspects d'un bouton : non enfoncé et enfoncé

- les retours d'information de niveau informationnel. Il s'agit par exemple d'un interrupteur à deux états. Dans ce cas, le retour d'information sert à guider l'utilisateur dans sa compréhension du fonctionnement de l'objet d'interaction : *suis-je en train d'allumer ou d'éteindre cet interrupteur* ?

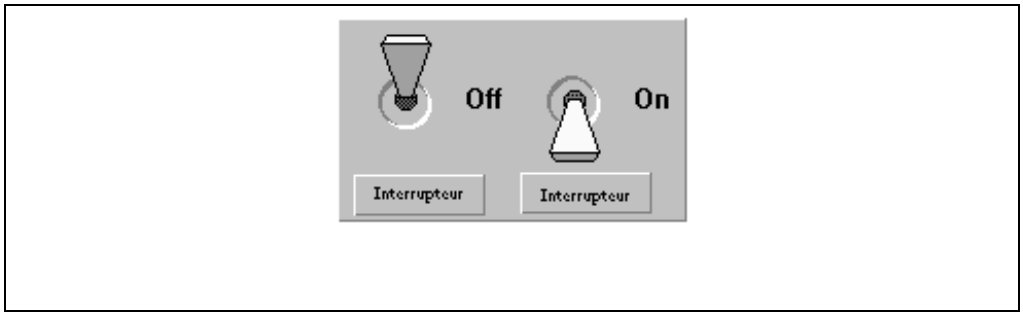


Figure 9-2 : Les deux aspects d'un même interrupteur

- *les retours d'information de niveau fonctionnel*. Il s'agit par exemple d'un système composé d'un interrupteur et d'un dispositif d'affichage. Dans ce cas, le retour d'information sert à aider l'utilisateur dans sa compréhension du fonctionnement du système modélisé : *quels sont les effets perceptibles sur le système quand j'actionne l'interrupteur* ?

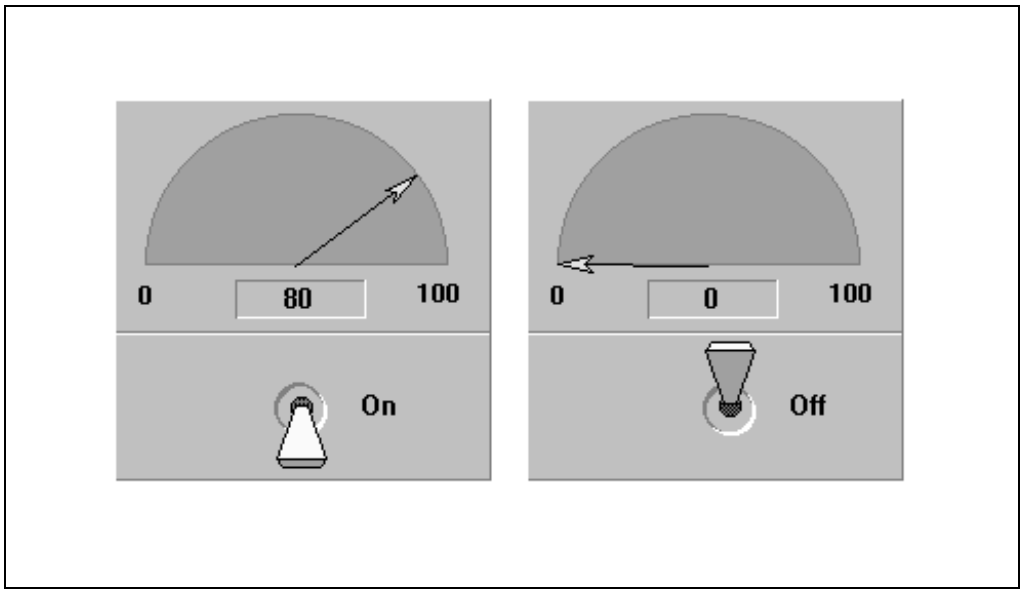


Figure 9-3 : Commande d'un système par interrupteur

Les trois modes de retour d'information que nous venons de décrire peuvent être représentés sous la forme d'un pipeline, comme le montre la figure 9-4.

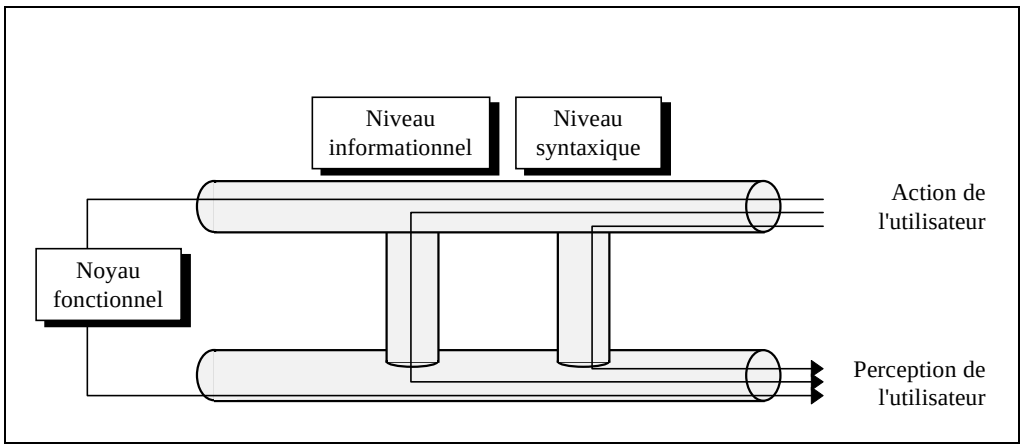


Figure 9-4 : Le modèle pipeline appliqué aux objets d'interaction

Sur cette figure, on constate qu'une même action de l'utilisateur sur un objet peut donner lieu à des retours d'information de nature différente, selon qu'elle est interprétée au niveau syntaxique, au niveau informationnel ou au niveau fonctionnel.

9.2.3 Typologie des objets de représentation

Chaque objet de représentation peut présenter simultanément un ou plusieurs de ces trois modes. Nous proposons donc une première classification des objets d'interaction en fonction des modes qu'ils proposent :

- les *objets non interactifs* sur lesquels l'utilisateur ne peut pas agir et dont l'état perceptible ne se modifie jamais. Rentrent dans cette catégorie, par exemple, les images fixes, les zones de texte expositif, les graphismes, ...
- les *objets déclencheurs* qui peuvent éventuellement offrir des retours d'information de niveau syntaxique ou informationnel, et qui communiquent avec le noyau fonctionnel pour lui fournir des informations en entrée relatives aux actions de l'utilisateur. Rentrent par exemple dans cette catégorie, les boutons, les champs de saisie, les objets permettant la manipulation directe, etc.
- les *objets percepteurs* sur lesquels l'utilisateur ne peut pas agir, et que le noyau fonctionnel utilise pour fournir des informations en sortie vers l'utilisateur. Rentrent par exemple dans cette catégorie, les champs d'affichage textuel ou numérique, les objets plus sophistiqués de visualisation (vumètres, ..) ou des objets multimédias tels que des zones permettant l'affichage dynamique d'une vidéo ou d'une animation graphique.
- les *objets Déclencheurs/Percepteurs* qui cumulent les deux rôles précédents.

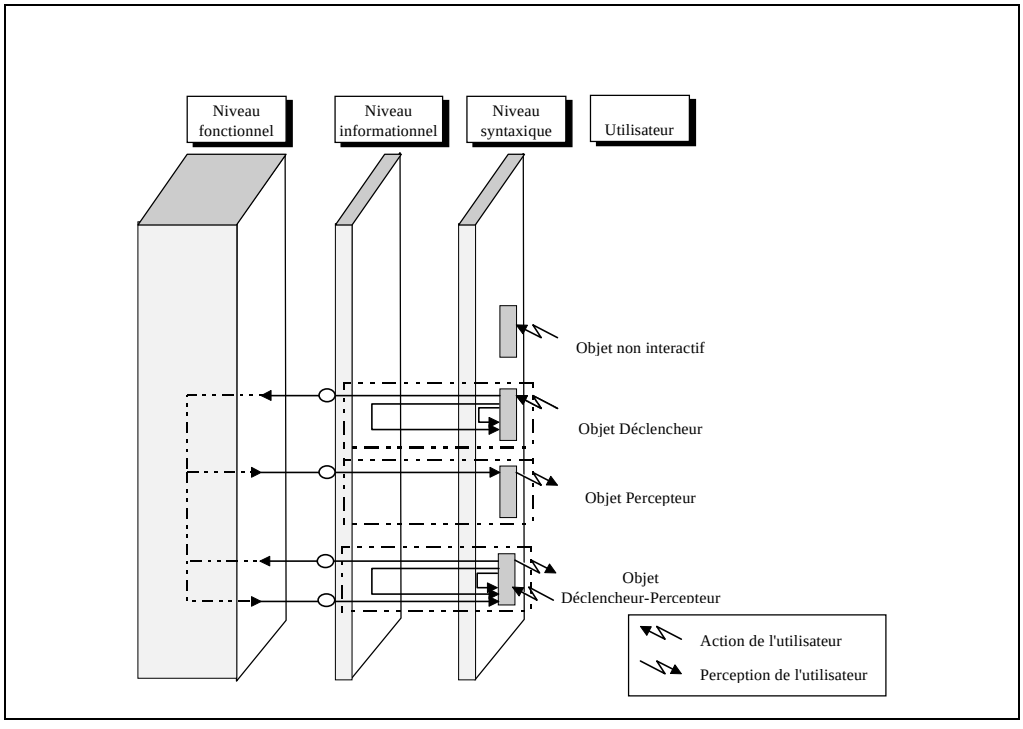


Figure 9-5 : Objets de représentation et niveaux

Cette classification permet de définir plus précisément ce que nous entendons par objet d'interaction :

> Un objet d'interaction est une unité informationnelle autonome caractérisée:
> - d'une part, par un comportement réactif aux actions de l'utilisateur
> - d'autre part, par sa capacité à fournir ou à exploiter des informations au sein de l'environnement dans lequel il est intégré

Nous choisissons de classer les objets de représentation de la façon suivante :

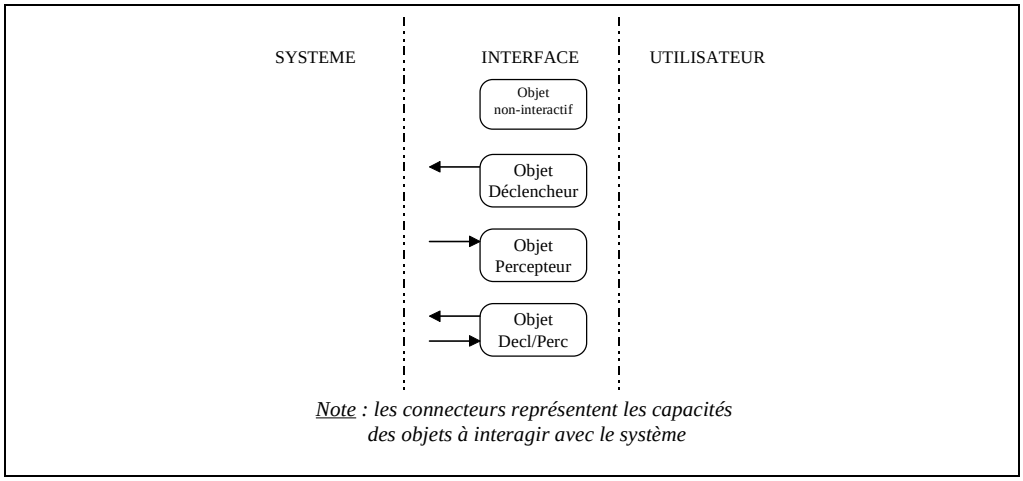


Figure 9-6 : Une classification des objets de représentation

9.2.4 Représentation du modèle et représentation du scénario

Les objets déclencheurs, percepteurs ou déclencheur/percepteurs peuvent être encore caractérisés en fonction du composant logique du noyau fonctionnel avec lequel ils sont appelés à communiquer.

De façon classique, dans les modèles de communication homme-machine, on opère une séparation formelle entre la partie abstraite de l'application (le noyau fonctionnel, ou modèle, ou abstraction selon les cas) et sa partie manipulable par l'utilisateur (l'interface, ou présentation ou représentation selon les cas). Dans notre modèle M.A.R.S, nous proposons dès l'étape initiale d'analyse du problème d'établir une distinction très nette entre deux parties indépendantes du noyau fonctionnel que nous appelons "Modèle" pour la partie qui concerne l'expertise du domaine, et "Scénario" pour ce qui concerne l'exploitation pédagogique du modèle. Les objets de représentation, qu'ils soient déclencheurs ou percepteurs, communiquent avec l'un des deux composants. Nous pouvons donc distinguer :

- Les objets *représentants du modèle*. Il s'agit par exemple des boutons, curseurs, graphismes, etc. , permettant de manipuler le modèle de simulation
- Les objets *représentants du scénario.* Il s'agit par exemple d'un bouton déclenchant le démarrage de l'exercice, ou d'un message textuel ou sonore de nature pédagogique.

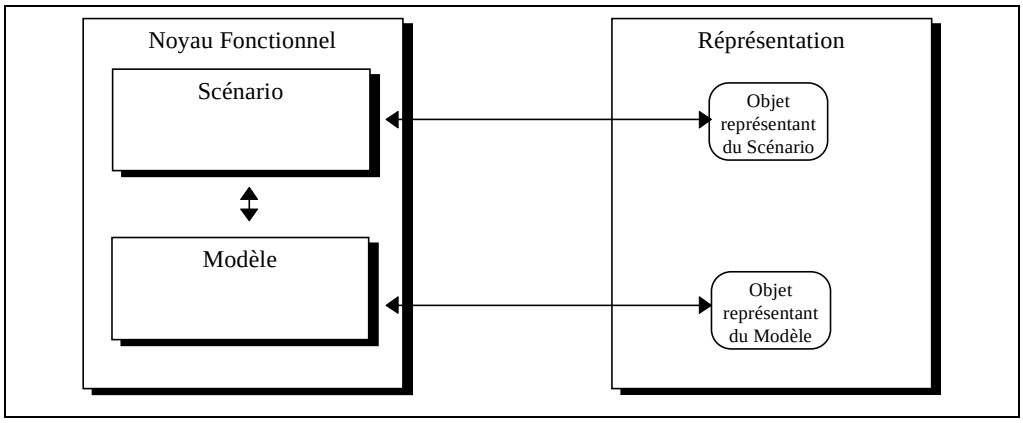


Figure 9-7 : Représentants du Modèle et du Scénario

La coexistence au sein d'une même interface d'objets au comportement local identique mais réservés à des finalités différentes pose de nombreux problèmes. Par exemple, deux boutons de même apparence peuvent déclencher soit une action vers le modèle, soit une action vers le scénario. De la même façon, deux messages d'allure identique pourront être délivrés par l'un ou l'autre des deux composants. Ceci peut entraîner pour l'utilisateur certaines erreurs de manipulation ou ambiguïtés d'interprétation. Pour régler ces problèmes, on peut envisager plusieurs solutions:

- *La spécialisation des objets*. Dans ce cas, le concepteur utilise des objets au comportement local identique mais à l'apparence différente. Le respect de certaines règles de présentation peut ainsi fortement réduire les risques d'ambiguïté.
- *La répartition topographique des objets*. Cette autre solution consiste pour l'auteur à appliquer certaines règles de placement (régions de l'écran, fenêtres différentes) permettant d'identifier de façon claire la fonction des objets [SEF 93a].
- *La multimodalité*. Une troisième solution consiste à réserver certaines modalités à l'une ou l'autre des communications. Par exemple, on peut imaginer que les messages délivrés par le modèle sont fournis de façon textuelle, alors que ceux délivrés par le scénario le sont de façon sonore [SEF 93b].

Ces solutions peuvent relever d'un choix explicite de l'auteur. Nous proposons quant à nous qu'elles puissent être prises en charge de façon automatisée par l'environnement de conception d'interfaces.

9.2.5 Les bibliothèques d'objets.

Nous avons vu que l'une des caractéristiques essentielles des UIMS récents était de proposer des bibliothèques d'objets d'interaction réutilisables. Nous voulons donc reprendre cette approche dans l'espace de conception "Représentation" en proposant au concepteur des mécanismes basés sur la manipulation directe lui permettant de :

- disposer de repères explicites permettant d'identifier rapidement la fonction des objets à importer (non interactif, déclencheur, percepteur ou déclencheur-percepteur).
- tester le comportement d'un objet de représentation avant d'en décider l'importation (1).
- importer dans son interface les objets qu'il a choisis (2).
- modifier interactivement certains paramètres des objets après leur importation (3). En particulier, il doit être possible de modifier certains aspects graphiques de ces objets sans en modifier le comportement.
- définir de nouveaux objets d'interaction (4)
- mettre en catalogue les nouveaux objets d'interaction créés (5). Cette mise en catalogue a pour conséquence un enrichissement progressif des bibliothèques d'objets interactifs au sein d'un même domaine ou l'élargissement de ces bibliothèques à de nouveaux domaines.

Ces propositions sont résumées dans la figure 9-8.

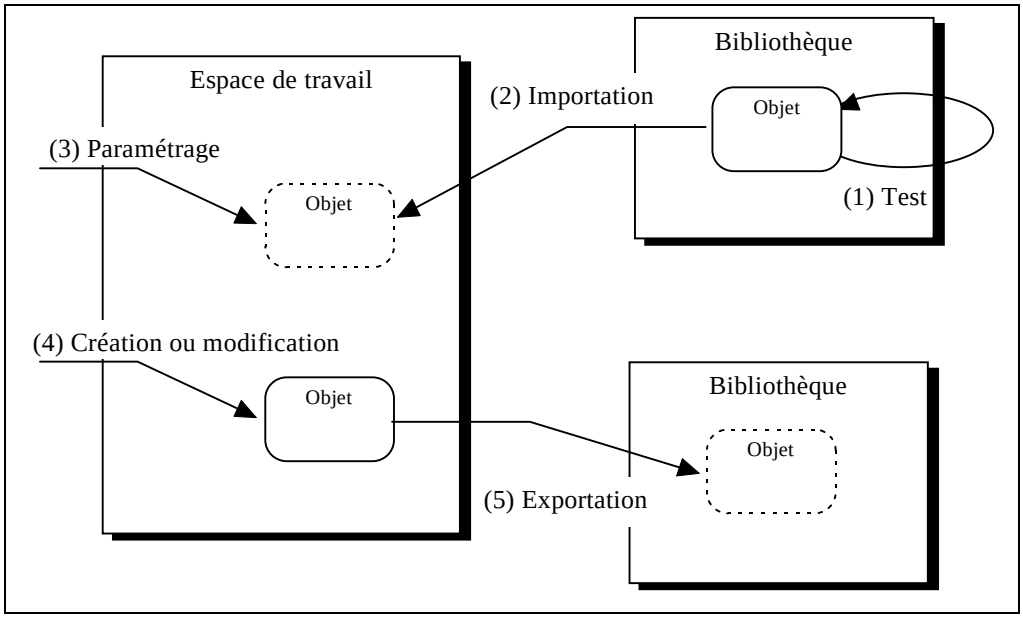


Figure 9-8 : Importation et Exportation d'objets interactifs

9.3 CONCEPTION DE NOUVEAUX OBJETS DE REPRESENTATION

Nous avons déjà défini la notion d'objet, lorsque nous avons étudié l'espace de conception du modèle de simulation. Nous avons vu que la partie abstraite d'un objet était caractérisée par :

1. *un ensemble d'attributs définissant sa structure.*
2. *un ensemble d'opérations définissant son comportement.*
3. *un Statechart de Harel définissant son évolution temporelle.*

Les objets d'interaction sont, quant à eux des objets concrets et manipulables par l'utilisateur. On doit donc leur ajouter une quatrième caractéristique :

4. *une représentation permettant à l'utilisateur de le manipuler ou d'en visualiser l'état interne.*

9.3.1 Modèles PAC et MVC

Nous retrouvons ici les principes énoncés dans les modèles de conception ou d'architecture d'interface tels que PAC [COU 90] ou MVC [GOL 84]. Un objet interactif se caractérise ainsi par :

- Une image qui définit un comportement perceptible pour l'utilisateur. Pour PAC, il s'agit de *la présentation* alors que MVC répartit le comportement en deux notions distinctes : le comportement en entrée (*Contrôleur*) et le comportement en sortie (*Vue*).
- Des fonctions ou des attributs fonctionnels (c'est le côté abstrait visible des autres constituants logiciels). PAC parle *d'Abstraction* alors que MVC utilise le terme de *Modèle*.
- La gestion des liens entre côtés abstraits et présentation : c'est le *Contrôle* de PAC. Dans MVC, le contrôle est inexistant : ses fonctions sont diluées entre le modèle, la vue et le contrôleur.

La figure 9-9 indique les différences entre les deux modèles.

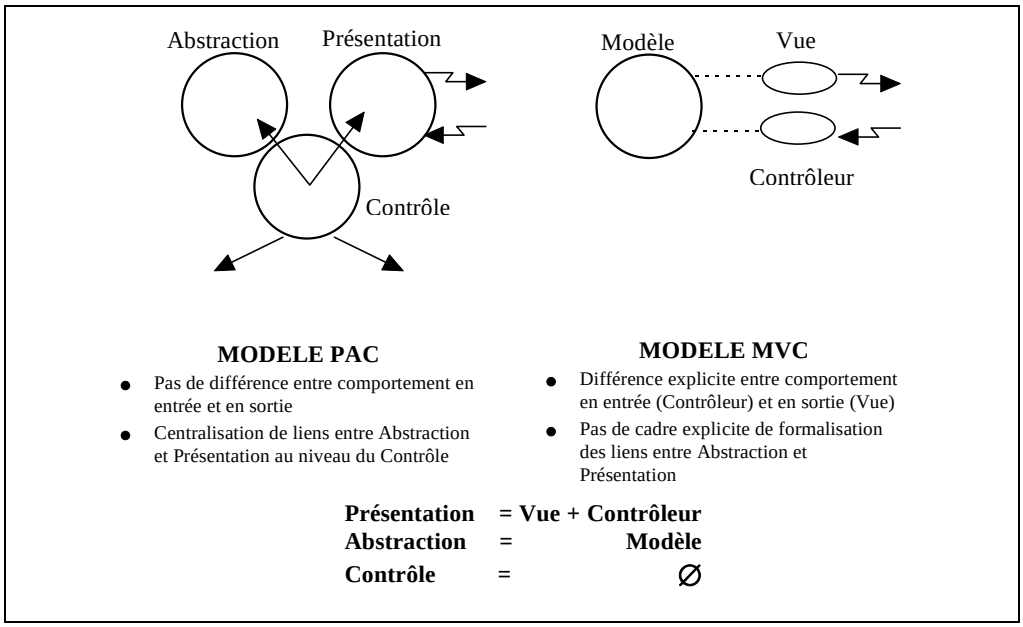


Figure 9-9 : Modèle PAC et Modèle MVC

Notre démarche personnelle nous pousse à nous rapprocher du modèle MVC pour les raisons suivantes :

- Nous nous situons globalement dans une approche par objets. Rappelons que MVC est très lié aux techniques de programmation par objets puisque faisant partie intégrante du langage Smalltalk. Par contre, PAC qui se définit comme un cadre conceptuel, se veut libre des contraintes pratiques de l'outil de réalisation.
- Nous avons défini plus haut la nécessité de répartir clairement les traitements en entrée (les déclencheurs) et les traitements en sortie (les percepteurs) pour permettre une intégration ultérieure plus aisée des objets d'interactions.
- Notre cadre général, le modèle MARS s'intéresse à pallier les manques de MVC en tentant de formaliser les liens directs qui peuvent être effectués entre le modèle et la représentation.

9.3.2 Un objet construit comme une application

Nous proposons donc d'étendre la démarche générale que nous avons adoptée pour MARS à la conception des objets d'interaction eux-mêmes. En utilisant notre terminologie, un objet d'interaction peut être vu comme une petite application de simulation pédagogique où :

- Le Modèle correspond au fonctionnement abstrait de l'objet.
- La Représentation représente le comportement perceptible pour l'utilisateur. On distingue le comportement en entrée et le comportement en sortie.

- Les Associations représentent les liens entre :

  ⇒ le comportement en entrée et le Modèle.

  ⇒ le Modèle et le comportement en sortie.

- Le Scénario est inexistant.

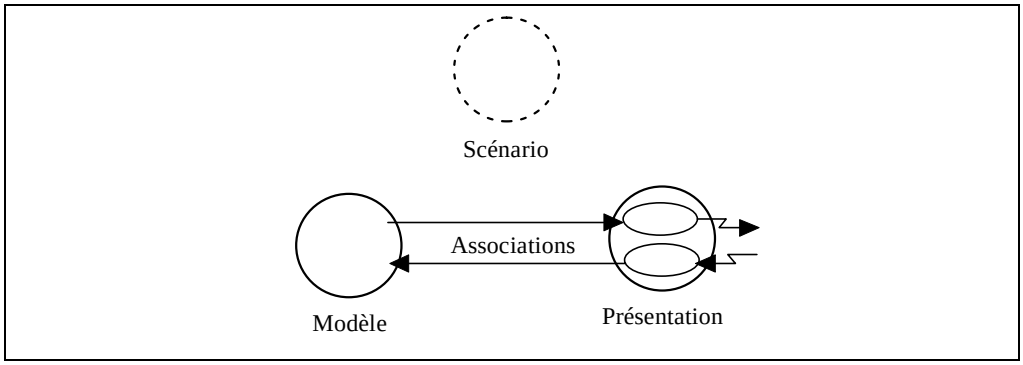


Figure 9-10 : Un objet d'interaction vu en tant qu'objet MARS

Le fait d'assimiler un objet de présentation à une application présente deux intérêts. Tout d'abord, il nous permet de proposer des interfaces identiques de spécification aux différents concepteurs (spécification de la représentation et des associations), et de ce fait, de renforcer l'homogénéité de l'environnement. Ensuite, cette possibilité nous permet d'introduire une définition récursive d'une application MARS, définition basée sur l'existence d'objets d'interaction élémentaires à partir desquels peuvent être produits tous les autres.

9.3.3 Un objet construit par réutilisation d'une application

Ce point de vue récursif nous permet, en particulier, d'envisager la transformation automatique d'une application de simulation pédagogique en une entité autonome de type objet interactif complexe pouvant être réutilisée ultérieurement dans une autre application.

Prenons, par exemple, une application de simulation pédagogique ayant pour sujet l'apprentissage du fonctionnement d'un ordinateur. Cette application possède un modèle, une représentation et un scénario de contrôle pédagogique. Imaginons maintenant que nous voulions développer une autre application pédagogique concernant la communication par réseau entre deux ordinateurs. Il pourrait être intéressant dans ce cas de considérer chaque ordinateur comme un objet d'interaction sophistiqué, et ne spécifier dans le modèle que les aspects propres à la communication, comme le montre la figure 9-11.

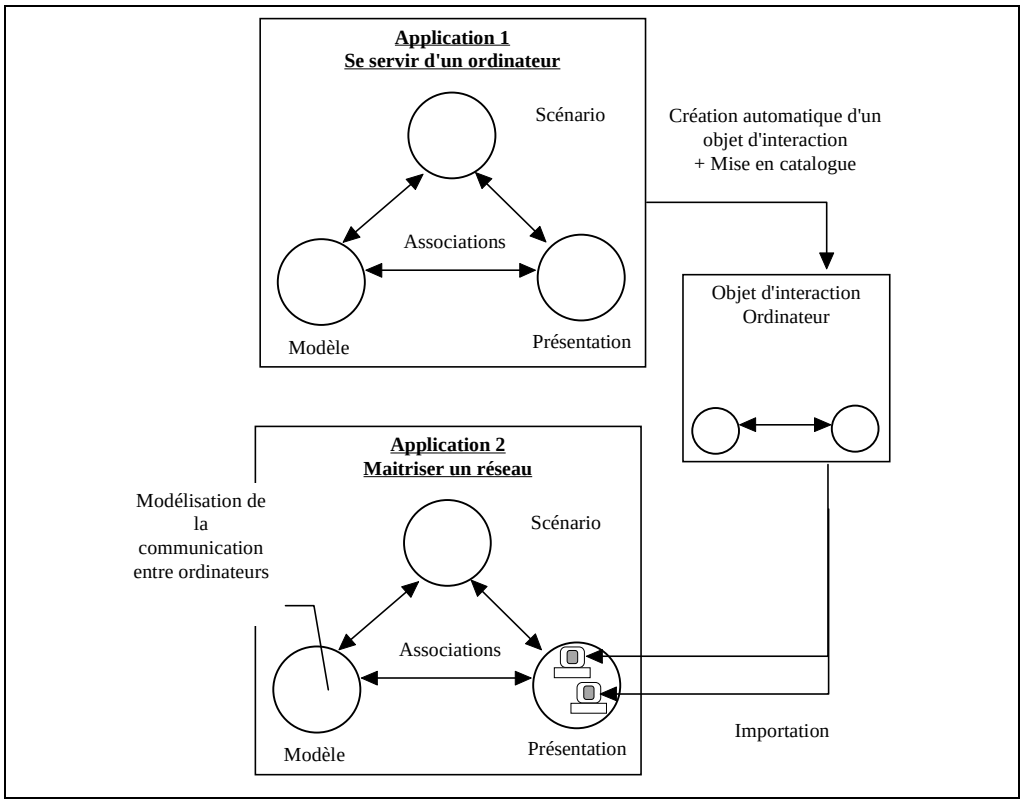


Figure 9-11 : La réutilisation d'une application en tant qu'objet d'interaction

9.3.4 Un objet construit par connexion d'objets d'interaction

La connexion des objets d'interaction sera examinée en détail lors du chapitre suivant consacré aux associations. Nous pouvons tout de même signaler qu'il peut exister deux types de connexions (Cf. Fig. 9-12) :

- connexion entre un objet d'interaction d'une part et le modèle ou le scénario d'autre part. Ce type de connexion représente une coopération entre la présentation et le noyau fonctionnel.
- connexion entre objets d'interaction. La réalisation de ce type de connexion correspond à la définition d'un objet d'interaction complexe, possédant lui aussi ses propres connecteurs.

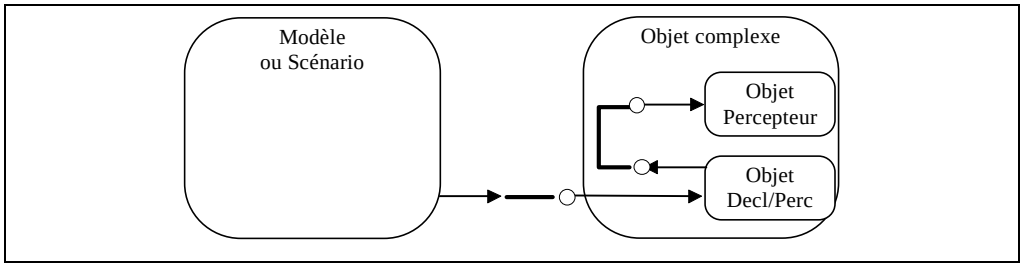


Figure 9-12 : Connexionx inter-objets et avec le noyau fonctionnel

9.4 RESUME

Dans ce chapitre, nous avons abordé l'espace de travail Représentation.

En premier lieu, nous avons établi un panorama des systèmes de gestion d'interfaces qui correspondent à un domaine largement étudié. Nous avons repris de ces systèmes le concept central d'objet d'interaction (widget), et nous avons proposé une classification de ces objets en fonction de leur capacité à fournir ou exploiter des informations au sein de l'environnement dans lequel ils sont insérés. Nous distinguons ainsi : les objets non-interactifs, les objets déclencheurs, les objets percepteurs et les objets déclencheurs-percepteurs.

Nous avons également établi une seconde classification des objets en fonction du composant logique du noyau fonctionnel (Modèle ou Scénario) avec lequel ils sont appelés à coopérer. Sur cette base, nous proposons quelques règles permettant de gérer la coexistence de ces objets au sein du même environnement.

Ensuite, nous avons défini les mécanismes d'importation et d'exportation des objets d'interaction, puis nous nous sommes attachés à en décrire les différents modes de construction. Basée sur le modèle MVC, notre approche consiste à présenter un objet d'interaction comme une petite application MARS, dont le composant Scénario serait absent. Cette proposition permet d'introduire une définition récursive d'une application MARS et d'offrir de puissants mécanismes de réutilisation.

10. LA SPECIFICATION DES ASSOCIATIONS DE COOPERATION

Le dernier espace de travail que nous devons examiner est celui réservé à l'établissement des associations entre les trois espaces précédemment définis. Cet espace de travail doit permettre l'intégration des résultats partiels obtenus lors de la conception du modèle de simulation, du scénario pédagogique et de la représentation perceptible par l'utilisateur.

10.1 DEFINITION DU CONCEPT DE COOPERATION

Notre objectif est de fournir des formalismes opérationnels permettant d'exprimer la *coopération* entre ces différents espaces de travail. La figure 10-1 montre que cette coopération peut être examinée à plusieurs niveaux d'abstraction :

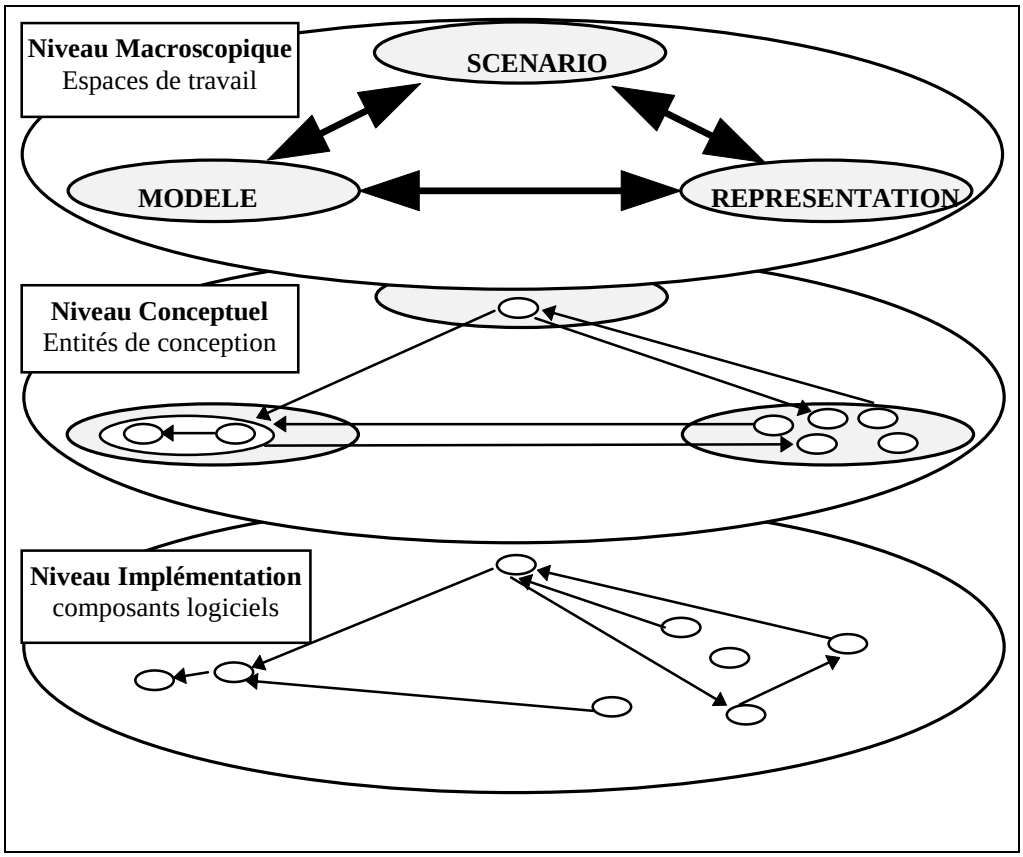


Figure 10-1 :Coopération et niveaux d'abstraction

- le premier niveau, ou *niveau macroscopique,* s'intéresse à caractériser de façon globale la nature de la coopération entre les espaces de travail.
- Le second niveau, ou *niveau conceptuel,* permet à l'intégrateur[2] de décrire de façon précise la coopération entre les diverses entités opérationnelles[3] qui ont été définies lors des étapes de conception précédentes.
- Le troisième niveau, ou *niveau d'implémentation,* décrit en termes informatiques la nature de la coopération entre les divers composants logiciels (échange de messages entre objets, par exemple). Ce niveau dépendra essentiellement des mécanismes de génération utilisés pour traduire la coopération définie au niveau conceptuel en mécanismes informatiques de communication.

---

[2] Nous employons dans ce chapitre le terme d'intégrateur pour désigner le responsable de la conception des associations

[3] Nous utiliserons dans ce chapitre le terme d'entité opérationnelle ou plus simplement entité plutôt que objet ou instance qui nous paraissent être trop liés à l'approche par objets.

10.1.1 Niveau macroscopique : coopération entre espaces de travail

A un niveau d'abstraction élevé, nous pouvons considérer le résultat obtenu dans chaque espace de conception comme un composant indépendant. On peut ainsi distinguer le *modèle* de simulation, le *scénario* pédagogique et la *représentation*. Après leur mise en correspondance, il existe entre ces composants un certain mode de coopération au moment de l'exécution. Pour chaque couple de composants, on peut ainsi préciser la nature de la coopération et le type de résultat obtenu par leur intégration.

- *Modèle et Représentation du Modèle*. Certains objets de représentation permettent à l'apprenant d'activer le modèle, d'autres permettent d'en visualiser l'état. On peut donc dire que la représentation permet à l'apprenant de contrôler le modèle. Une fois intégrés, le modèle et la représentation constituent une simulation exécutable.
- *Scénario et Représentation du Scénario*. De la même façon, certains objets de la représentation permettent l'activation du scénario, d'autres permettent de prendre connaissance des informations pédagogiques. Une fois intégrés, ces deux composants permettent l'accès par l'apprenant au composant pédagogique de la simulation.
- *Modèle et Scénario*. Le scénario permet également de contrôler le modèle, mais cette fois-ci du point de vue de l'exécution pédagogique. Il peut s'agir par exemple de placer le modèle dans l'état initial prévu par le scénario, d'observer les états du modèle successivement atteints. La mise en correspondance du modèle et du scénario permet d'instancier le scénario abstrait dans les termes concrets du modèle. A chaque état ou opération décrit de façon abstraite dans le scénario, on fera correspondre un état concret ou une opération concrète du modèle. Le scénario abstrait devient ainsi un scénario concret, exécutable.

Enfin, l'intégration finale des trois composants permet de définir la notion d'application de simulation pédagogique.

10.1.2 Niveau conceptuel : coopération entre entités

Le second niveau, le niveau conceptuel, intéresse l'intégrateur qui est chargé de concevoir et réaliser les liens entre les entités opérationnelles définies lors des étapes de conception indépendantes. Dans les chapitres précédents, nous avons identifié pour chaque espace de travail, les entités suivantes :

- Dans l'espace de modélisation

  ⇒ les *objets* du système.
  ⇒ le *modèle* résultant de la coopération des objets du système.

- Dans l'espace de scénarisation

  ⇒ les *scénarios*, décrivant des relations structurelles et temporelles liant des états ou des actions abstraits.

- Dans l'espace de représentation

  ⇒ les *objets déclencheurs.*

  ⇒ *les objets percepteurs.*

  ⇒ *les objets hybrides déclencheurs/percepteurs*.

Nous tenterons, à travers un exemple, de dégager une typologie des coopérations existant entre ces diverses entités (Cf. § 10.2 et 10.3).

10.1.3 Niveau implémentation : un système multi-agents

Le troisième niveau, le niveau implémentation, n'est pas accessible par le concepteur. Le processus automatique de génération associé aux différents outils, permet de traduire les entités et coopérations définies par les concepteurs en composants logiciels et en mécanismes de communication internes.

Le choix d'implémentation que nous avons retenu est basé sur la notion de système multi-agents où :

- chaque entité opérationnelle correspond à un agent défini par des propriétés et des méthodes.
- la communication est réalisée par échange de messages entre les agents.

Nous constatons que, si ce troisième niveau n'est pas visible par le concepteur, certaines contraintes liées aux choix d'implémentation remontent inévitablement au niveau supérieur. En particulier, le concepteur devra exprimer dans son espace de travail les mécanismes et interfaces de communication entre entités. L'environnement devra lui proposer le maximum d'assistance dans cette tâche, et lui fournir des concepts simples et aisément manipulables.

Certains de ces aspect internes sont décrits au Chapitre 11.

10.2 PRESENTATION D'UN EXEMPLE SIMPLE : LE FOUR

Pour illustrer plus concrètement notre propos, nous considérons un exemple d'école : la simulation d'un four ménager. L'objet de cette simulation et les résultats obtenus après les phases de conception partielle sont présentés de façon succincte dans l'encadré suivant :

Exemple du four

On veut proposer une simulation d'un four électrique pour apprendre à un utilisateur à le préchauffer correctement avant d'y mettre un plat cuisiné. Le four est muni d'une commande de marche-arrêt, d'un bouton de commande de puissance réglable de 1 à 8, d'un dispositif de visualisation de la température et d'un témoin de chauffe.

Le travail demandé consiste à régler le four à une température stable de 140°C.

Résultat obtenu après la conception du Modèle

Le four est modélisé sous la forme d'un seul objet caractérisé par :

- les attributs *TempératureRéglée* et *TempératureCourante*
- les opérations *RéglerPuissance (PuissanceDésirée)*
- un diagramme états/transitions comportant 3 états (*Eteint, EnTrainDeChauffer, EnTrainDeRefroidir*) et 3 événements (*MettreEnMarche, Arrêter, Comparer*)

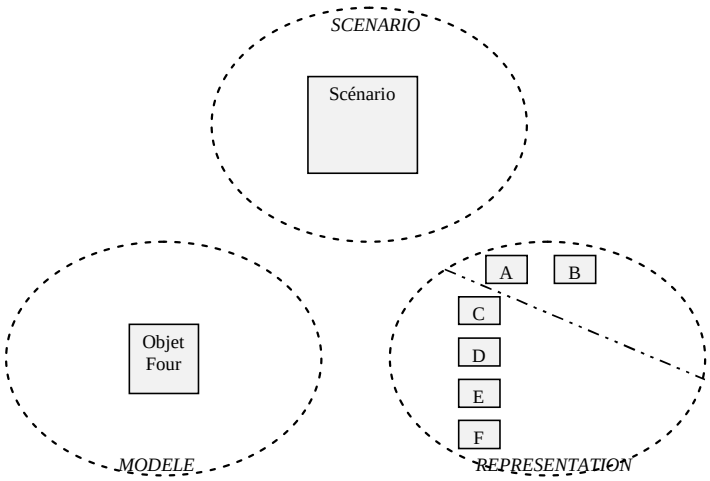
Résultat obtenu après la conception du Scénario

Le scénario est caractérisé par un état initial (*four éteint*) et un objectif (*Amener la température à 140°C*)

Résultat obtenu après la conception de la Représentation

La représentation est composée de 6 objets :

- un bouton A qui permettra de démarrer l'exercice
- un zone de message B réservée aux messages pédagogiques
- un bouton C qui permettra d'Allumer/Eteindre le four
- un voyant D à deux états qui sera allumé lorsque le four est en train de chauffer
- un thermomètre E qui permettra de visualiser la température du four en °C
- un bouton de puissance F qui permettra de régler la puissance de 0 à 8



10.3 TYPOLOGIE DES COOPERATIONS ENTRE ENTITES

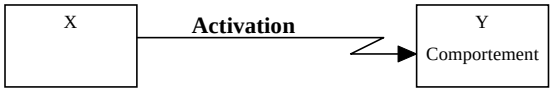
Après avoir identifié les entités opérationnelles définies lors des phases de conception partielle, il faut caractériser les différents types de coopération qui devront exister entre elles lors de l'exécution de l'application intégrée.

L'étude de divers exemples rencontrés dans le contexte Hewlett-Packard permet de dégager quatre types de coopération :

1. l'*activation.*
2. la *consultation.*
3. le *flot de données.*
4. la *synchronisation*.

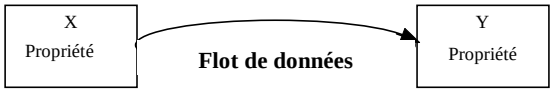
Nous en proposons les définitions suivantes :

L'activation d'une entité Y par une entité X correspond au fait qu'une sollicitation de X produit le déclenchement d'un certain comportement propre à Y.

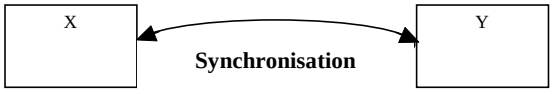


La consultation d'une entité Y par une entité X correspond au fait que pour remplir ses obligations, X doit connaître les valeurs de certaines propriétés de Y.

Le flot de données entre une entité X et une entité Y traduit le fait que les valeurs de certaines propriétés de Y sont directement fonction de valeurs de propriétés de X



La synchronisation entre une entité X et une entité Y traduit le fait qu'à un moment donné, l'évolution concurrente des deux entités X et Y ne peut plus être assurée de façon indépendante.



Décrivons donc pour notre exemple, ces coopérations par couples d'entités concernées (Four, Scénario, A, B, C, D, E, F) :

- quand on presse le bouton *A*, on déclenche le démarrage du *scénario*.
- quand le *scénario* doit donner une indication à l'élève, il active l'affichage d'un message dans la zone de texte *B*.
- quand on presse le bouton *C*, on provoque la mise en marche ou l'arrêt du *four*.
- quand le *four* est en train de chauffer, le voyant *D* est allumé, sinon il est éteint.
- la valeur de la température du *four* est toujours reproduite sur le thermomètre *E*.
- quand on règle le bouton de puissance *F* et si le four est allumé, on provoque l'augmentation de la température du *four*.
- le *scénario* consulte en permanence l'état du *four* pour déterminer si l'objectif est atteint.

La figure 10-2 reprend cet exemple en faisant apparaître les coopérations entre entités et en indiquant leur type.

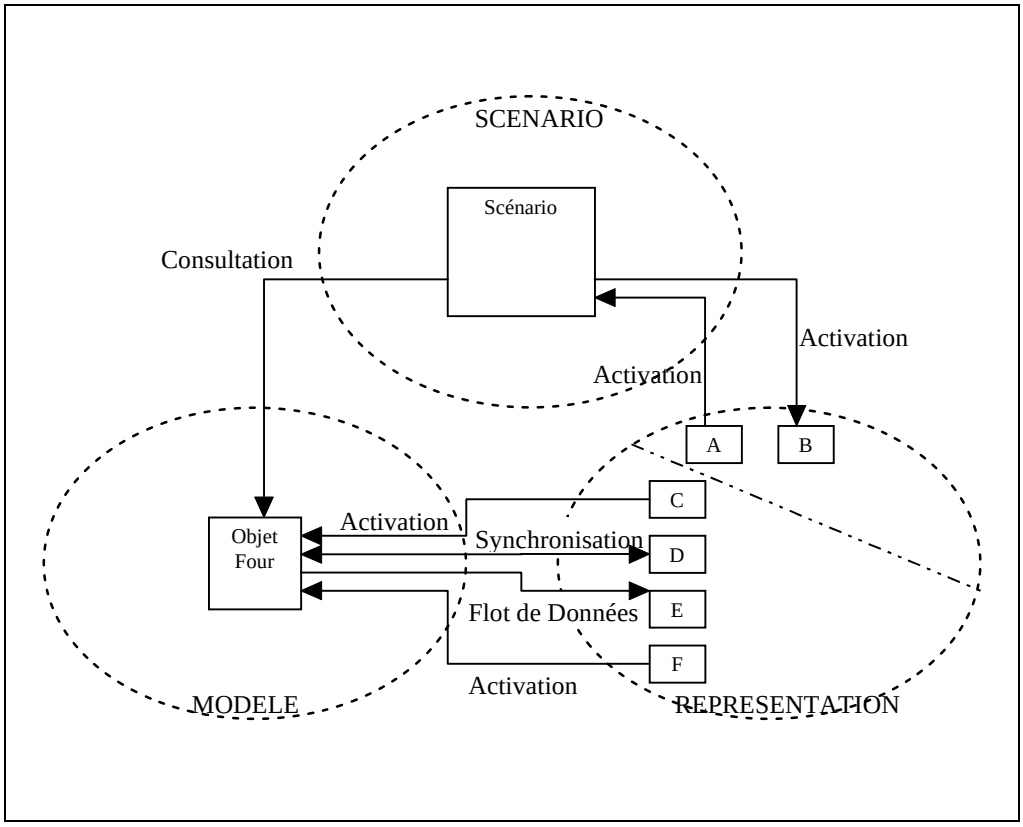


Figure 10-2 : Le réseau de coopération entre les entités de l'exemple "Four"

10.4 LA NOTION D'ASSOCIATION DE COOPERATION

La typologie que nous venons de définir permet de décrire la nature des coopérations qui doivent exister entre les entités au moment de l'exécution. La question est maintenant de proposer à l'intégrateur, des formalismes permettant de spécifier ces différents types de coopération au moment de la mise en correspondance des entités déjà définies dans les espaces de travail. Pour cela, nous introduisons la notion d'association de coopération :

> *Une association de coopération est un lien formel entre deux entités réelles X et Y permettant de décrire précisément la nature de leur coopération au moment de l'exécution. Ce lien peut comporter des données descriptives fixant les détails de cette coopération.*

Contrairement à la définition employée de façon courante dans les méthodes de modélisation (modèle entité-association, modèle par objets), notre notion d'association ne définit pas a priori la nature abstraite de la collaboration entre des entités qui n'existent pas encore. Pour nous, la finalité d'une association est de traduire *a posteriori* la façon dont vont coopérer des *entités réelles déjà existantes,* (i.e. des instances) définies et munies d'un comportement propre et d'une capacité intrinsèque à coopérer.

Nous nous proposons maintenant de formaliser les caractéristiques communes à chaque entité traduisant cette capacité à coopérer ainsi que les mécanismes de coopération entre entités.

10.4.1 Caractéristiques des entités à lier

Pour pouvoir proposer un cadre générique d'association, nous devons être capables de décrire de façon identique les différentes entités concernées. Nous avons vu dans les chapitres précédents que nous convergions vers une représentation par objets, représentation dégradée puisque ne fournissant de façon claire ni les concepts de classe, ni ceux d'instanciation ou d'héritage. Nous caractérisons chaque entité opérationnelle du Modèle et de la Représentation par :

- une *référence* qui permet de la distinguer de façon unique parmi les autres entités.
- un ensemble de *propriétés (ou attributs)* permettant de décrire sa structure.
- un ensemble d'*opérations* permettant depuis l'extérieur de manipuler ou d'accéder à ces propriétés.
- une description de la dynamique par un Statechart de Harel. Cette description est en particulier formalisée par une propriété spécifique de l'entité, *l'état courant* et

par une *liste des états possibles*. Depuis l'extérieur, le comportement dynamique de l'entité est modifiable par l'activation d'un événement choisi parmi une *liste des événements possibles*.

Ce cadre peut être étendu à l'espace de travail Scénario. En effet, nous pouvons également décrire une entité opérationnelle Scénario caractérisée par un certain nombre de propriétés (la description de sa structure et de la réactivité associée), par un certain nombre d'opérations (l'activation, l'arrêt du scénario par exemple), par un comportement dynamique (la progression au moment de l'exécution). La figure 10-3 montre donc le cadre général de description d'une entité opérationnelle.

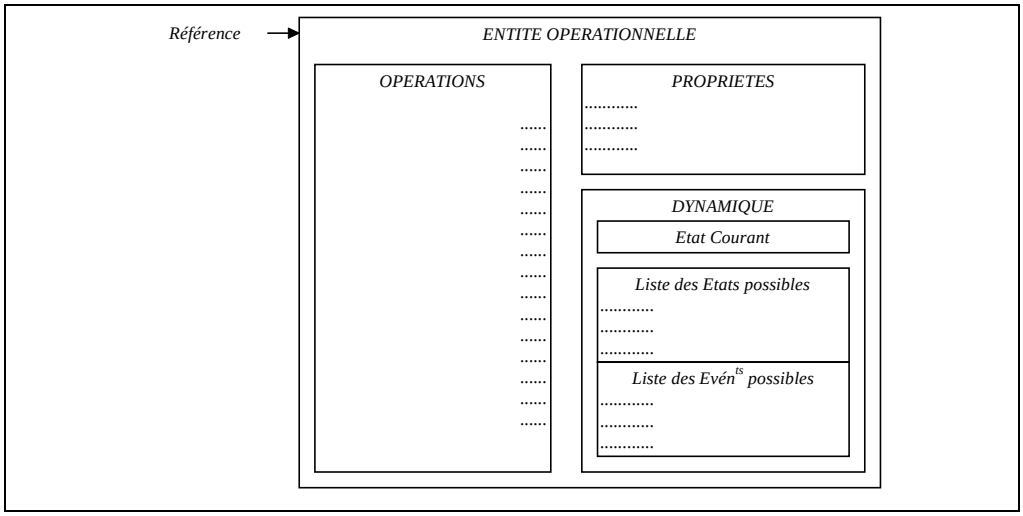


Figure 10-3: Les caractéristiques communes des entités opérationnelles à lier

10.4.2 Les formalismes d'expression de la coopération entre entités

Nous devons maintenant examiner comment les différents types de coopération que nous avons définis plus haut sont traités dans la littérature. Nous allons concentrer notre attention plus particulièrement sur :

- Les modèles à objets.
- Les systèmes multi-agents et les langages d'acteurs.
- Les bases de données actives.

Cet étude est guidée par deux axes d'appréciation :

- Dans quelle mesure les domaines abordés traitent-ils des types de coopération (activation, consultation, flot de données, synchronisation) ?
- Comment les concepts développés dans ces modèles peuvent-ils être exprimés à l'aide des caractéristiques présentées ci-dessus (référence, propriété, opération, état, événement) ?

Pour illustrer notre propos, nous essaierons de traduire les concepts dégagés à l'aide des conventions de représentation graphique suivantes :

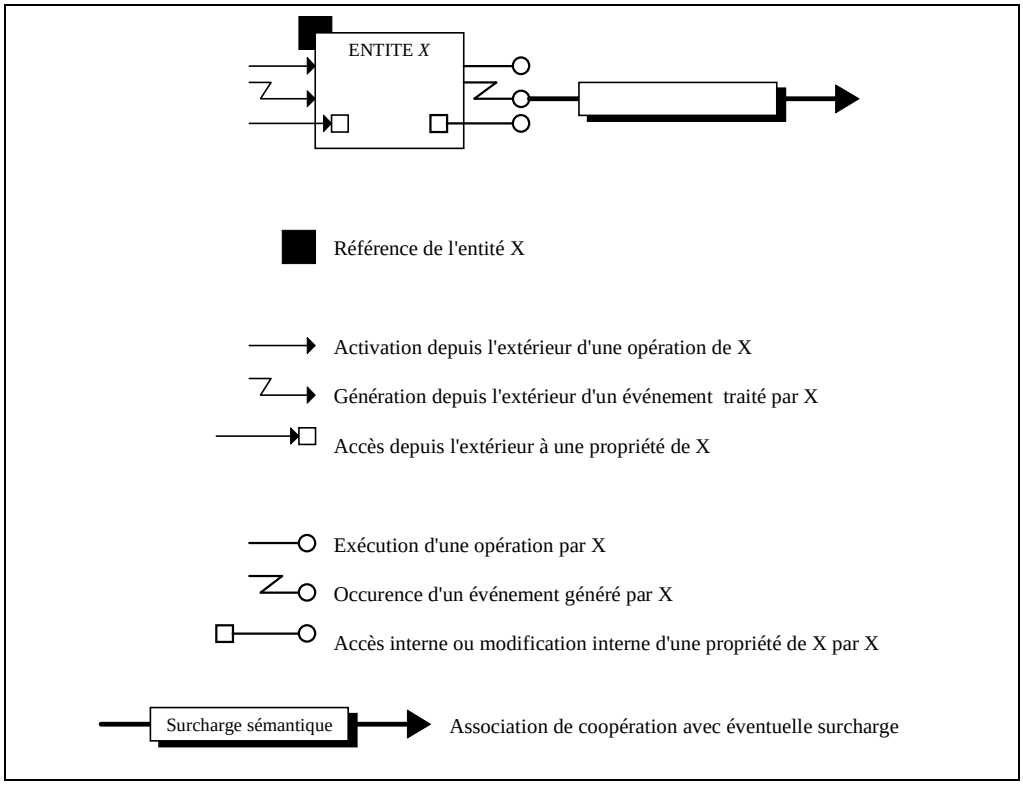


Figure 10-4 : Conventions de représentation graphique des entités

Le résultat de l'étude qui est présentée ci-dessous est abordée de façon plus approfondie dans le rapport de DEA de C. Campani [CAM 95].

10.4.2.1 Les modèles à objets

Notre attention s'est portée sur les méthodes à objets les plus couramment utilisées. Nous nous sommes intéressés plus particulièrement à la méthode OMT de J.Rumbaugh [RUM 91] parce que : (1) elle s'attache à préciser les diverses phases de développement (conception du système, conception des objets et implémentation) ; (2) elle propose une très grande variété de concepts (peut-être même un peu trop grande) et (3) elle associe une représentation graphique aux différents concepts proposés.

Examinons maintenant comment ces méthodes traitent des quatre types de coopération repérés.

L'activation

L'activation est bien entendue abordée dans les modèles à objets par le mécanisme d'envoi de message. Certaines méthodes proposent des notations

graphiques pour représenter l'envoi d'un message depuis un objet vers un autre : connexion de messages de OOA [COA 91], graphe d'interactions de Fusion [COL 94], diagramme d'interactions de OOAD [BOO 94] ou diagramme de propagation des opérations dans OMT. En général, ces notations sont très limitées au niveau sémantique, car elles ne précisent que très peu les conditions d'envoi du message. Les plus riches, comme celles de Fusion ou OOAD, indiquent le nom de la méthode de l'objet émetteur, le nom de la méthode invoquée dans l'objet récepteur, un éventuel prédicat de sélection et l'ordre d'envoi des messages. La définition précise des conditions d'envoi est reportée au stade de la conception détaillée ou de l'implémentation.

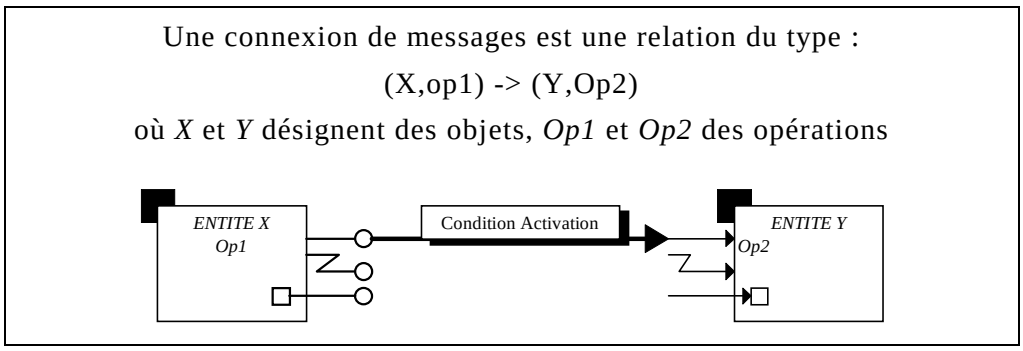


Figure 10-5 : Représentation d'une connexion de message

Le flot de données

La majorité des méthodes par objets proposent des formalismes pour exprimer des flots de données. OMT fournit par exemple le concept d'objets acteurs décrit dans le modèle fonctionnel. Cette notion permet d'établir une relation entre la valeur d'une propriété p1 d'un objet X et celle d'une propriété p2 d'un objet Y. Un traitement peut surcharger éventuellement cette relation par une fonction de transformation.

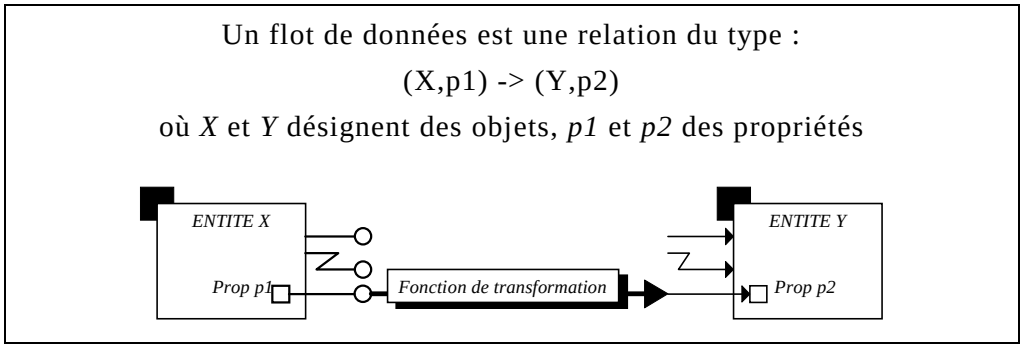


Figure 10-6 : Représentation d'un flot de données

Cette notation graphique nous parait satisfaisante pour exprimer le flot de données ; elle permet de concentrer la complexité éventuelle des relations de

dépendance au niveau du lien lui-même, et non de la reporter sur les objets.

La synchronisation

Nous avons vu plus haut comment était abordée la notion de synchronisation dans le processus de modélisation par objets (cf. Chapitre 7). Nous avons retenu la notation proposée par OMT, basée sur les Statecharts de Harel, car elle nous semble un bon compromis entre la puissance d'expression et la lisibilité. Toutefois, nous devons noter qu'elle se prête mal à l'expression graphique de la synchronisation d'entités déjà existantes. En effet, si chaque entité dispose de son propre Statechart, la méthode préconise de construire un nouveau Statechart regroupant ceux de chacune des entités concernées et intégrant les relations de concurrence et de synchronisation.

Une solution partielle à ce problème peut être l'établissement d'une association entre les deux Statecharts précisant les relations de dépendance par une relation du type :

(X,s) -> (Y,e)

où *X* et *Y* désignent des objets, *s* un état et *e* un événement

La sémantique d'une telle association serait :

A chaque fois que l'état de l'objet X devient égal à s,
appliquer l'événement e sur l'objet Y.

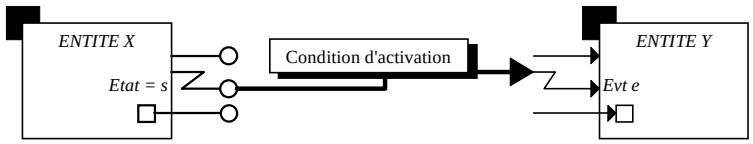


Figure 10-7 Représentation d'une synchronisation de deux Statecharts

La consultation

Dans l'approche par objets, la consultation est traitée de façon traditionnelle en répartissant la liste des attributs en deux catégories : celle des attributs publics accessibles depuis l'extérieur et celle des attributs privés non accessibles. Ainsi un objet X peut accéder à et consulter toutes les propriétés publiques d'un objet Y. Certains modèles règlent ce problème en forçant à passer par une opération publique pour accéder à la valeur d'une propriété.

Il n'existe pas à notre connaissance de formalisme graphique pour traduire précisément les conditions d'une consultation d'un attribut public d'un objet Y par un objet X. L'assimilation de cette consultation à l'activation d'une opération donnant un résultat équivalent nous permet de revenir au premier cas décrit, celui de l'activation.

10.4.2.2 Les systèmes multi-agents et les langages d'acteurs

Les modèles multi-agents ont été développés à l'origine dans le domaine de l'intelligence artificielle distribuée. Un système multi-agents est conçu comme une société d'agents autonomes travaillant en commun (selon certaines règles de coopération, de résolution de conflits et de concurrence) pour aboutir à un objectif global (résolution d'un problème, établissement d'un diagnostic, construction d'un plan, etc.). Toute décision est le fruit d'un compromis réalisé par négociation entre les agents (proposition, contre-proposition, acceptation, refus, etc.).

Pour Ferber [FER 89b], un agent est vu comme "*une entité physique ou abstraite qui est capable d'agir sur elle-même et sur son environnement, qui dispose d'une représentation partielle de cet environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de sa connaissance et des interactions avec les autres agents*".

La coopération est une préoccupation essentielle des systèmes multi-agents. Ceci nous a naturellement conduit à étudier les modes de communications sur lesquels ils s'appuient. De façon classique, on distingue deux approches : l'approche de type partage d'informations [ENG 88], et l'approche par envoi de message. La première approche, basée sur le partage d'un espace commun (blackboard) contenant toutes les informations nécessaires à la résolution du problème, nécessite la présence d'une structure supplémentaire de communication, et ne sera donc pas abordée ici.

Les systèmes multi-agents utilisant la communication par message, distribuent totalement à la fois les connaissances, les résultats partiels et les méthodes utilisées pour aboutir à un résultat. Cette approche a été systématisée (par la prise en compte, en particulier, de la concurrence) par les travaux de Hewitt [HEW 77] et de Agha [AGH 86] sur les langages d'acteurs. Les concepts essentiels de ces systèmes sont [FER 89b] :

- *le traitement local* : un agent ne peut que manipuler sa base de connaissance locale, et envoyer des messages à des agents qu'il connaît, ou *accointances*.
- *l'envoi de message avec continuation* : quand un agent envoie un message, il précise à quel agent la réponse de ce message doit être envoyée.
- la possibilité de définir des traitements appelés *réflexes*, associés non pas à la réception de message mais à l'occurrence d'événements spécifiques déclenchés de façon automatique. Les principaux types d'événements susceptibles de déclencher un réflexe sont l'accès ou la modification de la valeur d'un attribut, la création d'un objet ou la modification de sa structure.

Parmi les modes de coopération que nous avons identifiés plus haut, les modèles multi-agents basés sur les messages s'intéressent essentiellement aux coopérations de type *activation* ou *synchronisation*. Les mécanismes spécifiques qu'ils offrent sont le plus souvent complexes et difficilement adaptables à notre problématique. La raison principale en est qu'ils cherchent à décrire un monde où les agents, les attributs, les données sont constamment créés, supprimés, modifiés pour viser un but commun unique. Cette instabilité n'est pas propice à la description *a priori* des relations qui unissent les agents. On peut toutefois retenir deux idées intéressantes :

- La notion *d'accointances* permet de définir l'activation d'un ensemble d'objets {Y} par un objet X par une simple relation *X.ListeDesAccointances -> {Y.Référence}*. Mais cette possibilité induit que l'objet émetteur X est préprogrammé pour communiquer.
  Une connexion d'accointance peut donc être considérée comme une relation multivaluée, comme le montre la figure suivante.

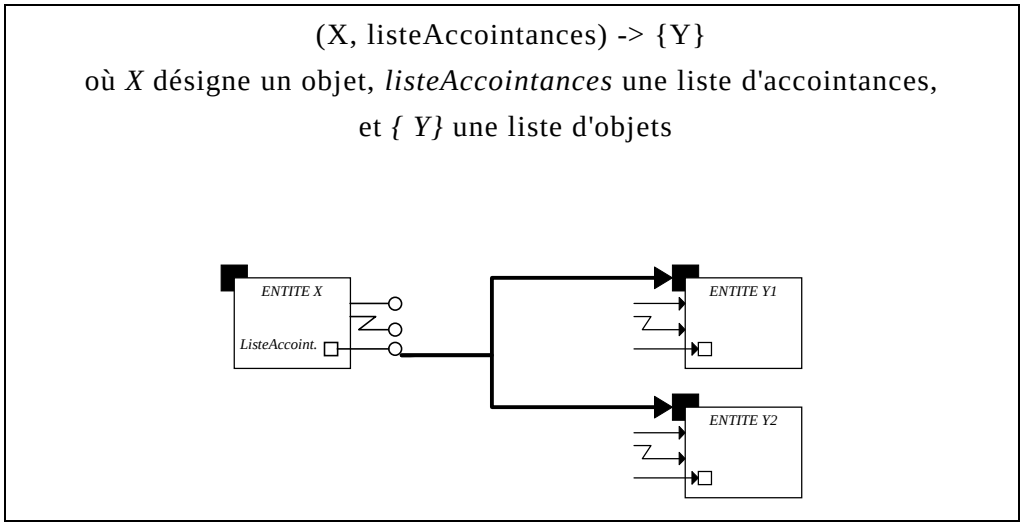


Figure 10-8 : X communique avec ses accointances Y1 et Y2

- La notion *de réflexe* permet d'envisager par exemple l'envoi d'un message à un objet à chaque modification de la valeur d'un attribut. Cette possibilité peut être intéressante dans notre contexte pour exprimer que l'envoi d'un message est subordonné à l'état de l'objet émetteur. Cette notion est approfondie par les règles événement/condition/action présentées dans le paragraphe suivant.

10.4.2.3 Les bases de données actives

Nous nous sommes intéressés au domaine des bases de données actives parce qu'elles ont pour objectif de traiter des objets informatifs *sans demande explicite* de la part des utilisateurs. Cette approche recouvre en partie nos préoccupations de recherche de *mécanismes* automatiques d'activation.

Cette approche est basée sur le concept de déclencheur, ou trigger [BRO 81, DIT 86], et est en particulier mise en œuvre dans des systèmes de base de données par objets tels que HiPAC, ODE, O2 ou SAMOS [ADI 93]. Un déclencheur peut être considéré comme un triplet <***événement, condition, action***> où :

- *l'événement* correspond à une transaction sur un objet et recouvre les cas suivants :
  ⇒ l'insertion de nouvelles données.
  ⇒ la modification de données.
  ⇒ la suppression de données.
  ⇒ l'exécution d'une méthode.
- la *condition* est, en général, spécifiée par un prédicat exprimé dans le langage d'interrogation du système de gestion de bases de données.
- les seuls types *d'actions* sont, dans la majorité des systèmes, des instructions spécifiées dans le langage de manipulation des données. Elles peuvent parfois être enrichies de constructeurs tels que "*if ... then ... else ...*" ou "*while ... do ...*".

Les déclencheurs sont essentiellement utilisés pour assurer l'intégrité des bases de données. Mais ils peuvent aussi être utilisés pour les déclenchements d'actions en cascade (lorsque je modifie A alors j'exécute B) ou d'actions qui doivent apparaître à un instant précis (j'exécute A à la date t). De façon générale, on peut distinguer trois types d'événements [ADI 93] :

- les événements *externes* qui sont provoqués par une intervention de l'utilisateur.
- les événements *internes* qui sont générés par le système lui-même (répercussions des événements externes notamment).
- les événements *temporels* qui apparaissent à une certaine date ou à des intervalles de temps réguliers.

Cette approche basée sur les règles Evénement/Condition/Action (E.C.A.) peut être adaptée à notre problématique de coopération. Nous pouvons par exemple la présenter sous la forme :

Evénement = Accès interne à un attribut de l'objet X émetteur
Condition = Prédicat portant sur l'état de X
Action = Activation d'une opération de l'objet Y destination

Une telle règle peut se représenter comme une relation du type :

(X,p) -> (Y,Op)
où *X* et *Y* désignent des objets, *p* une propriété et *Op* une opération

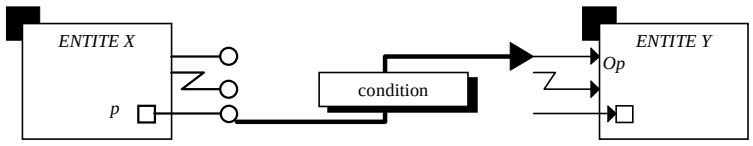


Figure 10-9 : Représentation d'une règle E.C.A. de coopération

10.4.3 Conclusions sur ces formalismes

Nous venons d'examiner un certain nombre de formalismes adaptés à l'expression de la coopération et nous avons essayé de les représenter sous forme graphique en termes d'association entre entités coopérantes.

Dans certains cas, nous avons défini une surcharge sémantique comme un ensemble d'informations liées à une association de coopération permettant de la rendre exécutable. En l'absence de cet ensemble d'informations, ou en cas d'informations incomplètes, l'environnement est dans l'impossibilité de générer le code informatique traduisant la coopération souhaitée. C'est le cas, par exemple, de la connexion de messages. Quand on spécifie une connexion de message (Cf. Figure 10-5), il n'est pas possible de déterminer de façon précise quand et comment sera envoyé le message si ce n'est en indiquant l'endroit du code source où doit être inséré cet envoi. Parce que nous voulons soulager l'intégrateur d'une partie des soucis de programmation, en lui évitant de manipuler le code source, nous abandonnons ce type de coopération, et de façon plus générale, les associations dont l'origine est une opération.

Il est également apparu une distinction entre deux politiques de gestion de la

coopération, selon le type (simple ou intelligent) des associations :

- Une *association simple* se contente d'établir un lien référentiel entre deux entités. Cette approche peut résulter soit de la simplicité de la coopération à décrire, comme par exemple dans le cas du flot de données sans calcul, soit d'un choix de répartition de l'"*intelligence*". Les systèmes multi-agents font ainsi le choix explicite de la répartition des connaissances et de l'intelligence au sein même des agents ; cette volonté exclut, de fait, la surcharge d'information sur les communications entre agents.
- Une *association intelligente* permet d'établir une relation entre deux entités en l'enrichissant d'un ensemble d'informations précisant les conditions ou la nature de la coopération. Dans les cas que nous avons étudiés, cette surcharge implique le plus souvent une certaine limitation de l'intelligence des entités impliquées, tout du moins en ce qui concerne leur capacité à coopérer.

La figure 10-10 illustre ce problème du niveau de complexité des associations en proposant deux solutions de coopération entre le *bouton A* et l'objet *four* de notre exemple décrit plus haut.

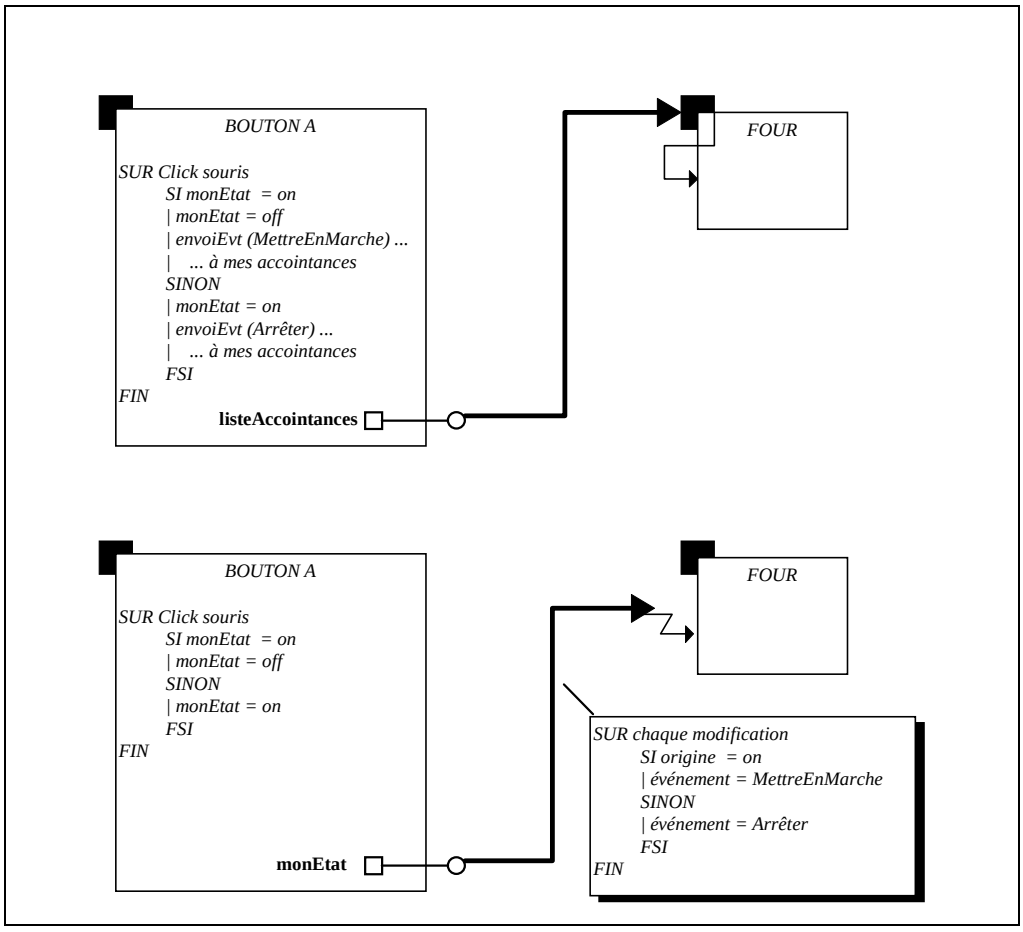


Figure 10-10 : Association simple et Association intelligente

Dans le premier cas, la coopération est répartie en deux endroits. En premier lieu, l'objet émetteur possède une partie de la connaissance de la coopération potentielle. Il sait qu'il devra envoyer l'événement *MettreEnMarche* ou *Arrêter* à une certaine liste d'objets *listeAccointances*. Dans un deuxième temps, cette coopération est matérialisée par l'association réalisée entre *listeAccointances* et l'objet *FOUR*. Cette solution suppose qu'il existe un protocole de communication préalable et que l'événement *MettreEnMarche* a un sens pour l'objet *FOUR*. Dans le cas contraire, la coopération supportée par l'association n'aura pas de signification.

Dans le second cas, la coopération est concentrée au niveau de l'association. Le bouton A est conçu uniquement pour remplir ses obligations ; il n'a pas de connaissance préalable d'un futur rôle de coopération. Par contre, l'association contient toutes les informations nécessaires à l'activation de la coopération et n'a de sens que pour ce couple précis *(bouton A, Four)*.

Cet exemple montre qu'il existe en fait deux politiques bien différentes de traitement de la coopération :

- la *coopération répartie*. Cette approche suppose que chaque entité du système prise isolément a une connaissance précise de ses capacités à coopérer.

  L'avantage principal de cette solution est :

  ⇒ la simplicité de création des associations et la facilité de la tâche d'intégration.

  ⇒ la possibilité de définir des traitements génériques de coopération (grâce au concept d'accointances).

  Par contre, les inconvénients majeurs sont :

  ⇒ l'obligation lors de la conception de chaque entité de connaître par avance l'environnement dans lequel elle sera insérée (ce qui entraîne une hyperspécialisation des composants et ne favorise pas la réutilisation).

  ⇒ L'impossibilité de traduire des types de coopération non prévus lors de la conception des composants.

  ⇒ Un certain manque de maîtrise de l'ensemble de la coopération dû à la répartition des traitements.

- La *coopération centralisée*. Cette approche concentre l'effort de coopération au moment de l'intégration des composants. Lors de leur conception, les composants n'ont pas été prévus pour répondre à des besoins spécifiques de coopération.

  Les avantages de cette solution sont :

  ⇒ l'indépendance des activités de conception et d'intégration.

  ⇒ une plus grande finesse dans la définition d'une association de coopération.

⇒ une meilleure maîtrise de la définition globale de la coopération.

⇒ une meilleure aptitude à la réutilisation des composants qui ne sont pas spécialisés dans un rôle de coopération déterminé.

Les inconvénients correspondent de façon symétrique aux avantages avancés pour la solution répartie :

⇒ la spécification relativement complexe des associations de coopération.

⇒ la définition redondante d'associations de coopérations de même nature. Si dans l'exemple, le bouton A devait commander en même temps dix fours identiques, les dix mêmes associations devraient être répétées.

Les choix de processus de développement que nous avons présentés dans le modèle M.A.R.S nous poussent naturellement à opter pour la solution centralisée qui permet l'indépendance des activités de conception et d'intégration. Notre approche consiste à vouloir limiter les inconvénients induits par cette approche.

10.5 UNE PROPOSITION. FORMALISMES GRAPHIQUES

Notre démarche réside donc dans les points suivants :

- Nous voulons proposer à l'intégrateur un nombre réduit de formalismes lui permettant d'exprimer de façon précise les différents types de coopération que nous avons répertoriés (activation, consultation, flots de données et synchronisation).
- Nous voulons réduire au maximum les tâches répétitives de définition d'associations de coopération.

10.5.1 Choix d'un mécanisme de coopération et d'un formalisme uniques

Nous avons introduit dans le paragraphe précédent, le mécanisme d'activation automatique basé sur les règles E.C.A. dans le domaine des bases de données actives. Nous avons décidé, dans un premier temps, de généraliser ce mécanisme dans notre modèle M.A.R.S afin qu'il serve de référence conceptuelle à l'intégrateur. Quand ce dernier doit créer une association de coopération entre deux entités, nous voudrions qu'il la spécifie dans les termes suivants :

- Quel événement est à l'origine d'une coopération entre les deux entités X et Y ?
- Sous quelles conditions cette coopération est-elle effective ?
- Quel est le résultat de cette coopération ?

Le tableau suivant montre pour chacun des types de coopération que nous avons définis une traduction possible en termes <événement/condition/action> :

Type de coopération	Evénement	Condition	Action	Autres Informations
Activation	Modification ou accès à une propriété P de X	aucune ou un prédicat portant sur les propriétés de X	Activation d'une opération Op de Y	Pour chaque paramètre de Op: une valeur fixe ou calculée (fonction portant sur les valeurs des propriétés de X)
Flots de données	Modification d'une propriété P1 de X	aucune ou un prédicat portant sur les propriétés de X	Affectation de la propriété P2 de Y à la valeur calculée F(X.P1)	Une fonction F de calcul
Synchronisation	Modification de la propriété Etat de X	aucune ou un prédicat portant sur les propriétés de X	Envoi d'un événement à Y.	
Consultation	Modification ou accès à une propriété de X	aucune ou un prédicat portant sur les propriétés de X	Activation d'une opération de lecture par X d'une propriété de Y	

Si l'on examine ce tableau, on s'aperçoit que seules diffèrent les deux dernières colonnes. Nous pouvons encore simplifier la colonne *Action* en ramenant les quatre cas au premier d'entre eux. Nous devons alors faire les suppositions suivantes :

- Flots de données : pour chaque propriété *Prop* accessible en écriture, il existe une opération *Ecrire_Prop (valeur)* permettant de modifier sa valeur.
- Synchronisation : pour chaque entité, il existe une opération *ActiverEvénement (nomEvénement)* qui peut être activée.
- Consultation : pour chaque propriété *Prop* accessible en lecture, il existe une opération *Lire_Prop ()* retournant sa valeur.

Le tableau précédent devient alors :

Type de coopération	Evénement	Condition	Action	Autres Informations
Activation	Modification ou accès à une propriété P de X	aucune ou un prédicat portant sur les propriétés de X	Activation d'une opération *Op* de Y	Pour chaque paramètre de l'opération une valeur fixe ou calculée (fonction portant sur les valeurs des propriétés de X)
Flots de données			Activation de l'opération *Ecrire_P2 (F(P))*	
Synchronisation			Activation de l'opération *ActiverEvénement (nomEvénement)*	
Consultation			Activation de l'opération *Lire_P2*	

Nous arrivons ainsi à un formalisme unique de règle E.C.A. où nous avons :

Evénement = Modification ou accès à une propriété de l'objet X émetteur
Condition = Prédicat portant sur les propriétés de l'objet X émetteur
Action = Activation d'une opération de l'objet Y destination

Ce formalisme peut facilement être représenté de façon graphique. Chaque entité est munie de connecteurs répartis en deux catégories :

- les connecteurs "origine" qui correspondent à un nom de propriété (cette propriété peut être l'*état* de l'entité).
- les connecteurs "destination" qui correspondent à un nom d'opération (cette opération peut correspondre à l'activation d'un événement ; la lecture ou l'écriture d'une variable).

Les associations sont des flèches orientées reliant un connecteur origine à un connecteur destination. Elles précisent les conditions d'activation, et contiennent les informations nécessaires à la description précise de l'activation.

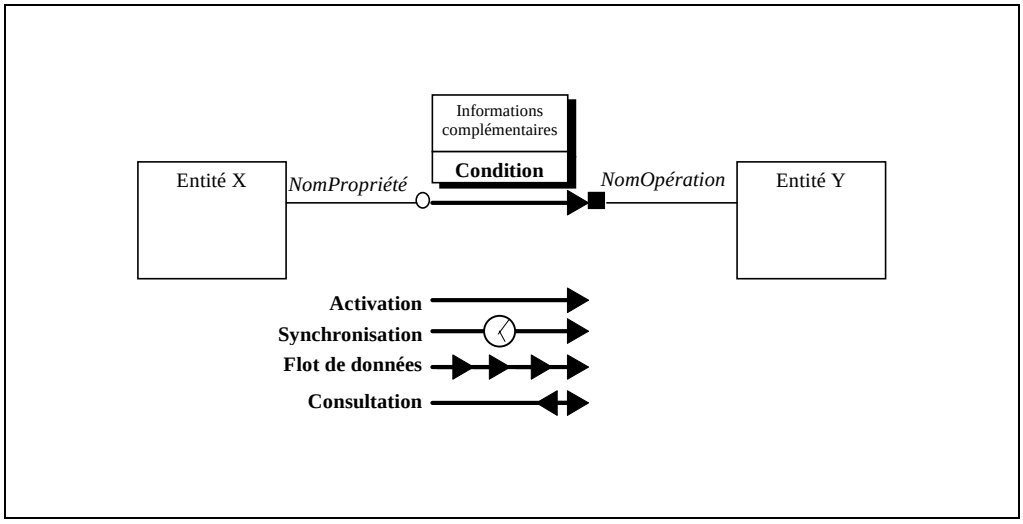


Figure 10-11 : La représentation graphique d'une association de coopération

En établissant ce type d'associations pour un système complet, on obtient un réseau qui permet d'avoir une vue d'ensemble de la coopération du système. L'application de cette notation à notre exemple "Four" est présentée à la figure 10-12.

Nous nous sommes inspirés de ce formalisme et de cette représentation graphique dans l'environnement MELISA présenté au Chapitre 11.

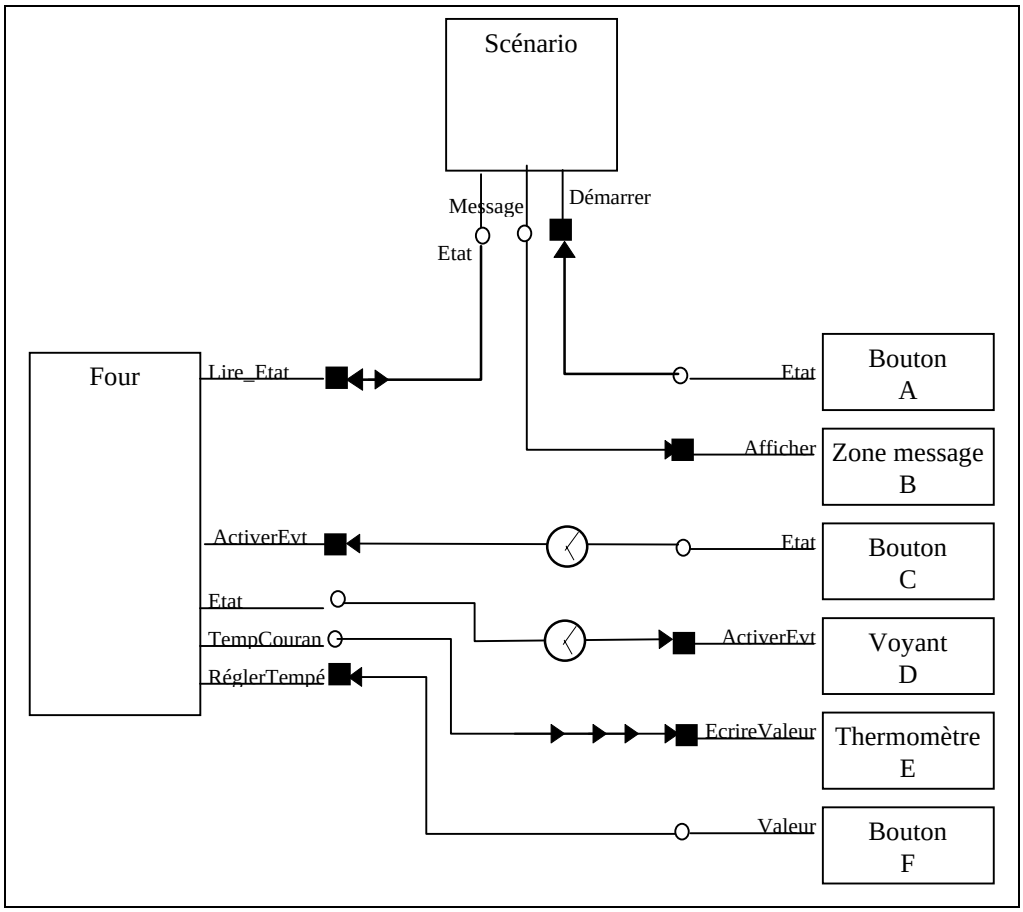


Figure 10-12 : Le réseau de coopération de l'exemple "Four"

10.5.2 La factorisation des associations : les collections

Nous avons souligné la difficulté à spécifier et à modifier plusieurs associations de type identique. Par exemple, pour réaliser une simulation de clavier informatique comportant 102 touches, il sera nécessaire d'établir 102 associations entre chacune des touches et l'objet abstrait "Ordinateur". Cette solution parait très fastidieuse et se prête très mal à toute évolution de la coopération.

Une autre solution consiste à s'appuyer sur la notion de *collection* introduite dans certains modèles par objets comme Fusion [COL 94]. Une collection est un ensemble d'objets (i.e. d'instances) issus de la même classe et susceptibles d'être impliqués dans une même association. Le nombre d'objets dans une collection peut être variable, voire nul. Ce concept de collection est décrit de façon plus approfondie dans le document [CAM 95]. Il peut être appliqué à notre problématique pour résoudre le problème de la multiplicité des associations de coopération de même type. Une association de coopération entre collections est alors notée :

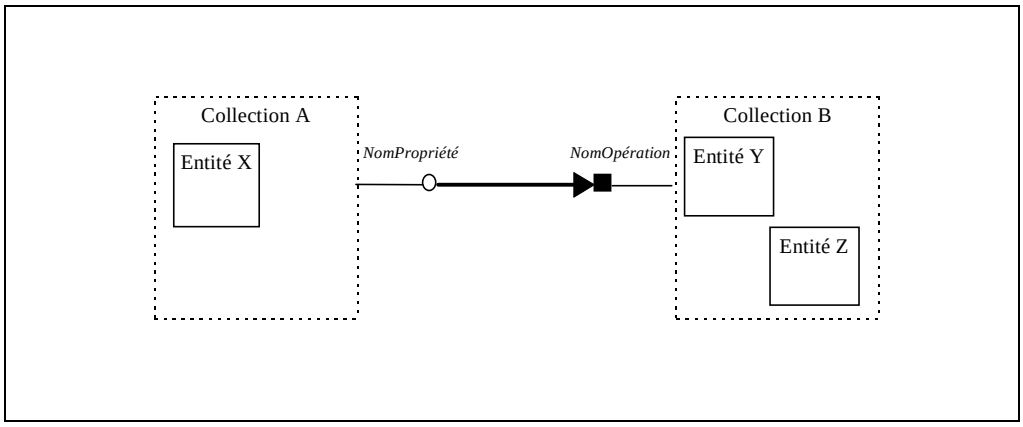


Figure 10-13 : La représentation d'une association de coopération entre collections

10.6 INTERFACES DE SPECIFICATION ET DE VALIDATION

Comme dans les autres espaces de travail, nous proposons des éditeurs spécialisés permettant de créer et modifier les réseaux de coopération. Ces éditeurs doivent permettre de raisonner à deux niveaux :

- un niveau global afin de disposer d'une vue d'ensemble du réseau de coopération, avec en particulier le concept de collection. Les conventions graphiques sont celles que nous avons retenues dans ce chapitre.
- un niveau spécifique permettant de préciser, modifier la nature de chaque association.

Il n'est pas nécessaire de prévoir d'outil spécifique de validation des résultats produits. En effet, une fois les associations réalisées, le résultat produit est manipulable à travers l'application finale elle-même.

Les mécanismes que nous avons décrits dans ce chapitre ont fait l'objet d'expérimentations dans le cadre du projet MELISA. Certains des concepts ont été mis en œuvre, en particulier la généralisation des règles ECA comme moyen d'exprimer la coopération. Le chapitre 11 présente un exemple complet de conception utilisant de telles associations.

10.7 RESUME

Dans ce chapitre, nous avons abordé le dernier espace, l'espace de travail Association.

Après avoir donné une définition générale du terme de coopération, nous avons distingué trois niveaux d'abstraction : le niveau macroscopique, le niveau conceptuel et le niveau implémentation. Dans la suite du chapitre, nous nous sommes essentiellement attachés à décrire le niveau conceptuel, celui qui sera manipulé par le concepteur des associations ou intégrateur.

A partir d'un exemple représentatif de la classe de problèmes que nous voulons résoudre, nous avons dégagé quatre types de coopération différents : l'activation, la consultation, le flot de données et la synchronisation.

Après avoir évalué différents formalismes d'expression de la coopération entre entités (modèles objets, systèmes multi-agents, langages d'acteurs, bases de données actives), nous avons défini pour les quatre types de coopération, un formalisme unique de coopération basé sur les règles Evénement/Condition/Action où :

- l'événement représente la modification ou l'accès à une propriété de l'objet émetteur.
- la condition représente un prédicat portant sur les propriétés de l'objet émetteur.
- l'action représente l'activation d'une opération de l'objet destination.

Nous avons également défini une représentation graphique du formalisme de coopération proposé, ainsi que les interfaces de spécification correspondantes.

PARTIE 4
MISE EN OEUVRE

Cette quatrième et dernière partie est consacrée à la mise en œuvre du modèle MARS.

Le chapitre 11 présente l'environnement auteur MELISA développé pour les besoins propres de la société Hewlett-Packard.

11. L'ENVIRONNEMENT MELISA

11.1 RAPPEL DU CONTEXTE

Le projet MELISA[1] (**M**ethodology and **E**nvironment for developing **L**earning, **I**nstruction and **S**imulation **A**pplications) est un projet de recherche et développement, établi entre le laboratoire CLIPS-IMAG et la division TPEC (Technical Planning and Education Center), principal centre européen de formation interne de l'entreprise Hewlett-Packard.

Le projet MELISA en est à sa troisième année et a permis, en particulier, le financement des travaux de recherche présentés dans cette thèse. La première année a été consacrée à une étude poussée des besoins et à l'élaboration de maquettes permettant de valider certaines hypothèses de travail. La seconde année a permis le développement du premier prototype opérationnel (Version 0.2) décrit dans ce chapitre. Ce prototype permet de couvrir l'ensemble du processus de développement décrit dans cette thèse, mais certaines des fonctionnalités qu'il propose sont réduites ou incomplètes. La troisième année, l'année 1996, est consacrée à la fiabilisation du prototype opérationnel (Version 1.0), à l'expérimentation de cette version sur de véritables projets de formation, et à l'élargissement de certaines fonctionnalités au vu des résultats d'expérimentation (Version 2.0).

[1] *L'environnement MELISA utilise des termes anglais, la langue anglaise étant la langue de communication interne à la société Hewlett-Packard*

Rappelons brièvement les objectifs du projet MELISA qui ont été présentés en détail dans le chapitre 1.

La société Hewlett-Packard est fréquemment confrontée à certaines difficultés de formation de ses personnels techniques. Ces problèmes sont principalement dus à :

- une mauvaise adéquation dans le temps de la formation centralisée.
- la difficulté de transmettre certains savoir-faire.
- la difficulté d'évaluation des compétences effectives acquises après une formation.

Le recours à la simulation pédagogique parait une solution intéressante pour résoudre en partie ces problèmes mais difficile à mettre en œuvre pour les raisons suivantes :

- la responsabilité du développement de telles simulations est du ressort du responsable de produit (Product Engineer) dont les compétences, ni celles de son équipe, ne couvrent le développement traditionnel de logiciel (maîtrise d'un langage de programmation, en particulier).
- Il n'existe pas aujourd'hui d'environnement de niveau "auteur" permettant le développement de simulations pédagogiques.

L'objectif essentiel du projet MELISA est donc de définir et de développer un environnement intégré de production de simulations pédagogiques adapté à un tel profil d'auteur. Cet environnement doit répondre aux critères suivants :

- Aptitude à créer rapidement des applications.
- Aptitude à créer des simulations "simples" à vocation pédagogique.
- Aptitude à spécifier le contrôle pédagogique à effectuer sur les simulations produites.
- Aptitude au prototypage et à la réutilisation.
- Aptitude à créer des applications réalistes.
- Simplicité et pertinence des interfaces proposées aux auteurs.
- Sûreté des applications produites.

Après étude de plusieurs solutions techniques, il a été décidé de construire cet environnement sur la base du logiciel Toolbook CBT, déjà choisi par l'entreprise Hewlett-Packard comme plate-forme générique de développement d'applications pédagogiques.

11.2 PRESENTATION GENERALE

11.2.1 Architecture logicielle

Comme il a été indiqué au chapitre 3, l'environnement MELISA est composé de surcouches construites autour du système Toolbook (Cf. Figure 11-1).

Cette solution présente les avantages suivants :

- le logiciel Toolbook est un environnement interprété, ouvert et adaptable. L'écriture de la surcouche MELISA correspond à l'ajout de nouvelles fonctionnalités au niveau auteur. Le choix de l'ajout ou du retrait de cette surcouche indépendante peut être réalisé en une seule opération utilisateur.
- les applications produites avec l'environnement MELISA génèrent des composants logiciels écrits dans le langage OpenScript, langage interne de Toolbook. Elles sont donc compatibles avec la version de base : l'environnement MELISA n'est pas nécessaire pour leur exécution, ce qui facilite la diffusion des simulations créées.
- L'ajout de la surcouche MELISA ne masque en rien les fonctionnalités de base proposées par le système. Ceci garantit aux utilisateurs habituels de Toolbook de toujours pouvoir accéder à l'interface et au langage proposés dans la version de base du système.

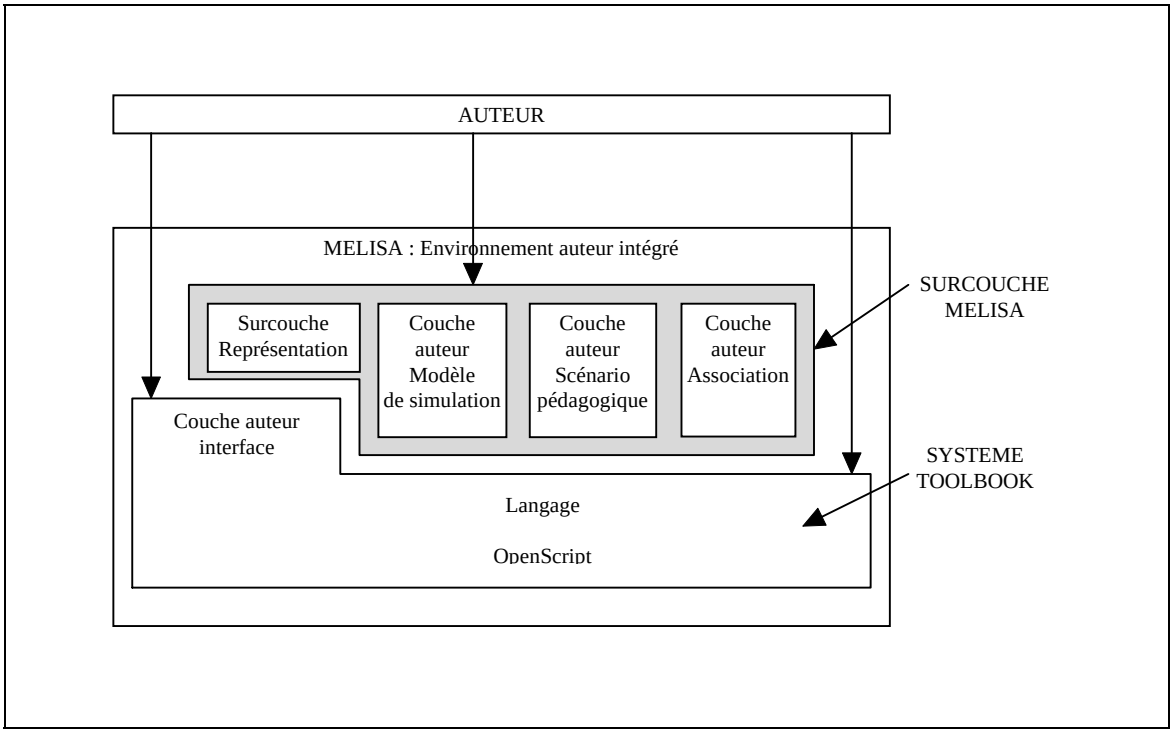


Figure 11-1 : MELISA : Architecture logicielle et visibilité de l'auteur

11.2.2 Le logiciel Toolbook

Le logiciel Toolbook appartient à la classe des générateurs d'applications hypermédias que nous avons décrite au chapitre 3. Rappelons brièvement ses caractéristiques.

En premier lieu, Toolbook est un système qui permet la création rapide d'interfaces par manipulation directe. Chaque application créée est structurée selon la métaphore du livre. Toolbook propose un mode de création, ou mode *auteur*, et un mode d'exécution, ou mode *lecteur*. Chaque application est considérée comme un *livre*. Chaque livre contient des écrans indépendants, ou *pages*. Plusieurs pages peuvent partager le même *fond*, ou arrière-plan. Chaque page contient des *objets* de présentation qui peuvent être de différents types (texte, bouton, graphique, image, scène multimédia). Au sein d'une même page, ces objets peuvent être associés en groupes ou en sous-groupes. Il existe donc, pour chaque application, une *hiérarchie d'objets* dans laquelle on peut distinguer les objets de structuration et les objets de présentation. Cette hiérarchie peut éventuellement être complétée par un ou plusieurs *livres système*. Chaque livre système est en fait une application Toolbook indépendante contenant la définition de traitements qui pourront être communs à plusieurs applications. C'est en particulier grâce au concept de livre système que peut être réalisée une surcouche auteur.

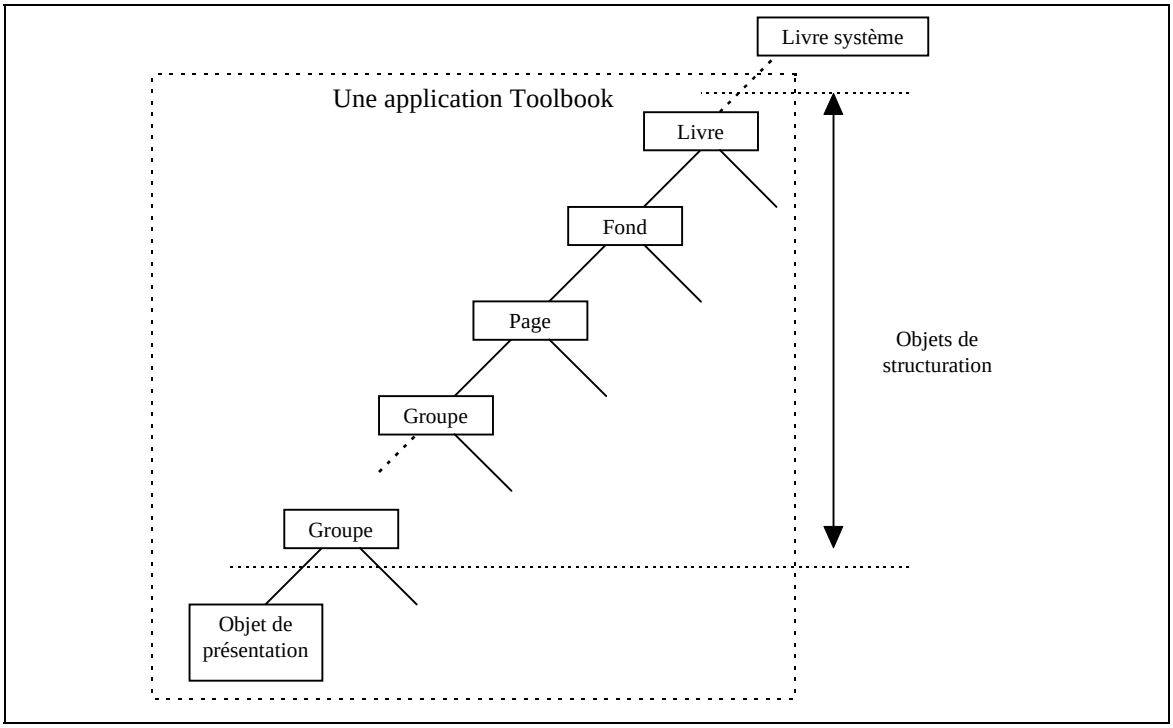


Figure 11-2 : La hiérarchie des objets Toolbook

Les objets créés de façon interactive par l'environnement Toolbook sont des objets inertes : aucun traitement ne leur est associé. Toolbook permet d'attacher à chacun des objets un script décrivant son comportement. Chaque script est composé de *handlers*. Un handler est un morceau de code écrit en langage OpenScript et identifié par un nom : ce code sera activé à chaque fois que l'objet recevra un message du même nom. Les messages dans Toolbook peuvent être générés de façon automatique par le système ; c'est le cas par exemple à chaque fois que le lecteur réalise une opération à l'aide du clavier ou de la souris. Les messages peuvent également être générés par l'exécution d'un script dans lequel l'auteur a spécifié de façon explicite un envoi de message qu'il a lui-même défini. Toolbook permet également d'attacher à chaque objet des propriétés dont les valeurs sont modifiables et consultables.

Toolbook est présenté par la société Asymetrix comme un générateur d'application hypermédia orienté-objets. Les concepts et le langage qu'il propose le rapprochent en fait davantage des systèmes à objets prototypes (Cf. Chapitre 7), que des systèmes à objets classiques. La notion de classe y est en effet absente. Le fonctionnement du système repose sur une communication par messages entre objets instanciés ou copiés, et sur le principe de délégation de messages. Quand un objet ne peut traiter un message qu'il reçoit (c'est-à-dire qu'il ne possède pas le handler correspondant), il en délègue le traitement à son parent dans la hiérarchie des objets. Ce mécanisme permet d'assurer la mise en commun de traitements à plusieurs objets, et la concentration des traitements en un seul endroit pour les objets composites. Il a été exploité de façon systématique dans l'environnement MELISA.

Les relations entre l'approche Toolbook et les approches orientées objets sont décrites dans le rapport [PER 93a].

11.3 LES CONCEPTS ET LES FONCTIONNALITES

11.3.1 MELISA et le modèle MARS

Dans la partie 3 de ce document, nous avons défini le modèle MARS ainsi que les concepts à proposer dans chacun des espaces de travail indépendant. La version actuelle de l'environnement MELISA a été développée de façon parallèle à la définition du modèle MARS. De ce fait, certains concepts proposés dans la version actuelle de MELISA ne recoupent pas totalement ceux qui ont été décrits pour MARS. Examinons maintenant ces différences et les extensions prévues :

- MELISA propose la démarche de conception suivante :

⇒ spécification globale de l'application.

⇒ conception automatisée du modèle de simulation.

⇒ conception automatisée de la représentation.

⇒ mise en correspondance automatisée du modèle et de la représentation pour constituer une simulation exécutable.

⇒ utilisation de la simulation exécutable pour décrire l'état initial et l'état final du scénario.

⇒ adaptation et modification du scénario généré.

- MELISA propose à l'auteur les quatre espaces de travail de MARS. L'espace MODEL, l'espace SCENARIO, l'espace PRESENTATION se présentent sous la forme d'éléments du menu MELISA (Cf. Figure 11-3). La définition des ASSOCIATIONS (Links) est répartie en fonctionnalités intégrées dans des sous-menus.

- Dans l'espace MODEL, le système à simuler est représenté sous la forme d'un seul objet. Cet objet est décrit par un ensemble de propriétés et d'opérations. La dynamique du système est représentée non pas à l'aide d'un Statechart de Harel, mais à l'aide d'un simple diagramme états/transitions à plat. Nous ne pouvons donc dans cette version de MELISA éviter l'explosion combinatoire des états et des transitions, dès que les modèles deviennent complexes. L'intégration des StateCharts est prévue pour la version V2.0.

- Dans l'espace SCENARIO, la description d'un scénario comprend actuellement un état initial du système, l'objectif à atteindre et la réactivité associée. La mise en place de la structure permettant l'intégration d'étapes intermédiaires, telle que décrite au Chapitre 8, est prévue pour la version V1.0.

- Les Associations entre le modèle et la présentation sont spécifiées dans MELISA grâce à la définition de règles E.C.A. (cf. Chapitre 10). Il existe aujourd'hui dans la version actuelle de MELISA, une restriction sur les paramètres de l'action décrite dans la règle E.C.A. De façon théorique, la valeur de ces paramètres peut être dépendante de la valeur de n'importe quelle propriété de l'objet origine. Aujourd'hui, elle ne peut dépendre que de la valeur de la propriété déclenchante. Cette restriction sera levée dans la version V1.0 du logiciel.

11.3.2 Liste des fonctionnalités

Comme le montre la figure 11-3, la surcouche auteur MELISA se présente aujourd'hui comme un simple menu supplémentaire intégré à la barre de menu auteur Toolbook.

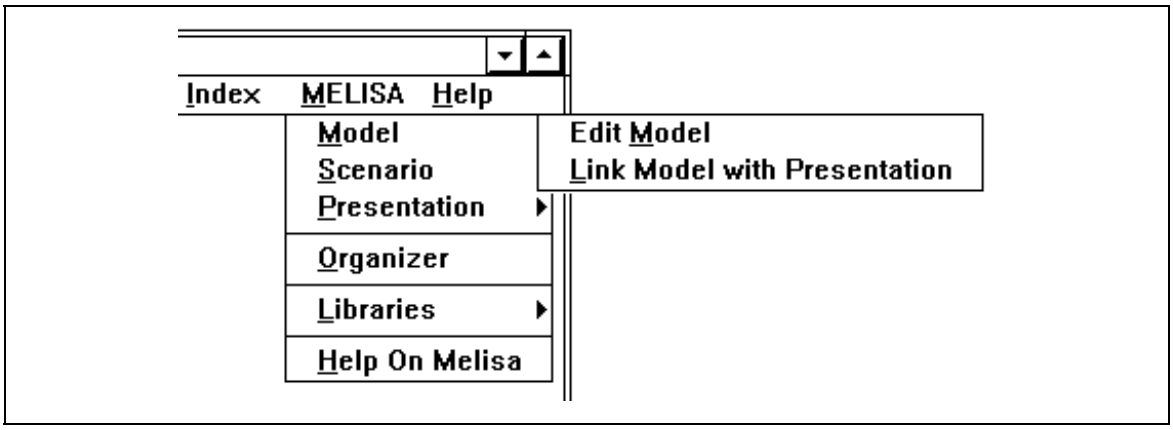


Figure 11-3 : Le menu auteur MELISA intégré à l'environnement Toolbook

Ce menu MELISA propose un ensemble de fonctionnalités réparties en sous-menus. Décrivons brièvement les fonctionnalités proposées.

- Sous-menu **MODEL**

  ⇒ Item **Edit Model**. Ce choix permet de définir ou de modifier le modèle de simulation. En l'activant, on peut définir les propriétés et opérations du modèle, définir et tester la dynamique, spécifier les connecteurs qui permettent la coopération du modèle.

  ⇒ Item **Link Model with Presentation**. Ce choix permet de gérer les associations entre le modèle et la représentation. En l'activant, on peut, par exemple, établir un lien entre une propriété déclenchante d'un objet de présentation et une opération du modèle.

- Sous-menu **SCENARIO**

  ⇒ Item **Create Scenario.** Permet la création automatique d'un scénario. Cette création se fait en deux temps : (1) enregistrement de l'état initial et (2) enregistrement de l'état final correspondant à l'objectif.

  ⇒ Item **Edit Scenario**. Permet de spécifier la réactivité et de modifier interactivement certaines caractéristiques du scénario.

- Sous-menu **PRESENTATION**

  ⇒ Item **Import Object**. Permet l'importation d'objets de représentation. Ces objets peuvent être passifs, déclencheurs, percepteurs ou déclencheurs-percepteurs (Cf. Chapitre 9). Ils sont accessibles à travers des bibliothèques les regroupant par catégories.

  ⇒ Item **Edit Object**. Permet la création et la modification d'objets d'interaction. Un objet d'interaction est créé de la même façon qu'un objet du modèle de simulation. On dispose, en plus, de possibilités de spécifier les interactions en entrée et en sortie (Cf. Chapitre 9).

⇒ Item **Replace an Object by another**. Permet de modifier l'apparence externe d'un objet sans en modifier le comportement. Par exemple, en phase de mise au point finale d'une application, ce choix permet de modifier l'apparence d'un objet d'interaction en remplaçant un graphisme élémentaire par une image de qualité photographique.

- Sous-menu **ORGANIZER.** Ce fonctionnalité n'est pas spécifique à la conception de l'application de simulation pédagogique. Elle a pour but d'améliorer certains aspects de l'interface Toolbook (vision générale de l'application, accès optimisé au script, etc.).
- Sous-menu **LIBRARIES**

⇒ Item **Export to user library.** Permet de classer un objet d'interaction que l'on a créé ou modifié dans une bibliothèque utilisateur.

⇒ Item **Delete object from library.** Permet de supprimer un objet d'interaction d'une bibliothèque utilisateur.

11.4 PRESENTATION DE L'ENVIRONNEMENT PAR UN EXEMPLE

Après cette description succincte des fonctionnalités, nous nous proposons maintenant de présenter l'environnement MELISA de façon plus détaillée à travers un exemple.

11.4.1 L'objectif de la simulation pédagogique

L'exemple que nous présentons ici est relatif à l'amorçage (boot) d'un ordinateur de type IBM PC lors de son démarrage. Sur certaines machines, il existe un dispositif matériel (un cavalier à deux positions situé sur la carte mère) permettant d'indiquer le comportement au démarrage :

- En position "Disque Maître", l'amorçage du système s'effectue toujours depuis le disque dur, qu'une disquette soit ou non présente dans le lecteur (Cas 1).
- En position "Disque Non Maître", trois possibilités peuvent se produire :

⇒ Il n'y a pas de disquette présente dans le lecteur : l'amorçage du système s'effectue depuis le disque dur (Cas 2).

⇒ La disquette introduite dans le lecteur est une disquette système : l'amorçage du système s'effectue depuis cette disquette (Cas 3).

⇒ La disquette introduite dans le lecteur n'est pas une disquette système : l'amorçage du système ne peut s'effectuer, et un message d'erreur apparaît à

l'écran (Cas 4).

La présence de ce type de cavalier n'est pas systématique et même plutôt rare pour ce type de machines. En l'absence de ce dispositif, l'amorçage s'effectue en mode "Disque Non Maître" (Cas 2, 3 et 4).

Il peut découler de la présence optionnelle de ce cavalier, un risque de mauvaise appréciation de la part d'un technicien de maintenance. Constatant qu'après l'introduction d'une disquette système dans le lecteur, l'amorçage du système s'effectue sur le disque dur, le technicien devrait considérer deux possibilités :

1. Il existe peut-être un cavalier placé en position "Disque Maître".
2. Le lecteur de disquette ne fonctionne plus. Il y a peut-être un problème de contrôleur.

S'il ne vérifie pas le point 1, et que la machine comporte effectivement un cavalier placé en position "Disque Maître", le technicien va sans doute changer le contrôleur de disquette tout à fait inutilement.

Pour éviter ce problème et pousser un technicien de maintenance à s'assurer de la présence du cavalier, on se propose de créer une application de simulation pédagogique.

11.4.2 Le processus de conception

Nous distinguons 5 étapes dans le processus MELISA :

Le première étape consiste à préciser les objectifs du logiciel à développer, et à spécifier de façon globale le Modèle de simulation et la Représentation.

Les étapes de conception détaillée du Modèle et de la Représentation peuvent être menées parallèlement avec une validation croisée des résultats intermédiaires.

L'étape de mise en association du Modèle et de la Représentation permet d'obtenir une simulation exécutable.

L'étape de définition du scénario se fait d'abord en créant un scénario de façon automatique par démonstration, puis en modifiant et adaptant le résultat obtenu.

Nous allons maintenant détailler chacune de ces étapes, en montrant directement les manipulations effectuées dans l'environnement MELISA.

11.4.3 Spécification globale de l'application

L'application de simulation pédagogique que nous désirons développer a pour but d'inciter un technicien de maintenance à s'assurer de la présence et de la position du cavalier sur un ordinateur, en cas de non-amorçage sur une disquette système. Pour ce faire, on va proposer à l'élève un exercice. Pour des raisons pédagogiques, on va lui proposer une situation et un objectif qui ne vont par l'orienter directement vers la bonne solution. On propose le scénario pédagogique suivant :

- L'état initial est le suivant :

  ⇒ L'élève se trouve devant un ordinateur éteint et dont le lecteur de disquette est vide. Cet ordinateur est en état de fonctionnement et utilise la version 6.0 du système DOS. Il possède un cavalier d'amorçage de système en position "Disque Maître".

  ⇒ L'élève dispose de deux disquettes sur la table. L'une est une disquette "Application" et l'autre est une disquette système contenant la version 2.0 de DOS.

  ⇒ L'élève peut avoir accès à la carte-mère et modifier la position du cavalier d'amorçage.

- La consigne que l'on donne à l'élève est la suivante :
  Vous arrivez chez un client qui veut faire fonctionner une application qui tourne uniquement sous la version 2.0 de DOS. Vous disposez d'une temps maximum de cinq minutes pour amorcer l'ordinateur avec cette version de système.

L'objectif de cet exercice est donc de tester la réaction du technicien face à une situation imprévue. S'il opère de façon normale, sans vérifier la présence du cavalier, il ne disposera que d'un temps très court pour réagir, modifier la position du cavalier et réamorcer le système.

Nous pouvons donc maintenant spécifier de façon globale ce que devront être les composants Modèle et Représentation de l'application.

Le *modèle de simulation* devra permettre d'exprimer les point suivants :

- L'ordinateur peut être ou non raccordé à une prise de courant.
- L'ordinateur peut être allumé ou éteint.
- Le cavalier peut être en position "Disque Maître" ou "Disque Non-Maître".
- La phase d'amorçage de l'ordinateur devra être simulée.
- Le lecteur de disquette peut être vide, occupé par une disquette système ou occupé par une disquette non-système.

Le *représentation* devra permettre à l'élève de :

- Brancher ou débrancher l'ordinateur.
- Allumer ou éteindre l'ordinateur.
- Poser un disquette dans le lecteur ou sur la table.
- Accéder à la carte-mère en ouvrant le capot.
- Refermer le capot.
- Modifier la position du cavalier.
- Visualiser les messages du système sur un écran.

Ces spécifications succinctes vont servir de base aux tâches de conception détaillée du modèle de simulation et de la représentation.

11.4.4 Conception du modèle de simulation

Pour définir le modèle de simulation de notre ordinateur il faut choisir l'item Edit Model. La fenêtre suivante s'affiche alors :

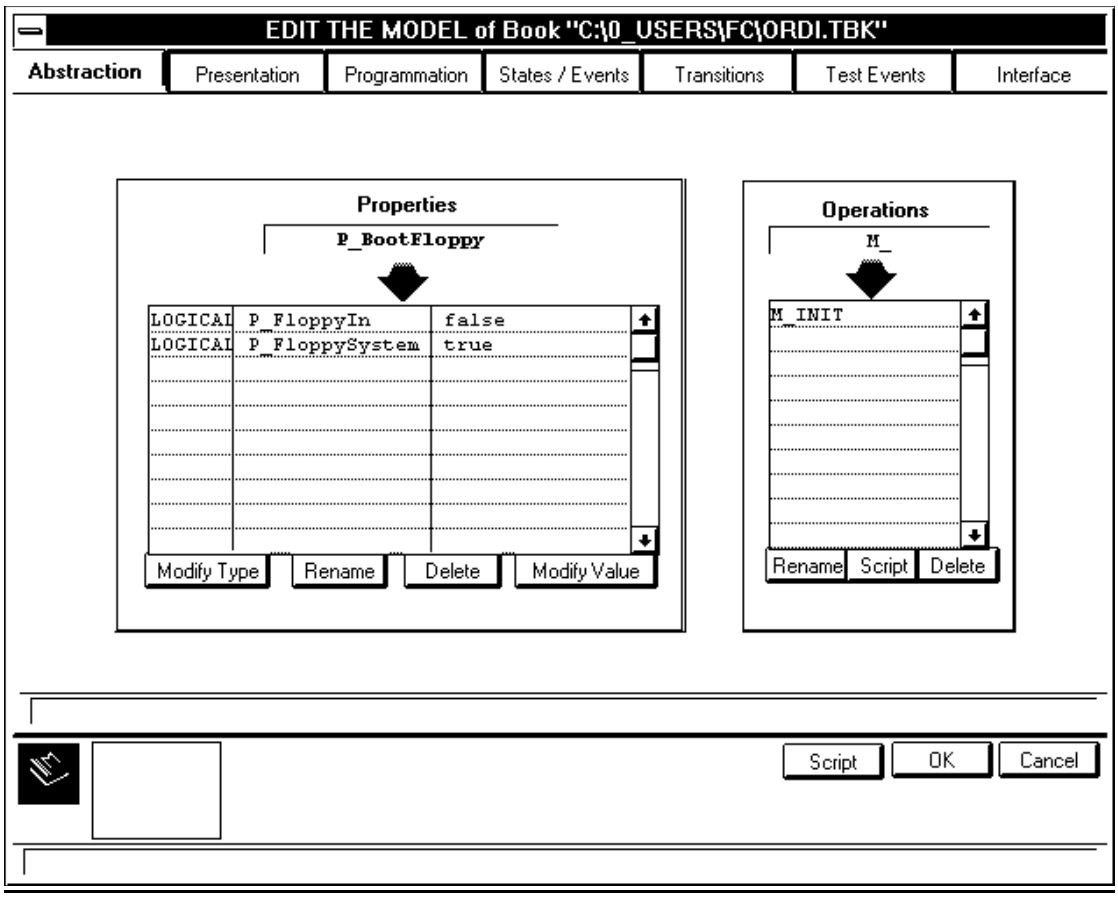


Figure 11-4 : La fenêtre EDIT MODEL

Cette fenêtre propose 7 onglets, dont 6 seulement sont activables. Le choix **Presentation** est réservé aux objets d'interaction qui, nous l'avons vu, peuvent être créés ou modifiés de façon identique à ceux du modèle.

L'onglet **Abstraction** permet de définir les propriétés et opérations du modèle.

L'onglet **Programmation** permet de définir des scripts complexes et d'avoir recours à toute la richesse du langage de programmation OpenScript. Il ne sera pas utilisé dans notre exemple.

Les onglets **States/Events**, **Transitions** et **Test Events** permettent de définir et tester le diagramme d'état représentant la dynamique du Modèle.

L'onglet **Interface** permet de définir les connecteurs qui vont permettre de lier le Modèle à la Représentation.

11.4.4.1 Définition des propriétés et opérations du modèle

En cliquant sur l'onglet "Abstraction" on peut par manipulation directe, entrer des noms de propriétés dans la zone **Properties.** Nous avons à définir trois propriétés :

- la propriété *P_FloppyIn* de type *LOGICAL* qui détermine s'il existe ou non une disquette dans le lecteur.
- la propriété *P_FloppySystem* de type *LOGICAL* qui détermine si la disquette introduite est ou non une disquette système.
- la propriété *P_BootFloppy* de type *LOGICAL* qui détermine si le cavalier d'amorçage est dans la position "Disque Maître" ou non.

Après saisie des deux premières propriétés, on obtient le résultat affiché sur la figure 11-4. La saisie de la propriété *BootFloppy* s'opère en deux temps comme le montre la figure 11-5.

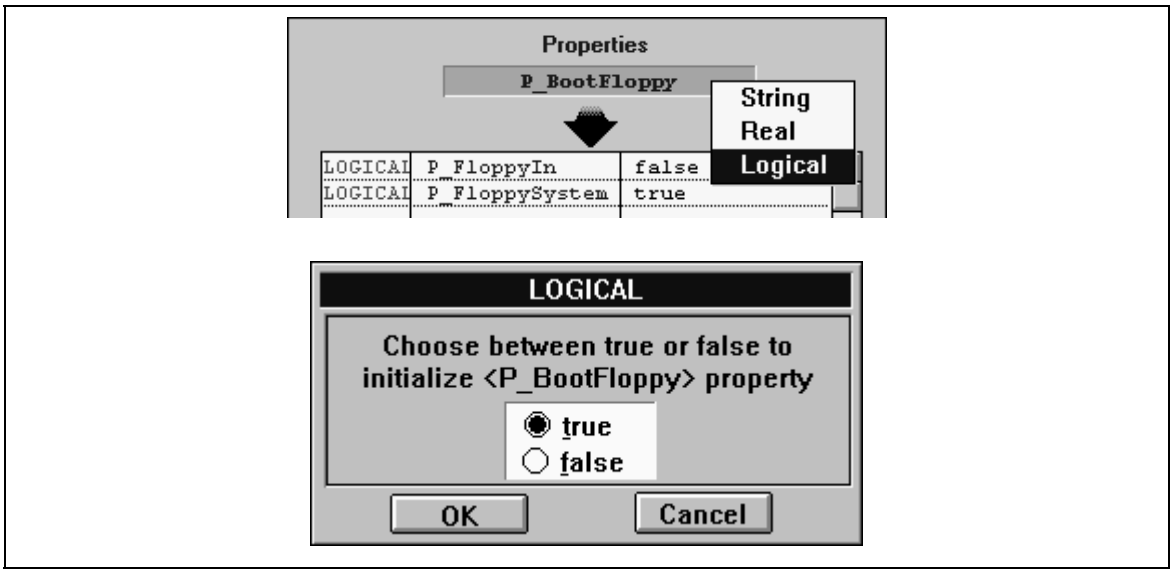


Figure 11-5 : Définition d'une propriété, de son type et de sa valeur initiale

La définition des opérations du modèle s'opère de la même manière dans la zone **Operations.** Nous avons dans notre cas trois opérations à définir : *M_SetFloppyIn*, *M_SetFloppySys* et *M_SetBootFloppy*. A chaque saisie, l'environnement invite à entrer le code correspondant à l'opération en langage OpenScript. La syntaxe de ce code ainsi que certaines règles d'écriture sont vérifiés par MELISA.

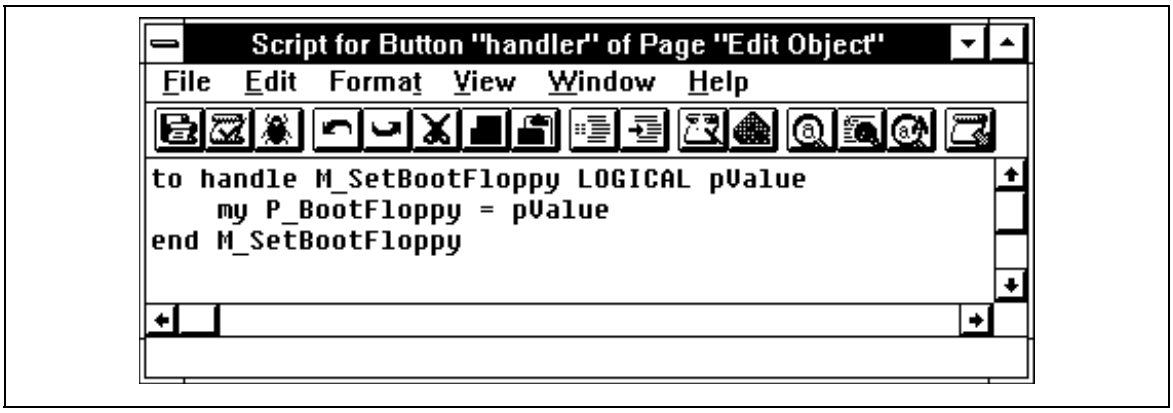


Figure 11-6 : Exemple de code OpenScript

11.4.4.2 Définition de la dynamique du modèle

En cliquant sur l'onglet **States/Events**, on accède à une fenêtre permettant de saisir les noms des états et des événements du diagramme d'état ainsi que l'état initial du diagramme. On obtient ainsi :

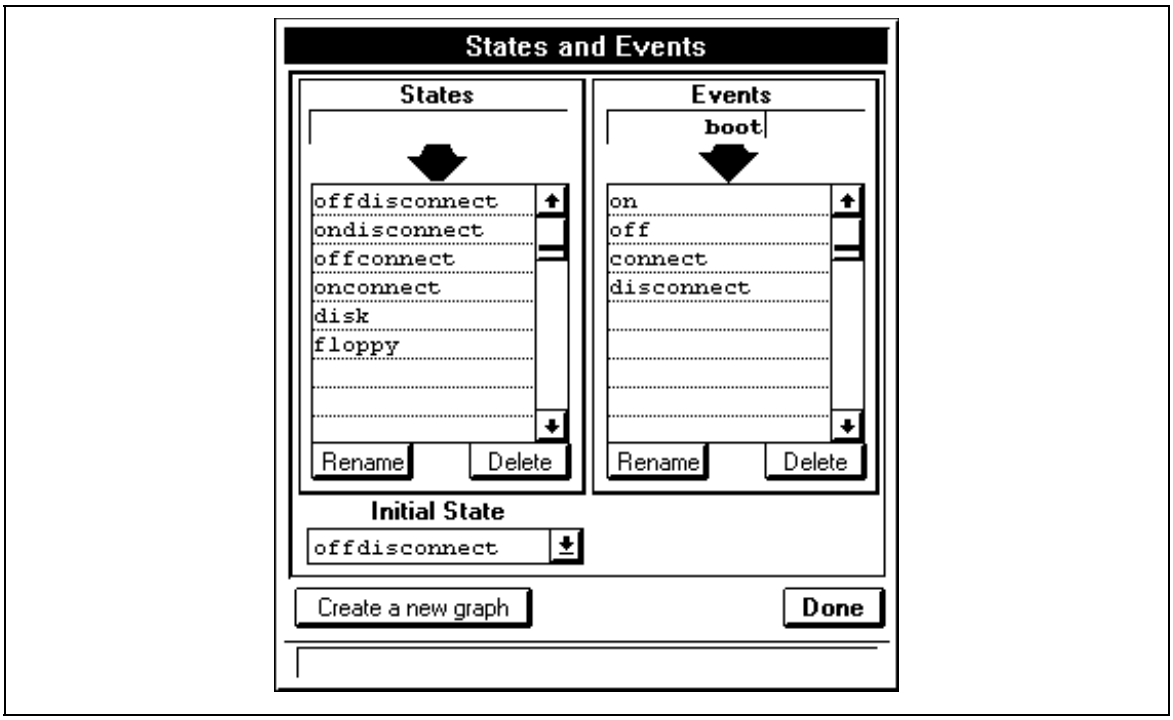


Figure 11-7 : Saisie du nom des états et des événements

L'étape suivante consiste à définir les transitions (onglet **Transitions**) à l'aide d'un éditeur interactif permettant de dessiner les états définis et de les relier par manipulation directe par des arcs orientés étiquetés par le nom des événements. On obtient après ces manipulations le résultat suivant :

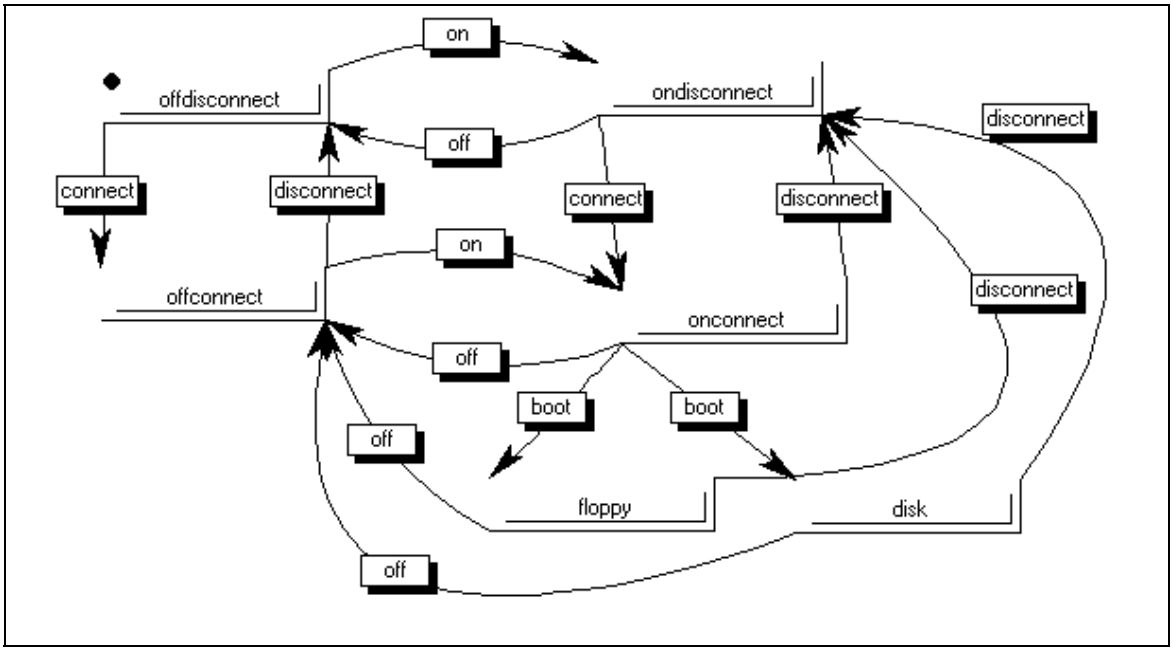


Figure 11-8 : Editeur interactif de diagramme d'état

On peut constater sur ce graphe qu'il existe des transitions caractérisées par le même état d'origine et le même événement (c'est le cas par exemple pour l'état *onconnect* et l'événement *boot)*. MELISA fournit la possibilité de définir dans ce cas des transitions gardées. On peut associer une condition en paramétrant de façon interactive la transition allant depuis l'état *onconnect* à l'état *floppy*. Dans la version actuelle, cette condition (liée au positionnement du cavalier et à la présence d'une disquette système dans le lecteur) est saisie sous la forme de code OpenScript.

MELISA permet également de définir des transitions automatiques. Par exemple, la transition précédente (depuis l'état *onconnect* à l'état *floppy*) doit être exécutée de façon automatique au bout d'un certain laps de temps. Cette information est saisie en paramétrant de façon interactive l'état *onconnect*.

11.4.4.3 Test de la dynamique du modèle

Une fois que le diagramme d'état a été complètement spécifié, il est possible de le tester en sélectionnant l'onglet **Test Events**. Cet item permet d'entrer interactivement une liste d'événements datés (Cf. Fig. 11-9) et d'évaluer l'application de cette liste sur le diagramme d'états. La visualisation dynamique de ce test est obtenue par une animation en temps réel du diagramme présenté sur la figure 11-8 : l'état actif est alors affiché dans une couleur différente de celle des autres états.

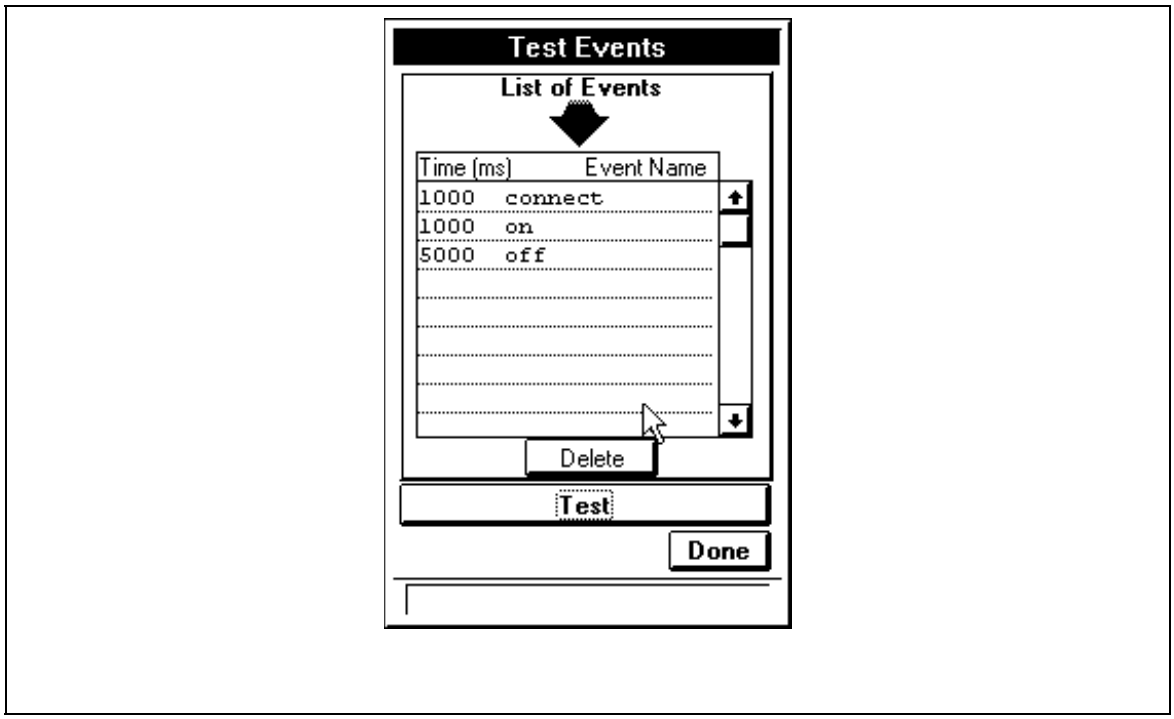


Figure 11-9 : Définition d'une liste d'événements à tester

11.4.4.4 Résultat obtenu à la fin de l'étape

A la fin de l'étape de modélisation, on a donc défini un modèle de simulation avec :

- un ensemble de propriétés typées et possédant chacune une valeur par défaut.
- un ensemble d'opérations dont le code a été entré en langage OpenScript.
- un diagramme d'états représentant la dynamique du système. Ce diagramme a pu être testé.

11.4.5 Conception de la représentation

La seconde étape de conception concerne la représentation. Cette étape peut se dérouler parallèlement à celle de conception du modèle de simulation.

MELISA fournit à l'auteur un ensemble de bibliothèques d'objets préprogrammés. Quand on sélectionne l'item **Import Objet**, la fenêtre suivante est affichée :

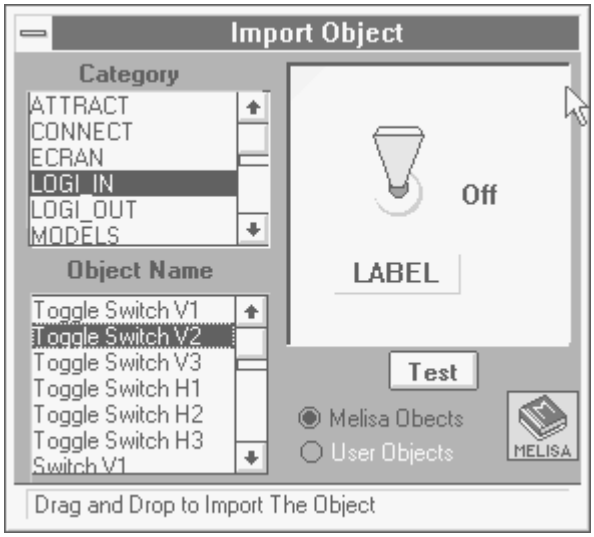


Figure 11-10 : Importation des objets de présentation

On peut choisir dans cette fenêtre des objets répartis par catégories. Par exemple, l'objet affiché *Toggle Switch V2* appartient à la catégorie *LOGI_IN* (LOGICAL INPUT). Cette catégorie regroupe un ensemble d'objets déclencheurs caractérisés par deux états (*true* ou *false*).

Le concepteur va donc chercher dans les bibliothèques, les objets dont le comportement se rapproche le plus de ceux qu'il veut proposer dans son interface.

Dans notre cas particulier, il va par exemple choisir :

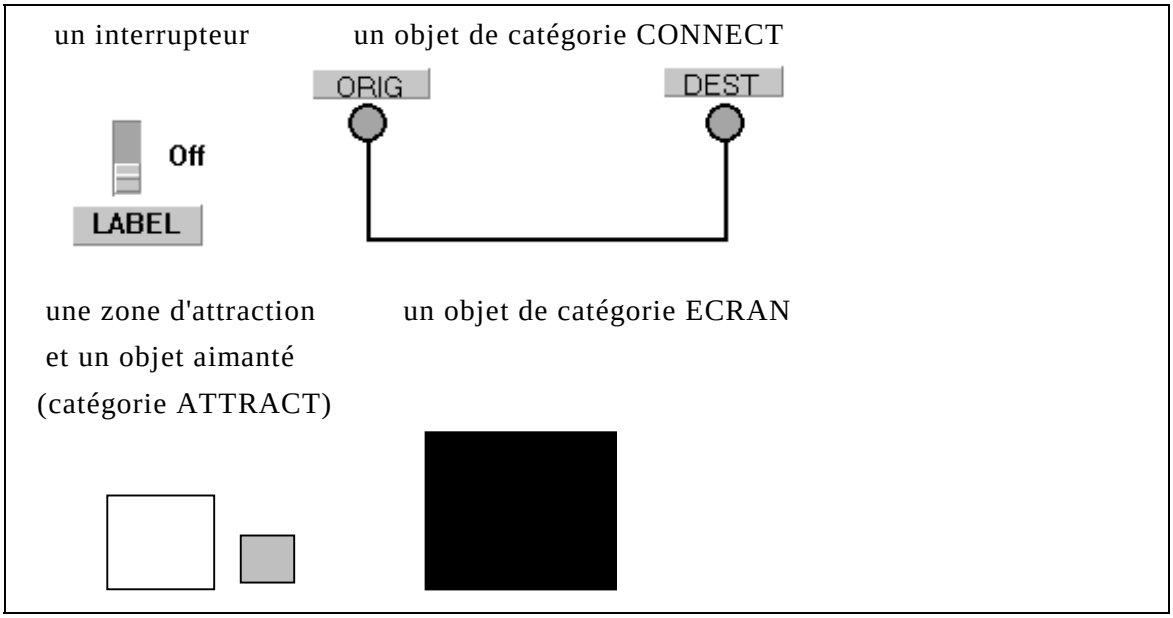


Figure 11-11 : Exemple d'objet importés

Il va donc placer ces objets sur son espace de travail, c'est-à-dire sur une page écran. Dès lors, ces objets sont opérationnels.

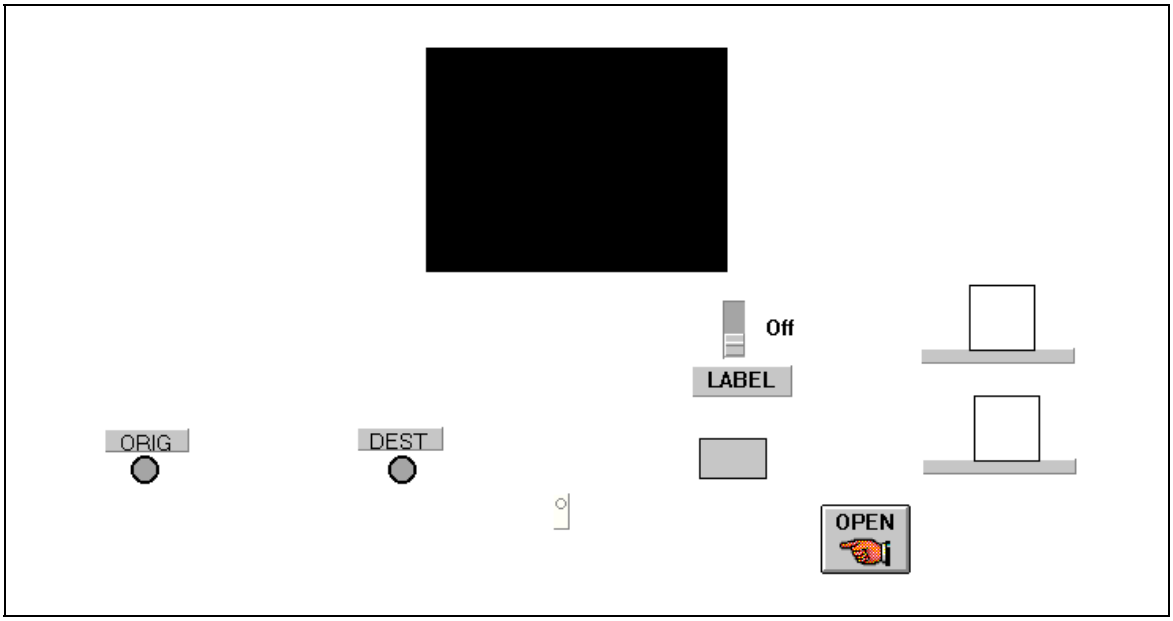


Figure 11-12 : Placement des objets sur la page

L'étape suivante consiste à personnaliser les objets. En particulier, l'item *Replace an object by another* permet le remplacement de composants graphiques d'un objet sans en modifier le comportement. C'est le cas, par exemple, de l'objet aimanté qui sera remplacé par un graphisme représentant une disquette. On obtient donc, après

cette personnalisation :

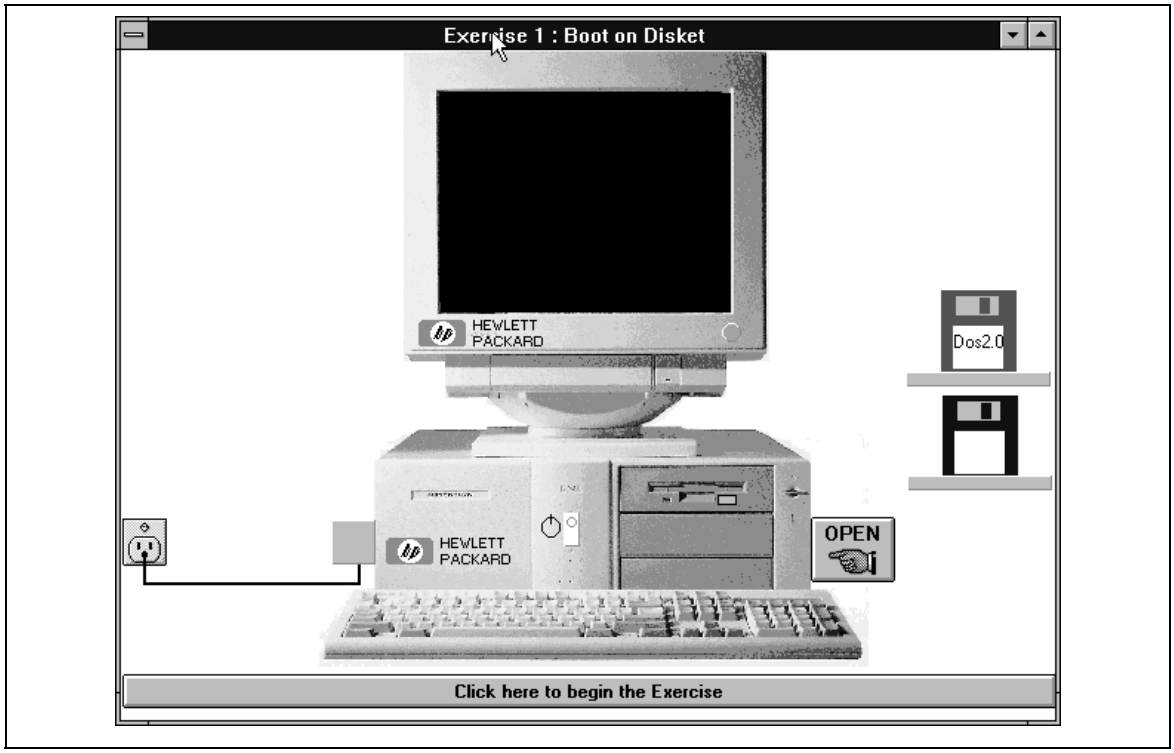


Figure 11-13: Personnalisation des objets de présentation

On dispose à la fin de cette étape d'une représentation non reliée avec le modèle de simulation. Le tâche suivante consiste donc à définir les associations entre le Modèle et la Représentation.

11.4.6 Définition des associations

L'étape de définition des associations requiert en premier lieu de préciser la visibilité du modèle (la visibilité des objets de présentation est prédéfinie). Pour cela, on dispose de l'onglet *Interface* de l'item *Edit Model* (Cf. Figure 11-14). Cet item permet de définir dans une boîte de dialogue :

- les opérations du modèle qui sont activables depuis l'extérieur (*Public Operations*).
- les propriétés déclenchantes (*Trigger Properties*) susceptibles d'être à l'origine d'une association de coopération.

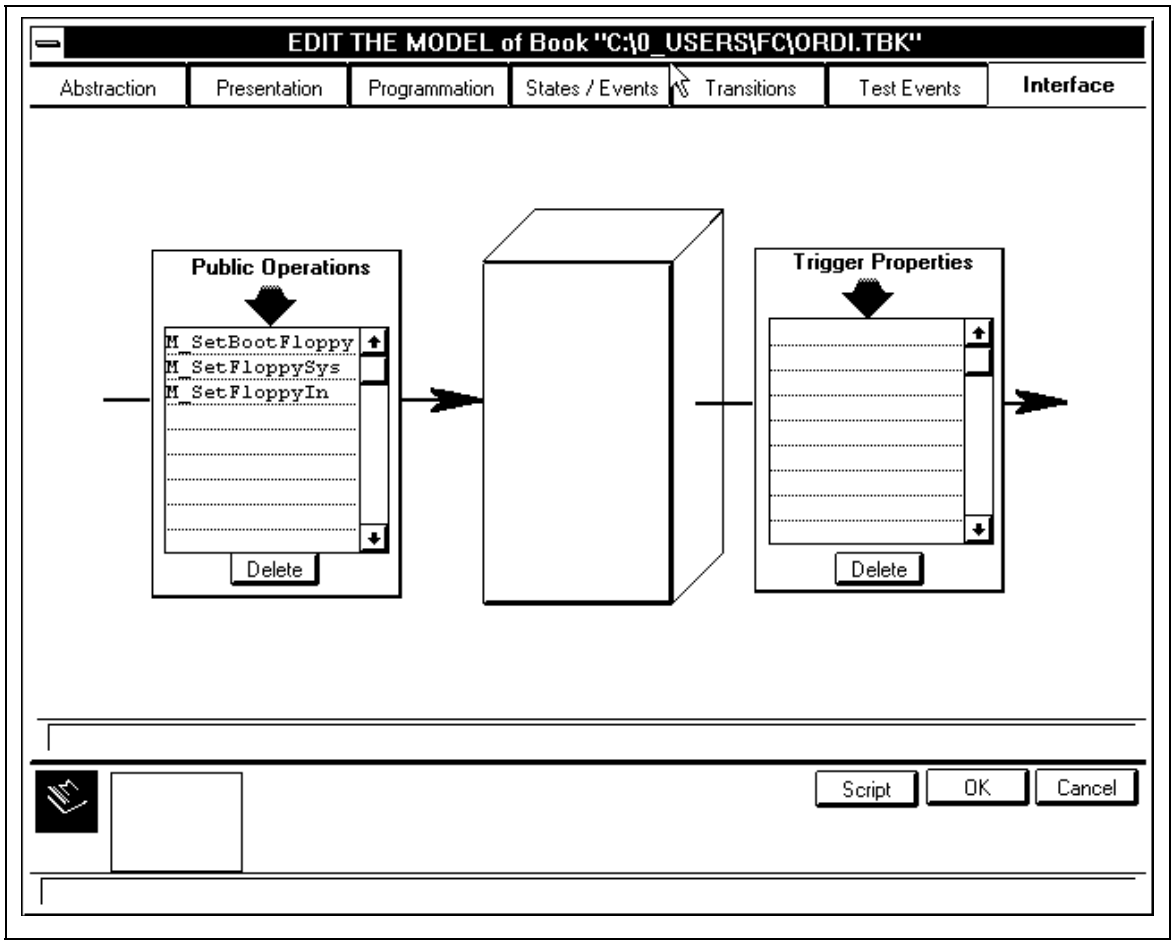


Figure 11-14 : Définition de la visibilité du modèle

Une fois cette opération effectuée, on peut accéder à la définition des associations grâce à l'item *Link Model with Representation.* Par exemple, le lien entre l'interrupteur ON/OFF de la machine et le modèle sera spécifié de la façon suivante :

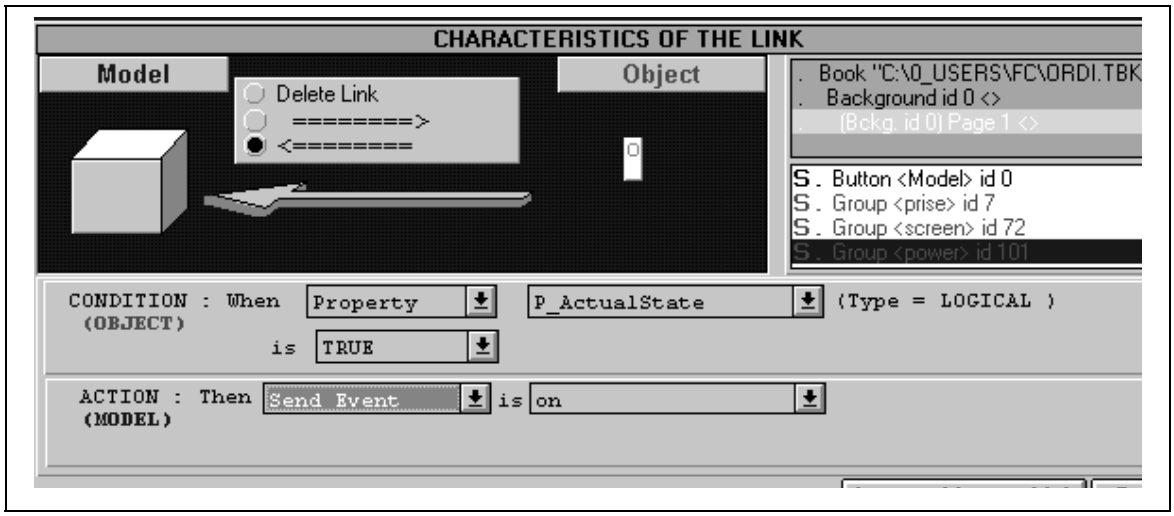


Figure 11-15 : La définition d'une association

Tout d'abord, l'auteur choisit un objet dans la liste des objets de représentation de l'application. Cet objet est alors visualisé, et l'auteur peut sélectionner le sens de son association en cliquant sur une flèche orientée. De façon automatique, MELISA remplit alors le contenu des zones *CONDITION* et *ACTION*.

Dans la partie *CONDITION*, le système propose de choisir une propriété déclenchante de l'objet origine (ici, *P_ActualState)*, ainsi qu'une contrainte sur les valeurs possibles de cette propriété (ici, la propriété *P_ActualState* doit être égale à *TRUE*).

Dans la partie *ACTION*, le système propose de choisir quelle opération doit être activée ou quel événement doit être envoyé à l'objet destination (ici, l'événement *on* du Modèle doit être activé).

Une fois tous les liens spécifiés, on obtient une liste d'associations que l'on peut gérer indépendamment (visualisation, suppression, modification). Ces associations sont opérationnelles et permettent d'obtenir une simulation directement exécutable.

11.4.7 Génération et adaptation du scénario

MELISA permet, comme nous l'avons dit, de générer un scénario de façon automatisée par démonstration. On dispose pour cela de l'item *Create Scenario* qui permet d'enregistrer, pendant l'exécution d'une simulation, un état initial (l'état dans lequel doit être placé l'apprenant au début de l'exercice) et un état final (l'objectif à atteindre par l'apprenant). Une fois ces opérations effectuées, on peut adapter le résultat produit grâce à l'item *Edit Scenario*, comme le montre la figure suivante.

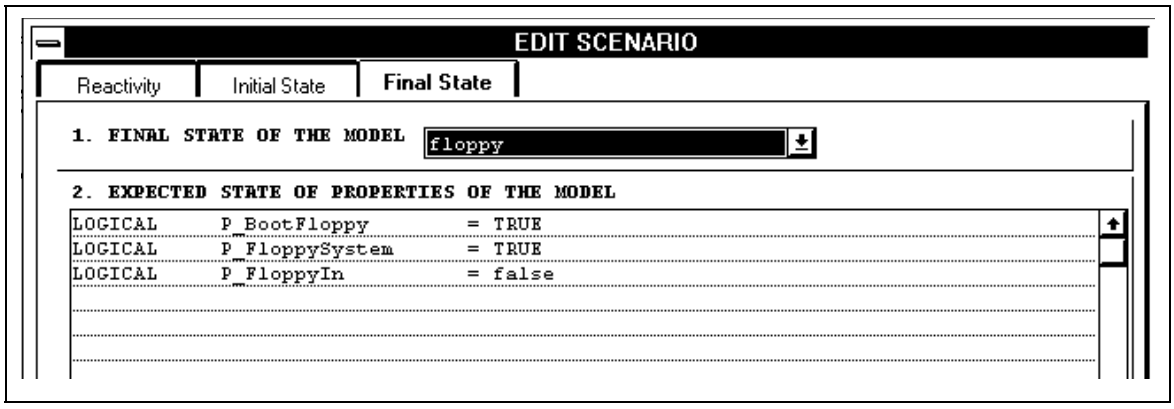


Figure 11-16 : La fenêtre Edit Scenario

Le concepteur peut modifier interactivement dans cette fenêtre les valeurs ou plages de valeurs acceptées correspondant à l'état final souhaité. De la même façon, il peut modifier les caractéristiques précises de l'état initial.

Enfin, l'auteur peut préciser la réactivité associée au scénario grâce à l'onglet *Reactivity*. Ce choix lui permet de fixer les informations textuelles, sonores ou autres (exécution d'une séquence vidéo, par exemple) à fournir à l'apprenant avant l'exécution du scénario, en cas de réussite ou d'échec.

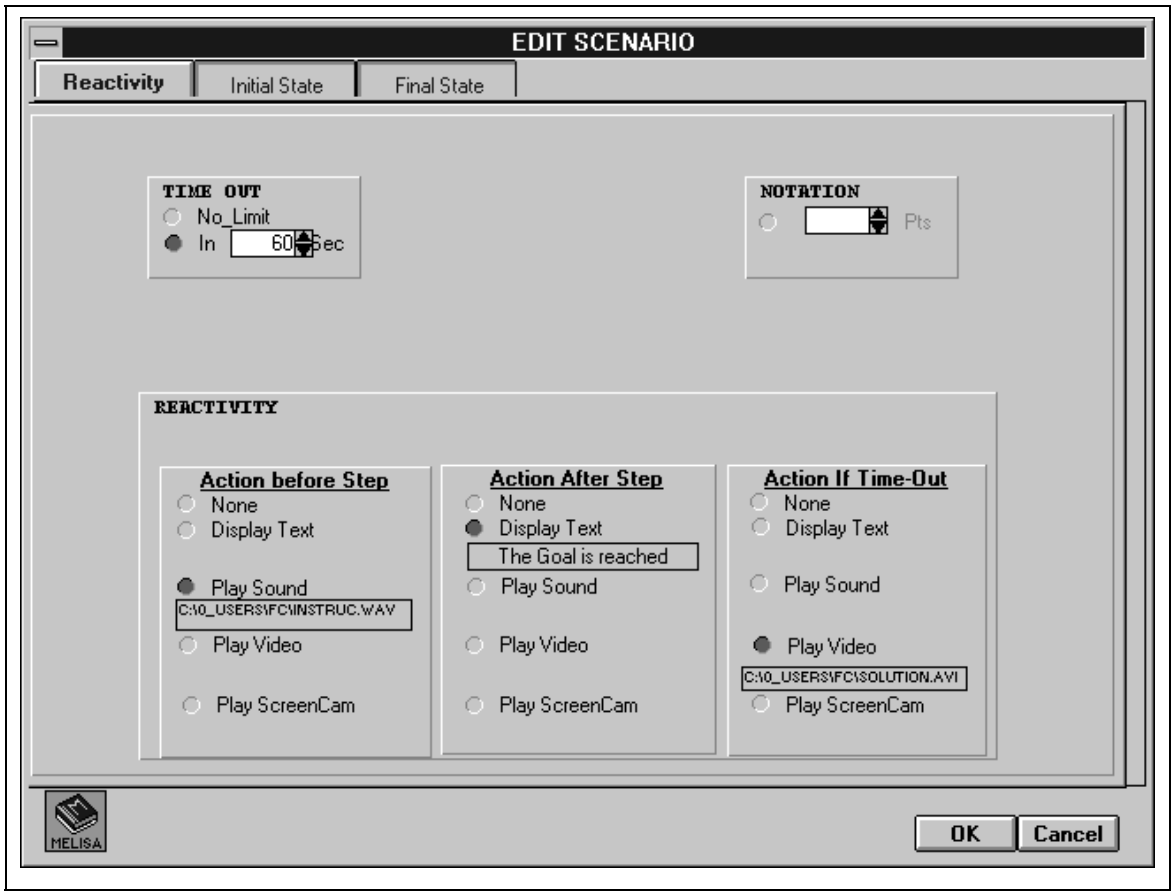


Figure 11-15 : La réactivité du Scénario

Après avoir spécifié l'état initial, l'état final et la réactivité, l'auteur peut tester directement le résultat de sa spécification, en cliquant sur un bouton lié au scénario.

11.4.8 Mise au point finale

A la fin des étapes précédentes, l'auteur dispose d'une application de simulation pédagogique exécutable. On peut estimer qu'une telle application peut être complètement développée et testée en une demi-journée, pour un auteur connaissant l'environnement. L'effort de programmation textuel se réduit à l'écriture relativement répétitive d'expressions simples en langage OpenScript.

L'auteur peut encore faire une mise au point finale en effectuant un réglage fin de son application. Il peut, en particulier :

- Modifier certains aspects de présentation, en remplaçant par exemple, certains graphismes par des images de qualité photographique.
- Régler de façon fine le scénario pédagogique, en ajustant les délais, en modifiant les messages à délivrer, etc.
- Régler certains paramètres d'exécution du modèle de simulation, à condition que ceux-ci ne soient pas exploités de façon explicite par le Scénario ou la Représentation.

11.5 LE FONCTIONNEMENT INTERNE

Il nous a paru intéressant d'illustrer succinctement le fonctionnement interne du système MELISA à l'aide de certains aspects de l'exemple précédent. Nous allons détailler les aspects suivants :

- La création du modèle et l'ajout d'un propriété et d'une opération au modèle.
- L'importation d'un objet de présentation.
- La modification de l'aspect d'un objet.
- La définition d'une association.
- L'exécution d'une association.

La création du modèle

Toolbook, comme nous l'avons vu, est un système à objets. Chaque objet est caractérisé par un certain nombre de propriétés prédéfinies (propriétés graphiques ou d'identification). L'auteur peut définir de nouvelles propriétés et les opérations qui vont permettre de traiter les messages.

Dans une application spécifique (un livre système Toolbook non accessible par l'auteur), nous avons défini un objet de référence auquel nous avons ajouté toutes les propriétés et opérations permettant de caractériser un objet de type modèle (matrice représentant le graphe d'états, liste des opérations publiques, des propriétés déclenchantes, etc.).

A chaque fois que l'auteur crée un modèle dans sa propre application, cet objet est en fait dupliqué par clonage (commande *CopyObject*) depuis l'application spécifique vers l'application en cours de développement. Cet objet Toolbook est rendu inaccessible à l'auteur et la seule visibilité qu'il pourra en avoir sera celle proposée par MELISA.

Quand le concepteur ajoute une propriété au modèle (Cf. Figure 11-17), cet ajout est répercuté sur l'objet dupliqué représentant le Modèle de son application.

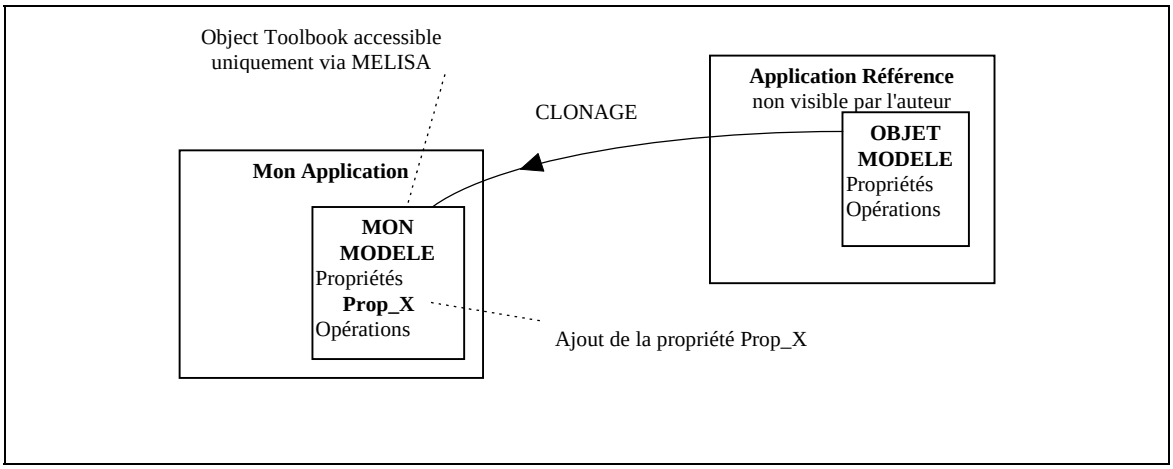


Figure 11-17 : Création d'un modèle

L'importation d'un objet de présentation

L'importation d'un objet de présentation (item *Import Object*) se fait également par clonage distant depuis des bibliothèques contenant les objets de référence. On peut noter que les clones produits gardent la référence de leur origine. Ce mécanisme permet de concentrer certains traitements qui doivent rester communs à tous les clones issus du même objet de référence. La figure 11-18 montre, par exemple, le cas du traitement de clic avec le bouton droit (message *rightButtonClick*) sur un objet de présentation. Par convention adoptée dans MELISA, ce clic a pour effet de faire apparaître un menu local de paramétrage de l'objet. Décrivons précisément la séquence des opérations :

1. L'objet de référence est dupliqué par clonage dans l'application destination. Le clone garde la référence de l'objet dont il est issu (*P_RefOrigin*). Il est à noter que ni l'objet de référence, ni par conséquent son clone, ne contiennent de handler traitant le message *rightButtonClick*. Dès que la copie est effectuée, le clone vient s'insérer automatiquement dans la hiérarchie des objets Toolbook.
2. L'utilisateur clique sur l'objet avec le bouton droit. Le message *rightButtonClick* est alors envoyé de façon automatique par Toolbook à l'objet *Clone*.
3. Le script de l'objet *Clone* ne comportant pas le handler correspondant au message *rightButtonClick*, le message est alors transmis à son parent. Les ascendants successifs ne traitant pas non plus le message, celui-ci remonte jusqu'au sommet de la hiérarchie.

4. Au sommet de la hiérarchie, se trouve un livre système défini par MELISA, partagé par toutes les applications. Celui-ci contient un handler *rightButtonClick* dont le script correspondant est exécuté. Ce script teste si l'objet qui a reçu en premier le message (*target*) a été obtenu par clonage (*P_RefOrigin of target <> NULL*). Dans ce cas, il envoie le message *localMenu* à l'objet dont est issu le clone avec pour paramètre la référence du clone (*send localMenu (target) to (P_RefOrigin of target)*).
5. Le message *localMenu* est reçu par l'objet de référence qui ne contient pas de handler correspondant. Le message est alors transmis à la hiérarchie.
6. Le parent de l'objet de référence possède un script permettant de traiter le message *localMenu*. Le traitement correspondant sur l'objet *Clone* dont la référence a été passée en paramètre peut alors être effectué.

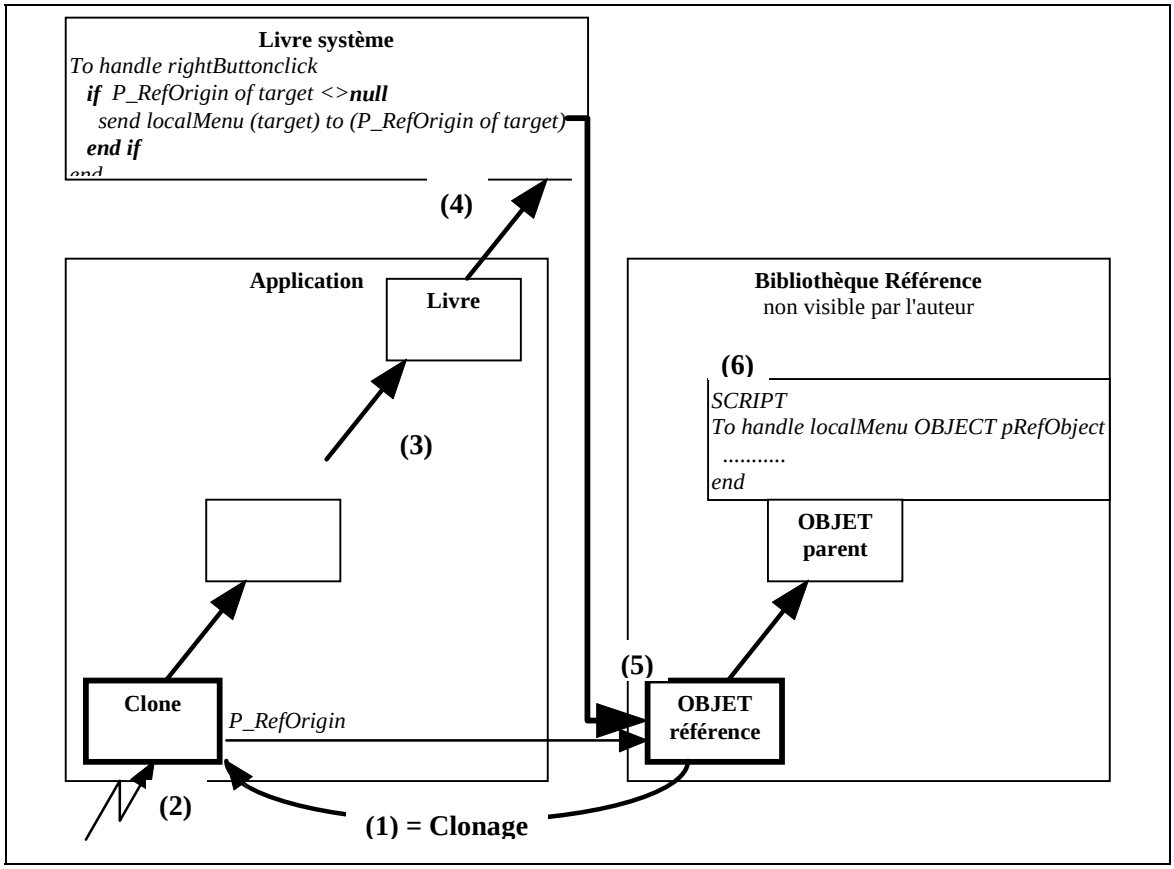


Figure 11-11-18 : L'activation d'un comportement générique

Cette solution peut paraître complexe mais elle présente les avantages suivants :

- Le code représentant le comportement commun n'est pas dupliqué dans tous les clones. Ceci allège considérablement la taille des applications produites.
- Les traitements sensibles peuvent être localisés dans des applications où l'auteur

n'a pas accès en écriture (les livres système, les bibliothèques de référence). Cette précaution permet d'assurer un certain niveau de sécurité à l'environnement. Nous situant dans un environnement interprété, la modification du script de référence (correction d'erreur, évolution), sera automatiquement répercutée sur tous les clones déjà existant issus de l'objet correspondant.

- Le traitement générique peut toujours être spécialisé au niveau d'un objet précis par l'écriture du handler correspondant. Dans notre cas, l'écriture d'un handler *rightButtonClick* dans le script de l'objet *Clone* suffira à désactiver la mécanique précédente.

La modification de l'aspect d'un objet d'interaction

MELISA utilise également le mécanisme de délégation de message pour permettre la modification de l'aspect externe d'un objet d'interaction sans en modifier le comportement (item *Replace an Object*). Prenons, par exemple, le cas d'un interrupteur à deux états (ON et OFF). Cet objet d'interaction est un groupe composé de deux objets Toolbook *ON* et *OFF*. Le comportement est concentré au niveau du script du groupe. Cette solution permet, par exemple, le remplacement de l'objet *OFF* par un autre objet de présentation *newOFF*. Ce choix entraîne l'insertion de l'objet *newOFF* dans le groupe, la recopie des valeurs des propriétés non-graphiques (y compris le nom et le script s'il existe) de l'objet *OFF* vers celles de l'objet *newOFF*, et pour terminer la suppression de l'objet *OFF*.

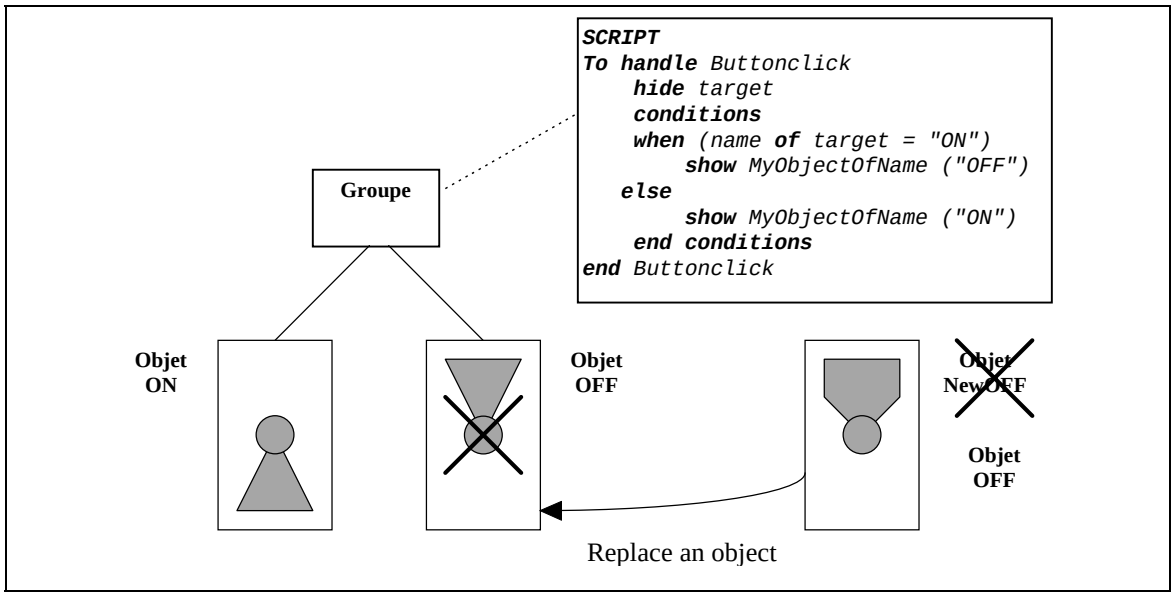


Figure 11-19 : Remplacement d'un objet par un autre

Cette solution garantit la possibilité de mettre au point les aspects de détail de présentation d'une application interactive après en avoir précisé le fonctionnement, et

sans risque de détruire les comportements déjà définis.

La définition d'une association entre le modèle et un objet de présentation

Nous avons vu que grâce au choix "Interface", l'auteur pouvait spécifier pour chaque objet quelles étaient les propriétés déclenchantes et les opérations publiques. Chaque objet tient donc à jour deux listes *ListOfTriggerProperties* et *ListOfPublicOperations* décrivant cette interface.

Quand on définit une association, par exemple entre le bouton ON/OFF et le modèle, MELISA offre à l'utilisateur la possibilité de choisir la propriété déclenchante du bouton ON/OFF et une opération du modèle. Un fois que l'auteur a spécifié les conditions précises d'activation de l'association de coopération, les caractéristiques de cette association sont stockées dans une structure *ListOfAssociations* de l'objet origine (ici, le bouton ON/OFF).

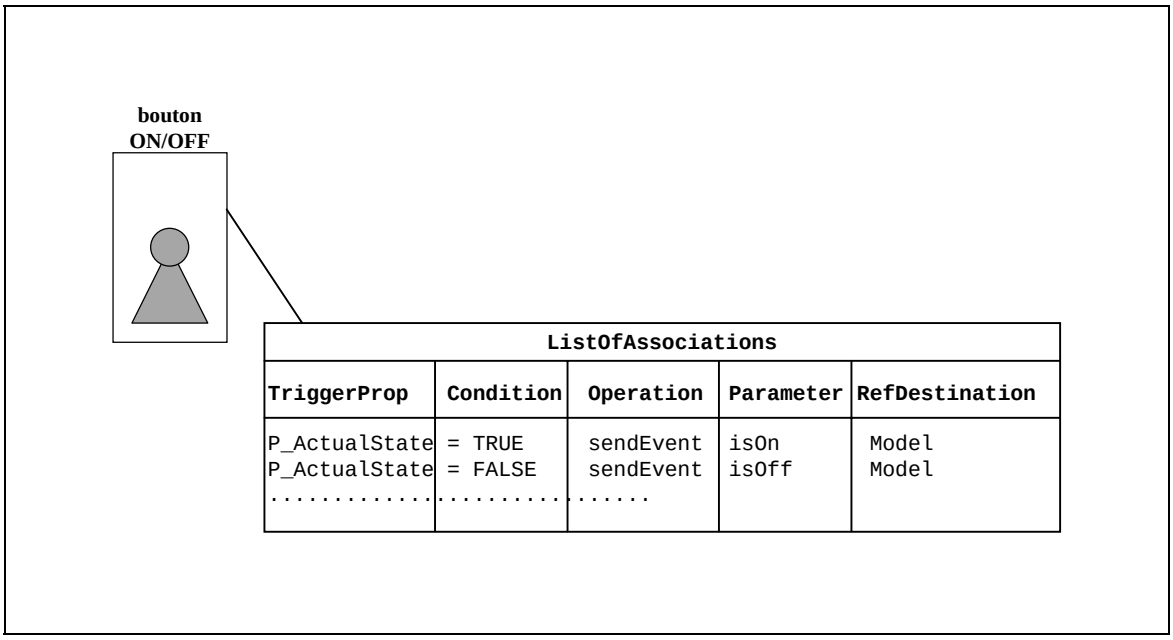


Figure 11-20 : Stockage des associations dans l'objet origine

L'exécution d'une association

L'exécution d'une association de MELISA est également basée sur le principe de délégation. Le système Toolbook propose un mécanisme permettant la génération d'un message à chaque fois qu'une propriété (ayant un nom précis) d'un objet est affectée. De la même façon que dans l'exemple décrit à la figure 11- 18, un script est défini au plus haut niveau, dans un livre système, permettant de traiter de façon générique la modification de la valeur d'un propriété.

```
                                    Livre système
to set P_ActualState to pValue
   if  P_ListOfAssociations of target <> null
        -- CHERCHER SI LA PROPRIETE P_ActualValue EST ORIGINE D'UNE
        -- ASSOCIATION
        ......
        -- SI OUI, VERIFIER SI LA CONDITION EST REMPLIE
        .....
        -- SI OUI, ENVOYER LE MESSAGE OperationName AVEC LA VALEUR CALCULEE
        -- DU PARAMETRE A L'OBJET RefDestination
        ...
   end if
end
```

Figure 11-21 : Un script d'exécution d'une association

Dans cet exemple, à chaque fois que la valeur de la propriété *P_ActualState* d'un objet X est modifiée, le script vérifie si cette propriété est définie comme propriété déclenchante dans la liste des associations de l'objet X. Si c'est le cas, et après avoir vérifié la condition précisée dans l'association trouvée (*P_ActualState* = TRUE), le script exécute l'action correspondante.

11.6 PREMIER BILAN DU PROJET MELISA

L'environnement MELISA en est aujourd'hui au stade du prototype opérationnel. De premiers retours ont été obtenus pendant les étapes de mise au point des maquettes. Il faut noter que nos interlocuteurs de la division TPEC ne sont pas des informaticiens, mais des experts en formation technique ayant une connaissance approfondie des environnements de production de didacticiels. Les échanges techniques que nous avons eu ont essentiellement concerné la définition de la visibilité "auteur", en tentant d'allier puissance d'expression et simplicité de mise en œuvre des concepts à proposer aux auteurs.

Ces échanges ont fortement guidé certains de nos choix. Citons, par exemple, les formalismes de représentation de la dynamique des modèles (StateCharts de Harel), ou le concept de scénario tel qu'il a été défini en fin du Chapitre 7. Ces échanges ont également permis la mise au point de procédés automatiques que nous n'avions pas imaginés, tels que la génération automatique de scénario par démonstration.

L'année 1996 sera consacrée à l'expérimentation en vraie grandeur de MELISA, dans des contextes de formation réels. Cette expérimentation connaîtra une nouvelle dimension avec l'intégration de notre équipe de recherche au sein du projet européen Ariadne auquel est également associé la division TPEC de Hewlett-Packard.

CONCLUSION GENERALE

CONTRIBUTION DE LA THESE

Aspects théoriques

Le résultat essentiel de ce travail est la définition du *modèle MARS*, *cadre conceptuel commun* qui permet de guider les différents intervenants dans le processus de développement de simulations pédagogiques, que ce soit dans l'optique d'une production "lourde" nécessitant une équipe de spécialistes ou dans celle d'une production "légère" assurée par un auteur isolé. MARS permet à un concepteur de circonscrire à un moment donné son activité à un certain aspect du logiciel en cours de construction : spécification du modèle, du scénario pédagogique, de la représentation ou de l'intégration des différents résultats. Le modèle MARS permet donc de structurer l'approche de conception.

Nous voulions aussi que ce modèle ne reste pas seulement conceptuel, mais qu'il devienne opérationnel. Nous nous sommes de ce fait efforcé d'en définir les conditions d'applicabilité.

Nous avons donc décrit un *processus de développement*, présenté au chapitre 5, basé sur le déroulement en parallèle d'activités de conception organisées chacune selon un *cycle de prototypage automatisé*. En particulier, nous avons abordé le problème de *l'intégration des résultats*, ou mise en associations, en utilisant la même démarche de prototypage.

Nous nous sommes aussi intéressés à l'architecture des environnements de production. Nous avons proposé au chapitre 5 la notion *d'environnement intégré adaptatif* qui permet de rendre davantage générique notre approche. Les quatre espaces de travail du modèle MARS peuvent ainsi proposer des fonctionnalités ou des niveaux d'accès adaptés à certaines contraintes particulières : domaine technique étudié, niveau de compétences des auteurs, contexte de production, type de suivi pédagogique envisagé, etc.

Pour définir les concepts et les interfaces à proposer aux différents concepteurs, nous avons décrit au chapitre 6 une *démarche de validation* basée sur un ensemble de règles simples. Cette démarche, fondée sur un processus itératif, suggère de s'intéresser de près aux relations qui existent entre le profil des concepteurs, les concepts mis en œuvre et les interfaces. Les "bonnes" solutions émergeront après un certain nombre de parcours de ce processus et d'expérimentations réalisées dans des cadres opérationnels.

Dans chacun des espaces de travail, nous avons amorcé cette démarche de validation.

En ce qui concerne l'espace "Modèle" (Chapitre 7), l'accent a été porté sur le fait de proposer aux auteurs une *représentation simplifiée de l'approche par objets*, basée sur la définition d'objets, de propriétés, d'opérations et de diagrammes d'états structurés.

Dans l'espace "Scénario" (Chapitre 8), nous avons établi une définition précise de ce que nous entendons par *scénario de contrôle de simulation*. Pour permettre l'expression des objectifs assignés à l'apprenant et des étapes intermédiaires à observer, nous avons adapté les formalismes IHM des modèles de tâches. Nous avons enrichi ces formalismes pour qu'ils supportent également l'expression de la réactivité pédagogique que doit assurer le système. Nous avons volontairement restreint la puissance d'expression des formalismes obtenus afin d'en assurer la maîtrise par les auteurs visés.

L'espace "Représentation" (Chapitre 9) concerne un domaine où il existe déjà des modèles et outils reconnus de gestion automatisée d'interfaces utilisateur. Nous avons concentré notre attention sur la gestion des objets de représentation et nous avons défini : une *typologie des objets de représentation*, la *façon de les construire*, *l'amélioration des mécanismes de réutilisation et d'adaptation*.

L'espace "Association" (Chapitre 10) représente un thème central du modèle M.A.R.S parce qu'il s'intéresse à l'intégration des résultats obtenus dans les trois autres espaces. Nous avons défini le concept essentiel *d'association de coopération* ainsi qu'un *mécanisme générique de coopération* basé sur des règles Evénement / Condition / Action. Nous avons également proposé une *typologie et une représentation graphique* des différentes associations de coopération.

Il est à noter que durant le déroulement de cette thèse, deux projets de DEA ont été consacrés respectivement au Scénario [BOR 95] et aux Associations [CAM 95]. La co-direction en était assurée par Viviane Guéraud et par moi-même.

Aspects pratiques

Tout au long du déroulement de cette thèse, nous avons confronté nos réflexions théoriques avec des réalisations pratiques auxquelles nous avons participé. Cela a été le cas pour le développement du logiciel expérimental "Alternateur" (Chapitre 4 et 5), mais surtout à l'occasion du projet MELISA en collaboration avec Hewlett-Packard (Chapitre 11).

La définition et le développement de l'environnement MELISA (50.000 lignes de code OpenScript produites depuis trois ans) a demandé de ma part un important investissement avec le support occasionnel de stagiaires (15 hommes*mois). Le

résultat disponible aujourd'hui l'a été grâce à la production régulière de petites maquettes permettant de valider certaines hypothèses établies conjointement avec la division TPEC de Hewlett-Packard. Ce résultat, tout en s'appuyant sur les principes théoriques du modèle MARS, en constitue une mise en œuvre adaptée aux auteurs et aux besoins de ce contexte.

Le principal apport de MELISA est de prouver qu'il est possible aujourd'hui de proposer un environnement auteur intégré de production de simulation pédagogique permettant de *réduire considérablement les temps de production*, tout en laissant la possibilité à l'auteur de *maîtriser la complexité de l'application produite* et de *modifier, adapter et personnaliser très rapidement* le résultat produit.

MELISA permet aujourd'hui de développer une simulation pédagogique grâce à un processus de génération automatique de code à partir des spécifications décrites par l'auteur. Toutefois, le recours à une programmation textuelle est encore nécessaire, en particulier en ce qui concerne la description des opérations du modèle.

Même dans notre contexte pédagogique, les modèles abordés peuvent devenir complexes. A notre avis, il n'existe pas de moyen de rendre simples des problèmes complexes. Nous pouvons *aider à la structuration de la complexité*, par le recours à la modularité ou des formalismes graphiques, mais *nous ne pouvons pas réduire la complexité* sans dénaturer le problème initial. Un des apports de MELISA est de *circonscrire cette complexité à un niveau précis*.

PERSPECTIVES

Sur le plan pratique

La première tâche que nous avons à réaliser aujourd'hui concerne la fiabilisation et l'expérimentation de l'environnement MELISA. Actuellement, un projet mené par quatre étudiants de DESS a pour but de consolider l'environnement. L'expérimentation de l'outil est prévue au sein de Hewlett-Packard pour le courant de l'année 1996, afin d'en dégager les premiers enseignements.

D'autre part, notre équipe est impliquée à compter du 1er Janvier 1996 dans le projet ARIADNE (programme *Telematics Applications* de la Communauté Européenne) dont le but est de construire un vivier informatique de connaissances accessibles à distance ainsi qu'une famille d'outils adéquats. Notre rôle dans ce projet est de fournir des prototypes d'outils-auteurs basés sur le modèle MARS (*Work Package n°8.201a*) et de piloter des expérimentations en vraie grandeur. L'expérimentation se déroulera en particulier, dans le cadre du CAFIM (Centre d'Auto-Formation et d'Innovations Multimédias du premier cycle de l'Université

Joseph Fourier), auprès d'enseignants de diverses disciplines scientifiques en milieu universitaire (*Work Package n°15-202*) et dans le cadre de TPEC de Hewlett-Packard (*Work Package n°12*).

Un projet de DEA en Sciences Cognitives confié à F. Coudret démarre pour aider à l'analyse des expérimentations que nous allons commencer en 1996.

Sur le plan théorique

Comme nous l'avons dit plus haut, nous avons amorcé le processus de validation des concepts dans chacun des espaces de travail de MARS. Nous envisageons dès aujourd'hui d'approfondir certains aspects théoriques :

- Une nouvelle thèse (G. Cortès) est déjà engagée sur le problème de formalisation du scénario pédagogique. En effet, nous avons vu que nous devons converger après des expérimentations, vers un compromis entre puissance d'expression, simplicité d'utilisation et adéquation avec les besoins.
- D'autres travaux sont prévus pour mener une réflexion de fond sur les notions d'objets prototypes ou de collections d'objets, pour aider à la conception du modèle de simulation.
- Une amélioration des techniques de représentation, surtout en ce qui concerne la visualisation et la manipulation des associations (recours à des représentations 3D, accès à différents niveaux de visualisation).
- Un élargissement du modèle MARS à d'autres usages que celui de la simulation pédagogique scientifique ou technique. On peut envisager cet élargissement sous plusieurs angles :

  ⇒ élargissement vers d'autres disciplines telles que les langues par exemple, en s'appuyant sur le concept d'environnement adaptatif. Ceci est actuellement l'objet de travaux menés en collaboration avec l'Université Stendhal-Grenoble 3 (par le biais de Travaux d'Etudes et de Recherche de maîtrise d'anglais, mention Traitement Automatique de la Langue).

  ⇒ redéfinition de la composante Scénario à d'autres fins que le contrôle pédagogique tel que nous l'avons défini. On peut, par exemple, envisager, grâce à MARS, la gestion d'hyperdocument : le modèle représente alors les connaissances brutes non mises en forme ; la représentation, les unités d'information perceptibles par l'utilisateur ; et le scénario, les différentes stratégies d'accès aux connaissances, c'est-à-dire le réseau conceptuel.
- prise en compte de la conception simultanée et de l'échange d'informations entre les concepteurs (collecticiel).

BIBLIOGRAPHIE
CLASSEE PAR THEMES

Un certain nombre de thèmes sont abordés dans cette thèse. Afin de permettre au lecteur un meilleur accès à la bibliographie, nous proposons ici un classement par thème. Les thèmes retenus sont les suivants :

- Génie logiciel et Processus de développement
- Environnements Interactifs d'Apprentissage par Ordinateur
- Environnements de production de logiciels
- Processus de modélisation
- Scénario. Modèles de tâche
- Interaction homme-machine. Conception d'interfaces
- Coopération et associations

Pour retrouver les références bibliographiques citées dans la thèse, le lecteur doit consulter la partie "*Références bibliographiques*" qui se trouve à la suite de ce classement par thèmes.

THEME : Génie logiciel et Processus de développement

Généralités

Boehm B. W., *A Spiral Model of Software Development and Enhancement*, ACM Software Engineering Notes, p.14-24, Août 1986

Boehm B. W., *Software Engineering Economics*, Prentice-Hall, 1981

Boehm B. W., *Software Engineering*, IEEE Transactions on Computers, Vol.25, n°12, p.1226-1241, Dec. 1976

Brooks F.P., *The Mythical Man-Month : Essays on Software Engineering*, Addison-Wesley, 1975

Coad P., Yourdon E., *Object-Oriented Analysis*, Yourdon Press, Prentice-Hall, 1990

Coad P., Yourdon E., *Object-Oriented Design*, Yourdon Press, Prentice-Hall, 1991

Coleman D. and all, *Object-oriented development : the Fusion Method*, Prentice-Hall, 1994

Dahl O., Dijkstra E., Hoare C.A.R., *Structured Programming*, Academic Press, London, 1972

Davis A.M., A comparison of techniques for the specification of external system behavior, Communications of ACM Vol. 31, n°9, p.1098-1115, Sep. 1988

Davis A.M., Bersoff E.H., Comer E.R., *A Strategy for Comparing Alternative Software Development Life Cycle Models*, IEEE Transactions on Software Engineering, Vol.14, n°10, Oct. 1988

De Marco T., *Controlling Software Projects*, Yourdon Press, 1982

Fernstrom C., Ohlsson L., *The ESF Vision of Software Factory*, Proc of Int. Conf. On software Development Environments & Factories, Berlin, Mai 1989

Ghezzi C., Jazayeri M., Mandrioli D., *Fundamentals of Software Engineering*, Prentice-Hall, 1991

Gilb T., *Principles of Software Engineering Management*, Addison-Wesley, 1988

Henderson P., S*ystem design : analysis*. Infotech State of the Art Report Series 9, number 6 : System design, Pergamon Infotech Ltd., n°2, p.5-163, 1981

IEEE Standard Glossary of software engineering terminology, IEEE Standard 729, 1983

Jackson M., *System Development*, Prentice-Hall international, 1983

Partsch H.A., *Specifications and Transformation of Programs*, Springer-Verlag, 1990

Pressman R.S. *Software Engineering*, Mc Graw-Hill, p.148-160, 1987

Royce W.W., *Managing the development of large software : Concepts and Technics* in Proc. WESCON Aou. 1970

Rzepka W., Ohno Y., *Requirements Engineering Environment*, Computer, Vol. 18, n°4, special issue, Apr. 1985

Approche par prototypage

Balzer R., *A 15 Years Perspective on Automatic Programming*, IEEE Transactions on Software Engineering, Vol.11, n°11, Nov. 1985

Balzer R., *Software technology in the 1990's : Using a New Paradigm*, Computer IEEE, Vol. 16, n°11, Nov. 1983

Bischofberger W., Pomberger G., *Prototyping-Oriented Software Development*, Springer-Verlag, 1992

Blaschek G., *Type-Safe Object-Oriented Programming with Prototypes - The concepts of Omega*, Structured Programming, Vol. 12/4, Springer-Verlag, 1991

Blaschek G., *Object-Oriented Programming with Prototypes*, Springer-Verlag, 1994

Boar B., *Application Prototyping*, Wiley-Interscience, 1984

Boehm B. W., *A Spiral Model of Software Development and Enhancement*, ACM Software Engineering Notes, p.14-24, Aug. 1986

Chambers C., Ungar D., Lee E., An Efficient Implementation of SELF, a Dynamically-Typed Object-Oriented Language Based on Prototypes, Proc. of OOPSLA'89, 1989

Davis A.M., Bersoff E.H., Comer E.R., *A Strategy for Comparing Alternative Software Development Life Cycle Models*, IEEE Transactions on Software Engineering, Vol.14, n°10, Oct. 1988

Floyd, *A Systematic Look at Prototyping*, Springer-Verlag, 1984

Taivalsaari A., *Kevo - a prototype-based object oriented language based on concatenation and module operations*, Technical Report LACIR 92-02, University of Victoria, B.C., Canada, 1992

Ungar D., Chambers C., Chang B-W., HölzleU., *Organizing Programs Without Classes*, Lisp and Symbolic Computation, Vol. 4, n°3, 1991

THEME : Environnements Interactifs d'Apprentissage par Ordinateur

Généralités : EAO, EIAO et Simulation

Bratley P., Bennet L.F., Schrage L.E., *A Guide to Simulation*, Springer-Verlag, 1983

Cagnat J.M., Guéraud V. & Peyrin J.P., *The Arcade Laboratory : an environment to help teach algorithms*, ACM SIGSE Bulletin, Vol. 22, n°4, 1990

Cellier F.E., *Continuous System Modelling*, Springer-Verlag, 1991

Cornu B., *L'ordinateur pour enseigner les maths*, Nouvelle Encyclopédie Diderot, Editions UPF, 1992

Crowder R.G., *Principles of learning and memory*, L.Erbaum (Ed). Hillsdale, New Jersey, USA, 1976

De Jong T., *Learning and instruction with computer simulations*, Education and computing, n° 6, p.217-229, 1991

Kodratoff Y., *Leçons d'Apprentissage Symbolique Automatique*, Cepadues-Editions, Toulouse, 1986

Gibaud 0., *Contribution au concept de MicroMondes pour l'Enseignement Assisté par Ordinateur*, Thèse de l'Ecole Centrale de Lyon, 1993

MacCalla G.I., Greer J.E, *Intelligent Advising in Problem solving Domains : the SCENT-3 architecture*, 1st Conference Intelligent Tutoring systems, p.124-131, Montreal, Canada, Jun. 1988

Major N., *Teachers and Intelligent Tutoring systems*, 7th conference PEG '93,p.168-177, Scotland, Jul. 1993

Nicaud J.F., Vivet M., *Les Tuteurs Intelligents. Réalisations et tendances de recherche*, Technique et science informatiques, Vol. 7, n°1, 1988

Papert S., *Mindstorms, children, computers and powerfull ideas*, Basic books, New York, 1980

Skinner B.F., *About behaviorism*, Vintage Books, New York, 1976

Vivet M., *Expertise pédagogique et Usage de tuteurs intelligents*, 13èmes Journées francophones sur l'Informatique, Formation Intelligemment Assistée par Ordinateur, Jan. 1991

Applications

Boufaïda M., *Génération d'interfaces et Conduite du dialogue : le système GEODE*, Thèse de l'Université Paris 6, 1993

Brette J.F., *PASCAL/V : un environnement pour l'apprentissage de la programmation par découverte guidée*, Thèse d'état à Paris 6, 1994

Desmoulins C., *Etude et réalisation d'un système tuteur pour la construction de figures géométriques*, Thèse de l'Université Joseph Fourier, Grenoble, 1994

Fleurkens H., *ESCAPE : A Flexible Design and Simulation Environment*, Rapport interne de Design Automation Section, Eindhoven University of Technology, 1995

Guéraud V., *A Role-Playing Game to learn (programming) methodology*, Conference CATS '90, Barcelona, Spain, 1990

Guéraud V., Cagnat J.M. & Peyrin J.P., *Teaching and Learning made easier by the Arcade Laboratory*, Conference CALISCE '91, Lausanne, Switzerland, 1991

Guéraud V., *Un jeu de rôle pour l'enseignement de la programmation*, Thèse de l'Université Joseph Fourier, Grenoble, 1989

Herzog. J.M., Forte E.N., A *Goal Oriented Simulation in Chemical Thermodynamics*, Conference CALISCE 94, Paris, 1994

Liem I., *Visualisation de l'exécution des programmes pour l'enseignement de la programmation*, Thèse de l'Université J.Fourier, Grenoble, 1989

Njoo M., De Jong T., *Learning preocesses of students working with a computer simulation in mechanical engineering*. Conference EARLI, Madrid, Spain, 1991

Normand X., *Train driving simulators using computer generated images*, Conference CISS, Zurich, Switzerland, Août 1994

Teutsch P., *Environnements interactifs et Langues Etrangères MARPLE : système d'évaluation et suivi de formation*, Thèse de l'Université du Maine, 1994

THEME : Environnements de production de logiciels

Généralités sur la production de logiciels

Fernstrom C., Ohlsson L., *The ESF Vision of Software Factory*, Proc of Int. Conf. On software Development Environments & Factories, Berlin, Mai 1989

Lonchamp J., Godart C., Derniame J-C., *Les environnements intégrés de production de logiciel*, Technique et science informatiques, Vol. 11, n°1, p.31 à 95, 1992

Wasserman A.I., *Tool Integration in Software Engineering Environments*, LNCS 467, Springer-Verlag, p.137-149, Sep. 1989

Production de logiciels pédagogiques

Azzi R., *Environnement de développement de simulateurs pédagogiques de procédés industriels*, Thèse de L'INSA, Lyon, 1995

Beltran T, *Etude d'une interface "multimode" pour la production de "Systèmes hypermédias éducatifs" par l'Atelier de Génie Didacticiel Intégré*, Thèse de l'Université P.Sabatier, Toulouse, 1991

Brette J.F., *PASCAL/V : un environnement pour l'apprentissage de la programmation par découverte guidée*, Thèse d'état à Paris 6, 1994

Burelle I., *Etude et Réalisation d'un logiciel d'aide à l'apprentissage de la compilation dans un monde graphique à objets*, Thèse CNAM, Grenoble, 1993

Canut M.F., *Spécification formelle de systèmes d'Enseignement Intelligemment Assisté par Ordinateur pour l'Atelier de Génie Didactique Intégré (AGDI)*, Thèse de doctorat de l'Université Paul Sabatier, 1990

Choquet C., *Représentation et manipulation des connaissances dans un atelier de génie didacticiel : Application au projet DIGITEF*, Thèse de l'Université P.Sabatier, Toulouse, 1993

David J.P., *Une approche par objets pour la modélisation et la réalisation de micromondes d'apprentissage*, Thèse de l'Université Joseph Fourier, Grenoble, 1994

De Jong T., *Learning and instruction with computer simulations*, Education and

computing, n° 6, p.217-229, 1991

Dillembourg P. & al., *De la généralisibilité d'un environnement d'apprentissage*, Actes des 3èmes journées EIAO de Cachan : Environnements Interactifs d'Apprentissage avec Ordinateur, Ed. Eyrolles, p.169-167, Feb, 1993

Gouardères D., *Le projet d'Atelier de Didactique Intégré*, Technique et Science Informatiques, Vol. 9, n°5, Oct. 1990

Guéraud V., Peyrin J.P., Cagnat J.M., David J.P. et Pernin J.P., *Software environments for Computer Aided Education*, Sigcse Bulletin, Vol. 26, n°2, ACM Press, Jun. 1994

Guéraud V., Peyrin J.P., David J.P. et Pernin J.P., *Environnements logiciels pour une intégration quotidienne de l'E.A.O. dans l'enseignement*, conférence Hypermédias et Apprentissage, Lille, Mar. 1993

Marcenac P., *EDDI : Contributions aux Environnements de Didacticiels*, Thèse de l'Université du Nice, 1990

Mengelle T., *Etude d'une architecture d'environnements d'apprentissage basés sur le préceptorat avisé*, Thèse de l'Université P.Sabatier, Toulouse, 1995

Nicaud J.F., Vivet M., *Les Tuteurs Intelligents. Réalisations et tendances de recherche*, Technique et science informatiques, Vol. 7, n°1, 1988

Njoo M., De Jong T., *Learning preocesses of students working with a computer simulation in mechanical engineering*. Conference EARLI, Madrid, Spain, 1991

Nodenot T., *MAGE : Une méta-atelier de génie éducatif*, Thèse de l'Université Paul Sabatier, 1992

Péninou A., *MACT : Un modèle d'agents centrés tâches pour la production de systèmes tuteurs intelligents par l'atelier de génie didacticiel integré*, Thèse de l'Université P.Sabatier, Toulouse, 1993

Pernin J.P., *Assisted design and automatic generation of pegagogical simulations*, 3rd conference Computer Aided Engineering Education, Bratislava Slovakia, Sep.1995

Pernin J.P., *Generalizing Object-Oriented Approach for Authoring Systems*, conference Computer Based Learning in Science, Wien Austria, Dec. 1993

Pernin J.P., Guéraud V., *MARS, Modèle-Associations-Représentation-Scénario, un modèle de conception d'applications pédagogiques interactives*, Conférence IHM'95, Toulouse, Oct. 1995

Pernin J.P., *Vers une généralisation de l'approche par objets dans les systèmes auteurs hypermédias*, Rapport de DEA, Université Joseph Fourier, 1993

Seffah A., *Propositions pour un Atelier de Génie Didacticiel Intégré : Concepts, Démarche & outils*, Thèse de l'Ecole Centrale de Lyon, 1993

Seffah A., Ouadou K., David B.T., *Impacts des techniques de génie logiciel sur la méthode de réalisation d'un logiciel d'EAO*, Le Génie Logiciel et ses Application, 6èmes Journées Internationales, Paris, 1993

van Rosmalen P., in Ton de Jong & al., *Design and Production of Multimedia and simulation-Based Training*, Kluwer Academic Publishers, June 1994

THEME : Processus de modélisation

Modélisation des systèmes

Bouzeghoub M., Gardarin G., Valduriez P., *Objets : de C++ à Merise Objet*, Editions Eyrolles, 1994

Brodie, M.L. *On Modeling Behavioural Semantics of Databases*, 7th VLDB, p.32-41, Sep. 1981

Chen P.P.S., *The Entity-Relationship Model, Toward a Unified View of Data*, CACM Trans. On DB Systems, Vol. 1, n°1, p9-36, 1976

Dahl O., Dijkstra E., Hoare C.A.R., *Structured Programming*, Academic Press, London, 1972

De Marco T., *Controlling Software Projects*, Yourdon Press, 1982

Flavin M., *Fundamental concepts of Information Modeling*, Prentice-Hall, 1981

Jackson M., *Principles of Program Design*, Academic Press, Orlando, 1975

Jackson M., *System Development*, Prentice-Hall international, 1983

Jinxiong C., Lobo N., Hughes C., Blau B., LI X., *Distributed Virtual Environment Real-Time Simulation Networks*, Advances in Modelling and analysis, AMSE periodicals, n°42, p.1-7, Jan. 1994

Marca D., McGowan C., *SADT : Structured analysis and Design Technique*, Mc Graw-Hill, 1988

Ross D.T., *Applications and extensions of SADT,*, IEEE Transactions on Software Engineering, Vol.18, n°4, p.25-34, 1985

Ross D.T.,*Structured Analysis (S.A.) : A language for communicating ideas,* IEEE Transactions on Software Engineering, Vol. SE-3, n°1, p.16-34, 1977

Tardieu, Rochfeld, Coletti, *La méthode MERISE*, 1982

Warnier J.D., *Logical Construction of Systems*, Van Nostrand Rienhold, 1981

Wasserman A.I. *Information system design methodology*, Journal of the American Society for Information Vol. 3, n°1, p5-24, 1980

Modélisation de la dynamique des systèmes

Genrich H.J., Lautenbach K., *The analysis of Distributed Systems by Means of Predicate/Transition Nets*, Semantics of Concurrent Computation, Evian 1979, G. Khan (Ed.), Lect. Notes in Computer Sciences, p.123-146, Springer-Verlag, 1979

Harel D., *On visual Formalisms*, Communications of ACM Vol. 31 n°5 p.514-530, May 1988

Harel D., *Statecharts : A Visual Formalism for Complex Systems*, Science of Computer Programming, Vol. 8, p 231-271, 1987

Harel D., *Statemate : a working environment for the dévelopment of complex reactive systems*, IEEE Transactions on Software Engineering, Vol.16, n°4, p.403-413, Apr. 1990

Jensen K., *Coloured Petri Nets and the Invariant Method*, ATPN, IF 66, Springer-Verlag, 1983

Petri C., *Kommunication mit automaten*, PhD dissertation, University of Bonn, 1962

Schlaer S., Mellor J.S., *Object-Oriented Systems Analysis : Object Lifecycles, Modeling the World in States*, Yourdon Press, 1991

Sibertin-Blanc C., *High-Level Petri Nets with Data Structure*, 6th European workshop on Petri Net and applications, Espoo, Finland, Jun. 1985

Wasserman A.I. *Extending Transition Diagrams for the Specification of Human-Computer interaction*, IEEE Transactions on Software Engineering, p326-345, Aug. 1985

Wood W.A., *Transition network grammars for natural language analysis*, Communication of ACM, Vol. 13, n° 10, p 591-606, Oct. 1970

Approche par objets

Booch G., *Object-Oriented Analysis and Design with Applications*, Benjamin-Cummings, 1994

Booch G., *Object-Oriented Design with Applications*, Benjamin-Cummings, 1991

Coad P., Yourdon E., *Object-Oriented Analysis*, Yourdon Press, Prentice-Hall, 1990

Coad P., Yourdon E., *Object-Oriented Design*, Yourdon Press, Prentice-Hall, 1991

Coleman D. and all, *Object-oriented development : the Fusion Method*, Prentice-Hall, 1994

Goldberg A., *Smalltalk 80, The Interactive Programming Environment*, Addison-Wesley, 1984

Meyer B., *Conception et Programmation par objets*, InterEditions, Paris, 1990

Rumbaugh J. & all., *Object Oriented Modelling and Design*, Prentice-Hall, 1991

Schlaer S., Mellor J.S., *Object-Oriented Analysis : Modeling the World in Data*, Yourdon Press, 1988

Schlaer S., Mellor J.S., *Object-Oriented Systems Analysis : Object Lifecycles, Modeling the World in States*, Yourdon Press, 1991

THEME : Scénario. Modèles de tâche

Balbo S., *Evaluation ergonomique des interfaces utilisateur : un pas vers l'automatisation*, Thèse de l'Université Joseph Fourier, Grenoble, 1994

Barthet M.F., *Logiciels interactifs et ergonomie, modèles et méthodes de conception*, Dunod Informatique, 1988

Bordon S., *Formalisme pour la représentation des scénarios dans les didacticiels*, Rapport de DEA, Université Joseph Fourier, 1995

Gray P., England D., McGowan S., *XUAN : Enhancing the UAN to capture temporal relations among actions*, Computing Science Research Report, IS-94-02, University of Glasgow, e-mail : reports@dcs.glasgow.ac.uk, Feb. 1994

Hartson H.R., Gray P.D., Temporal aspects of tasks in the User Action Notation, Human-Computer Interaction, Vol. 7, p.1-45, 1992

Mason R.E.A., Carey T.T., *Productivity Experiences with a Scenario Tool*, Proc IEEE Compcon, CS Press, Los Alamitos, Calif., p.106-111, 1981

Norman D.A., Draper S.W., *User Centered System Design*, Lawrence Erlbaum Associates, Publishers, 1986

Normand V., *Le modèle SIROCO : de la spécification conceptuelle des interfaces utilisateur à leur réalisation*, Thèse de l'Université Joseph Fourier, Grenoble, 1992

Pernin J.P., *Assisted design and automatic generation of pegagogical simulations*, 3rd conference Computer Aided Engineering Education, Bratislava Slovakia, Sep. 1995

Pernin J.P., Guéraud V., *MARS, Modèle-Associations-Représentation-Scénario, un modèle de conception d'applications pédagogiques interactives*, Conférence IHM'95, Toulouse, Oct. 1995

Sébilotte S., *Décrire des tâches selon les objectifs des opérateurs? De l'interview à la formalisation*, revue Travail Humain, tome 54, n°3, p.193-223, 1991

Wasserman A.I. *Extending Transition Diagrams for the Specification of Human-Computer interaction*, IEEE Transactions on Software Engineering, p326-345, Aug. 1985

THEME : Interaction homme-machine. Conception d'interfaces

Balbo S., *Evaluation ergonomique des interfaces utilisateur : un pas vers l'automatisation*, Thèse de l'Université Joseph Fourier, Grenoble, 1994

Barthet M.F., *Logiciels interactifs et ergonomie, modèles et méthodes de conception*, Dunod Informatique, 1988

Coutaz J., *Interfaces homme-ordinateur*, Dunod, 1990

Dix A. & all, *Human Computer Interaction*, Prentice-Hall, 1993

Goldberg A., *Smalltalk 80, The Interactive Programming Environment*, Addison-Wesley, 1984

Hayes P.J., Szekely P.A. & Lerner R.A., *Design Alternatives fir User interface Management Systems Based on Experience with Cousin*, Proc. CHI'85 Conf. Human Factors in Computing Systems, ACM, p.177-183, 1985

Hix D., *Generations of User-Interface Management Systems, IEEE Software*, Sep. 1990

Kieras D., Polson G., *A generalised transition network representation for interactive systems*, In proceedings of CHI'93, Human factors in computing systems,

p.103-106, 1983

Myers B.A., *Creating Dynamic Real-time interface Design*, Proc. CHI'87 Conf. Human Factors in Computing Systems, ACM, p.271-278, 1987

NeXT step Reference, Addison-Wesley Publ., 1991

Nigay L., *Conception et modélisation de systèmes interactifs*, Thèse de l'Université Joseph Fourier, Grenoble, 1994

Normand V., *Le modèle SIROCO : de la spécification conceptuelle des interfaces utilisateur à leur réalisation*, Thèse de l'Université Joseph Fourier, Grenoble, 1992

Palanque P., *Modélisation des objets coopératifs interactifs d'interfaces homme-machine dirigées par l'utilisateur*, Thèse de l'Université Toulouse 1, 1992

Pfaff G.E., *User Interface Management Systems*, Eurographics seminars, Springer-Verlag, 1985

Preece J., *Human Computer Interaction*, Addison-Wesley, 1994

Sibert J.L. & all, *An object oriented user interface management system*, SIGGRAPH'86

Sibert J.L. & all, *Design and Implementation of an Object-Oriented User Interface Management System*, in Advances in Human-computer Interaction, Vol. 2, Hartson & Dix, eds., Ablex Publishing, Norwood, N.J., p175-213, 1988

Wasserman A.I. *Extending Transition Diagrams for the Specification of Human-Computer interaction*, IEEE Transactions on Software Engineering, p326-345, Aug. 1985

Wood W.A., *Transition network grammars for natural language analysis*, Communication of ACM, Vol. 13, n° 10, p 591-606, Oct. 1970

THEME : Coopération et associations

Adiba M. et al., *Triggers Systems : Different Approaches*, Rapport Aristote, Jun 1993

Agha G., *Actors : a Model of Concurrent Computation in Distributed Systems*, Series in Artificial Intelligence, MIT Press, 1986

Blatschek G., *Object-Oriented Programming with Prototypes*, Springer-Verlag, 1994

Booch G., *Object-Oriented Analysis and Design with Applications*, Benjamin-

Cummings, 1994

Booch G., *Object-Oriented Design with Applications*, Benjamin-Cummings, 1991

Campani C., *Cooperation et Associations dans un AGL de Conception et de Réalisation d'Application de Simulation Pédagogique*, Rapport de DEA, Université Joseph Fourier, 1995

Coad P., Yourdon E., *Object-Oriented Analysis*, Yourdon Press, Prentice-Hall, 1990

Coad P., Yourdon E., *Object-Oriented Design*, Yourdon Press, Prentice-Hall, 1991

Dittrich K.R. & al., *An Event : Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases*, 1986

Dittrich K.R., *Supporting Semantics Rules by a Generalized Event/Trigger Mechanism*, 1st EDBT, p.76-91, Mar. 1988

Engelmore R.S. & Morgan A.J.,*Blackboard Systems*, Addison Wesley, 1988

Ferber, *Objets et agents : Une étude des structures de représentation et de communication en I.A.* Thèse d'état, Paris 6, 1989

Harel D., *Statecharts : A Visual Formalism for Complex Systems*, Science of Computer Programming, Vol. 8, p 231-271, 1987

Hewitt C.E., *Viewing Control Structures as Patterns of Passing Messages*, Journal of Artificial Intelligence, Vol 8, n° 3, p.323-364, 1977

Kieras D., Polson G., *A generalised transition network representation for interactive systems*, In proceedings of CHI'93, Human factors in computing systems, p 103-106, 1983

NeXT step Reference, Addison-Wesley Publ., 1991

Palanque P., *Modélisation des objets coopératifs interactifs d'interfaces homme-machine dirigées par l'utilisateur*, Thèse de l'Université Toulouse 1, 1992

Pernin J.P., *Vers une généralisation de l'approche par objets dans les systèmes auteurs hypermédias*, Rapport de DEA, Université Joseph Fourier, 1993

Pernin J.P., *Assisted design and automatic generation of pegagogical simulations*, 3rd conference Computer Aided Engineering Education, Bratislava Slovakia, Sep. 1995

Pernin J.P., Guéraud V., *MARS, Modèle-Associations-Représentation-Scénario, un modèle de conception d'applications pédagogiques interactives,* Conférence IHM'95, Toulouse, Oct. 1995

Petri C., *Kommunication mit automaten*, PhD dissertation, University of Bonn, 1962

Portejoie P., *Planification en univers mono et multi-agents*, Thèse de l'Université de Rennes, 1991

REFERENCES BIBLIOGRAPHIQUES

[ADI 93] Adiba M. et al., *Triggers Systems : Different Approaches,* Rapport Aristote, Jun. 1993

[AGH 86] Agha G., *Actors : a Model of Concurrent Computation in Distributed Systems,* Series in Artificial Intelligence, MIT Press, 1986

[AZZ 95] Azzi R., *Environnement de développement de simulateurs pédagogiques de procédés industriels,* Thèse de L'INSA, Lyon, 1995

[BAE 89] Bauer F.L. and all, *Formal Program construction by Transformations, computer aided, intuition-guided programming,* IEEE Transactions on Software Engineering, Vol. 15, n°2, p.165-180, Feb. 1989

[BAL 83] Balzer R., *Software technology in the 1990's : Using a New Paradigm,* Computer IEEE, Vol. 16, n°11, Nov. 1983

[BAL 85] Balzer R., *A 15 Years Perspective on Automatic Programming,* IEEE Transactions on Software Engineering, Vol.11, n°11, Nov. 1985

[BAL 94] Balbo S., *Evaluation ergonomique des interfaces utilisateur : un pas vers l'automatisation,* Thèse de l'Université J. Fourier, Grenoble, 1994

[BAR 88] Barthet M.F., *Logiciels interactifs et ergonomie, modèles et méthodes de conception,* Dunod Informatique, 1988

[BEL 91] Beltran T, *Etude d'une interface "multimode" pour la production de "Systèmes hypermédias éducatifs" par l'Atelier de Génie Didacticiel Intégré,* Thèse de l'Université P.Sabatier, Toulouse, 1991

[BIS 92] Bischofberger W., Pomberger G., *Prototyping-Oriented Software Development,* Springer-Verlag, 1992

[BLA 91] Blaschek G., *Type-Safe Object-Oriented Programming with Prototypes - The concepts of Omega,* Structured Programming, Vol. 12/4, Springer-Verlag, 1991

[BLA 94] Blaschek G., *Object-Oriented Programming with Prototypes,* Springer-Verlag, 1994

[BOA 84] Boar B., *Application Prototyping,* Wiley-Interscience, 1984

[BOE 76] Boehm B. W., *Software Engineering,* IEEE Transactions on Computers, Vol.25, n°12, p.1226-1241, Dec. 1976

[BOE 81] Boehm B. W., *Software Engineering Economics,* Prentice Hall, 1981

[BOE 86] Boehm B. W., *A Spiral Model of Software Development and Enhancement,* ACM Software Engineering Notes, p.14-24, Aug. 1986

[BOO 91] Booch G., *Object-Oriented Design with Applications*, Benjamin-Cummings, 1991

[BOO 94] Booch G., *Object-Oriented Analysis and Design with Applications*, Benjamin-Cummings, 1994

[BOR 95] Bordon S., *Formalisme pour la représentation des scénarios dans les didacticiels*, Rapport de DEA, Université Joseph Fourier, 1995

[BOU 93] Boufaïda M., *Génération d'interfaces et Conduite du dialogue : le système GEODE*, Thèse de l'Université Paris 6, 1993

[BOU 94] Bouzeghoub M., Gardarin G., Valduriez P., *Objets : de C++ à Merise Objet*, Editions Eyrolles, 1994

[BRA 83] Bratley P., Bennet L.F., Schrage L.E., *A Guide to Simulation*, Springer-Verlag, 1983

[BRE 94] Brette J.F., *PASCAL/V : un environnement pour l'apprentissage de la programmation par découverte guidée*, Thèse d'état à Paris VI, 1994

[BRI 86] Encyclopedia Britannica, 1986

[BRO 75] Brooks F.P., *The Mythical Man-Month : Essays on Software Engineering*, Addison-Wesley, 1975

[BRO 81] Brodie, M.L. *On Modeling Behavioural Semantics of Databases*, 7th VLDB, p.32-41, Sep. 1981

[BUR 93] Burelle I., *Etude et Réalisation d'un logiciel d'aide à l'apprentissage de la compilation dans un monde graphique à objets*, Thèse CNAM, 1993

[CAG 90] Cagnat J.M., Guéraud V. & Peyrin J.P., *The Arcade Laboratory : an environment to help teach algorithms*, ACM SIGSE Bulletin, Vol. 22, n°4, 1990

[CAM 95] Campani C., *Coopération et Associations dans un AGL de Conception et de Réalisation d'Application de Simulation Pédagogique*, Rapport de DEA, Université Joseph Fourier, 1995

[CAN 90] Canut M.F., *Spécification formelle de systèmes d'Enseignement Intelligemment Assisté par Ordinateur pour l'Atelier de Génie Didactique Intégré (AGDI)*, Thèse de doctorat de l'Université Paul Sabatier, 1990

[CEL 91] Cellier F.E., *Continuous System Modelling*, Springer-Verlag, 1991

[CHA 89] Chambers C., Ungar D., Lee E., *An Efficient Implementation of SELF, a Dynamically-Typed Object-Oriented Language Based on Prototypes*, Proc. of OOPSLA '89, 1989

[CHE 76] Chen P.P.S., *The Entity-Relationship Model, Toward a Unified View of Data*, CACM Trans. On DB Systems, Vol. 1, n°1, p.9-36, 1976

[CHO 93] Choquet C., *Représentation et manipulation des connaissances dans un atelier de génie didacticiel : Application au projet DIGITEF*, Thèse de l'Université P.Sabatier, Toulouse, 1993

[COA 90] Coad P., Yourdon E., *Object-Oriented Analysis*, Yourdon Press, Prentice Hall, 1990

[COA 91] Coad P., Yourdon E., *Object-Oriented Design*, Yourdon Press, Prentice Hall, 1991

[COL 94] Coleman D. and all, *Object-oriented development : the Fusion Method*, Prentice Hall, 1994

[COR 92] Cornu B., *L'ordinateur pour enseigner les maths*, Nouvelle Encyclopédie Diderot, Editions UPF, 1992

[COU 90] Coutaz J., *Interfaces homme-ordinateur*, Dunod, 1990

[CRO 76] Crowder R.G., *Principles of learning and memory*, L.Erbaum (Ed). Hillsdale, New Jersey, USA, 1976

[DAH 72] Dahl O., Dijkstra E., Hoare C.A.R., *Structured Programming*, Academic Press, London, 1972

[DAV 88a] Davis A.M., *A comparison of techniques for the specification of external system behavior*, Communications of ACM Vol. 31, n°9, p.1098-1115, Sep. 1988

[DAV 88b] Davis A.M., Bersoff E.H., Comer E.R., *A Strategy for Comparing Alternative Software Development Life Cycle Models*, IEEE Transactions on Software Engineering, Vol.14, n°10, Oct. 1988

[DAV 94] David J.P., *Une approche par objets pour la modélisation et la réalisation de micromondes d'apprentissage*, Thèse de l'Université J. Fourier, Grenoble, 1994

[DEJ 91] De Jong T., *Learning and instruction with computer simulations*, Education and computing, n° 6, p.217-229, 1991

[DeM 82] De Marco T., *Controlling Software Projects*, Yourdon Press, 1982

[DES 94] Desmoulins C., *Etude et réalisation d'un système tuteur pour la construction de figures géométriques*, Thèse de l'Université J. Fourier, Grenoble, 1994

[DIL 93] Dillembourg P. & al., *De la généralisibilité d'un environnement d'apprentissage,* Actes des 3èmes journées EIAO de Cachan : Environnements Interactifs d'Apprentissage avec Ordinateur, Ed. Eyrolles, p.169-167, Feb. 1993

[DIT 86] Dittrich K.R. & al., *An Event : Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases*, 1986

[DIT 88] Dittrich K.R., *Supporting Semantics Rules by a Generalized Event/Trigger Mechanism*, 1st EDBT, p.76-91, Mar. 1988

[DIX 93] Dix A. & all, *Human Computer Interaction*, Prentice Hall, 1993

[ENG 88] Engelmore R.S. & Morgan A.J.,*Blackboard Systems*, Addison Wesley, 1988

[FER 89a] Fernstrom C., Ohlsson L., *The ESF Vision of Software Factory,* Proc of Int. Conf. On software Development Environments & Factories, Berlin, Mai 1989

[FER 89b] Ferber, *Objets et agents : Une étude des structures de représentation et de communication en I.A.* Thèse d'état, Paris VI, 1989

[FLA 81] Flavin M., *Fundamental concepts of Information Modeling*, Prentice Hall, 1981

[FLE 95] Fleurkens H., *ESCAPE : A Flexible Design and Simulation Environment,* Rapport interne de Design Automation Section, Eindhoven University of Technology, 1995

[FLO 84] Floyd, *A Systematic Look at Prototyping*, Springer-Verlag, 1984

[GEN 86] Genrich H.J., Lautenbach K., *The analysis of Distributed Systems by Means of Predicate/Transition Nets*, Semantics of Concurrent Computation, Evian 1979, G. Khan (Ed.), Lect. Notes in Computer Sciences, p.123-146, Springer-Verlag, 1979

[GHE 91] Ghezzi C., Jazayeri M., Mandrioli D., *Fundamentals of Software Engineering*, Prentice Hall, 1991

[GIB 93] Gibaud 0., *Contribution au concept de MicroMondes pour l'Enseignement Assisté par Ordinateur*, Thèse de l'Ecole Centrale de Lyon, 1993

[GIL 88] Gilb T., *Principles of Software Engineering Management*, Addison-Wesley, 1988

[GOL 84] Goldberg A., *Smalltalk 80, The Interactive Programming Environment*, Addison-Wesley, 1984

[GOU 90] Gouardères D., *Le projet d'Atelier de Didactique Intégré*, Technique et Science Informatiques, Vol. 9, n°5, Oct. 1990

[GRA 94] Gray P., England D., McGowan S., *XUAN : Enhancing the UAN to capture temporal relations among actions*, Computing Science Research Report, IS-94-02, University of Glasgow, e-mail : reports@dcs.glasgow.ac.uk, Feb. 1994

[GUE 89] Guéraud V., *Un jeu de rôle pour l'enseignement de la programmation*, Thèse de l'Université J. Fourier, Grenoble, 1989

[GUE 90] Guéraud V., *A Role-Playing Game to learn (programming) methodology*, Conference CATS '90, Barcelona, Spain, 1990

[GUE 91] Guéraud V., Cagnat J.M. & Peyrin J.P., *Teaching and Learning made easier by the Arcade Laboratory*, Conference CALISCE '91, Lausanne, Switzerland, 1991

[GUE 93] Guéraud V., Peyrin J.P., David J.P. et Pernin J.P., *Environnements logiciels pour une intégration quotidienne de l'E.A.O. dans l'enseignement*, Conférence Hypermédias et Apprentissage, Lille, Mar. 1993

[GUE 94] Guéraud V., Peyrin J.P., Cagnat J.M., David J.P. et Pernin J.P., *Software environments for Computer Aided Education*, Sigcse Bulletin, Vol. 26, n°2, ACM Press, Jun. 1994

[HAR 87] Harel D., Statecharts : A Visual Formalism for Complex Systems, Science of Computer Programming, Vol. 8, p.231-271, 1987

[HAR 88] Harel D., *On visual Formalisms*, Communications of ACM Vol. 31, n°5 p.514-530, May 1988

[HAR 90] Harel D., *Statemate : a working environment for the dévelopment of complex reactive systems*, IEEE Transactions on Software Engineering, Vol.16, n°4, p.403-413, Apr. 1990

[HAR 92] Hartson H.R., Gray P.D., *Temporal aspects of tasks in the User Action Notation*, Human-Computer Interaction, Vol. 7, p.1-45, 1992

[HAY 85] Hayes P.J., Szekely P.A. & Lerner R.A., *Design Alternatives for User interface Management Systems Based on Experience with Cousin*, Proc. CHI'85 Conf. Human Factors in Computing Systems, ACM, p.177-183, 1985

[HEN 81] Henderson P., S*ystem design : analysis*. Infotech State of the Art Report, Series 9, n°6 : System design, Pergamon Infotech Ltd., n°2, p.5-163, 1981

[HER 94] Herzog. J.M., Forte E.N., A *Goal Oriented Simulation in Chemical Thermodynamics*, Conference CALISCE 94, Paris, 1994

[HEW 77] Hewitt C.E., *Viewing Control Structures as Patterns of Passing Messages*, Journal of Artificial Intelligence, Vol 8, n° 3, p.323-364, 1977

[HIX 90] Hix D., *Generations of User-Interface Management Systems, IEEE Software*, Sep. 1990

[IEE 83] *IEEE Standard Glossary of software engineering terminology*, IEEE Standard 729, 1983

[ISO 89] ISON R., *An Experimental Ada Programming Support Environment in the HP CASEEdge Integration Framework*, LNCS, p.179-193, Springer-Verlag, 1989

[JAC 75] Jackson M., *Principles of Program Design*, Academic Press, Orlando, 1975

[JAC 83] Jackson M., *System Development*, Prentice-Hall international, 1983

[JEN 83] Jensen K., *Coloured Petri Nets and the Invariant Method*, ATPN, IF 66, Springer-Verlag, 1983

[JIN 94] Jinxiong C., Lobo N., Hughes C., Blau B., LI X., *Distributed Virtual Environment Real-Time Simulation Networks*, Advances in Modelling and analysis, AMSE periodicals, n°42, p.1-7, Jan. 1994

[KIE 83] Kieras D., Polson G., *A generalised transition network representation for interactive systems*, In proceedings of CHI'93, Human factors in computing systems, p.103-106, 1983

[KOD 86] Kodratoff Y., *Leçons d'Apprentissage Symbolique Automatique*, Cepadues-Editions, Toulouse, 1986

[LIE 89] Liem I., *Visualisation de l'exécution des programmes pour l'enseignement de la programmation*, Thèse de l'Université J.Fourier, Grenoble, 1989

[LON 92] Lonchamp J., Godart C., Derniame J-C., *Les environnements intégrés de production de logiciel*, Technique et science informatiques, Vol. 11, n°1, p.31-95, 1992

[MAJ 93] Major N., *Teachers and Intelligent Tutoring systems*, 7th conference PEG'93, p.168-177, Scotland, Jul. 1993

[MAR 88] Marca D., McGowan C., *SADT : Structured analysis and Design Technique*, McGraw-Hill, 1988

[MAR 90] Marcenac P., EDDI : Contributions aux Environnements de Didacticiels,

Thèse de l'Université du Nice, 1990

[MAS 81] Mason R.E.A., Carey T.T., *Productivity Experiences with a Scenario Tool*, Proc IEEE Compcon, CS Press, Los Alamitos, Calif., p.106-111, 1981

[McC 88] MacCalla G.I., Greer J.E, *Intelligent Advising in Problem solving Domains : the SCENT-3 architecture*, 1st Conference Intelligent Tutoring systems, p.124-131, Montreal, Canada, Jun. 1988

[MEN 95] Mengelle T., *Etude d'une architecture d'environnements d'apprentissage basés sur le préceptorat avisé*, Thèse de l'Université P.Sabatier, Toulouse, 1995

[MEY 90] Meyer B., *Conception et Programmation par objets*, InterEditions, Paris, 1990

[MYE 87] Myers B.A., *Creating Dynamic Real-time interface Design*, Proc. CHI'87 Conf. Human Factors in Computing Systems, ACM, p.271-278, 1987

[NAN 89] Nance R.E., *Comtemplation of a simulation navel or recognizing the seers among the peers*, Simulation Digest, Vol. 20, n°3, p36-45, 1989

[NEX 91] *NeXT step Reference*, Addison-Wesley Publ., 1991

[NIC 88] Nicaud J.F., Vivet M., *Les Tuteurs Intelligents. Réalisations et tendances de recherche*, Technique et science informatiques, Vol. 7, n°1, 1988

[NIG 94] Nigay L., *Conception et modélisation de systèmes interactifs*, Thèse de l'Université J. Fourier, Grenoble, 1994

[NJO 91] Njoo M., De Jong T., *Learning processes of students working with a computer simulation in mechanical engineering*. Conference EARLI, Madrid, Spain, 1991

[NOD 92] Nodenot T., *MAGE : Une méta-atelier de génie éducatif*, Thèse de l'Université Paul Sabatier, 1992

[NOR 86] Norman D.A., Draper S.W., *User Centered System Design*, Lawrence Erlbaum Associates, Publishers, 1986

[NOR 92] Normand V., *Le modèle SIROCO : de la spécification conceptuelle des interfaces utilisateur à leur réalisation*, Thèse de l'Université J. Fourier, Grenoble, 1992

[NOR 94] Normand X., *Train driving simulators using computer generated images*, Conference CISS, Zurich, Switzerland, Aoû. 1994

[OXF 86] *Dictionnary of Computing*, Oxford University Press, 1986

[PAL 92] Palanque P., *Modélisation des objets coopératifs interactifs d'interfaces*

homme-machine dirigées par l'utilisateur, Thèse de l'Université Toulouse 1, 1992

[PAP 80] Papert S., *Mindstorms, children, computers and powerfull ideas*, Basic books, New York, 1980

[PAR 90] Partsch H.A., *Specifications and Transformation of Programs*, Springer-Verlag, 1990

[PEN 93] Péninou A., *MACT : Un modèle d'agents centrés tâches pour la production de systèmes tuteurs intelligents par l'atelier de génie didacticiel integré*, Thèse de l'Université P.Sabatier, Toulouse, 1993

[PER 93a] Pernin J.P., *Vers une généralisation de l'approche par objets dans les systèmes auteurs hypermédias*, Rapport de DEA, Université Joseph Fourier, 1993

[PER 93b] Pernin J.P., *Generalizing Object-Oriented Approach for Authoring Systems*, Conf. Computer Based Learning in Science, Wien, Austria, Dec. 1993

[PER 95a] Pernin J.P., *Assisted design and automatic generation of pegagogical simulations*, 3rd conference Computer Aided Engineering Education, Bratislava Slovakia, Sep. 1995

[PER 95b] Pernin J.P., Guéraud V., *MARS, Modèle-Associations-Représentation-Scénario, un modèle de conception d'applications pédagogiques interactives*, Conférence IHM'95, Toulouse, Oct. 1995

[PET 62] Petri C., *Kommunication mit automaten*, PhD dissertation, University of Bonn, 1962

[PFA 85] Pfaff G.E., *User Interface Management Systems*, Eurographics seminars, Springer-Verlag, 1985

[POR 91] Portejoie P., *Planification en univers mono et multi-agents*, Thèse de l'Université de Rennes, 1991

[PRE 87] Pressman R.S. *Software Engineering*, MacGraw-Hill, p.148-160, 1987

[PRE 94] Preece J., *Human Computer Interaction*, Addison-Wesley, 1994

[ROB 92] *Dictionnaire Le Petit Robert*, 1992

[ROS 77] Ross D.T.,S*tructured Analysis (S.A.) : A language for communicating ideas*, IEEE Transactions on Software Engineering, Vol. SE-3, n°1, p.16-34, 1977

[ROS 85] Ross D.T., *Applications and extensions of SADT*, IEEE Transactions on

Software Engineering, Vol.18, n°4, p.25-34, 1985

[ROS 94] van Rosmalen P., in Ton de Jong & al., *Design and Production of Multimedia and simulation-Based Training*, Kluwer Academic Publishers, Jun. 1994

[ROY 70] Royce W.W., *Managing the development of large software : Concepts and Technics* in Proc. WESCON, Aug. 1970

[RUM 91] Rumbaugh J. & all., *Object Oriented Modelling and Design*, Prentice-Hall, 1991

[RZE 85] Rzepka W., Ohno Y., *Requirements Enginnering Environment*, Computer, Vol. 18, n°4, special issue, Apr. 1985

[SCH 88] Schlaer S., Mellor J.S., *Object-Oriented Analysis : Modeling the World in Data,* Yourdon Press, 1988

[SCH 91] Schlaer S., Mellor J.S., *Object-Oriented Systems Analysis : Object Lifecycles, Modeling the World in States,* Yourdon Press, 1991

[SEB 91] Sébilotte S., *Décrire des tâches selon les objectifs des opérateurs? De l'interview à la formalisation*, revue Travail Humain, tome 54, n°3, p.193-223, 1991

[SEF 93a] Seffah A., *Propositions pour un Atelier de Génie Didacticiel Intégré : Concepts, Démarche & outils*, Thèse de l'Ecole Centrale de Lyon, 1993

[SEF 93b] Seffah A., Ouadou K., David B.T., *Impacts des techniques de génie logiciel sur la méthode de réalisation d'un logiciel d'EAO*, Le Génie Logiciel et ses Application, 6èmes Journées Internationales, Paris, 1993

[SIB 85] Sibertin-Blanc C., *High-Level Petri Nets with Data Structure*, 6th European workshop on Petri Net & applications, Espoo, Finland, Jun. 1985

[SIB 86] Sibert J.L. & all, *An object oriented user interface management system*, SIGGRAPH'86, 1986

[SIB 88] Sibert J.L. & all, *Design and Implementation of an Object-Oriented User Interface Management System,* in Advances in Human-computer Interaction, Vol. 2, Hartson & Dix, eds., Ablex Publishing, Norwood, N.J., p175-213, 1988

[SKI 76] Skinner B.F., *About behaviorism*, Vintage Books, New York, 1976

[TAI 92] Taivalsaari A., *Kevo - a prototype-based object oriented language based on concatenation and module operations,* Technical Report LACIR 92-02, University of Victoria, B.C., Canada, 1992

[TAR 82] Tardieu, Rochfeld, Coletti, *La méthode MERISE*, 1982

[TEU 94] Teutsch P., Environnements interactifs et Langues Etrangères MARPLE : système d'évaluation et suivi de formation, Thèse de l'Université du Maine, 1994

[UNG 91] Ungar D., Chambers C., Chang B-W., HölzleU., *Organizing Programs Without Classes*, Lisp and Symbolic Computation, Vol. 4, n°3, 1991

[VIV 91] Vivet M., *Expertise pédagogique et Usage de tuteurs intelligents*, 13èmes Journées francophones sur l'Informatique, Formation Intelligemment Assistée par Ordinateur, Jan. 1991

[WAR 81] Warnier J.D., *Logical Construction of Systems*, Van Nostrand Rienhold, 1981

[WAS 80] Wasserman A.I. *Information system design methodology*, Journal of the American Society for Information Vol. 3, n°1, p5-24, 1980

[WAS 85] Wasserman A.I. *Extending Transition Diagrams for the Specification of Human-Computer interaction*, IEEE Transactions on Software Engineering, p326-345, Aug. 1985

[WAS 89] Wasserman A.I., *Tool Integration in Software Engineering Environments*, LNCS 467, Springer-Verlag, p.137-149, Sep. 1989

[WEB 77] *Webster's New Twentieth Century Dictionnary*, Colins World, 1977

[WOO 70] Wood W.A., *Transition network grammars for natural language analysis*, Communication of ACM, Vol. 13, n° 10, p.591-606, Oct. 1970

[YOU 89] Yourdon E., *Modern Structured Analysis*, Yourdon Press, Prentice-Hall, 1989

[YOU 94] Yourdon E., Object-Oriented Design : an integrated approach, Yourdon Press, 1994

Résumé

Notre recherche se situe dans le domaine des "Environnements Interactifs d'Apprentissage par Ordinateur", et concerne plus spécialement la conception et la production de simulations pédagogiques. Cette thèse se propose d'aborder la définition de nouveaux environnements de production d'un point de vue pragmatique en partant de cas concrets rencontrés dans la formation professionnelle technique en milieu industriel.

Le résultat essentiel de ce travail est la définition du modèle MARS (Modèle-Associations-Représentation-Scénario), cadre conceptuel permettant de structurer l'approche de conception. MARS propose trois espaces de travail dédiés à la spécification du modèle de simulation, du scénario pédagogique, de la représentation et un quatrième espace permettant l'intégration des différents résultats obtenus. La volonté de fournir des solutions opérationnelles nous a amené à définir un processus de développement basé sur le déroulement en parallèle des activités de conception organisées chacune selon un cycle de prototypage automatisé.

Nous abordons également la définition des environnements de production en insistant, dans chaque espace de travail, sur l'importance des interfaces et des concepts manipulés par les concepteurs. Nous définissons l'architecture d'un environnement intégré adaptatif qui permet de proposer différents niveaux d'accès adaptés aux compétences des auteurs, au domaine technique étudié, au contexte de production, etc.

Nous décrivons enfin un exemple d'environnement que nous avons développé pour TPEC, centre de formation de Hewlett-Packard. L'environnement MELISA propose aux responsables de formation technique un ensemble de méthodes et outils pour créer, modifier ou adapter, rapidement et à moindre coût, leurs propres applications de simulation, et résoudre ainsi des problèmes non couverts par les techniques traditionnelles de formation.

Mots-clés : Environnements Interactifs d'Apprentissage par Ordinateur, Simulation pédagogique, Développement par prototypage, Processus de conception automatisé, Approche par objets, Coopération inter-agents, Environnements auteurs, Interaction homme-machine

Abstract

Our research is related to the domain of Interactive Computer Based Learning Environments, and concerns more specifically the design and production of pedagogical simulations. This thesis attempts to approach the definition of new production environments from a realistic point of view by considering real professional technical training situations in an industrial context.

The main result of this work is the definition of the MARS Model (Model-Associations-Representation-Scenario), a conceptual framework that structures the design approach. MARS offers three workspaces dedicated to specifications of respectively the simulation model, the pedagogical scenario and the representation, and a fourth workspace dedicated to integration of the different results. Our wish to propose operational solutions led us to the definition of a development process based on parrallel design activities, each one following an automated prototyping cycle.

We also consider the definition of production environments. We insist, for each workspace, on the importance of interfaces and concepts proposed to the designer. We define the architecture of an adaptative integrated environmnent which gives the possiblity to offer different access levels according to the author's capabilities, the technical domain studied, the production context, etc.

Finally, we describe a specific environment that we developed for the Hewlett-Packard Training Center, TPEC. The MELISA environment offers to technical trainers a set of methods and tools to create, modify or adapt, rapidly and at low cost, their own simulation applications, thereby solving problems not covered by traditional training methods.

Keywords : Interactive Computer Based Learning Environments, Pedagogical Simulation, Prototyping, Automated Design Process, Object-Oriented Approach, Cooperation, Authoring Environments, Human-Computer Interaction.