THÈSE

présentée par

Marie Blandine WU YAO KUANG RASOAVOLOLONA

en vue de l'obtention du

DOCTORAT DE L'UNIVERSITE DE PARIS VIII

Spécialité : INFORMATIQUE

LA ROBUSTESSE DES SYSTEMES AUTEURS MULTIMEDIAS : CONTRIBUTION THEORIQUE ET MISE EN ŒUVRE

Date de soutenance : 01 Juillet 2000

Composition du jury :

Président :	M. Marc BUI
Directeur de thèse :	Mme Violaine PRINCE
Rapporteurs :	M. Guy GOUARDERES
	Mme Jocelyne NANARD
Examinateurs :	Mme Anh NGUYEN-XUAN
	Mme Brigitte de la PASSARDIERE

A ma mère,

Remerciements,

Je remercie très sincèrement les membres du jury qui ont fait un effort formidable afin de trouver ensemble un créneau pour ma soutenance en cette fin d'année universitaire bien chargée :

- *Madame Violaine PRINCE pour ses aides morales afin que je ne baisse pas les bras, pour sa présence quand je reviens, pour sa patience quand je m'éclipse, pour ses conseils pertinents et ses remarques perspicaces. Je lui témoigne toute ma reconnaissance pour la confiance qu'elle m'a accordé en acceptant d'être mon directeur de thèse.*
- *Monsieur Marc BUI, professeur à l'Université Paris 8, pour l'honneur qu'il me fait en présidant le jury.*
- *Monsieur Guy GOUARDERES pour avoir accepter d'être rapporteur de ce travail et pour ses remarques et conseils subtiles. Son soutien et encouragement depuis 1996 à Bayonne, m'ont toujours été d'un grand réconfort.*
- *Madame Jocelyne NANARD pour avoir accepté d'être rapporteur de ce mémoire, pour ses remarques minutieuses et ses suggestions judicieuses.*
- *Madame Anh NGUYEN-XUAN pour l'honneur qu'elle me fait en acceptant d'être membre du jury.*
- *Madame Brigitte de la PASSARDIERE d'avoir consenti à être membre du jury.*

Toute ma gratitude à Mourad CHEBILI pour ses nombreux conseils techniques, pratiques et moraux d'une extrême subtilité, ainsi que pour ses encouragements permanents et ses aides précieuses.

Je tiens aussi à remercier tous ceux qui m'ont accompagné, famille et amis pour leur compréhension, leur soutien inconditionnel, leur petit mot agréable pour m'encourager malgré le refrain presque obsessionnel « je fais ma thèse » à chaque rencontre.

Enfin, ce paragraphe est dédié à ma sœur, à ma mère et à tous ceux que j'aime, présents ou partis trop tôt, et à qui je pense et je penserai toujours.

Table des matières

"Je montre mon travail tout en sachant qu'il n'est qu'une partie de la vérité, et je le montrerais même en le sachant faux, parce que certaines erreurs sont des étapes vers la vérité. »

Robert Musil

1. INTRODUCTION

1.1. PRESENTATION DU SUJET

L'informatique en quelques années, est devenue une discipline incontournable dans presque tous les domaines. Ce succès fait que toutes les catégories socio-professionnelles sont aussi concernées par cette discipline que les informaticiens. Plusieurs recherches sont faites actuellement pour permettre à tout type de population d'accéder et de concevoir des moyens pour partager des informations, pour partager leurs connaissances : recherche linguistique, traitement de la parole, développement des systèmes experts, amélioration des interfaces homme-machine, invention de la domotique, arrivée de la réalité virtuelle, etc. L'émergence de ces domaines de recherche est accélérée par l'arrivée de l'Internet où l'informatique a donné la preuve de son imminence dans les systèmes d'information.

Des recherches dans ce sens ont été faites fin 70-début 80 dans le domaine de l'éducation. Elles ont donné lieu à des systèmes auteurs qui se définissent alors comme des éditeurs de didacticiel, donc des logiciels d'écriture de didacticiel déchargeant l'auteur du travail de programmation informatique, pour lui permettre de mieux se consacrer à la structuration pédagogique [Madaule, 87]. Du développement de ce type de concept se dégage plusieurs branches dont les plus connues sont l'EAO qui devient plus tard l'EIAO et l'AGD.
Depuis, la notion de système auteur a dépassé le domaine de l'éducation. Ainsi, les systèmes auteurs sont appelés aussi environnements d'aides à la conception et au développement [Nanard, 96]. Ces systèmes auteurs peuvent alors être évalués de deux manières : du point de vue du génie logiciel en tant qu'outil de conception ou du point de vue de l'interaction homme-machine.

Mais, l'impact ou le résultat de ces concepts n'est malheureusement pas proportionnel aux efforts et aux mérites des chercheurs et des professionnels qui s'y sont investis. L'une des principales causes de cette difficulté à s'imposer dans ce domaine vient de la robustesse.
La robustesse signifie le caractère de quelque chose de robuste donc de fort, solidement construit et capable de résister à des efforts extrêmes à différentes agressions et altérations.

Robustesse signifie également vigueur. Toutes ces définitions tirées du Petit Robert 1996 et du Larousse Encyclopédique 1996, associées à notre souhait de permettre aux non-informaticiens de concevoir leur propre logiciel pour partager leurs compétences, avec le minimum de contraintes, nous incitent à faire des recherches pour permettre à un système auteur de conserver un comportement conforme aux besoins exprimés en présence d'évènements non prévus et non souhaités.

La complexité des fonctions et des problèmes rencontrés par les personnes initiées ou non à la conception, la distribution physique et fonctionnelle des erreurs et l'hétérogénéité de moyens et de médias mis en œuvre ne facilitent pas l'adaptation des systèmes auteurs à des modifications de structures et à des évolutions de contexte et d'environnement. La robustesse incluant les concepts de fiabilité, de pérennité et de sûreté de fonctionnement est alors un des piliers manquants pour la réussite des systèmes auteurs.

Les moyens technologiques actuels mettent en avant la dynamique de l'interaction avec l'utilisateur pour construire une application multimédia à travers l'exploitation des découvertes faites dans l'intégration en informatique de certaines qualités humaines. Ces dernières concernent surtout la capacité de l'homme pour analyser, décider, anticiper, apprendre et réagir. La prise en considération de ces qualités améliore la robustesse des systèmes auteurs.

1.2. OBJECTIF DE LA RECHERCHE

Le développement correct d'un système informatique augmente les chances de réussite du projet et, par voie de conséquence, la qualité du produit final. Notre travail contribuera à modéliser une solution théorique validée par des simulations et des tests pratiques afin d'obtenir un environnement de création de logiciels robustes non pas en terme algorithmiques mais par rapport à ses qualités pour s'adapter à tout contexte d'utilisation.

Notre motivation consiste alors à trouver un moyen pour que :

- l'utilisateur du système auteur multimédia ne soit pas bloqué pendant la conception de son propre logiciel,
- le logiciel final soit dépourvu d'un maximum d'erreurs,

- les compétences que l'utilisateur n'a pas forcément au niveau programmation, donc ses lacunes soient intégrées de manière progressive.

En effet, tout acte créatif et innovant engendre des risques [Latour,99]. Notre motivation porte alors sur la réduction de l'ampleur de ces risques pendant la phase de conception, convaincues par le fait que la détection et la correction des erreurs humaines ou machines ne doivent pas intervenir après l'écriture du programme mais doivent au contraire jouer dans le processus même de sa création. L'interaction homme-machine, l'évaluation des risques ou plus directement des erreurs et la méthodologie de conception constituent alors l'axe principal de notre étude.

Nous allons adopter une approche émergente car à partir d'une méthode et d'un modèle, nous essayons de faire ressortir le réel : l'erreur sous toutes ses formes. La partie méthode consiste à développer un système sûr de fonctionnement se basant sur :

- La prévention des fautes qui sont les causes adjugées d'une erreur
- La tolérance aux erreurs pour continuer à fournir un service malgré les fautes
- L'élimination des erreurs pour réduire leurs nombres et criticités
- La prévision des erreurs pour estimer leurs présences et leurs conséquences.

De la modélisation de ce système robuste sera concrétisé un outil appelé ECLR pour Environnement de Création de Logiciels Robustes, que l'utilisateur informaticien ou néophyte doit mettre en œuvre en parallèle avec le système auteur pendant la conception de son application. Le programme ECLR fonctionne alors comme un applet autonome et évolutif en fonction du contexte.

L'ECLR peut être assimilé à un compilateur d'idées et de programmation en temps réel et ce, dans un environnement contextuel. De même, il peut être assimilé à un expert informatique qui assiste en permanence l'utilisateur du système auteur pour concevoir son application.

Nous allons distinguer, comme le montre la figure 1-1, deux positions dans lesquelles peut se trouver un individu face à un système auteur : le concepteur du système auteur versus l'utilisateur du système auteur pour concevoir. Le mot conception englobe ici toutes les opérations à effectuer pour réaliser un logiciel, de l'idée de départ pour développer, jusqu'à la

validation. Mais quelle que soit l'étape de conception, à laquelle l'individu concepteur se trouve, il faut remarquer qu'il a toujours un rôle d'AUTEUR.

Pendant notre étude, nous ne nous positionnons pas en tant qu'auteur de niveau 1 donc utilisateur des systèmes auteurs pour concevoir des logiciels destinés à des utilisateurs finaux. Cette position est intéressante à plusieurs degrés : conceptuel, usuel et comportemental. Notre champ d'étude en est alors enrichi car nous faisons face aux erreurs et problèmes qu'un individu peut rencontrer dans le SAM lui-même pendant son utilisation, dans son comportement lors de l'utilisation et de la conception, et dans son évolution dans son rôle d'auteur d'un nouveau logiciel.

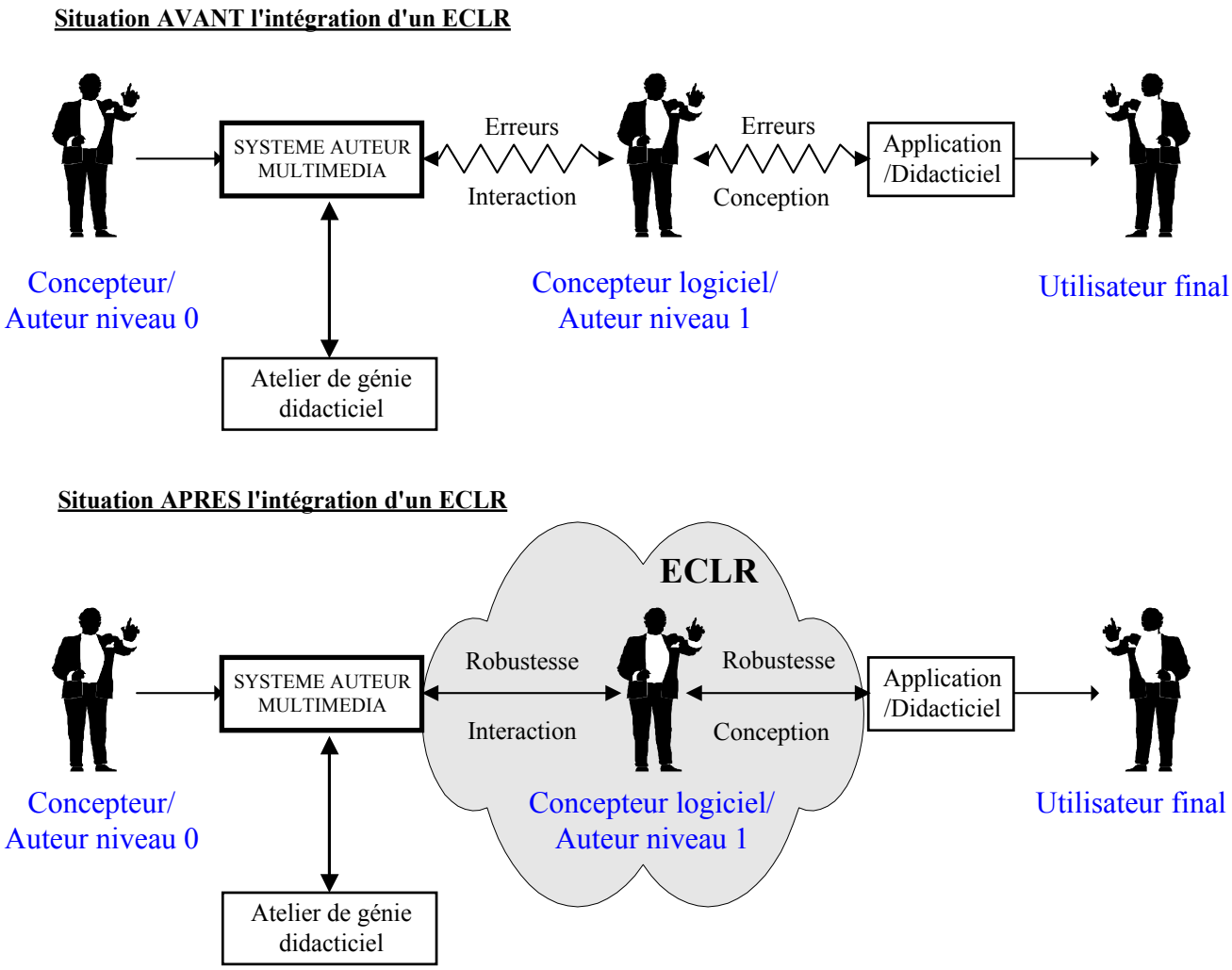


Figure 1-1 : Apport de l'ECLR

L'intérêt de l'ECLR par rapport aux autres outils disponibles pour offrir la robustesse porte sur la dynamique d'une architecture simple et performante qui s'adapte à l'évolution du système, du contexte et de l'environnement. Notre outil présente également l'avantage d'être en adéquation avec le réseau Internet grâce à son architecture qui est inspirée de celle des réseaux TCP/IP et ATM. En revanche, nous n'appliquons pas tous les concepts utilisés dans ces domaines telles que la notion de durée de vie d'un agent ou la notion de transfert d'un agent d'un module vers un autre. Cet outil n'est pas destiné à combler les failles d'un logiciel en terme de sécurité ou de vulnérabilité aux problèmes de protections (intrusion, duplication, crackage…).

Pour mettre en œuvre l'ECLR, nous allons nous appuyer sur les différentes recherches faites sur la fiabilité, les tolérances aux fautes, les niveaux d'organisations dans la société humaine, les systèmes multi-agents , etc. La synthèse et la combinaison de ces domaines nous aident à faire ressortir l'émergence organisatrice entre les composants de l'ECLR et de palier les problèmes d'agrégation de ces composants lors de la constitution d'organisations.

1.3. CONTENU DE LA THESE

Ce document sera constitué de cinq parties :

- La première partie est une synthèse sur les recherches et le statut des systèmes auteurs actuels. Pour ce faire, nous allons commencer par donner la définition et les composants d'un système auteur. Les différentes catégories de systèmes auteurs existants y sont également exposés afin d'établir les critères de sélection utilisés pour évaluer un outil d'aide à la conception d'un logiciel.

Selon le choix effectué, nous allons décrire les avantages et inconvénients de la conception à travers un système auteur. Cela nous autorisera à mettre en valeur notre idée de concevoir un outil pour offrir un environnement de conception robuste. De plus, cet état de l'art sur les systèmes auteurs est essentiel car il permet de mieux cadrer les périmètres d'action de nos travaux. En effet, ce chapitre sera la base de nos inspirations pour développer un environnement permettant à tout individu de concevoir un logiciel avec le moins de contraintes possibles.

• La seconde partie détaille les tenants et aboutissants de notre recherche : créer un environnement de création de logiciels robustes. Nous nous intéressons aux problématiques vis-à-vis de l'erreur de conception et de développement d'un logiciel. Nous adopterons une vision symptomatique de l'erreur pour permettre plus tard à notre outil ECLR d'envisager des procédures de récupération simples de ces erreurs.
Nous allons ensuite définir notre notion de robustesse que nous allons mettre en œuvre dans l'ECLR. Nous verrons que la robustesse est étroitement liée à l'interaction homme-machine et son évaluation est dépendante du niveau de perception et d'impact sur l'utilisateur.
Pour reprendre les termes utilisés en télécommunication, nous y démontrons le fait que la robustesse est un critère fondamental pour assurer la qualité de service ou QoS d'un logiciel dès sa conception.

• Dans la troisième partie, nous allons donner la définition d'un logiciel permettant de concevoir un système socio-technique complexe robuste appelé ECLR pour Environnement de Création de Logiciels Robustes. Le terme environnement est expressément mentionné dans le nom du logiciel afin de rappeler son rôle primordial pour une bonne conception à l'aide d'un système auteur multimédia et par conséquent obtenir un bon logiciel final. Afin de compléter cette définition, nous présentons en détail les entités qui composent et qui définissent un ECLR. Enfin, nous verrons qu'il existe différentes approches qui peuvent être faites par un auteur de niveau 1 dont l'approche cognitive qui nous intéresse particulièrement dans le sens où nous cherchons à créer un logiciel fournissant les mêmes services qu'un expert informatique qui aide à concevoir un logiciel.

• Le transfert d'une fonction cognitive de l'homme vers la machine est appelé communément automatisation [Boy, 95]. C'est ce que nous allons développer de manière théorique dans la quatrième partie de ce document. La modélisation de l'ECLR commence par la plus petite entité qui compose l'ECLR soit l'agent. En effet, nous mettons en œuvre un système multi-agents ou SMA, où nous ferons la distinction entre l'organisation et l'interaction des agents, pour intégrer les qualités d'analyse, de décision, d'anticipation, d'apprentissage présentes chez un expert humain. Nous présenterons ensuite les différentes hiérarchies entre les agents dans le groupe auquel ils appartiennent et par rapport aux autres groupes voisins. Le mode de fonctionnement des composants nous incite à adopter une architecture globale en trois dimensions de l'ECLR.

• Dans la cinquième et dernière partie, la modélisation informatique de l'outil ECLR sera concrétisée par un programme dont nous présenterons les grandes lignes. Nous verrons que les connaissances des agents évoluent stochastiquement malgré les bases de travail fixés dès le début de la conception par l'auteur de niveau 1. Cela impose un comportement heuristique aux agents dans le sens où ils doivent adopter une méthode d'approche progressive afin d'optimiser leur analyse sur la relation cause à effet de chaque erreur. Nous y développerons la résolution distribuée d'erreurs à travers l'architecture fonctionnelle de notre modèle ECLR.

2. ETAT DE L'ART DES SYSTEMES AUTEURS

2.1. INTRODUCTION

Les dernières années ont vu l'apparition de l'outil informatique en enseignement et de travaux en didactique sur l'informatique. Les premiers résultats de ces travaux ont permis à l'E.A.O.[1] de s'instaurer, début des années 70, comme une discipline fondamentale pour le développement de l'informatique dans le système éducatif. Des études sur les méthodes et outils pour la construction de logiciels conviviaux et de qualité ont permis par la suite aux E.I.A.O.[2] [Eloi, 96], génie logiciel et génie didacticiel, à partir de 1975, de se développer.

Le concept de système auteur, toujours dans un but didactique, cherchant à aider les enseignants à créer eux-mêmes des logiciels servant d'appui et de complément à leurs cours, est alors apparu fin des années 70. Le multimédia, né dans les années 80, grâce aux avances technologiques considérables de l'informatique, a donné un coup de pouce au système auteur dans d'autres domaines d'activités. Mais l'intégration du multimédia donne parfois des résultats médiocres essentiellement en raison des manques de pluridisciplinarité des équipes qui ont la charge de la conception et qui ne sont pas assez sensibilisés aux facteurs humains de la présentation de l'information [Nanard, 96].

Un état de l'art sur les systèmes auteurs multimédia, que nous désignons par la suite par le sigle SAM, est nécessaire car il nous permettra de :

- comprendre la manière dont un SAM est conçu,
- prendre connaissance de la variété de SAM existant actuellement,
- définir les critères de sélection d'un SAM,
- comprendre la conception à travers un système auteur.

Ainsi, nous pouvons au cours de notre étude délimiter les périmètres de recherche, détecter plus facilement les problèmes à éviter et combler les lacunes éventuelles d'un SAM.

[1] Enseignement Assisté par Ordinateur

[2] Enseignement Intelligemment Assisté par Ordinateur ou Environnement Interactif d'Apprentissage par Ordinateur

Une présentation des systèmes auteurs multimédias, vus par leurs concepteurs nous permettra de comprendre la manière dont ils sont composés. Suivant la motivation et l'objectif de l'auteur du projet, différentes méthodes de conception peuvent être adoptées. Toutefois, il existe quelques grands traits auxquels les auteurs font très souvent référence. Il est intéressant de les exposer car, par rapport à l'objectif de notre recherche, cette étude nous aide à faire une synthèse sur les méthodes de conception à travers les SAM disponibles actuellement. Cette taxinomie sur les SAM a été possible grâce à différents documents et à des articles publiés dans des revues de recherche, des livres et Internet.

Le premier avantage perçu des systèmes auteurs est qu'il n'est généralement pas nécessaire de connaître un langage de programmation pour pouvoir les utiliser. Cet avantage constitue en fait un inconvénient puisque la simplicité d'utilisation se fait généralement au détriment de la richesse d'expression. Quelques rares systèmes offrent des possibilités de programmation plus étendues, mais leur complexité d'utilisation augmente d'autant [Ibrahim & al, 96]. Nous allons réserver une partie de cet état de l'art pour exposer par catégories les systèmes auteurs existant actuellement. Ensuite, nous présentons les critères de sélection d'un système auteur. En effet, en dehors des soucis d'implémentation du logiciel à partir d'un SAM, l'utilisateur doit effectuer des choix logistiques, fonctionnels et techniques. Nous étudierons la réalisation d'un projet afin d'avoir une vue sur le comportement de l'auteur de niveau 1 face à un SAM. Ainsi, nous pouvons faire une pseudo-comparaison sur la conception de logiciels, des deux types d'auteurs : le premier auteur qui correspond au concepteur du SAM, et le second correspondant à ceux qui utilisent le SAM pour créer leurs didacticiels. Nous nous sommes appuyés sur les critiques et les FAQ[3] pour développer cette partie.

2.2. DEFINITION D'UN SYSTEME AUTEUR MULTIMEDIA OU SAM

La réponse donnée dans les FAQ sur Internet[4] lorsque l'on pose la question « qu'est-ce qu'un système auteur ? » est : un système auteur est une application disposant d'éléments pré-programmés et utilisés pour développer des produits multimédias. Le terme multimédia a été adopté simplement à cause du fait qu'il est difficilement concevable aujourd'hui d'avoir des programmes destinés à des utilisateurs finaux informaticiens ou non, n'utilisant que des

[3] FAQ (Frequently Asked Questions) : Liste des questions-réponses sur un sujet déterminé et publiées sur Internet.

[4] http://www.tiac.net/users/jasiglar/MMASFAQ.HTML

données textuelles, et donc ne profitant pas des différentes possibilités et progrès offerts par la technologie [Gouyet & al, 95]. En revanche, multimédia implique aussi travail d'organisation et de programmation supplémentaire pour le concepteur et l'utilisateur d'un système auteur.
Cette définition est complétée par la théorie suivante : un système auteur permet à des gens qui ne savent pas programmer, ou qui veulent gagner du temps dans la programmation, ou intéressés par la programmation, de créer des logiciels [Ackerman, 92].

Mais ces définitions nous paraissent incomplètes aujourd'hui. En effet, pour permettre à un utilisateur profane ou néophyte en informatique, de créer une application à part entière, dans un domaine spécifique, il faut trouver des algorithmes capables de s'adapter à toutes les manipulations possibles de l'utilisateur, des interfaces homme-machines non bloquantes, bref un système produit avec un très haut niveau de conception. De même, le multimédia ne sera avantageux pour un utilisateur que si le besoin d'avoir des images et sons est incontournable. En effet, le multimédia utilisé abusivement peut pénaliser la performance du logiciel. Il est impossible de pouvoir satisfaire chaque individu susceptible de travailler sur le système auteur, d'autant plus que le terme système auteur lui-même peut être interprété ou défini différemment. Tout individu est apte à donner sa propre définition sur ce terme, surtout qu'un système auteur est sensé, par définition, être exploitable par tout type d'utilisateur. Aussi, allons nous donner notre définition du terme système auteur par rapport au cadre de notre étude : ***un système auteur multimédia est un logiciel[5] permettant à tout individu souhaitant partager ses connaissances de le faire en lui facilitant la création de logiciels éducatifs multimédias.***

Population cible

L'informatique d'aujourd'hui n'est plus réservée à certaines catégories d'individus, d'ailleurs on parle de la pluridisciplinarité de l'informatique. Un système auteur, est par définition, destiné à tout individu souhaitant réaliser un logiciel, ce qui confirme cette pluridisciplinarité. Aujourd'hui, les plus intéressés sont principalement :
Les entreprises comme les industries de construction automobile, les sociétés travaillant sur l'enseignement assisté par ordinateur ou EAO et le multimédia : cette attraction vers les systèmes auteurs est justifiée par le gain de productivité, et des résultats répondant au marché

[5] « Un logiciel est l'ensemble des programmes, procédés et règles, et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitement de l'information » arrêté ministériel du 2 décembre 1981.

actuel. Avec un système auteur, pour un non informaticien, le temps d'apprentissage et de développement est inférieur au temps requis en utilisant un autre système de programmation pour développer certains logiciels multimédias. Nous pouvons penser par exemple à la création d'un bouton, émettant un son lorsque nous cliquons dessus, avec le logiciel Director. Un informaticien de base, ne connaissant pas le logiciel Director a mis environ 2 minutes pour réaliser cette tâche. Il aurait mis au moins 15 minutes pour réaliser l'équivalent en langage C. Un non informaticien, a mis 15 minutes environ avec Director, alors qu'avec le langage C, il aurait fallu qu'il apprenne les bases (fonctions, boucles, compilations, etc.), et leurs syntaxes. Cela nécessitera plusieurs heures voire des semaines de travail.
Les universités de différents pays, comme Stanford (USA) ou Laval (Canada), se penchent sur ce sujet depuis quelques années, et cela en rapport à des recherches informatiques, médicales, enseignements ou autres. L'intérêt vient généralement du souci du partage des connaissances avec les moyens modernes. D'ailleurs, un système auteur est considéré parfois, comme une tentative d'industrialisation de certaines productions comme les didacticiels, avec des outils performants [Bessière, 88].

Afin de pouvoir profiter des progrès effectués en multimédia, un SAM exige au moins la configuration suivante :

- un écran graphique couleur,
- 32 Mo de mémoire vive étant donné que les systèmes d'exploitation (Windows, Unix) sur lesquels s'appuieront les logiciels de SAM, exigent au moins cette capacité.
- Une grande capacité de disque dur pour le stockage de fichiers images (30 secondes d'images animées non compressées nécessitent 1 Mo de place disque), sons ou textuels.
- Des périphériques spéciaux comme une carte son, des enceintes, un scanner, etc.
- Des logiciels de traitement d'images, de graphes et de sons.

2.2.1. Objectifs principaux d'un SAM

Pour inciter les individus ayant des compétences dans divers domaines à créer des logiciels, il faut mettre à leur disposition des outils complets, simples et utilisables par des auteurs n'ayant en principe aucune compétence informatique [De La Passardière, 85]. Par conséquent, étudier les problématiques dans le but d'utiliser un système auteur, signifie entre autres, chercher et prendre en considération tout critère pouvant mettre l'utilisateur dans une mauvaise situation.

Plusieurs attributs susceptibles d'influencer une méthode de développement, quel que soit le type et le niveau de l'utilisateur du système auteur, doivent être étudiés soigneusement et considérés par le concepteur. Nous allons présenter ci-dessous les critères que nous considérons comme principaux, d'après nos expériences et études :

- ***La convivialité*** : La première approche avec un logiciel a un rôle psychologique fondamental pour tout utilisateur, même chez les informaticiens confirmés. En effet, si l'écran ou la présentation du logiciel n'est pas soigné, l'individu peut être plus ou moins bloqué pour la suite. L'importance de l'interface homme-machine justifie la généralisation des icônes et l'intégration du multimédia dans les SAM. La transposition graphique, au sens visuel, est incontournable quel que soit le domaine, pour celui qui veut projeter son idée, faire comprendre, expliquer, fabriquer ou modéliser [Bourron & al., 95]. Le concepteur doit par conséquent attacher une attention particulière à la présentation de son produit.
- ***La transparence*** : La gestion des données, et le fonctionnement interne au point de vue relation entre les différents éléments (variables, objets, etc.) doivent être complètement transparents à l'utilisateur. Certes, la sélection des données à inclure dans le logiciel construit à partir du système auteur lui est réservé, mais s'il doit y avoir des compressions (fichiers images par exemples) ou autres traitements (emplacement sur disque des données), l'utilisateur ne doit pas se soucier ni du moment ni de l'endroit, ni du type de fonctions et librairies à inclure et à lancer pour l'exécution de ces diverses tâches. En effet, si l'utilisateur doit comprendre le fonctionnement des registres de données, la notion de système auteur n'a plus lieu d'être !
- ***L'assistance*** : Comme un SAM est supposé, entre autres, être un outil de programmation pour les non informaticiens, il est fondamental d'assurer un minimum d'aide en ligne ou au moins une partie explicative des principales tâches ou commandes ou icônes du système. Même pour des tâches devenues banales comme la saisie de données textuelles, le système doit indiquer au moins à l'utilisateur dans quelle fenêtre et quand il peut le faire.
- ***L'interactivité*** : La communication entre le système auteur et son utilisateur, souvent sous la forme d'une assistance, test et contrôle automatique des tâches de l'utilisateur, doit être particulièrement soignée car elle est vitale pour l'utilisateur et pour la réussite sur le marché du produit. Un système auteur doit intéresser son utilisateur et non l'ennuyer.
- ***La fiabilité*** : Le système doit être le moins bloquant possible quelle que soit l'action menée par l'utilisateur sur le logiciel. Ce dernier doit pouvoir revenir en arrière, modifier

ou supprimer sans difficulté tout composant de son logiciel. De même, il doit pouvoir assembler les pièces qui constituent son application dans un ordre pas trop contraignant, indépendamment les uns des autres et ce à n'importe quel moment du développement.

- ***La précision*** : Les fonctions spécifiques relatives à l'objectif du SAM doivent être claires et précises. Cela permet de minimiser des oublis et de prévoir les impossibilités techniques éventuelles lors de la phase de programmation.

2.2.2. Les principes de conception offerts par un système auteur multimédia

Un système auteur doit pouvoir différencier et traiter séparément les données de base du SAM et celles que l'utilisateur inclura dans son produit final. Un système auteur, comme tout logiciel, doit pouvoir être modifié ou amélioré constamment afin de l'adapter aux nouveaux besoins et le faire évoluer indépendamment des données mises par l'utilisateur. La programmation d'un système auteur nécessite alors une algorithmique et une méthodologie de conception pointues, surtout pour le traitement des données.

Il n'existe pas beaucoup de méthodes permettant de concrétiser cette fonction de gestion des données à travers un programme. En effet, aucune méthode de développement complète entièrement graphique n'est actuellement disponible bien qu'un graphique exprime autrement qu'un langage textuel les enchaînements d'actions séquentielles et conditionnelles qui caractérisent la programmation [Bessière, 88]. Les deux méthodes les plus couramment utilisées sont :

- Une programmation avec les méthodes traditionnelles ou structurées.
- Une programmation avec des méthodes plus évoluées, modulaires mais plus complexes, se basant sur la méthode objet.

Mais quelle que soit la méthode de programmation adoptée, le fait d'intégrer des fonctionnalités multimédia, acte important dans le cadre de notre étude, influence systématiquement le résultat final, c'est-à-dire le SAM.

2.2.2.1.Conception traditionnelle

Beaucoup de systèmes auteurs multimédias ont été développés avec les langages de base comme Pascal ou plus encore, le langage C. Le programme est alors constitué par diverses *fonctions* ou *procédures* qui cohabitent suivant un algorithme bien déterminé. Cette méthode

est plus facile à acquérir pour un individu car le programme est organisé de manière *séquentielle* donc, le développeur peut garder une ligne directive qui définit la *structure* de son projet. Il sait alors quand, comment et où ses données vont être traitées.
En revanche, lorsqu'il s'agit de gérer simultanément plusieurs bases de données, indépendantes les unes des autres, et ce à différents niveaux, cette méthode de développement peut devenir obsolète. Les données relatives au SAM, donc au niveau du logiciel lui-même, sont les icônes, les liens et les différents outils que manipulera l'utilisateur pendant la conception de son logiciel. Les données des utilisateurs, à un niveau supérieur, sont ici les différents textes, sons et images à mettre en œuvre pour le fonctionnement de leurs logiciels construits avec le système auteur.
Parfois, cela peut consister à créer une certaine interaction entre des bases de données de différents niveaux. A titre d'exemple, nous pouvons penser que le traitement du son inclus dans le système utilise les mêmes outils ou programmes que les sons mis volontairement par l'utilisateur du SAM. La solution est alors de considérer chaque base, voire chaque donnée comme un objet.

2.2.2.2.Rapprochement entre SAM et Programmation orientée objet[6]

La conception objet a permis de faire un grand pas dans l'évolution des systèmes auteurs car les graphiques, les icônes, le son ou autre composant des systèmes auteurs multimédia actuels sont manipulés et conçus comme des objets. Dans le cadre de notre étude orientée vers l'enseignement, la méthode de programmation par objet semble être la meilleure solution disponible aujourd'hui. La caractéristique de l'éditeur pédagogique est d'appliquer la désignation à des objets logiques des actions informatiques [Bessière, 88].

La programmation orientée objet ou POO est devenue rapidement la méthode la plus utilisée pour développer des SAM. Nous retrouvons les concepts principaux de la POO :

- ***L'objet*** : virtuellement, pour un utilisateur tout composant de son futur logiciel est un objet. Les données et les procédures sont regroupées dans une même entité appelée objet [Masini & al, 89]. Un SAM peut être considéré comme un assemblage d'objets. En effet, il doit considérer que tout ce qui sera manipulable par l'utilisateur est un objet car « an object is a region of storage » [Stroustrup & al, 90]. L'utilisateur peut considérer que le

[6] Notée POO par la suite.

bouton image est un objet qu'il peut ensuite associer à un autre comme l'objet son. Pour le concepteur, les boutons son et image représentent deux classes différentes, regroupant chacune plusieurs fichiers images (*.gif, *.bmp, *.jgp ou autres) ou son (*.mid ou *.wav) et des fonctions permettant l'affichage ou l'utilisation de ces fichiers à des instants donnés. L'affectation d'un objet à un autre objet qui veut dire en POO la recopie des valeurs des champs de données (sauf pour les pointeurs) est associé à la duplication d'un objet par l'utilisateur du SAM. L'inverse de cette action de duplication consiste à détruire automatiquement les objets temporaires créés lors d'un appel au constructeur de l'objet dès qu'ils sont inutiles.

- ***L'encapsulation*** : afin que les objets puissent être associés les uns aux autres et avoir leur vrai sens en POO, il faut penser à leur encapsulation, ce qui signifie que l'accès à chaque objet ne peut se faire que par le biais de procédures ou ***méthodes*** pré-définies. Cela est représentée par des macro-instructions mises à la disposition des utilisateurs d'un SAM. Ces macro-instructions décrivent les méthodes pour accéder à chaque objet.
- ***La classe*** : c'est la description d'une famille d'objets ayant une même structure et un comportement identique [Masini & al, 89]. Il s'agit d'une généralisation de la notion de type définie par le programmeur, dans lequel sont associées des données et des fonctions. Comme en POO, le concepteur comme l'utilisateur du SAM peut déclarer publique ou privée une classe afin de protéger certaines données. La différence vient du fait que l'utilisateur n'écrit pas un programme et donc n'a pas à savoir des syntaxes parfois compliquées comme cette simple déclaration d'une classe en C++ :

```
class image
{
int x ;
int y ;
public :
void deplacer (int, int) ;
void afficher () ;
private :
void sauverchoix (int, int) ;
}
```

Figure 2-1 : Création d'une classe en POO

En revanche, la notion de constructeur et destructeur en POO peut être rapidement ambiguë pour l'utilisateur du SAM qui prendra ces termes dans leur sens courant (langage

naturel). Cela n'est pas un mauvais côté car ces concepts lui auraient vite compliqué la tâche pour construire son logiciel. Pour le développeur du système, l'auteur de niveau 0, ces notions gardent leurs sens et propriétés comme en POO.

- ***L'héritage*** : ce concept, un des fondements de la POO, permet de dériver à partir d'un objet parent une hiérarchie d'objets descendants, qui héritent des fonctions du parent. La relation d'héritage permet ainsi d'organiser les classes hiérarchiquement car une classe hérite alors des informations (variables et méthodes) d'une classe supérieure ou superclasse. Cela permet une représentation graphique facile à mettre en œuvre, souvent sous forme d'un arbre dont la racine est la classe la plus générale, détenant le comportement commun à tous les objets. Une classe qui hérite de plusieurs superclasses relève de la notion d'héritage multiple qu'il faut utiliser avec précaution en matière de gestion.

  Dans un SAM, la notion d'héritage est très utilisée dans le sens où chaque classe créée par l'utilisateur du SAM hérite des propriétés des classes incluses dans le système. Nous pouvons penser par exemple à une classe « photos » ayant comme superclasse « imagesfixes » et des objets possédant les variables « nom », « titre », « ordonnée » et « abscisse ». Les deux derniers objets ordonnée et abscisse sont hérités de la classe « imagesfixes ».
- ***Le polymorphisme*** : signifie une fonction ayant un seul nom, mais qui a différents effets selon les objets auxquels elle est associée. Dans un SAM, une fonction « retour » peut, par exemple, signifier « ligne précédente » pour un texte, « secteur précédent » pour une image animée.

2.2.2.3.Interface multimédia

Chaque projet multimédia est « *comme un petit sommet des Nations Unies* » [Salamone, 96] car la conception d'un projet multimédia regroupe en général plusieurs types de profils comme : les graphistes, les directeurs de projet, les producteurs de scénarios animés et les programmeurs.

Les concepteurs de produits multimédias utilisent en général trois outils pour regrouper dans un seul logiciel les résultats donnés par ces différents profils :

- Les *organigrammes* qui représentent ce qui va se passer pendant l'exécution du programme et non comment va se présenter celui-ci. Un organigramme permet d'avoir une vue de la disposition des procédures, des décisions et leurs finalités.
- Les *scénarios* que le concepteur rassemble et modifie à sa guise. Chaque composant du produit final est associé à une trame. Ces trames sont ensuite rassemblées dans un des différents scénarios composants le produit final. Les SAM permettant de construire des scénarios utilisent souvent le système de « drag and drop », facile à utiliser pour constituer et regrouper les trames.
- Les *outils auteurs* qui gèrent les composants du produit multimédia dans une base de données que le programmeur utilise pendant le développement de l'application. Ces outils sont par exemple des outils de traitement d'images animés ou de mixage de sons. Ils sont généralement intégrés dans la plupart des SAM afin d'enlever les soucis de l'utilisateur sur tout ce qui compose le multimédia.

Du point de vue psychologique, des études ont montré qu'il est nécessaire de mettre en jeu plusieurs moyens de communication (visuel, auditif, etc.) pour améliorer l'acquisition de connaissances [Bourron & al, 95]. Le multimédia est alors porteur de sens, en plus de son côté attrayant.

Mais, la richesse des médias peut malheureusement donner un effet d'alourdissement de la charge de travail pour l'utilisateur d'un SAM. De même, une mauvaise utilisation peut être dangereuse car elle peut détourner complètement le plan de travail initial. Une mauvaise association des médias risque également de compromettre le résultat final. Il faut alors que l'esthétique reste fonctionnelle et non pas un simple gadget. Le concepteur doit par conséquent être vigilant dès le début, dans le choix de l'interface homme-machine qu'il veut mettre en place.

2.3. TAXINOMIE DES SYSTEMES AUTEURS OU PARADIGMES AUTEURS

Chronologiquement, en dehors du langage Assembleur que seuls des informaticiens spécialisés dans les applications de couches basses maîtrisent actuellement, les langages de programmation tels que Pascal, C, LISP, etc. sont les premiers outils de développement de systèmes auteurs.

Un langage de programmation permet à tout individu de créer des logiciels. Mais cela nécessite souvent un long parcours d'apprentissage. Différentes méthodes sont étudiées et

mises en exploitation pour pallier les problèmes de contrainte d'apprentissage en programmation.

La capacité technique et les fonctions offertes par les systèmes auteurs multimédias ont permis de les classer en plusieurs catégories. Nous les appelons également paradigmes auteurs dans le sens où, ces catégories de SAM peuvent être substituées les uns aux autres dans un contexte donné.

Nous allons ensuite les classer suivant un ordre que nous considérons croissant selon les objectifs principaux d'un SAM [7], c'est-à-dire : la convivialité, la transparence, l'assistance, l'interactivité, la fiabilité et la précision.

2.3.1. Les langages de niveau 1

Ils sont souvent désignés par le nom de *3GL* ou Third Generation Language, et les plus connus sont le C, le Pascal ou le Basic. Les bibliothèques de fonctions ou de macro-instructions permettent à l'utilisateur d'éviter certains soucis ou détails de programmation parfois difficilement réalisables comme la gestion de l'entrée-sortie standard ou bien la gestion de plusieurs fenêtres.

Ce sont des langages souvent assez complets et qui offrent une certaine flexibilité. Ils sont moins proches de tout ce qui est matériel (hardware), contrairement à l'assembleur que nous appelons langage de niveau 0. Des connaissances informatiques et algorithmiques sont nécessaires pour arriver à constituer un bon programme à partir de ces langages. En effet, ils se présentent intérieurement eux-mêmes en tant qu'une couche entre le système d'exploitation et l'utilisateur. Leur avantage principal vient du fait qu'ils ne sont pas dédiés à des objectifs donnés, c'est-à-dire que l'utilisateur de ce type de SAM pour concevoir, peut réaliser un logiciel quel que soit son objectif, son domaine et la population visée.

Mais la richesse de leurs bibliothèques de fonctions malgré la complexité de leurs syntaxes et la lenteur en terme d'acquisition de savoir-faire en développement, nous poussent à les considérer comme des langages auteurs. L'orientation objet de ces langages de programmation

[7] Voir chapitre 2.2.2.2

de niveau 1, n'a malheureusement pas réussi à simplifier leur utilisation ni l'approche à la programmation d'un non informaticien.

2.3.2. Les langages générateurs de codes

Les générateurs de codes sont également appelés progiciels ou langages de script. Les programmeurs utilisent des outils souvent présentés sous formes de menus textuels. Chaque composant d'un menu est un ensemble de programmes, souvent écrits avec des langages de plus bas niveaux, beaucoup plus complexes syntaxiquement. Ces progiciels sont souvent spécifiques à un domaine comme la gestion par exemple. Cela est dû au nombre limité des menus ou des macro-instructions mis à la disposition de l'utilisateur pour programmer.

L'avantage de ces générateurs de codes est le gain de temps en apprentissage et en développement. Ils peuvent également être enrichis par des librairies 3GL externes afin de pallier au problème de limitation en commandes et donc de fiabilité. Mais cela exige une connaissance d'un langage 3GL ou du moins de la manière d'intégrer les outils et ne suffit malheureusement pas à combler cet inconvénient majeur.
Voici le programme nécessaire en langage C, placé dans la catégorie langage de niveau 1, pour créer une fenêtre contenant un bouton qui permet de détruire l'ensemble.

```
/* Appel à des librairies de fonctions pré-définies */
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <X11/Xlib.h>
#include <stdio.h>
/* Déclaration des variables globales */
char *getenv();
int screen;
Display *d;
Window fenetre;
XEvent e;
GC gc,gc1;
XFontStruct *fd,*fd1;
XGCValues gcv,gcv1;
XColor gris,grise;
int main()
{
        /* ouverture du display */
        if((d = XOpenDisplay(getenv("DISPLAY"))) == NULL){
        printf("Impossible de contacter le serveur\n");
```

```
exit(1);
}
/* allocation de la couleur */
XAllocNamedColor(d,DefaultColormap(d,screen), "grey70",&gris,&grise);

/* création de la fenêtre racine */
fen = XCreateSimpleWindow(d,
    RootWindow(d,DefaultScreen(d)),0,0,50,50,10,WhitePixel(d,DefaultScreen(d)),gris.pixel);

/* création d'une fenêtre fen */
ecr = XCreateSimpleWindow(d,fen,30,10,
    30,50,2,BlackPixel(d,DefaultScreen(d)),WhitePixel(d,DefaultScreen(d)));
/* affichage de la fenêtre */
XMapWindow(d,fen);

/* création des contextes graphiques */
gcv.font = fd1 -> fid;
gcv.foreground = WhitePixel(d,DefaultScreen(d));
gcv.background = BlackPixel(d,DefaultScreen(d));
gcv1.font = fd -> fid;
gcv1.foreground = BlackPixel(d,DefaultScreen(d));
gcv1.background = WhitePixel(d,DefaultScreen(d));
gc = XCreateGC(d,fen,GCFont | GCForeground | GCBackground,&gcv);

/* sélection des événements possibles dans la fenêtre */
XSelectInput(d,fen,ExposureMask |ButtonPressMask | ButtonReleaseMask);

/* sélection des événements */
for(;;){
        XNextEvent(d,&e);
        switch(e.type) {
                default:break;
                /* à l'ouverture de la fenêtre */
                case Expose:
                XDrawString(d,fen,gc,260,60, "***MA FENETRE***", strlen("***MA FENETRE ***"));
                /* gestion de la souris */
                case ButtonPress:
                if(e.xbutton.window == non){
                        XdestroyWindow(d,fen);
                        XcloseDisplay(d);
                        exit(0);
                }
                break;
        }
```

```
        }
}
```

Figure 2-2: Code de création d'une fenêtre en langage C

Il faut environ 50 lignes d'instructions pour créer une fenêtre en langage C. Nous pouvons constater immédiatement, en regardant les lignes d'instructions tk/tcl, un langage générateur de code sous l'environnement graphique X11 d'Unix, la différence de complexité, de gain de temps et d'approche pour réaliser la même fenêtre.

```
#!/usr/local/bin/wish –f
#
button .button \\
-text "***MA FENETRE***" \\
-bg "#cccccc" \\
-command \\
{
        # detruire tout et quitter
        destroy .
}
# mettre le bouton dans la fenetre root
pack append . .button {expand}
```

Figure 2-3 : Code de création d'une fenêtre en TK/TCL

De ces deux illustrations, nous pouvons noter que le fait de travailler dans les détails pour chaque action comme l'exige le langage C n'améliore pas forcément la compréhension du concepteur. Au contraire, cela risque de l'embrouiller rapidement, sans compter le temps requis pour l'apprentissage et le développement.

2.3.3. Les langages de programmation visuelle

Les langages de programmation visuelle comme le Visual Basic ou 4ème Dimension, simplifient les tâches du développeur dans le sens où ils peuvent avoir une vision globale, pendant la programmation, du résultat de leurs codes. L'avantage vient surtout dans le gain de temps pour réaliser chaque interface directement à l'écran à partir d'un éditeur simple à utiliser, et de codes faciles à utiliser souvent constitués de générateurs de codes 3GL ou de macro-instructions. L'intégration de macro-instructions écrites avec des langages 3GL externes au langage est possible. Ce type de langage intègre progressivement les atouts du multimédia et est surtout exploité pour les logiciels accordant une grande importance à l'interface homme-machine.

Par exemple, pour construire dans l'environnement Access, une fenêtre que l'on peut re-dimensioner à sa guise avec la souris, il suffit de cliquer sur le menu Insertion et Feuille, pour obtenir l'écran suivant :

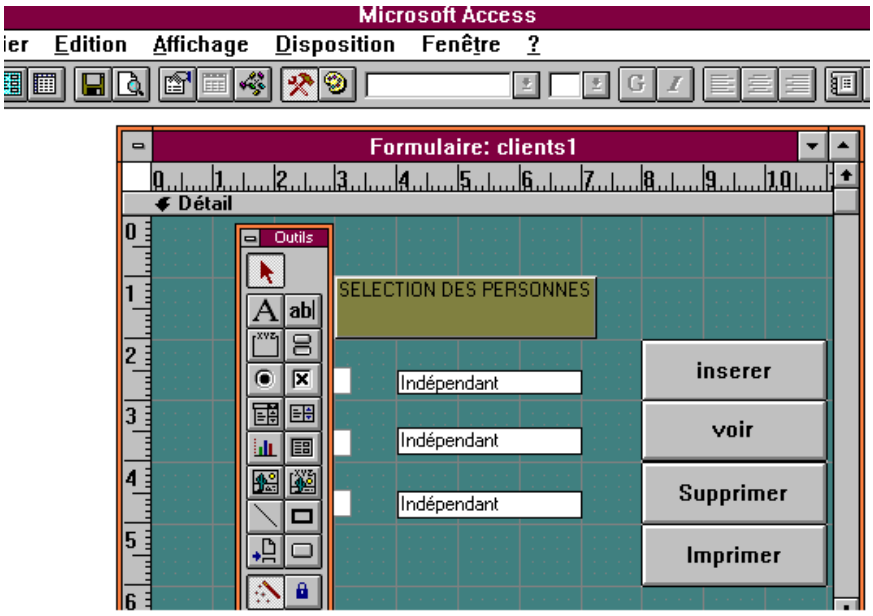


Figure 2-4 : Fenêtre initiale pour la création d'un programme Access

Un autre exemple intéressant concerne le langage 4ème Dimension. Ce langage de programmation visuelle, permet à l'utilisateur d'avoir le schéma global de la structure de son programme, avec les liens qui mettent automatiquement les différents éléments le constituant. Grâce à ces liens, l'utilisateur connaît les conséquences de chaque action sur une rubrique.

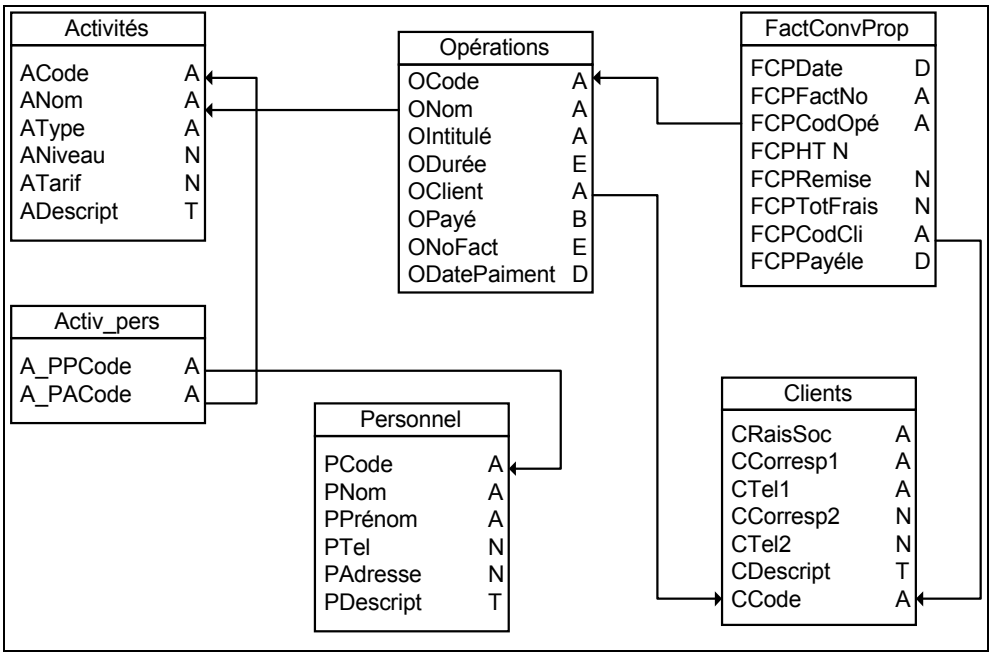


Figure 2-5 : Structure d'un programme en langage 4D

Exemple, à une opération est associée un nom qui fait référence à une activité détaillée dans la table « Activités » dont les détails sont décrits dans une autre table « Clients ». Ainsi, lors de l'utilisation de ce logiciel, on ne peut saisir une opération si le client associé n'existe pas.

2.3.4. Les métaphores à carte

L'intégration et l'interaction des médias se font autour du concept de la carte qui représente alors l'unité logique de travail. Le fonctionnement de l'application est centré sur des liens entre des objets situés sur une même carte ou sur des cartes différentes. Les applications les plus connues des métaphores à carte sont l'hypercards et l'hypertexte.

Les **hypercards** sont utilisés pour développer des logiciels non multimédia rapidement et facilement. Les logiciels sont développés à partir de fiches ou cadres (*card* ou *frame* en anglais) qui sont des graphes indépendants les unes des autres, ayant des propriétés différentes, et que l'on empile (*stack*) les uns sur les autres de manière cohérente.
Pour réaliser son application, le développeur doit parfois écrire des lignes d'instructions ou scripts en langage 3GL qu'il associera à des événements comme le clic ou le mouvement d'une souris sur la fiche par exemple. Généralement, les actions ou événements les plus courants sont automatisés, par exemple lors de la construction d'un bouton, les scripts pour cliquer sont déjà intégrés dans ce bouton.

Les ***hypertextes*** sont très proches des hypercards sauf qu'ils utilisent des textes au lieu de graphes et images. Cette catégorie de système auteur est surtout exploitée pour réaliser des systèmes d'aides dans des logiciels comme Microsoft Word, ou des systèmes d'exploitation comme Windows 95. La méthode hypertexte[8] consiste à construire plusieurs fichiers que l'on peut appeler (afficher à l'écran) automatiquement grâce à des liens représentés par un ensemble de mots mis en évidence (soulignés, en gras ou en couleur) cliquables contenus dans le fichier en cours d'utilisation.

[8] Le concept d'hypertexte a été décrit par Vanner Bush en 45 et revu par Ted Nelson dans les années 60. Cela démontre l'importance de la convivialité et de l'interactivité en informatique.

Le succès des hypertextes a été démontré par leur généralisation dans les menus d'aides mais également par Internet, car le langage HTML utilisé pour construire des pages Web, n'est rien d'autre qu'un hypertexte plus complexe et enrichi (gestion des images, sons, etc.).

2.3.5. Les montages par diagramme d'icônes

Les langages de programmation avec des icônes sont les plus évolués pour aider les non-informaticiens à créer des logiciels. Les écrans et objets y sont liés et s'enchaînent en fonction d'un organigramme défini et par conséquent autorisé par le concepteur du SAM. En effet, le développeur utilise les icônes pour constituer l'ossature de son programme, donc la succession des actions souhaitées. Cela est un avantage considérable pour un non informaticien qui sera naturellement moins bloqué face à une palette d'icônes qu'à un langage de programmation avec des codes textuels.

Les SAM offrant des palettes d'icônes peuvent réaliser des tâches complexes. Cela n'est pas forcément un avantage, car l'utilisateur doit consacrer un temps d'apprentissage conséquent dès qu'il veut réaliser des actions élaborées. En plus, selon leur capacité technique, ces systèmes sont souvent très chers, nécessitent de grande capacité machine donc, souvent accessibles uniquement aux professionnels. Les SAM utilisant des icônes possèdent souvent les avantages des autres catégories de SAM citées ci-dessus, à savoir : la possibilité de tester pendant le développement, de rajouter des macro-instructions externes, d'intégrer des opérations multimédias, etc.
Un système auteur utilisant exclusivement des icônes, malgré tous ces avantages, est plutôt utopique et même déconseillé car il sera moins riche potentiellement dans le sens où l'utilisateur sera limité par le nombre de fonctions proposées avec les icônes qui sont également en nombre limité.

2.3.6. Evaluation des paradigmes auteurs de quelques SAM

Cette étude des différentes catégories de langage de programmation, ainsi que nos différentes expériences en programmation, nous permettent de donner une évaluation relative à plusieurs critères relatifs à l'interaction homme-machine à savoir :

- l'interface homme-machine qui se décline en terme de convivialité qui regroupe les critères de visibilité, de transparence, de prévisibilité, d'intuition, de cohérence, de

concision, d'adaptativité et d'adaptabilité [Meinadier, 91], d'ergonomie, de simplicité d'utilisation, d'assistance et de navigation pour accéder à des fonctions du SAM.

- les potentialités fonctionnelles du SAM c'est-à-dire la puissance du langage de programmation, la précision des actions, les variétés de fonctions mises à la disposition de l'utilisateur. Bref, il s'agit de la capacité du SAM à permettre à l'utilisateur de développer toutes les fonctions qu'il souhaite, et offrir plusieurs choix pour ce faire.

L'attribution des valeurs (faible, moyenne ou élevée), dépend du niveau de puissance que le SAM offre et les types d'applications qu'il permet de réaliser.

Catégorie	**IHM**	**Potentialité**
LP* de niveau 1	Faible	maximale
LP générateurs de code	Faible	élevée
LP visuels	Moyenne	élevée
LP à liens	Elevée	moyenne
LP avec des icônes	Elevée	moyenne

* LP : Langage de Programmation

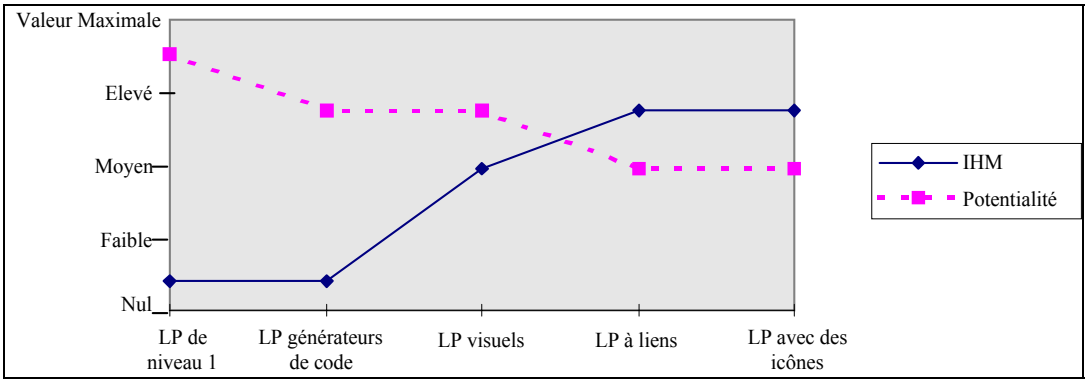


Figure 2-6 : Evaluation des paradigmes auteurs

Catégorie	**Système d'exploitation**	**Nom**	**Particularités**
LP (1) Générateurs de code	Unix	tk/tcl	Génère des codes en C, C++
	Windows/Dos	Protogen	Génère des codes en Turbo C++
		Boilerplace	Génère des codes en Turbo Pascal
LP Visuels	Unix	tk/tcl	Principe identique au shell d'Unix

	Windows/Dos	Visual Basic (Microsoft)	Création des interfaces visuelles et en direct
	Macintosh	4ème Dimension (ACI)	Schématisation de structure du programme
LP à liens	Unix	HyperAuthor	Hypertexte et hypercard
	Windows/Dos	EasyHelp (Eon Solutions)	Aide en ligne (2)
		Tollbook (Asymetrix)	Hypercard multimédia
	Macintosh	HyperCard (Apple)	Facilité d'utilisation
LP avec icônes	Unix	Icon Author (Aim Tech)	Multimédia(3), déboggeur (4), Internet(5), base de données (6), système expert (7), etc.
	Windows/Dos	Icon Author (Aim Tech)	
		Authorware (Macromédia)	Richesse de fonctions et opérations proposées
		Director (Macromédia)	Logiciels multimédia, déboggeur, Internet
		ScrMach (Owen & Co)	Ecran de veille (8)
	Macintosh	Icon Author (Aim Tech)	

Figure 8 : Exemple de systèmes auteurs

(1) LP : Langage de Programmation

(2) Aide en ligne : Le LP permet de créer des fichiers d'aide que l'on peut consulter à partir d'un menu. Selon la puissance du SAM, on peut faire des indexations ou des recherches sur les fichiers.

(3) Multimédia : Le LP permet de créer tout type de logiciel multimédia. Quand le LP est spécialement orienté vers un domaine précis, le nom du domaine est souvent mentionné, exemple EAO multimédia.

(4) Débogueurs : le LP est enrichi d'une fonction de détection et correction d'erreurs de codage. Cette fonction se résume souvent en la signalisation à l'utilisateur des erreurs rencontrées, avec des précisions sur la cause et la localisation dans le programme.

Internet : Le LP permet de concevoir des applications multimédia interactives, exportables sur Internet. Cela suppose également une architecture ouverte du système de stockage du contenu de l'application dans le sens ou les différents fichiers (pages) peuvent être localisés dans différents endroits.

Base de données : Le LP grâce à la séparation de la structure et du contenu permet à l'utilisateur d'interroger et de modifier une base de données à travers une ou plusieurs interfaces à construire librement.

(7) Système expert : Le LP permet la gestion d'un élève (suivi, résultat, progression) à travers des exercices pré-construits (vrai/faux, association, choix multiples, réponses ouvertes).

(8) Ecran de veille : Le LP permet de créer librement un écran de veille multimédia.

2.3.7. Critères de sélection d'un système auteur

Les utilisateurs souhaitant concevoir des logiciels rapidement et facilement, font généralement appel à un système auteur suite à une rapide démonstration ou lecture d'un article ou par un commercial. Actuellement, ce sont des instituts (entreprises ou universités) spécialisés en informatique ou dans l'enseignement, qui adoptent le plus cette solution.
Il y a également le fait que les gens considèrent un SAM comme un logiciel permettant d'avoir un résultat de grande qualité. En réalité, c'est souvent par effet psychologique, car on est facilement en admiration devant la capacité d'une machine à intégrer intelligemment son, images fixes ou animées et texte. Mais souvent, cette attraction est mélangée d'un refus psychologique à tout ce qui concerne l'informatique, car on considère que cette matière signifie uniquement écrire des codes plus ou moins compréhensibles.

L'utilisateur doit par conséquent, après avoir défini son objectif, se poser en permanence les six questions de la méthode quintilienne[9] adoptée en communication [Balbo,92]: qu'est ce que je cherche à faire ? Pour qui ? Pourquoi ? Comment ? Quand ? Où ?

Plusieurs questions peuvent être posées et évaluées avant de s'investir dans un SAM que nous désignerons dans la liste qui suit par uniquement le terme *logiciel*. En dehors de tout point de vue commercial, ces questions peuvent être d'ordre logistique, fonctionnel ou/et technique. Une partie de ces questions sont tirées des questionnaires publiés par le Ministère de l'Education Nationale et de la Formation Professionnelle sur Internet[10] intitulés « Questions à poser lors du choix d'un logiciel ».

2.3.7.1.Raisons logistiques

Les questions logistiques concernent tout ce qui a trait à l'approvisionnement, à l'entretien, au transport des équipements, qu'il s'agisse d'une simple barrette de mémoire vive, d'une carte

[9] Marcus Fabius Quintilianus : enseigna la rhétorique à Rome vers les années 100 après JC

[10] Http ://www.men.lu/_didac/choisir.html

son, de l'ordinateur ou des logiciels. Nous allons proposer quelques questions relatives à notre domaine d'intérêt, les logiciels SAM :

- Le logiciel est compatible avec quel type de CPU et quel système d'exploitation ? Des problèmes rencontrés avec l'incompatibilité entre des processeurs Intel et Risc, pour certaines opérations montrent l'intérêt de poser cette question.
- Le logiciel nécessite-t-il des périphériques particuliers pour tourner ? Si oui, sont-ils faciles à intégrer ?
- Pour ne pas restreindre l'utilisateur final à travailler sur un environnement de travail particulier, est-il possible de porter le résultat produit sur d'autres plates-formes ?
- Le fournisseur du logiciel donne-t-il la garantie de la durée de vie du logiciel, dans le sens où le produit peut être amélioré ?

2.3.7.2. Fondements fonctionnels

Le mode de fonctionnement d'un SAM est primordial, surtout si l'utilisateur est néophyte en informatique. Les quelques questions ci-dessous semblent alors être incontournables :

- Le logiciel est-il basé sur les connaissances de l'utilisateur en matière de didactique et de pédagogie ?
- Est-ce que le logiciel est facile à utiliser ? Y a-t-il des assistances à l'utilisateur ?
- Existe-il des interactions entre le logiciel et l'utilisateur ?
- Y a-t-il plusieurs approches possibles suivant la compétence de l'utilisateur ? Le logiciel accroît-il la motivation de l'apprenant ?
- Le système permet-il un auto apprentissage, dans le sens où il garde en mémoire ce que l'utilisateur a fait et agit en conséquence ?
- Le contenu du logiciel est-il modifiable, peut-on y ajouter des éléments nouveaux, l'enrichir et l'adapter aux besoins et/ou capacités de l'utilisateur ?

Au point de vue fonctionnel, un système auteur est, selon nous, considéré globalement valide à partir du moment où il respecte les objectifs fixés par le concepteur, indépendamment des moyens techniques utilisés. Il reste encore beaucoup de domaines que la technologie informatique ne maîtrise pas tels que le traitement de la parole, la prévision des réactions des individus, etc. Cela nous amène à accorder une certaine tolérance sur la validation du choix d'un SAM par un individu.

2.3.7.3.Choix techniques

Les questions techniques sont aussi importantes que celles relatives à la logistique et au fonctionnement. Voici quelques exemples :

- Le logiciel est-il un système ouvert afin de permettre à l'utilisateur d'intégrer des programmes externes au système auteur ?
- Le logiciel met-il à la disposition de l'utilisateur les outils nécessaires à des traitements spécifiques comme le graphisme ?
- La génération automatique (compilation) d'applications à partir du SAM est-elle possible ?
- L'application ainsi générée est-elle capable de traiter des requêtes spécifiques ou des données nouvelles que l'utilisateur final veut intégrer au fur et à mesure dans le programme ? Par exemple, si le logiciel n'accepte que les images en format bitmap (bmp), et que l'utilisateur dispose de certaines images gif.
- Peut-on donner une intelligence au logiciel ? Et cette intelligence peut-elle progresser ?

2.4. CONCEPTION A TRAVERS UN SYSTEME AUTEUR

La notion de concept peut être caractérisée par trois ensembles étroitement liés [Vergnaud, 89] :

- un ensemble de problèmes pour lequel le SAM fournit un moyen fiable et efficace de résolution,
- un ensemble de procédures et d'outils pour la résolution de ces problèmes,
- un ensemble de signifiants dans un langage opératoire pour la description des problèmes et des procédures.

La conception d'un logiciel utilise ces trois caractéristiques. En effet, l'auteur d'un système commence par constater un certain nombre de problèmes qu'il cherche à résoudre le plus simplement possible : il s'agit de la définition de l'objectif d'un projet, ainsi que la détermination de la population cible. Il est important de noter que l'auteur du système, fait partie lui-même, inconsciemment ou non, de la population cible car ses goûts et souhaits vont intervenir sur sa méthodologie de travail, sur la structure du logiciel, donc sur le résultat final. Ensuite, l'auteur fait une étude fonctionnelle qui permet de cerner les champs d'action ou d'utilisation du logiciel qu'il souhaite mettre à disposition dans son système, donc de définir l'ensemble des problématiques et en décrire les conséquences. Relativement aux

spécifications fonctionnelles, l'auteur donne les spécifications techniques permettant la concrétisation de ces fonctions.
Le développement et la validation ne font que concrétiser et respecter au mieux ces différentes étapes de la conception.

2.4.1. Etude fonctionnelle

Avant de mener toute action, un individu qui se propose de mettre en place un projet pour concevoir un système, pense à la faisabilité fonctionnelle et technique de son système. Faisabilité signifie principalement ici, cohérence des idées et possibilité technique. Selon la population visée, faisabilité peut signifier également réussite du système par rapport à sa distribution, son coût, ses exigences, etc. Nous avons inclus volontairement dans cette phase traitant l'étude fonctionnelle, l'étude conceptuelle appliquée dans la méthode Merise. En effet, nous considérons que les deux phases sont inséparables dans le sens où, naturellement, le concepteur pense systématiquement aux résultats donc aux fonctions à mettre en place.
Par conséquent, dès cette étape, le concepteur doit bien définir son objectif. En effet, bien que l'informatique soit une discipline plutôt ouverte à tous types de sujets, et arrive même parfois à résoudre des problèmes relativement peu usuels, comme la réalité virtuelle, il existe des limites. Pour vérifier la faisabilité d'un projet, les informaticiens ont recours depuis longtemps à l'organigramme c'est-à-dire un schéma représentant l'organisation générale du projet, et faisant ressortir les liens entre ses différents éléments. Dans certaines méthodologies de programmation, cette opération est souvent appelée étude organisationnelle.

Le principe de l'organigramme est appliqué dans différentes méthodes de programmation. Dans Merise par exemple, on applique une double démarche [Morejon, 89] :

- par niveaux pour la formalisation du futur système,
- par étapes pour la hiérarchisation des décisions à prendre.

La conception à travers un système auteur n'est pas une exception car le concepteur peut avoir une vision globale de son projet grâce à l'organigramme : les fonctions sont mises en relation entre elles de manière explicite. Cette méthode offre l'avantage d'être simple à mettre en œuvre car on peut détailler chaque partie de l'organigramme, en d'autres sous-modules ou sous-organigrammes. De plus, elle facilite la vérification de l'ensemble sachant que l'union des sous-modules forment l'organigramme complète originale [Saxena &al, 99]. Cela permet

d'estimer assez facilement les moyens techniques et le temps requis pour réaliser chaque point du projet.

La spécification fonctionnelle se termine par la définition de l'apparence finale du logiciel à savoir les écrans principaux, les types de données traitées et les opérations auxquelles les utilisateurs du logiciel, donc du système auteur, ont droit. Il s'agit alors de définir l'ensemble des maquettes[11].

2.4.2. Spécification technique

Cette seconde phase de la conception sert à concrétiser l'étude fonctionnelle, donc à en chercher et estimer les solutions techniques. Cette étape de la conception est alors importante, car on y définit principalement :

- Les équipements informatiques nécessaires, c'est-à-dire l'ordinateur, les cartes graphiques, les cartes sons, les scanners, etc. Les choix des équipements sont importants car souvent le produit final en dépend, par exemple du type de processeur utilisé.
- Le système d'exploitation qui sert d'interface entre le programme et la machine. Il est encore rare de trouver des logiciels fonctionnant sur différentes plates-formes, c'est-à-dire différents systèmes d'exploitation. Il est déterminant car le produit final, donc le SAM, tournera presque exclusivement sur le même environnement. Actuellement, les systèmes les plus adoptés sont ceux qui exploitent les micro-ordinateurs tels Windows et Linux. Cela s'explique par le fait que ces environnements sont accessibles facilement et sont plus faciles à utiliser. Ils sont souvent conviviaux grâce à l'intégration des techniques d'interface homme machine telle que l'application massive du principe de WYSIWYG[12], donc correspondant à la majorité des populations cibles des SAM.
- Le langage de programmation utilisé pour le développement du système auteur fait partie principalement des langages de bas niveau[13] tels que les langages C ou C++.

[11] Voir chapitre « robustesse - partie intégrante de la méthodologie de conception »

[12] WYSIWYG : What You See Is What You Get

[13] Langage proche de la machine, plus évolué que l'assembleur

2.4.2.1. Utilisation d'un langage auteur

Un ***langage auteur*** est défini comme étant un langage de programmation allégé grâce à des commandes et syntaxes moins complexes que les langages de programmation comme le C ou le Pascal. Mais comme les autres langages, il se base sur une syntaxe de codage plus ou moins complexe. Un langage auteur est constitué de macro-instructions de programmes. Il comporte des jeux d'instructions spécialisés : gestion d'écran, analyse de réponse, suivi, etc.

Nous pouvons citer ici le langage LINGO proposé par le SAM Director de la société américaine Macromédia, qui peut être rapidement difficile à mettre en oeuvre. De même, un langage auteur, utilise des variables ou des objets, comme les autres langages.

Les programmes de développement des macro-instructions intégrées dans ces langages auteurs sont souvent orientés objet. Cela offre une fiabilité dans le sens où tout est modulaire, et une simplicité pour l'utilisateur qui n'aura pas à avoir des connaissances approfondies de la programmation objet.

2.4.2.2. Utilisation d'outils graphiques

Les outils graphiques sont constitués d'icônes ou de schémas, regroupés dans des fenêtres ou palettes de fonctions, assurant chacun une ou plusieurs fonctions particulières, et pouvant être reliés les uns aux autres. Il existe deux types d'outils graphiques :

- Ceux qui peuvent être le reflet des fonctionnalités offertes par le logiciel comme les outils graphiques pour un lecteur de CD audio. Chaque bouton mis à la disposition de l'utilisateur final représente une fonction spécifique ou un ensemble d'instructions du programme. Un bouton est alors un outil graphique faisant appel à une fonction ou une opération déterminée.

Figure 9 : Interface pour le lecteur de CD audio de Windows 95

Cet outil graphique montre en plus l'avantage de présenter des boutons qui sont presque des standards sur tous les équipements éléctro-ménagers tels le magnétoscope, le magnétophone, le baladeur, etc. Cet avantage est considérable car l'utilisateur ne se trouve pas face à un écran

inconnu, et se sentira beaucoup moins bloqué, d'où l'intérêt de faire un rapprochement du monde quotidien avec les outils.

- Ceux qui donnent la structure du programme, représentant donc uniquement les grands traits ou la manière dont est constitué le programme. Dans le cas de l'exemple sur la figure 2-2 ci-dessous, dans le SAM Director, on constate la présence de scénarios et de distribution des images dans la constitution du programme. Chacun de ces éléments est ensuite détaillé en plusieurs fonctions. L'utilisateur peut avoir une visibilité générale des différentes images constituant un scénario.

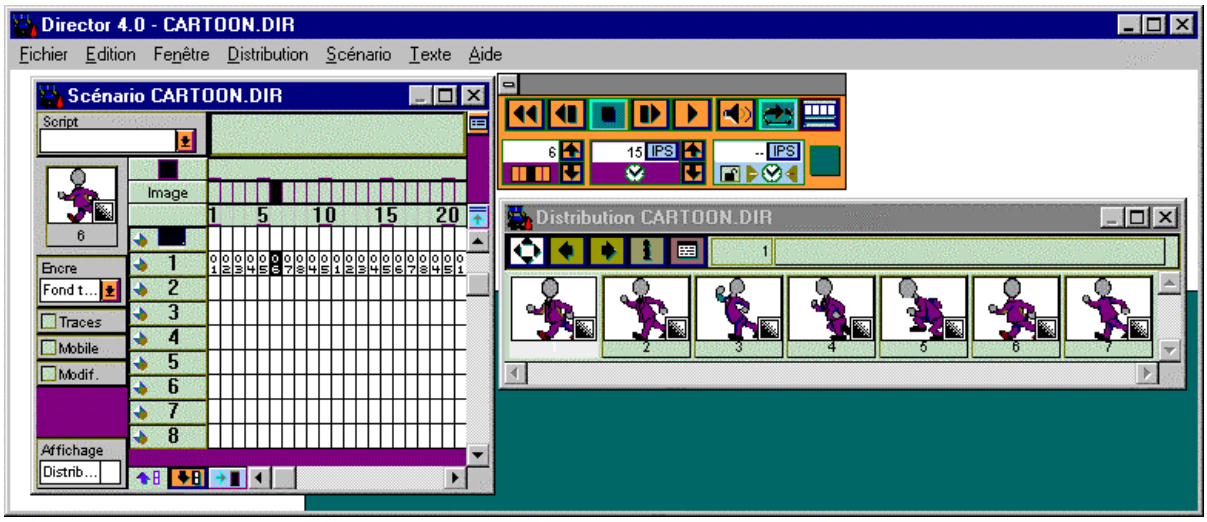


Figure 2-7 : Outils du SAM Director pour réaliser un logiciel multimédia

2.4.2.3. Traitement des informations

Les données traitées par les SAM sont souvent de différents types et peuvent alors être intégrées de différentes manières. Par exemple, si un utilisateur veut intégrer dans son logiciel des modules de questions-réponses, se basant sur un système-expert et comprenant des analyseurs syntaxiques, tout en explorant les capacités du multimédia ; il faut mettre à sa disposition les outils permettant de réaliser ces tâches, dont les plus connus sont :

Le traitement textuel pour intégrer des données sous forme texte que nous appellerons passif ou actif : **actif** dans le sens où l'utilisateur peut agir sur ce texte ou éventuellement, le texte s'adapte au fur et à mesure à l'utilisateur ; **passif** lorsque le texte est présent juste à titre d'information.

Le graphisme qui permet de représenter la logique de conception dans le sens où un objet graphique regroupant plusieurs données simule généralement une action. De plus, le graphisme est une partie intégrante du multimédia et influence directement la relation homme-machine. Le caractère synthétique des schémas aide beaucoup les utilisateurs qu'ils soient informaticiens ou non, car la représentation se réalise dans un espace à deux dimensions, donc on perçoit une logique non linéaire, plus facilement compréhensible. De ce fait, les compétences ergonomiques et graphiques de l'auteur de niveau 0 sont essentielles.

Des outils complémentaires d'intégration de données, souvent des logiciels à part entière, sont intégrés dans le SAM : traitement du son et éventuellement de la parole, compression des images, animation, etc. Il s'agit souvent d'une fonction demandant l'exécution (lancement) d'un autre logiciel tel que QuickTime pour les images animées.

2.4.3. Développement d'un logiciel à partir d'un SAM

Il n'existe aucune méthode à imposer aux individus pour réaliser son propre logiciel ou sa propre animation ou son jeu multimédia à partir d'un SAM. En revanche, les conseils donnés dans la majorité des systèmes auteurs pour écrire son logiciel [Librik, 94] sont, quelles que soient les catégories de systèmes utilisées, une fois le choix du système auteur décidé en fonction du besoin, globalement les suivants :

1. Diviser l'application en différentes étapes, et chaque étape en plusieurs fonctions.
2. Sélectionner les procédures que l'on peut généraliser.
3. Constituer les différents composants de l'application finale : les données textuelles à saisir, le scan des photos, la construction des schémas, etc.
4. Se familiariser avec le SAM : son langage, ses icônes, etc. Les subtilités et complexités comme les syntaxes seront à travailler plus tard dans les démarches et doivent donc être vues superficiellement pour ne pas bloquer l'approche avec le système.
5. S 'appuyer sur les exemples donnés généralement avec le système, afin de s'inspirer de certains points se rapportant à son aventure.
6. Partir de ces exemples, enlever les points inutiles et ajouter petit à petit des fonctions spécifiques.
7. Enfin, développer les parties complexes et intégrer les bibliothèques et routines externes au SAM.

8. Commenter le maximum de codes et des modifications éventuelles.

Mais, d'autres questions[14] se posent : ces démarches sont-elles applicables par tout type d'utilisateur ? Si cette liste avait été plus précise, est-ce que cela aurait amélioré la compréhension et la démarche de chaque utilisateur ? Bref, ces démarches ne sont-elles pas les sources de certains problèmes rencontrés lors de l'utilisation d'un système auteur multimédia ?

En dehors de ces éventuelles contraintes, il ne faut pas oublier qu'avant de pouvoir réaliser un produit à partir d'un système auteur, l'individu doit se familiariser avec la machine sur lequel il travaille, en comprendre les rouages et en interpréter les messages [De La Passardière, 93]. Notre étude est orientée vers l'enseignement, c'est-à-dire tout ce qui est logiciel destiné à l'apprentissage. Par conséquent, la réalisation d'un logiciel éducatif à partir d'un système auteur nécessite des efforts pédagogiques et ergonomiques de la part du concepteur du logiciel final.

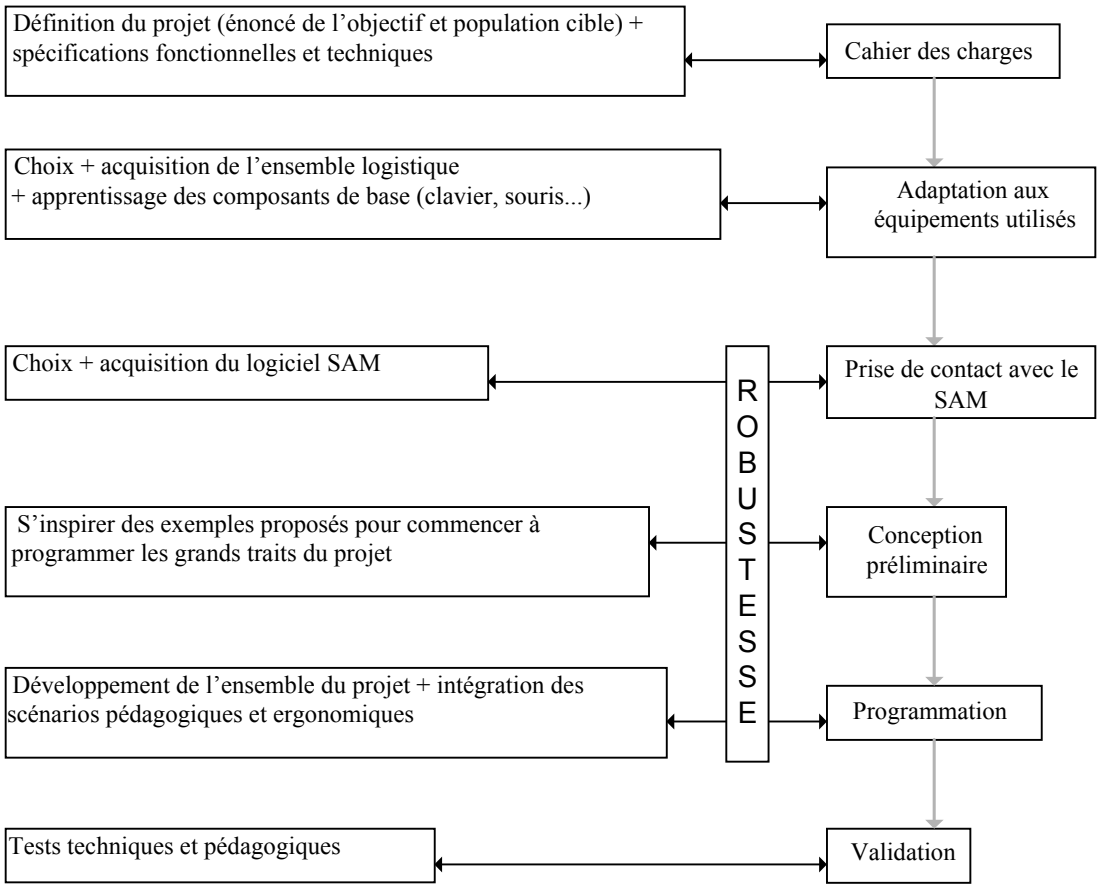


Figure 2-8 : Processus de réalisation d'un logiciel éducatif à partir d'un SAM

[14] Nous essayerons de répondre à cette question dans la seconde partie de ce mémoire.

La figure ci-dessus nous donne une vision globale des étapes de conception de logiciel à partir d'un SAM. Nous remarquons que la notion de robustesse que nous développerons dans la seconde partie de notre étude concerne surtout :

- l'acquisition du SAM car si le SAM ne répond qu'à une partie des besoins définis dans le cahier des charges il est difficile de combler les lacunes
- la conception préliminaire car les premières approches sont essentielles pour la continuité de la conception
- la programmation où la robustesse est primordiale pour maintenir les motivations de l'auteur de niveau 1 et surtout pour aboutir à un résultat robuste.

2.4.4. Utilisateur constructeur de sa connaissance informatique

Les difficultés pour apprendre à programmer un logiciel à partir d'un SAM ou non, peuvent être séparées en 5 parties [Boulay, 86] :

1 - La première et la plus générale difficulté concerne l'orientation, c'est-à-dire comprendre l'objectif d'un programme, trouver les moyens pour y parvenir.

2 - Il existe aussi une difficulté à acquérir la notion de machine, c'est-à-dire comprendre les propriétés d'un ordinateur, prévoir les effets de certaines actions , etc.

3 - Des difficultés sur les multiples caractéristiques des langages de programmation peuvent également se poser. En effet, l'individu doit apprendre la syntaxe et la sémantique d'un langage.

4 - La compréhension des structures de programmes telles que les boucles. Il s'agit alors ici de l'algorithmique de développement d'une application.

5 - La pragmatique du programme, c'est-à-dire l'apprentissage de l'habilité à spécifier, développer, tester et débuguer un programme.

La liste ci-dessus nous montre alors que lorsqu'un individu ayant une compétence dans d'autres domaines que l'informatique utilise un système auteur, il construit lui-même sa connaissance concernant le système. En effet, pour pouvoir réaliser son logiciel à partir d'un SAM, il doit explorer son environnement, participer activement à la construction de l'espace, du temps et de la causalité [Balacheff, 92].

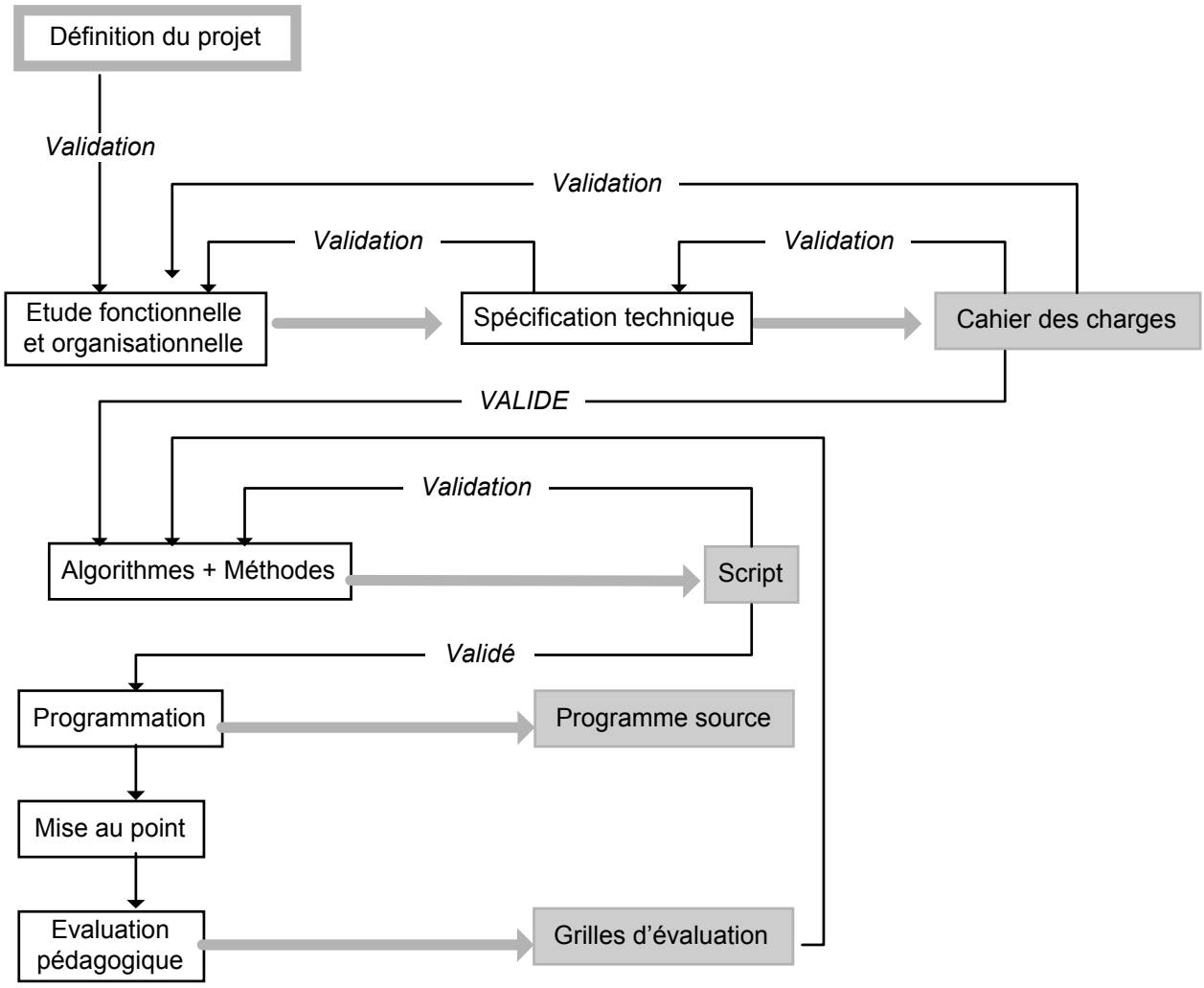


Figure 2-9 : Modèle de développement à partir d'un système auteur[15]

Une définition détaillée de la structure du programme est alors importante afin de permettre à l'auteur du logiciel ou à un utilisateur, de créer facilement des procédures de validation selon l'objectif, la population et tous les critères soulevés pendant la phase de spécification fonctionnelle.

La validation peut se faire soit étape par étape selon l'avancement du projet, soit seulement à la fin de la réalisation. Les expériences ont montré que la technique de validation systématique des étapes, schématisée à la figure 2-4 ci-dessus, est plus intéressante. En effet, elle minimise les risques d'incohérence par rapport à l'objectif et par rapport aux autres étapes.

[15] Modèle inspiré du modèle de développement d'un logiciel éducatif [Madaule &al, 92]

Cette validation est en plus des critères habituels (simplicité, fiabilité, etc.), conditionnée, pour un SAM par deux critères fondamentaux : l'ergonomie et la pédagogie du logiciel.

Lorsqu'un individu néophyte en informatique utilise un système auteur, il apprend inconsciemment une partie de la discipline informatique. En effet, quels que soient les outils manipulés ou testés, un concept informatique reste le même. Le fait d'utiliser un SAM, ne change pas, par exemple, la manière d'intégrer du son dans un logiciel. Cela peut réduire ou simplifier certaines tâches, mais les principes globaux restent. De même, l'utilisateur va s'accoutumer petit à petit aux expressions spécifiques au domaine, par exemple les mots processus ou variable.
Lorsque l'utilisateur commence à concevoir son logiciel, il apprend inconsciemment à programmer. Au fur et à mesure qu'il avance, même sans respecter un ordre précis, il finit par acquérir une certaine méthodologie, et améliore alors progressivement son approche vers le SAM ainsi que le développement de son produit informatique.

Pour optimiser la conception du nouveau logiciel à partir d'un SAM, il est alors essentiel d'avoir un outil qui assimile et apprend progressivement le contexte dont le comportement de l'utilisateur et le SAM.

2.5. CONCLUSION

Cette partie de notre étude nous a permis de constater qu'un système auteur multimédia est un système socio-technique complexe car il s'agit d'un environnement dans lequel des individus, des groupes sociaux, des processus à la fois technologiques et intellectuelles coopèrent pour réaliser un objectif [Prince, 98]. Un SAM permet une conception participative c'est-à-dire que l'utilisateur peut être un innovateur et participe à la conception. Il sert à aider un utilisateur dans sa conception dans le sens où l'auteur peut profiter pleinement des outils mis à sa disposition. A titre d'exemple, pour intégrer du son dans une partie du programme à réaliser, le programme exploitera un outil de traitement de sons déjà inclus dans le système. Il y a par conséquent, une sorte d'assistance dans les démarches d'un individu utilisant un SAM pour concevoir, permettant ainsi de diminuer le stress ou l'approche négative de certains individus sur l'informatique.

Pendant cette synthèse sur les SAM, nous constatons que malgré la timidité du grand public face à ce type de systèmes, les SAM sont en constante évolution technologique et ne tardent pas à intégrer toutes les nouvelles découvertes réalisées tant au point de vue matériel que logiciel.
Par ailleurs, l'arrivée du multimédia et l'expansion d'Internet au niveau mondial, a donné un véritable coup de pouce au lancement des SAM auprès des professionnels spécialisés dans ce domaine. Les SAM commencent à bien s'implanter dans certains domaines d'activité telles que l'édition de CD-Interactive ou de CD-ROM multimédia et l'enseignement.

Cet état de l'art nous a également permis de bien cerner l'utilité et le rôle d'un SAM afin d'aider l'auteur de niveau 1 à faire le bon choix et surtout à acquérir de façon optimale les connaissances nécessaires pour obtenir un logiciel dépourvu d'erreurs et robuste.

Le fait que les autres activités ainsi que le grand public hésitent à franchir le pas démontrent en partie, le manque de maturité des SAM malgré les soins que les concepteurs de ces logiciels essaient de porter pour arriver à sa définition principale : permettre à quiconque de développer des logiciels multimédias avec le minimum de contraintes possibles.

3. CONCEPT D'ERREUR ET DE ROBUSTESSE DANS L'UTILISATION DES SAM

3.1. INTRODUCTION

Un système logiciel peut appartenir à plusieurs catégories d'applications : transactionnelle, relationnelle, temps réel, système expert, protocole de communication, etc. Un système auteur, peut prétendre être le fruit de plusieurs années de travail et d'expérience dans ces différentes catégories de systèmes, dans le sens où il exploite la majorité des spécificités constituant ces derniers. Cela présente l'avantage de fiabiliser et d'agrandir les champs d'utilisation des systèmes auteurs : création de modèles de voitures, de CD-ROM et d'autres produits, en particulier des logiciels éducatifs.

Or le système auteur lui-même est un logiciel fait par des informaticiens. Et comme le dit ce dicton derrière lequel se réfugient souvent les programmeurs et concepteurs de logiciel "il n'y a pas de logiciel qui n'a pas de bogues", aucun système auteur n'est à l'abri d'erreurs plus ou moins graves. Il existe par conséquent des raisons pour lesquelles tout logiciel est bogué. Ces raisons sont à définir afin de pouvoir résoudre les problèmes de robustesse. Elles ne sont pas forcément techniques et peuvent être de conception ou d'utilisation. Dans le sens où l'objectif de notre recherche est la création d'un environnement de création de logiciels robustes, il est intéressant de se pencher sur la détection et la correction de ces erreurs.

Il est à savoir qu'actuellement, un SAM permet surtout d'améliorer la productivité en quantité, et non pas en qualité. Cette qualité du logiciel dépend du nombre et de la gravité des erreurs que l'on peut rencontrer surtout au cœur du système, c'est-à-dire des erreurs de conception par rapport à un objectif donné. Il s'agit alors d'appliquer au mieux une méthode de conception, et qui dit méthode dit génie logiciel [Ermine, 93].
Le but du génie logiciel est de fournir les méthodes et les outils nécessaires pour la construction de logiciels « de qualité » [Surcin & al, 95]. Nous nous intéressons ici aux concepts fertiles en ce sens, et comme notre recherche sera orientée vers les logiciels éducatifs, notre motivation porte surtout sur les ateliers de génie didacticiel ou AGD étant donné qu'il s'agit du secteur qui exploite le plus les SAM actuellement. De plus, l'AGD nous

intéresse particulièrement, car notre recherche implique également l'aide aux individus cherchant à partager leurs connaissances avec le minimum de contrainte au point de vue informatique.

Dans cette partie du document, nous allons présenter en premier lieu une épistémologie sur l'erreur. Nous verrons que la propagation d'une erreur peut entraîner des dégâts considérables et conduire à plusieurs défaillances successives. Le principe consiste alors à rassembler les causes de défaillances en essayant de répondre aux questions : pourquoi et comment commettons-nous des erreurs ? Il est intéressant d'adopter cette démarche car en conception, même un débutant est capable de résoudre le problème ou de corriger l'erreur « à la main », produisant ainsi un résultat, sans connaître pour autant les procédures mises en œuvre pour y arriver [Hoc, 92].

Ces études nous permettront de mieux cerner la notion de robustesse que nous développerons par la suite. Nous verrons que la robustesse rejoint en grande partie la notion de fiabilité car il s'agit d'analyser la capacité d'un SAM à être utilisé et à rendre le service attendu malgré les perturbations éventuelles.

3.2. PROBLEMATIQUES VIS-A-VIS DE L'ERREUR

Une erreur signifie, d'après le Larousse Multimédia Encyclopédique 1996, un acte de se tromper, d'adopter ou d'exposer une opinion non conforme à la vérité, à une norme, à une règle ; de tenir pour vrai ce qui est faux. Cette action peut être inconsidérée, regrettable ou maladroite. Une erreur désigne également ce qui est inexact, par rapport au réel ou à une norme définie. La norme[16] ANSI/IEE-STD-729 préconise d'utiliser le terme erreur pour exprimer une différence entre une valeur ou un état calculé, observé ou mesuré et la valeur ou l'état réel, spécifié ou en principe correct.

Dans le cadre d'une conception de logiciel avec l'aide ou non d'un SAM, on peut considérer comme erreur tout ce qui ne correspond pas à l'objectif défini lors de la mise en place du projet et décrit dans le cahier des charges. Cela impliquerait de disposer d'une

[16] ANSI/IEEE-STD-729 : Glossaire standard de la terminologie du développement de logiciel, 1983

présentation formelle d'un cahier des charges. Il est à noter que nous ne tenons pas compte des erreurs éventuelles commises lors de la rédaction du cahier des charges.

Les manières et les moyens pour atteindre cet objectif sont décrits dans le cahier des charges à travers les spécifications fonctionnelles et techniques[17]. Ces spécifications désignent alors les normes ou règles à suivre et à respecter tout au long du développement du logiciel.
Il est à noter qu'il ne faut pas confondre le terme erreur avec la richesse fonctionnelle et technique du SAM. Cela signifie qu'un SAM qui ne présente pas une palette d'opérations nécessaires à la création d'un logiciel ne peut être considéré comme rempli d'erreurs. Dans ce cas, nous disons que le choix du SAM a été mal évalué ou que le SAM est pauvre en fonctionnalités.

3.2.1. Expression de l'erreur

L'erreur peut être détectée selon trois mécanismes de base [Jambon, 96] :

1. L'autocontrôle par l'auteur lui-même donc par l'utilisateur de niveau 1,
2. La détection par un tiers qui est un moyen efficace pour détecter les erreurs de diagnostiques,
3. La détection à partir d'indices de l'environnement appelés plus couramment des bogues.

La troisième manière de détecter une erreur nous intéresse particulièrement dans notre recherche car l'utilisateur de niveau 1 d'un SAM peut être néophyte en conception et donc n'aura pas le réflexe d'effectuer l'autocontrôle.

Une erreur en logiciel informatique se manifeste généralement lors de son utilisation. Les niveaux de criticité d'une erreur peuvent s'exprimer dans les termes suivants [Fournier, 93]:

- Catastrophique : erreur non tolérée par l'application
- Majeur : erreur qui modifie le service rendu
- Gênant : erreur qui transforme sans gravité le service rendu
- Mineur : erreur qui réduit le service rendu
- Bénin : erreur qui n'a aucun effet réel sur le service

[17] Voir chapitre 2.3.

Cette partie de notre étude sur l'erreur décrit les différentes manifestations de tout type d'erreurs, et présente les outils et méthodes pour les éviter et les résoudre.

3.2.1.1.Les différents types de bogues

Une panne ou un blocage résultant d'une erreur est appelé bogue en informatique. Généralement un bogue est une occurrence ou un fait qui n'a pas été prévu dans le programme [Dubrovsky, 97]. Voici la liste des principaux types de bogues auxquels nous avons associé à chacun une qualification :

- Bogues radicales :

Il s'agit des types de bogues qui peuvent être détectés même par un débutant. Il s'agit des erreurs créant un "plantage" soit du programme, soit de l'ensemble du système, donc l'ordinateur. Ils arrivent inopinément et sont difficiles à corriger ou à éviter.

- Bogues syntaxiques :

Ce nom définit les bogues provenant du non-respect du langage de programmation que l'ordinateur va coder. Ces problèmes sont souvent faciles à résoudre grâce au compilateur et interpréteur souvent très évolués. Nous pensons par exemple à l'analyseur syntaxique Lint sous Unix lorsque l'on veut lister les erreurs syntaxiques avant de compiler un programme en langage C. Ils apparaissent uniquement pendant le développement du logiciel, jamais lorsque le logiciel est compilé.

- Bogues pervers :

Ces types de bogues sont souvent bien camouflés et très difficiles à détecter même par les spécialistes. Nous pouvons penser au bogue concernant les premiers microprocesseurs Pentium, qui donne un résultat faux périodiquement, après un nombre défini d'opérations. Les bogues pervers mettent souvent énormément de temps pour se manifester, sont difficilement décelables et sont plus ou moins faciles à corriger.

- Bogues surprises :

Ces types de bogues se manifestent par surprise. Le programme exécute parfois des instructions complètement inattendues. Ce sont surtout les débutants qui ne maîtrisent pas suffisamment leurs outils de développement qui font face à ces bogues.

- Bogues stratégiques :

Ces types de bogues sont dûs à une erreur de logique dans le programme. Le programme pourra ainsi dévier de son objectif principal. Ils ne sont souvent détectés qu'à la fin du développement.

3.2.1.2.Outils et méthodes de prévention et de détection

Les bogues sont souvent communs à tous les domaines constituant la discipline informatique. Aussi, les systèmes auteurs ont-ils pu profiter de toutes les recherches effectuées pour résoudre les bogues, bien que ce ne soit pas en temps réel. Les méthodes et outils mis en œuvre aujourd'hui pour la détection sont :

- Le *debugger* offre un environnement qui prévient le développeur dès qu'une erreur est détectée. Mais les erreurs reconnues par ces debuggers sont prédéfinies et ne couvrent pas encore tous les cas d'erreurs possibles. Plusieurs systèmes auteurs tels que Director ou IconAuthor ont implémenté un debugger.
- Le *breakpoint* correspond à un traçage de l'exécution d'un programme. Le développeur marque la ligne d'instruction où l'exécution doit s'arrêter. Cette méthode lui permet de situer l'emplacement du problème. Ce sont surtout les langages de programmation de niveau 0 et les langages générateurs de code[18] qui utilisent cette méthode.
- Le *Single Step* est quasiment la même méthode que le breakpoint sauf que le traçage s'effectue ligne par ligne. L'utilisateur a alors la possibilité de tester le résultat de chaque instruction. Le single step est par conséquent plus précis au point de vue syntaxique mais n'aide pas le développeur au niveau logique de programmation et surtout à avoir une vision globale du programme.
- L'*Output* désigne une méthode visant à afficher volontairement sur la sortie standard ou dans un fichier certains résultats du programme. Cette méthode est souvent associée aux deux précédentes.
- Le *Commentaire* est la manière la plus simple, et la plus utilisée pour déboguer un programme. Cette méthode est peu efficace, surtout syntaxiquement. En échange, le fait de commenter le maximum de lignes d'instruction, représente une aide considérable dans le suivi de l'ossature du programme. Cela permet à l'utilisateur, qu'il soit novice ou expert, de ne pas se noyer pendant la phase de codage.

[18] Voir Chapitre 1.5.

D'autres méthodes moins directes, destinées principalement à la prévention, existent également [Chebili, 2000] :

- La *documentation* : selon sa qualité, elle peut être considérée comme le premier élément de secours en cas de problème.
- Les *messages d'erreurs* : ils permettent à l'utilisateur de diagnostiquer une mauvaise action ou une erreur de syntaxe qu'il a commise. Le plus souvent, ces messages sont d'ordre indicatif et ne sont pas approfondis. Ils ne sont donc pas suffisants pour permettre à un utilisateur débutant ou occasionnel de résoudre la difficulté à laquelle il est confronté. En revanche, les messages d'erreur sont avantageux par le fait qu'ils apportent une aide en temps réel.
- Les *définitions des fonctions* : on les rencontre souvent dans les interfaces graphiques ; au passage de la souris sur l'icône représentative de la fonction, une très brève définition s'affiche pour informer l'utilisateur de la tâche qui sera effectuée en cliquant sur l'icône en question.
- Les *tutoriels* : ils offrent des formations structurées aux utilisateurs pour apprendre l'utilisation de l'application [Shute-94]. Ils sont généralement utilisés hors session de travail et n'apportent pas d'assistance en temps réel à l'utilisateur en cas de problème.

3.2.2. Les facteurs d'erreurs

Des expériences réalisées en 1983 pendant un an [Bailey, 83], sur des informaticiens ayant une bonne connaissance de la programmation des logiciels et de l'architecture des ordinateurs, a permis d'évaluer les différents facteurs d'erreurs dans les systèmes informatiques en général. Cette évaluation n'est pas obsolète aujourd'hui malgré les progrès technologiques effectués depuis. En effet, ces résultats restent fiables particulièrement dans le cadre de notre étude car le concepteur qui utilise un SAM pour développer son logiciel peut être ou non un informaticien.

L'erreur humaine dans le schéma ci-dessous représente simultanément le manque d'attention, le manque de compréhension, le mauvais choix et le manquement de la part de l'individu, qui peut être associé à l'utilisateur du SAM dans notre étude.
La partie conception concerne toutes les erreurs provenant de la méthodologie de conception appliquée, ainsi que les erreurs commises lors de l'application de ces méthodologies.

Le codage est relatif aux erreurs incluses dans le langage de programmation.
La partie structure et organisation indique les erreurs fonctionnelles du logiciel, incluant l'algorithmique.

Les simulations et les tests signalent toutes les erreurs commises pendant la ou les phases de validation du logiciel.

- La partie matériel indique le pourcentage des erreurs provenant directement des équipements utilisés, indépendamment du système.
- Enfin, la partie interface homme-machine indique le taux d'erreurs dues à une mauvaise ergonomie et une pédagogie insuffisante.

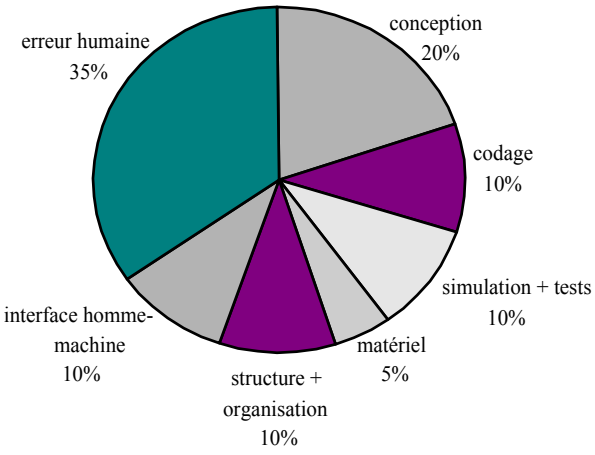


Figure 3-1 : Attribution des facteurs d'erreurs d'après [Bailey, 83]

Nous pouvons remarquer que la part de l'individu représente plus de 60% des erreurs qui se produisent ; tandis que les erreurs logistiques constituent environ 25% de l'ensemble.
L'amélioration de la qualité des logiciels grâce aux nouvelles technologies n'entraîne pas forcément une diminution des erreurs humaines. Le fait que 15 à 43% des erreurs recensées dans les systèmes d'exploitation proviennent des erreurs de codage (mauvaise utilisation des pointeurs et mauvaises allocations de tables) démontrent cette thèse [Ghosh & al, 1998]. Par conséquent, beaucoup d'erreurs sont communes à différents individus, mais diffèrent selon les tâches auxquelles elles sont appliquées.

3.2.2.1.Erreur humaine

« L'erreur n'est pas une pure négation, c'est-à-dire n'est pas le simple défaut ou manquement de quelques perfections, mais c'est une privation de quelques connaissances que l'individu

devrait avoir » Descartes, Méditations, IV, 1641. En effet, on peut considérer l'erreur comme un moyen permettant à un utilisateur de SAM, c'est-à-dire à l'auteur de niveau 1[19], de progresser dans ses développements. C'est en commettant des erreurs que les problèmes se posent, et que le concepteur utilisant un SAM améliore au fur et à mesure sa méthode de conception, de développement et de tests, bref son produit final.

Il existe différentes démarches pour prendre une décision lors de la phase conceptuelle d'un projet. Suivant la démarche adoptée, on obtient des styles de programmes différents. Nous nous sommes appuyés sur les trois types de démarches présentées par Forner [Forner, 90] :

- un *style rationnel* constitué d'une évaluation systématique et d'une délibération dans une perspective temporelle étendue. Cela implique une formulation du type « je pense à ce que je pourrai obtenir et ce à quoi je devrais renoncer en choisissant cela ». Cette démarche donne souvent la garantie d'aboutir à un produit assez satisfaisant, et répondant suffisamment aux besoins exprimés, mais nécessite des efforts considérables de la part du concepteur. Ici, l'expérience du concepteur est primordiale. Ce style de conception est fréquemment adopté par les utilisateurs ayant des connaissances en informatique.
- un *style intuitif* attachant de l'importance à l'imagination, au sentiment présent et à une conscience de soi émotionnelle. Le style intuitif donne la formulation « je choisis cela car je sens qu'il me correspond bien ». Ce réflexe peut devenir rapidement un handicap pour les débutants car, par expérience, les habitudes de développement se prennent très rapidement. De plus, le concepteur tend souvent à donner plus d'importance à l'interface homme-machine, qu'il considère rapidement comme le centre du projet. Le résultat donné par cette démarche est peu convaincant car la structure du programme est souvent inexistante réellement et les différentes fonctions ou opérations mises à la disposition de l'utilisateur final sont peu soignées.
- un *style dépendant* composé d'une passivité et d'une soumission à l'influence des attentes d'autrui, des événements et du marché. Cela signifie « ma décision de choisir cela a été fondée sur le conseil de personnes qui m'ont convaincu que c'était un bon produit ». Cette démarche peut être intéressante à condition que la personne, source de l'influence est un expert dans le domaine. Mais, cette méthode n'est bénéfique qu'à des individus ayant déjà de l'expérience dans le domaine.

[19] Concepteur de logiciel à partir d'un SAM - voir Introduction générale.

Les erreurs humaines résultent alors souvent du vécu de chaque individu. Le passé, c'est-à-dire l'expérience joue ainsi un rôle primordial dans la limitation des erreurs grâce à des contrôles intrinsèques de l'individu. En effet, l'expérience peut aider l'utilisateur du SAM à réduire ses erreurs comme elle peut l'influencer de telle manière qu'il en oublie les autres parties du projet.

3.2.2.2.Erreur logistique

Beaucoup de logiciels sont conçus avec une « pensée » orientée surtout vers l'ordinateur et le logiciel. Le concepteur a tendance à s'occuper uniquement des problèmes venant de la machine et de son programme [Bailey, 82]. En d'autres termes, il est plus souvent intéressé par tout ce qui est concret, voir palpable, dans un projet, et non pas vers les éléments abstraits telles les spécifications fonctionnelles correspondant à l'objectif final.
Cela peut réveiller l'idée que pour réaliser un programme, il faut être un informaticien ce qui signifie avoir au moins des connaissances sur un système d'exploitation et un langage de programmation. Généralement, l'environnement de travail d'un développeur n'est pas du tout convivial. Il utilise un éditeur tels que « vi » ou « emacs » dans Unix et « Edit » dans Dos. Le mot éditeur sous-entend déjà connaître quelques commandes, généralement rentrés au clavier (sauf sur quelques éditeurs plus conviviaux et ergonomiques, se basant sur des interfaces graphiques) pour saisir des données textuelles. Ainsi, ce type d'erreur peut provenir du mauvais choix sur un ou plusieurs éléments techniques, pendant la phase de spécification technique, du concepteur.

L'erreur logistique provient parfois du fait que l'utilisateur du système auteur accorde trop de confiance à la capacité de la machine. Le résultat n'est pas satisfaisant car cela débouche souvent sur une mauvaise gestion de l'espace disque, un ralentissement de l'exécution d'une opération multimédia, etc.

Ainsi, l'erreur logistique entraîne généralement un mauvais résultat à cause du fait que le concepteur peut s'écarter de façon incontrôlée de son projet. Le suivi du cahier des charges, primordial pour la réussite du projet, peut alors être affecté par l'erreur logistique. Cela entraîne également un étalement considérable du temps global de réalisation. De même, cette erreur induit d'autres erreurs qui peuvent être qualifiées d'erreurs humaines comme le fait de ne pas prendre en considération les différents niveaux des utilisateurs par exemple.

3.2.3. Portée systémique de l'erreur

Selon l'étape où l'on se situe dans le cahier des charges, l'erreur peut être qualifiée et quantifiée différemment. Elle peut être attribuée uniquement aux deux entités physiques qui permettent la création de logiciels comme le montre la figure 2-1 ci-dessous. L'individu désigne ici toute participation humaine en matière de logiciel, de la définition des besoins, à la méthodologie et l'algorithmique ; tandis que le terme logistique désigne tout ce qui a trait ou est directement en relation avec le matériel et la technologie utilisés.

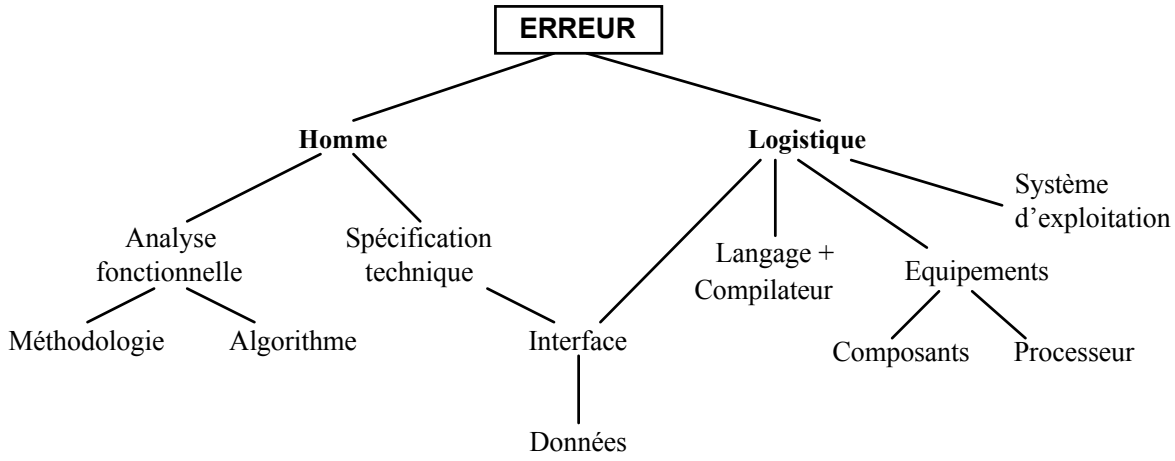


Figure 3-2 : Attribution des causes d'erreurs

Des études ont montré qu'au moins la moitié des erreurs qui se produisent lors d'un projet logiciel[20], proviennent de la conception du système [Bailey, 82]. Il existe plusieurs niveaux d'erreurs associés au concept de système auteur, dont principalement :

- erreur de conception et de développement du SAM lui-même,
- erreur de sélection du SAM approprié,
- erreur de conception du logiciel final à partir du SAM sélectionné,
- erreur de développement du logiciel final par l'utilisateur,
- erreur d'utilisation du logiciel final.

[20] Il est à remarquer que cela n'est pas à généraliser en informatique. Exemple, en réseaux informatiques, plus de 70% des erreurs proviennent du matériel.

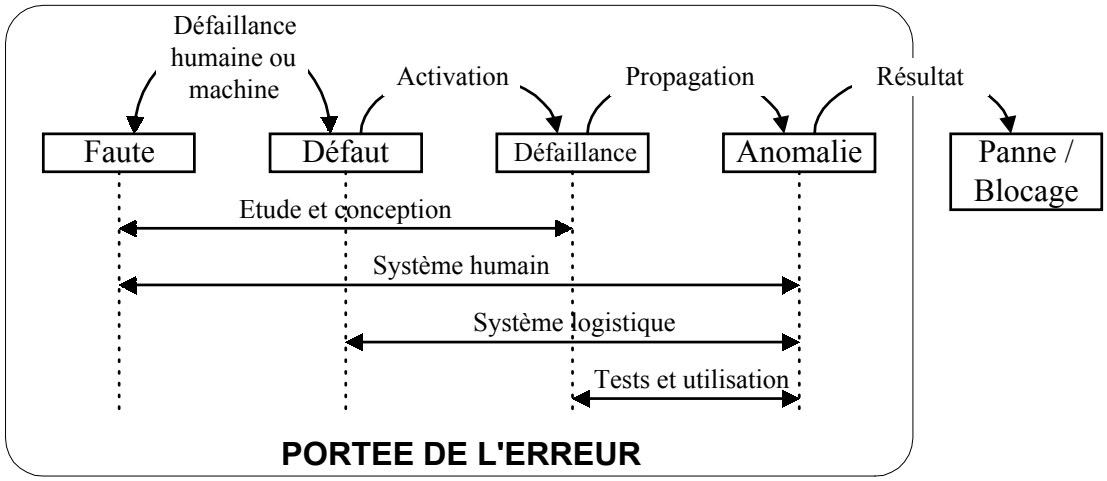


Figure 3-3 : Représentation systémique de l'erreur

La figure ci-dessus, inspirée du processus de défaillance défini par Fournier [Fournier, 93] nous montre l'étendue de ce que nous définissons comme erreur dans notre étude. Ainsi, le terme erreur peut signifier selon le contexte une **faute** lorsqu'elle provient d'une intervention humaine, un **défaut** lorsqu'une partie de la logistique (matériel, système ou logiciel) est inadaptée ou manquante, une **défaillance** pour marquer l'écart entre ce qui est attendu et ce qui est constaté, une **anomalie** lorsque l'utilisateur ressent l'effet ou s'aperçoit du dégât. Nous ne considérons pas comme erreur une panne ou un blocage qui représente uniquement l'état résultant de l'erreur.

Quelle que soit l'erreur à considérer, on constate qu'il s'agit surtout de manquement provenant de l'individu. Il est par conséquent important d'effectuer des études sur les erreurs humaines car la robustesse[21] du produit final dépend en grande partie de leurs nombres et de leurs gravités.

Enfin, une erreur implique obligatoirement la présence d'un problème plus ou moins évident à résoudre, car on détecte une erreur suite à un problème de fonctionnement. Avant de décrire l'erreur, il faut constater le fait qu'il y ait un problème et le comprendre. Et « comprendre un problème » signifie être capable de le situer, c'est-à-dire définir au moins le début et la fin de l'état critique [Nguyen & al, 78].

[21] Voir chapitre 2.3

3.2.4. L'erreur en interaction homme-machine

Les psychologues accordent beaucoup d'importance aux problèmes de mémorisation et d'apprentissage. Des recherches sur l'apprentissage tentent de préciser les mécanismes de différenciation et de généralisation des connaissances, leur automatisation et leurs règles de transfert. Les conséquences de ces recherches sont évidement d'importance puisqu'on peut en attendre des retombées rapides dans le domaine de l'éducation, de l'expertise et de la créativité [Tiberghien, 88].

Le succès d'un logiciel en terme d'apprentissage est directement lié à l'appréciation de son interface homme-machine [Norman, 91]. Une grande partie de cette interface impacte autant l'homme pour son apprentissage et la machine si cette dernière est dotée de techniques lui permettant d'acquérir ou de mémoriser des connaissances dans l'optique de les développer par la suite.
Cet apprentissage provient surtout de l'interaction entre l'homme et la machine. Cela amène Norman à donner des paramètres cognitifs, que nous allons mettre en œuvre dans la partie contribution théorique et applicative[22], pour optimiser cette interaction :

- Le degré d'exigence d'attention de la part des utilisateurs,
- la capacité des utilisateurs à mémoriser les commandes,
- la rapidité de mémorisation des commandes par les utilisateurs,
- le degré de participation de l'interface à la décision,
- le degré de participation de l'interface dans la résolution des problèmes,
- la capacité du système à réaliser les tâches.

Dans le cadre de notre étude, il est important ici de définir les partenaires[23] impliqués dans l'interaction :

1. ***l'homme*** c'est-à-dire l'utilisateur de niveau 1 qui développe en se basant sur un SAM
2. la ***machine*** c'est-à-dire le SAM

Bien qu'il n'y ait pas de consensus sur une classification fédératrice des erreurs, nous pouvons répartir les erreurs en interaction homme-machine en trois niveaux [Jambon, 96] :

[22] Chapitres 4-5-6

[23] Voir schéma explicatif de la fig1-1

- Conceptuel qui réfère aux hypothèses sur les mécanismes cognitifs impliqués dans la production de l'erreur.
- Comportemental relatif au fait que l'utilisateur ne maîtrise pas ses actions, exemple un mauvais ordonnancement des tâches
- Contextuel qui s'intéresse aux caractéristiques formelles de l'erreur mais aussi à des hypothèses causales.

Nous tiendrons compte de ces niveaux de classification dans la suite de nos travaux.

3.2.5. Reconnaissance et évaluation de l'erreur dans la conception

Il est important de savoir évaluer chaque erreur commise lors de la conception d'un logiciel à partir d'un SAM, après l'avoir détecté, afin de mieux la comprendre et par la suite la corriger bien qu'une erreur n'entraîne pas systématiquement une panne ou un blocage.
La détection de l'erreur peut s'effectuer de deux manières différentes :

- par la constatation ou la découverte, c'est-à-dire que l'utilisateur rencontre inopinément un bogue,
- par la détection automatique des erreurs grâce à un mécanisme interne ou externe au logiciel.

Sachant que la majeure partie des erreurs de conception est humaine, le travail à mener pour les déterminer et les évaluer, est laborieux. Afin de faciliter les tâches et surtout garantir les résultats de l'expertise, nous allons adopter deux axes de vue :

- Une vue verticale se basant sur la description des grandes catégories de fonctionnalités du logiciel final et donnant par conséquent l'architecture globale du système.
- Une vue horizontale qui exprime les relations entre les tâches et entre ceux qui les mettent en œuvre.

Nous remarquerons dans ce chapitre que l'analyse d'une erreur se fait uniquement par décomposition plus ou moins hiérarchique des fonctions et par raffinements successifs.

3.2.5.1. Vue verticale

Pour comprendre une erreur, il faut en connaître la source. Aussi, l'utilisateur doit-il définir le maximum de fonctions et d'opérations qu'il souhaite réaliser, dans le cahier des charges, lors de la phase spécification fonctionnelle.

La vue verticale consiste à détecter le moment et l'endroit où a lieu l'erreur et à remonter progressivement jusqu'à la source. Cette remontée peut se faire de différentes façons :

- la manière abstraite consiste à analyser uniquement les différentes actions / fonctions effectuées auparavant. Cela peut renvoyer l'utilisateur à la partie spécification fonctionnelle du cahier des charges.
- la manière concrète se base uniquement sur les détails techniques, de la syntaxe de la ligne d'instruction vers le bloc de fonction, ensuite vers le langage utilisé, le système d'exploitation, jusqu'au matériel. Cela renvoie alors l'utilisateur à la partie spécification technique du cahier des charges.
- la manière logique s'intéresse uniquement à la structure de l'ensemble du programme. L'algorithmique et la méthode de travail, normalement décrites dans la spécification organisationnelle du cahier des charges interviennent dans ce cas.

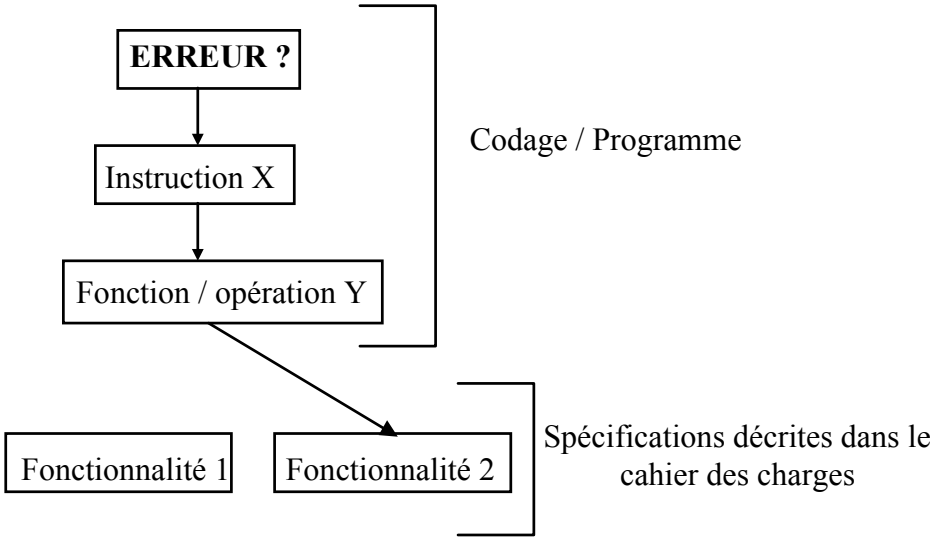


Figure 3-4 : Vue verticale suite à la détection d'une erreur

La vue verticale donne généralement une excellente évaluation de chaque erreur détectée. En effet, comme nous l'avons développé dans le chapitre sur les facteurs d'erreurs, la majorité des erreurs proviennent de l'individu. Il s'agit alors de suivre et de vérifier en détail chaque acte effectué pour la réalisation d'une fonction donnée. Cette vue qui offre la ramification

complète de la propagation d'une erreur dans un système, permet ainsi une vérification détaillée de l'ossature du programme et de l'enchaînement des idées.

3.2.5.2. Vue horizontale

La vue horizontale est moins pertinente que le fait d'aller en profondeur pour comprendre une erreur. En effet, il s'agit d'étudier les relations entre les différentes tâches se situant au même niveau. Par exemple, pour constituer une séquence d'images animées, la fonction servant à distribuer les différentes images et la fonction pour les inclure dans un scénario se situent au même niveau. Ainsi, lorsqu'il y a une erreur dans le scénario, l'utilisateur peut rechercher la cause dans la distribution ou dans la constitution des séquences d'images.

La vue horizontale peut être comparée à la programmation parallèle. Comme le montre le schéma ci-dessous, pour comprendre une erreur, on peut mettre en cause simultanément plusieurs fonctions.

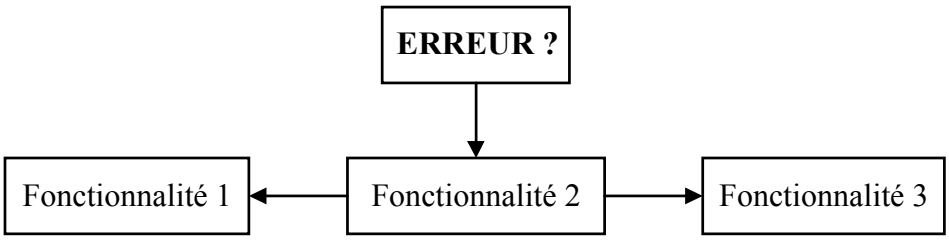


Figure 3-5 : Vue horizontale suite à la détection d'une erreur

Mais la navigation entre les différentes tâches peut se réduire à une simple ballade avec une faible mémorisation et une structuration superficielle des connaissances [Labat, 91]. En effet, l'utilisateur n'a aucun recul vis-à-vis de l'ensemble et risque même de tourner en rond car les autres fonctions ou opérations parallèles, auxquelles il a recours, peuvent être également erronées. L'évaluation d'une erreur peut alors être superficielle.

En revanche, la vue horizontale offre l'avantage de pouvoir vérifier et valider les liens entre les différentes fonctionnalités impliquées. Elle permet ainsi une consolidation globale du logiciel face à une erreur.

3.2.5.3.Principe d'héritage dans l'erreur

Les deux axes de vue décrits précédemment montrent la propagation de l'erreur, et ce à partir de la source de l'erreur jusqu'aux lignes de code correspondant. On obtient alors une sorte d'arborescence non binaire car un élément peut être associé par la relation « entraîne l'erreur sur », à plusieurs éléments qui lui sont dépendants. Cette arborescence permet de connaître la gravité de l'erreur et démontre une certaine hiérarchie dans l'erreur.
Nous pouvons faire un rapprochement avec la notion d'héritage de la programmation objet, car on retrouve les mêmes définitions et principes.

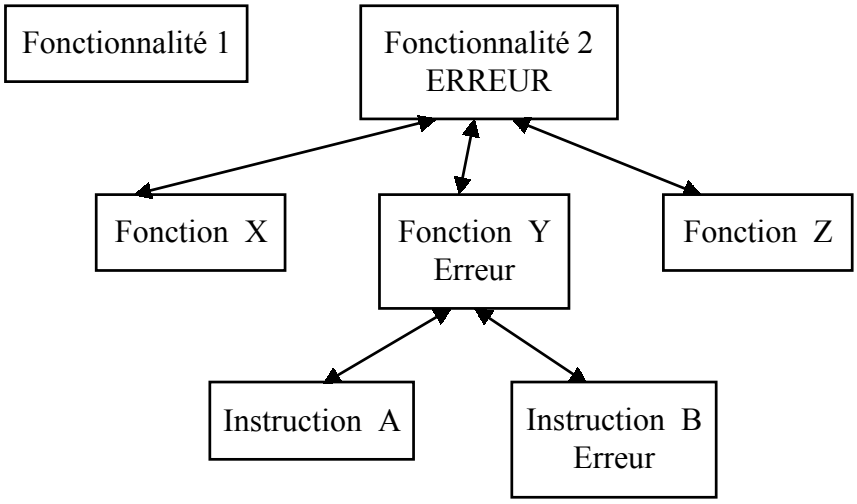


Figure 3-6 : Arborescence de propagation de l'erreur

L'ISO[24] a retenu le formalisme « entité-association » pour décrire, pendant la conception d'un système, l'aspect conceptuel des données, en distinguant les entités (éléments), des associations entre les entités.

- Une entité désigne un type ou une classe ou un ensemble d'objets ayant des caractéristiques communes. Les sous-entités héritent de ces caractéristiques.
- L'association représente une rencontre, une correspondance entre des entités.

Les études menées par Peter Chen (1976) décrivent par exemple, ce formalisme appliqué au langage naturel : les noms comme des entités, les verbes équivalents aux associations et les adjectifs comme des attributs des entités.
Cette illustration nous montre que l'on peut aller jusqu'à un niveau de détail pointilleux pour appliquer le principe d'héritage. En particulier, par rapport à notre étude, l'analyse de chaque opération effectuée est intéressante car elle peut révéler des erreurs dans l'opération elle-

[24] International Standardization Organisation

même ou dans d'autres opérations qui lui sont associées par la vue verticale ou par la vue horizontale.

3.3. NOTION DE ROBUSTESSE

Malgré le constat que développer un didacticiel ou une présentation interactive prend en moyenne huit fois moins de temps en utilisant un SAM que de compiler du code [Rousseau, 96], il ne faut pas oublier qu'il faut 1 à 2 secondes pour commettre une erreur mais 15 ou 30 minutes voire des jours pour la détecter et encore plus pour éventuellement la corriger [Bailey, 82].
La documentation opérationnelle est habituellement développée à la fin du développement (artefact). De ce fait, elle a tendance à ne pas tenir compte de certaines erreurs existantes dans le logiciel. En effet, le rédacteur du document est plutôt absorbé à découvrir ce dont est capable le logiciel au niveau des fonctions et opérations possibles et non pas aux erreurs et faiblesses d'utilisation, c'est-à-dire à la robustesse du logiciel.

Le terme « robustesse », d'après Le Petit Robert [1996], désigne la qualité de ce qui est robuste, donc à la fois fort et résistant, de par sa solide constitution. Est qualifiée de robuste une chose qui se développe dans des conditions difficiles. Le Larousse Multimédia Encyclopédique [1996] rajoute à cette définition le terme vigueur et applique le mot robuste à un objet solidement construit et capable de résister à des efforts extrêmes et à un usage prolongé.
Le terme « robustesse » appliquée aux SAM, prend en général son sens à partir de ces définitions. Il s'agit alors, lorsque nous parlons de SAM robuste, d'un logiciel dont :

- l'objectif a été défini clairement et précisément,
- les analyses effectuées pendant les différentes phases de conception, décrites par la suite dans un cahier des charges, ont été menées avec rigueur,
- le développement ou codage du logiciel a été solidement effectué,
- les tests de validation ont été faits en tenant compte du maximum de conditions d'utilisation possible du logiciel.

La figure 2-10 nous montre les étapes concernées par la robustesse pendant la conception. Nous remarquons que la validation y est fréquente.

Dans le cadre de notre étude, notre motivation première porte surtout sur la robustesse résultant des tests de validation. Pour confirmer notre vision sur le terme « logiciel robuste », nous nous sommes aidés des réponses obtenues à la question « Qu'est-ce qu'un logiciel robuste ? » que nous avons posé à une trentaine d'individus dont :

- 10 personnes profanes connaissant les termes utilisés dans le questionnaire et ayant déjà utilisé un ou plusieurs logiciels,
- 10 débutants en informatique, ayant les connaissances de base en informatique et ayant déjà fait de la programmation,
- 10 informaticiens confirmés, ayant faits plusieurs années d'études et ayant des expériences professionnelles en informatique.

Les personnes qui ont répondu au questionnaire ne sont pas averties de l'objectif de notre recherche ni du domaine auquel s'applique le terme logiciel. Chaque personne peut donner plusieurs réponses. Nous avons obtenu globalement les réponses suivantes :

Un logiciel robuste est un logiciel qui :	Profanes	Débutants	Confirmés
répond aux descriptions du logiciel			6
ne présente pas de conflits avec le système et le matériel			10
ne présente pas d'anomalies de fonctionnements		5	7
laisse l'utilisateur libre de ses actes quel que soit l'objectif	10	8	1
facile à utiliser	3		2

3.3.1. Robustesse et qualité de l'interface homme-machine

Comme nous l'avons présenté dans les chapitres précédents, un SAM doit offrir un certain nombre de fonctions auxquelles il faut rajouter un ensemble de contraintes et performances, pour satisfaire l'utilisateur. Parmi cet ensemble, on peut citer [Martin, 87] :

- les conditions d'exécution de chaque fonction : à la suite de quel événement, de quelle commande, de quelle combinaison de valeurs de données, la fonction est-elle exécutée ?
- la nature des interfaces homme-machine : par quels moyens les données en entrée sont-elles acquises ? les données en sortie sont-elles fournies ?

Les caractéristiques, souvent testées lors des contrôles de qualité d'un logiciel, sont liées à l'interface homme-machine qui se décline en deux parties :

- l'environnement d'exploitation
- l'environnement de maintenance et de suivi.

En ce qui concerne l'environnement d'exploitation, nous pouvons souligner :

- la *couplabilité* : aptitude du système auteur à communiquer et à fonctionner avec d'autres logiciels. Différents progrès ont été faits grâce à l'intégration du multimédia car on peut directement intégrer par exemple les résultats d'un logiciel de créations d'images gifs animés dans un SAM comme Netscape Composer.
- *l'efficacité* : aptitude du SAM à minimiser l'utilisation des ressources (matériels et système d'exploitation) disponibles. En effet, les systèmes auteurs sont déjà assez lents par rapport aux autres logiciels car il existe un module de runtime faisant office d'interpréteur de scripts [MediaSet, 94].
- la *maniabilité* : aptitude du système à être convivial et facile d'emploi quel que soit le niveau de l'utilisateur. Cette caractéristique est étroitement liée à l'ergonomie et à la pédagogie. L'arrivée des aides et boutons graphiques mis à la disposition de l'utilisateur ont énormément amélioré ce critère.
- la ***robustesse*** : aptitude d'un logiciel à conserver un comportement conforme aux besoins exprimés en présence d'événements souhaités ou non prévus et à supporter un maximum d'actions de la part d'un utilisateur. Ce critère de qualité et de fiabilité est étroitement lié à la capacité du SAM. Par exemple, la notion de feed-back, pour assurer la qualité du logiciel, donc sa robustesse doit être en permanence prise en compte par le concepteur du SAM.

La notion de robustesse implique un environnement de maintenance et de suivi résistant aux éventuels événements favorables ou non au bon fonctionnement du logiciel. Bastien et Scapin définissent, parmi leurs critères ergonomiques pour l'évaluation, la *protection contre les erreurs,* en d'autre terme *la robustesse,* qui concerne ‹‹ les moyens mis en place pour détecter et prévenir les erreurs d'entrées de données ou de commandes ou les actions aux conséquences néfastes ›› [Bastien et Scapin, 1993].

Les études et travaux pour pallier les problèmes de robustesse ont donné plusieurs résultats positifs. Les techniques les plus utilisées sont le prototypage et la simulation. Ces deux approches ont en plus l'avantage d'être basées sur une vision orientée utilisateur [Van-Eylen, Hiraclides, 96]. En effet, les deux méthodes donnent une vision rapide donc une ébauche du produit final. Elles favorisent l'expression des idées de l'utilisateur car il y une élimination rapide des choix non satisfaisants et une vérification du comportement du logiciel.

Le *prototypage* est une technique de construction d'implémentations partielles permettant aux utilisateurs de mieux cerner le problème et sa solution. Il présente également l'avantage de produire des logiciels plus simples à maintenir. Il existe deux approches :

- *les maquettes jetables* présentent l'avantage d'être applicable à n'importe quelle phase de développement. Il s'agit alors d'un simulateur du comportement du produit. Du moins, certains détails importants comme la pédagogie peuvent ne pas être considérés. Cette démarche présente alors l'avantage d'être rapide et provisoire mais manque de rigueur.
- *les prototypes évolutifs* sont repris et adaptés successivement jusqu'à satisfaction de l'utilisateur. L'inconvénient de cette démarche est qu'il est quasiment impossible d'improviser les actions. En revanche, elle minimise un certain laxisme de la part du concepteur et permet d'optimiser la flexibilité du produit.

Quelle que soit l'approche adoptée, les produits prototypés, bien que simples d'emploi, donnent des résultats moins satisfaisants en ce qui concerne les aspects fonctionnalités et robustesses [Jaulent, 94]. En effet, les prototypes sont limités aux fonctions souhaitées par le concepteur. Comme ce dernier peut émettre des oublis ou des erreurs, il n'y a aucune garantie de satisfaction totale de l'utilisateur sur la richesse et la robustesse des fonctionnalités proposées.

Le système de *simulation* permet d'obtenir toutes les procédures de résolution possibles [Roulin, 90]. Cela consiste à simuler chaque opération relative à chaque fonction décrite dans le cahier des charges. Cette approche permet à l'utilisateur de valider plus minutieusement le logiciel. Mais, cette validation qui augmente la qualité du logiciel, ne garantit pas sa robustesse car l'ossature de l'ensemble des opérations n'est pas vérifiée. De plus, les tests sont effectués suivant la logique et le goût du concepteur lui-même. Par expérience, les cahiers des charges survolent souvent les précieux détails relatifs à la robustesse.

3.3.2. La robustesse, partie intégrante de la méthodologie de conception

Un système auteur doit permettre à son utilisateur de construire les modules constituant son logiciel indépendamment les uns des autres et ce à n'importe quel moment du développement. Cette possibilité de manipuler à sa guise et librement les outils et fonctionnalités d'un système, n'est pas suffisamment exploitée ou maîtrisée dans les systèmes actuels.

Par exemple, nous avons parlé dans le sous-chapitre précédent de la notion de feed-back. Elle n'est pas systématiquement prise en considération, ou mal intégrée, dans les méthodes de conception. Or, elle offre à l'utilisateur une liberté d'actions, et parfois même la possibilité d'avoir une vision globale du système. Le feed-back permet également à l'utilisateur de corriger les erreurs rencontrées à n'importe quel moment et avec moins de contraintes dans ses manières d'explorer le SAM.

Les différentes séances de travaux pratiques[25] pour aider les étudiants en DEUG Informatique à acquérir les bases en langage C, nous ont montré qu'il y a une « sur-estimation » de certaine phase de la conception. Nous pouvons penser par exemple à un étudiant qui attache une importance particulière à des détails syntaxiques relatifs au langage de programmation qui sera utilisé, détails non pertinents pour une bonne compréhension de l'objectif du projet. On considère que l'individu, dans cette situation n'a pas atteint *un bon niveau d'abstraction*[26].
Ces mêmes expériences nous ont montré qu'il y a *transfert* d'une procédure connue à une situation nouvelle. L'étudiant tentera d'appliquer un programme qu'il a construit pour un autre problème, au nouveau problème qui lui est posé [Hoc, 79].
Il arrive aussi que l'étudiant assimile le nouveau problème à l'aide de connaissances élaborées dans un autre domaine : on parle alors de résolution par *analogie*, plutôt que de transfert.

Si nous nous référons au fait que les phases amont du cycle de vie d'un logiciel éducatif sont exclusivement pédagogiques [Marquesuzaa, Nodenot, 96], l'environnement conceptuel d'un didacticiel, ici le SAM ou l'AGD doit accorder une importance particulière à cette caractéristique, quel que soit l'objectif. Or, cet environnement très orienté vers la pédagogie, qui semble être naturel pour un enseignant peut devenir une contrainte pour le concepteur du didacticiel.
Par exemple, un historien aura tendance à s'intéresser de près aux fonctions multimédias pour animer son logiciel au lieu de se concentrer sur la cohérence de l'ensemble. Il en oublie parfois les connaissances de l'utilisateur au niveau informatique et plus grave au niveau de l'objectif de son logiciel. Cet enthousiasme peut même aboutir à des blocages voire des paniques suite à des essais techniques non concluants. Cela n'améliore bien évidemment pas le bon déroulement de son projet.

[25] TP de langage C avec une quarantaine d'étudiants de l'Université Paris 8 pendant deux ans.

[26] Cela démontre souvent une certaine peur de la programmation.

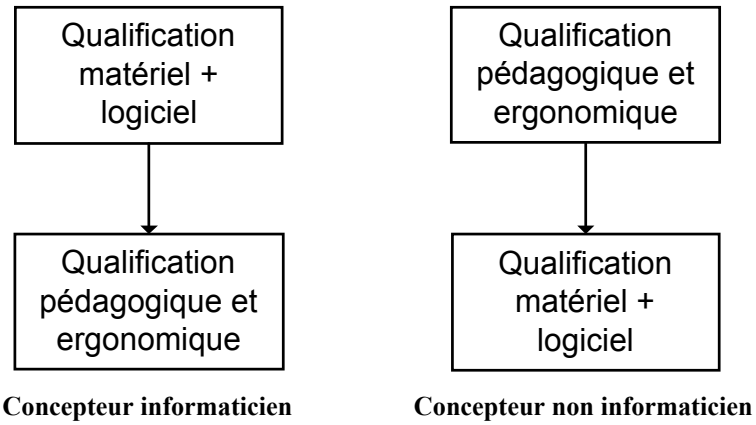


Figure 3-7 : Comparaison de deux types d'approches d'un logiciel

La figure 3-6 nous présente les deux types d'approches d'un logiciel. Comme nous venons de l'expliquer, les deux tendances peuvent être en défaveur de la robustesse d'un SAM et d'un AGD.

Pour mettre en œuvre la robustesse, nous devons jumeler les deux tendances. Pour ce faire, nous allons étudier les points à retenir dans les méthodes utilisées aujourd'hui particulièrement Merise. A chaque niveau d'analyse pour la définition des besoins, nous allons associer les questions correspondantes :

1. conceptuel : quoi ?
2. organisationnel : qui ? où ? quand ? combien ?
3. physique : comment ?

Chacune de ces trois questions concerne simultanément la qualification pédagogique, organique et la planification matériel et logiciel. Lorsque l'on analyse les questions, nous pouvons constater qu'il est difficile de faire la séparation de la qualification des besoins d'un projet en deux étapes différentes (figure 3-6).

La qualité des réponses apportées à ces questions fondamentales, a une influence considérable sur la robustesse du logiciel. De plus, le concepteur utilisant un système auteur doit constamment se poser la question *« Quelle est la réaction de l'utilisateur en voyant ou en manipulant telle ou telle partie de l'application ? »*

Par conséquent, la qualité de la méthode appliquée conditionne la robustesse d'un SAM, et réciproquement. La robustesse est donc, partie intégrante de la méthodologie de conception.

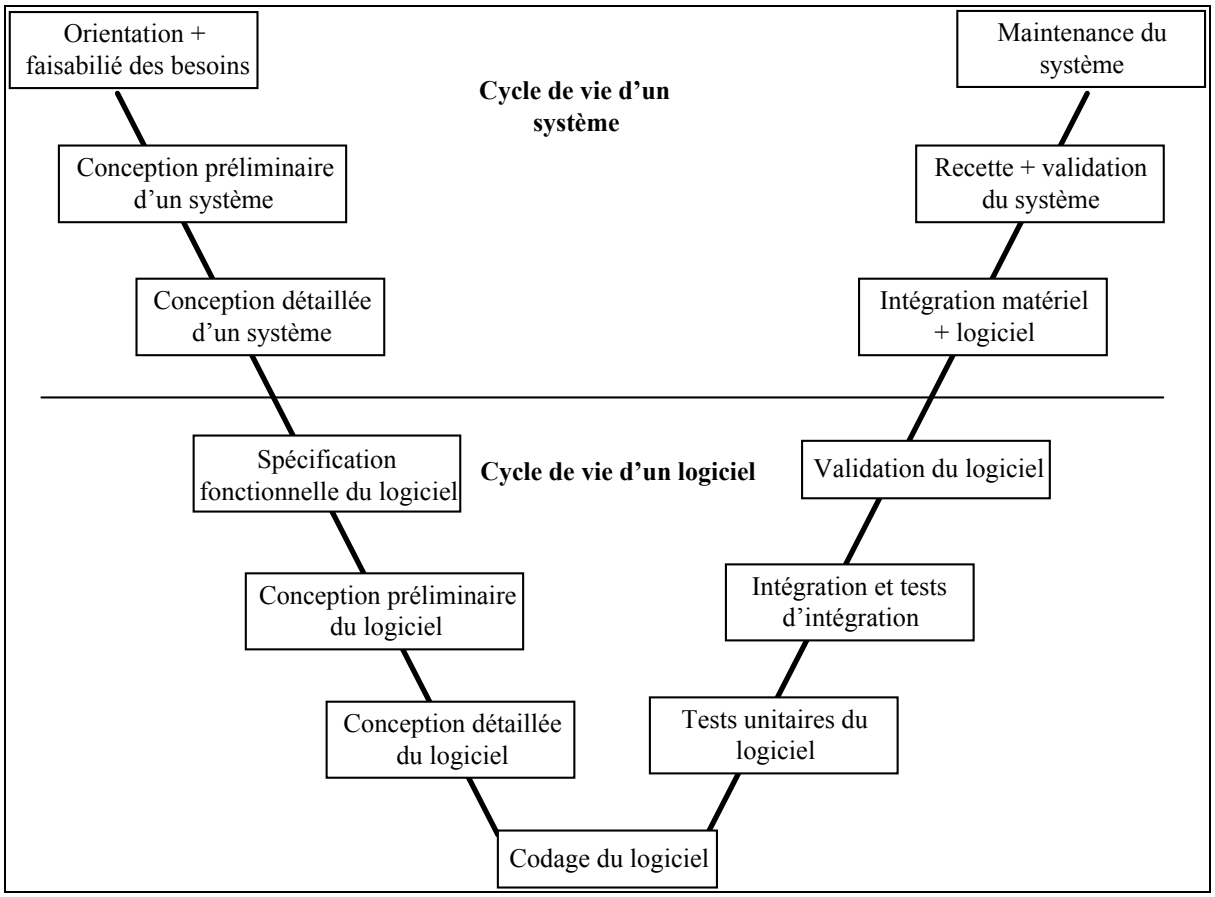


Figure 3-8 : Cycle de vie d'un système et d'un logiciel [27] [Jaulent, 94]

Une méthode signifie une formalisation du cycle de vie d'un logiciel [Ermine, 93]. Le schéma ci-dessus donne une vision globale de tout ce qui est relatif au cycle de vie d'un SAM. Les méthodes de conception disponibles aujourd'hui, mettent plus ou moins en évidence chaque élément de ce graphe. La raison principale vient du fait que si on prend en considération tous les éléments, la méthode devient très condensée, complexe, difficile à mettre en œuvre et à apprendre. De plus, il y a des risques de redondance dans certaines fonctions.

Or, pour être pertinent, un modèle méthodologique de développement d'un projet logiciel doit être : universel, évolutif et réutilisable [Coulette, 90]. Ces trois critères rentrent dans le cadre de la robustesse du système. Pour assurer leurs applications dans un projet de conception de logiciels à partir d'un SAM, le modèle doit respecter entièrement la partie ascendante (à droite) du schéma ci-dessus :

⇒ Maintenance du système permet de modifier le logiciel pour éliminer au fur et à mesure les logiciels et surtout pour faire évoluer le produit.

[27] Découpage normalisé DoD 2167 par l'ISO

⇒ Recette et validation du système consistent à simuler les grandes fonctionnalités décrites dans le cahier des charges afin de vérifier globalement si le système répond à l'objectif.

⇒ Intégration matériel et logiciel sert à vérifier la compatibilité entre les équipements choisis lors de la spécification technique, exploités par le logiciel.

⇒ Validation du logiciel permet de vérifier la bonne intégration et compatibilité des différentes fonctions constituant le logiciel.

⇒ Intégration et tests d'intégration contrôlent l'intégration des diverses informations que le logiciel peut traiter.

⇒ Tests unitaires du logiciel valident minutieusement l'ensemble.

3.3.3. Systèmes intelligents et robustesse

Si nous faisons référence au fait que les systèmes auteurs sont des systèmes interactifs dans lesquels l'auteur ne fournit que des informations pédagogiques, on peut conclure que ces systèmes sont dotés d'un ensemble de mécanismes assurant tout ce qui est relatif aux autres besoins d'un projet tel le codage, le stockage des informations, etc. En effet, les informations pédagogiques ne constituent que les données rentrées. Il faut par la suite les traiter et les faire « vivre ».

La robustesse implique une interactivité car l'utilisateur est actif, et suivant l'intensité de cette activité naît une émergence du sens. Dans cette interaction, l'utilisateur associe à chaque objet du système son interprétation, donc la signification qui lui importe [Vacherand-Revel, Bessière, 90].

L'intelligence d'un système dépend alors avant toute chose, de l'utilisateur, plus précisément de la satisfaction donnée par le système par rapport à l'attente de l'utilisateur. Ensuite, l'intelligence vient des différentes interactivités autorisées par le produit. La quantité, la richesse de ces fonctionnalités interactives démontrent la robustesse du SAM car ce dernier sera alors résistant à toutes actions volontaires ou non de l'utilisateur.

Le principe consiste alors à se rapprocher au maximum de l'utilisateur tel un système expert qui cherche à reproduire la démarche d'un expert humain vis-à-vis d'un problème [Laurière, 82]. Un expert humain a une grande quantité de connaissance et par la faculté de les extraire, de les combiner à un instant donné, il arrive à des performances particulières. Ces performances font partie des qualités requises pour la robustesse.

Par ailleurs, un système intelligent ne doit pas forcément aller à l'encontre du cycle de développement classique d'un logiciel. L'intelligence, pour nous se situe dans la phase conceptuelle et non dans la phase d'implémentation. En effet, les grandes fonctionnalités du produit final, ainsi que les éventuelles futures interactivités, sont définies lors de cette phase.
De même, l'analyse des éventuels problèmes et sources d'erreurs possibles se fait lors de cette étape.
Un système intelligent doit proposer une démarche projet pour gérer, concevoir mais également valider des produits, selon une approche coopérative [Jaulent, 94]. Une approche coopérative signifie faire travailler simultanément, suivant une démarche incrémentale et interactive, différents acteurs issus d 'une équipe pluridisciplinaire (spécialistes des disciplines techniques mises en jeu dont la pédagogie, l'ergonomie , etc.).

3.3.4. Champs d'application du concept de la robustesse

Dans un AGD, les thèmes suivants sont considérés comme primordiaux [Madaule & al, 92] :

- les interfaces homme-machine,
- les processus de conduite de projet,
- l'édition structurelle,
- l'audit des configurations,
- la maintenance et le diagnostic du logiciel,
- les tests unitaires,
- la qualimétrie du logiciel,
- l'évaluation des logiciels.

Ces thèmes sont relatifs à la phase conceptuelle du projet de réaliser un logiciel, éducatif ou non, à partir d'un AGD. Dans le cadre de notre étude, nous allons alors surtout nous intéresser à la robustesse qui découle de cette phase.

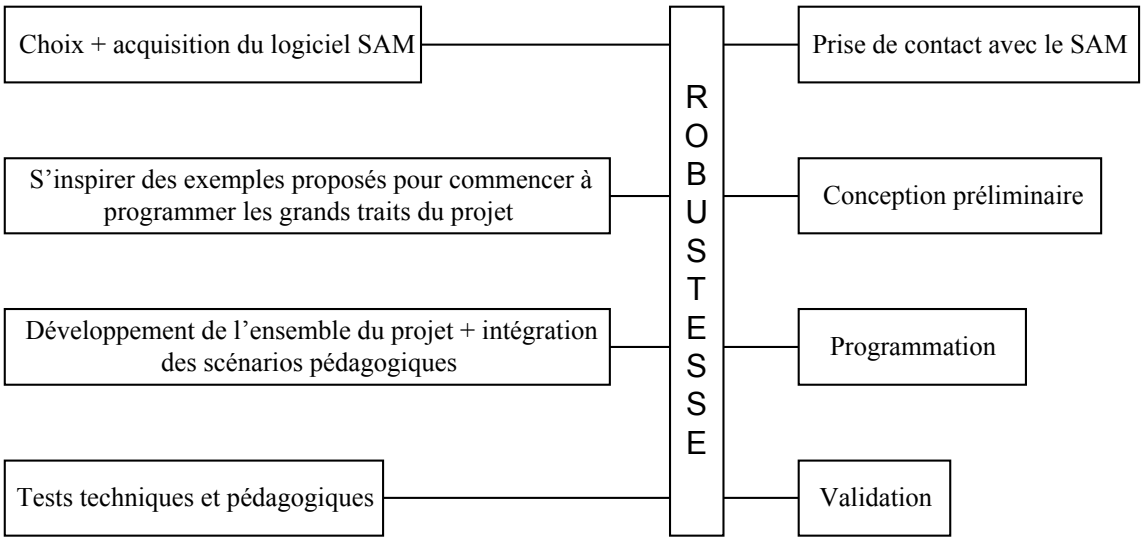


Figure 3-9 : Positionnement de la robustesse dans un AGD

La figure 2-10 nous montre l'emplacement de la robustesse dans les différentes phases de réalisation d'un logiciel. Le schéma ci-dessus donne le positionnement de la robustesse par rapport à la conception d'un didacticiel à partir d'un AGD. La différence entre les deux graphes vient du fait que le critère de robustesse couvre également la phase de validation dans un AGD.

Si l'auteur de niveau 1 ne fournit que des informations pédagogiques dans un SAM, cela suppose la forte orientation de ces systèmes vers l'enseignement. D'ailleurs, il faut considérer l'ordinateur comme un outil pédagogique supplémentaire au même titre que le document écrit ou audiovisuel [Madaule, 87].

3.3.5. Evaluation de la robustesse

Etant donné le champ d'application assez vaste[28] de la robustesse et surtout sa définition assez abstraite, nous ne voulons pas prétendre ici donner la méthode radicale pour faire l'évaluation complète de la robustesse. De plus, la robustesse est un critère qui n'est pas mesurable de manière objective. Les mesures qui peuvent être éventuellement réaliser donnent des statistiques inférentielles[29]. Ce sont :

- Les mesures communes telles que le taux d'erreurs, le temps d'apprentissage, etc.

[28] Voir figure 3-9

[29] Statistique inférentielle : statistique donnée comme conséquence d'un ou plusieurs faits.

- Les mesures particulières telles que la facilité de réalisation des tâches composées, la facilité d'obtention d'informations sur le SAM et le déroulement des tâches, la capacité du SAM à réaliser les tâches, etc.

Concernant les méthodes d'évaluation, nous allons nous appuyer sur celles déjà existants sachant qu'aucunes d'elles ne permettent de donner des appréciations subjectifs comme « le plaisir d'utilisation des interfaces » [Nanard, 90][Balbo, 92].
Dans les sous-chapitres précédents, nous avons montré les relations fortes entre la notion de robustesse et l'interaction homme-machine ainsi que la phase de conception. Interaction homme-machine appelle forcément au concept d'interface homme-machine. Ainsi, évaluer la robustesse revient à contrôler la qualité des interfaces utilisateurs. Pour les questions fondamentales concernant l'évaluation de la robustesse dans le cadre des conceptions à partir de SAM, nous allons nous appuyer sur les travaux de S. Balbo [Balbo, 92] :

➔ **De quoi s'agit-il?** Il s'agit de contrôler la qualité ergonomiques et fonctionnelles des interfaces utilisateur.

➔ **Qui effectue l'évaluation?** L'évaluation de la qualité ergonomique des logiciels est le fait d'experts ergonomes qui peuvent se faire aider d'utilisateurs représentatifs.

➔ **Où s'effectue l'évaluation?** L'évaluation doit se pratiquer en laboratoire ou sur le terrain, dans le contexte réel d'utilisation. Seule la compétence de l'évaluateur intervient (heuristiques et théories prédictives).

➔ **Quand s'effectue l'évaluation?** L'évaluation se pratique dans l'espace IHM du processus de développement.

➔ **Pourquoi évalue-t-on les interfaces utilisateur?** L'objectif est de concevoir à travers un SAM pour une interaction homme-machine efficace.

➔ **Comment procède-t-on pour évaluer les interfaces ?** Généralement, le processus commence par la définition d'une situation de référence. Il inclut ensuite une phase d'expérimentation puis une analyse comparative des performances attendues et effectives.

D'après J. Coutaz [Coutaz-95], il existe cinq domaines de préoccupation dans l'évaluation :

- Les moyens humains ,
- Les ressources matérielles ,
- Les connaissances requises,
- Les facteurs situationnels ,

- La nature des résultats.

Pour effectuer ces évaluations, J. Coutaz propose deux méthodes :

- La méthode empirique ou expérimentale : elle repose sur le recueil des données comportementales des sujets en situation. Les techniques les plus souvent rencontrées sont la manipulation directe, la vidéo, les interviews, les questionnaires ainsi que la capture d'actions. Les informations rassemblées seront interprétées et serviront par la suite à concevoir une bonne interface utilisateur.
- La méthode prédictive ou théorique : elle est limitée et ne fournit pas des résultats exacts. Cependant, elles assistent le concepteur d'IU à la mise en place de l'architecture de l'interface et formaliser quelques idées de conception. Ses méthodes se basent entièrement sur des actions de prédiction telles que les durées d'exécution d'une commande ou d'apprentissage et la qualité d'affichage.

De ces deux méthodes, la première nous semble la plus adaptée à la robustesse.

3.4. EVOLUTION DU SAM VERS L'AGD

On définit un système d'EAO par un ensemble de fonctionnalités pour manipuler des objets et des concepts essentiellement pédagogiques [Quere, 83]. L'Enseignement Assisté par Ordinateur, comme les systèmes auteurs, met en scène deux acteurs principaux :

- l'auteur qui conçoit le projet en définissant les objectifs pédagogiques et le contenu didactique,
- l'apprenant qui dialogue avec le système.

Grâce au principe d'interactivité, l'EAO est aujourd'hui abandonnée en tant que voie de recherche au profit de l'EIAO, qui signifiait au début Enseignement Intelligemment Assisté par Ordinateur et est défini aujourd'hui comme l'Environnement Interactif d'Apprentissage par Ordinateur. De plus, le terme d'enseignement assisté par ordinateur est généralement associé à l'utilisation de systèmes auteurs, qui ont le désavantage d'être relativement restrictifs et par conséquent de limiter la capacité créative des concepteurs de didacticiels [Aubord, Ibrahim, 91]

Un système auteur est un éditeur de didacticiel, donc un logiciel d'écriture de didacticiel déchargeant l'auteur du travail de programmation informatique, pour lui permettre de mieux

se consacrer à la structuration pédagogique [Madaule, 87]. Cela se rapproche beaucoup de la fonction d'un atelier de génie didacticiel (AGD) : fournir des outils spécialisés, mais intégrés dans un même poste auteur.
La terminologie AGD vise à souligner le lien technique entre l'EAO et les autres champs d'application de l'informatique [Vacherand-Revel, Bessière, 90]. La tâche de l'auteur consiste à produire un didacticiel, à l'aide de l'outil AGD, en prenant en compte par anticipation les apprenants auxquels il le destine.

Un AGD doit prendre en compte la diversité des objectifs pédagogiques et doit donc tenir en compte les stratégies suivantes [De La Passardière, 93] :

- Des exercices permettant à l'apprenant d'acquérir une connaissance par la pratique.
- Un tutoriel qui permet à l'élève d'avoir un minimum de fil conducteur dans ses parcours du didacticiel.
- Un simulateur pour permettre à l'étudiant de découvrir une heuristique et une méthodologie de travail dans l'objectif de consolider son savoir-faire.

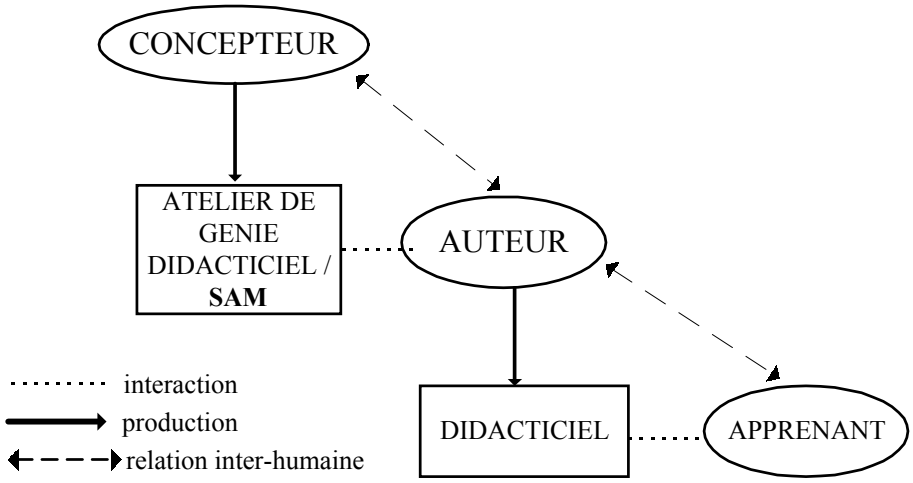


Figure 3-10 : Les acteurs et outils impliqués dans un AGD[30]

Sur le schéma ci-dessus, nous pouvons remarquer la proximité et la similitude entre AGD et SAM. Cela nous aide à comprendre l'importance de la pédagogie et de l'ergonomie qu'il faut implémenter dans notre logiciel «expert» qui aidera l'auteur de niveau 1 à créer son logiciel.

[30] Schéma inspiré de [Vacherand-Revel, 90]

Mais génie didacticiel renvoie automatiquement au terme génie logiciel. Le génie logiciel signifie les méthodes, les langages, les ateliers imposés ou préconisés par les normes adaptées à l'environnement d'utilisation afin de favoriser la production et la maintenance de composants logiciels de qualité [Jaulent, 90].

Les concepts de structuration indispensables pour satisfaire aux exigences de qualité requises, donc à la robustesse du produit, en génie didacticiel et en génie logiciel en général sont [Jaulent, 90] :

- la modularité qui permet de partitionner un système en un ensemble de constituants moins complexes. Ce découpage donne comme résultats des modules ou classes d'objets que l'on peut réutiliser pour construire d'autres systèmes.
- l'abstraction qui désigne l'interprétation d'un constituant du logiciel, permettant d'en extraire une ou plusieurs vues, selon les besoins du domaine. Exemple dans un SAM, les abstractions peuvent concerner :
    - des données : les fichiers images, sons et textes , etc.
    - des fonctions : revenir en arrière, supprimer, ajouter , etc.
    - des contrôles : taille de fichier maximale, nombre maximum d'images , etc.
- le masquage d'informations permettant au logiciel de s'adapter aux différents niveaux de connaissance des utilisateurs.
- la hiérarchie correspondant à une relation entre les différents constituants du système.

3.5. CONCLUSION

Les erreurs ne doivent pas être considérées comme des problèmes mais comme des procédures normales et inévitables, indicateurs des stratégies adoptées par l'utilisateur [Strevens, 69]. La synthèse des différents types de bogues informatiques et des méthodes pour les résoudre aident à définir l'application qui joue le rôle d'expert informatique pour aider un auteur de niveau 1 à créer un logiciel à partir d'un SAM ou d'un AGD. Cette application doit délimiter ses frontières à partir de la distinction entre les erreurs internes et externes, entre les erreurs de conception et les erreurs opérationnelles.

De même, l'étude des notions d'erreur et de robustesse nous donne la perspective de trouver une méthode à appliquer à tout type de logiciel pour le rendre robuste.

Maintenant, tout en sachant que les méthodes robustes garantiront que le résultat est bon pour une très grande collection de distributions sans pour autant être les meilleures pour une en particulier [Rappachi, 95], notre rôle consiste à :

- Eliminer les causes de défaillances par des méthodes de détection et de non-propagation de l'erreur.
- Réduire les nombres de défauts par application de méthodes de vérification,
- Eviter les fautes humaines.

Les études sur l'erreur et la robustesse nous ont montré que l'homme s'avère être paradoxalement le meilleur et le pire des éléments de sûreté d'un système, ces faits sont les conséquences des différents aspects de son fonctionnement cognitif et psychologique [Guignier, 91].

4. VERS DES ENVIRONNEMENTS DE CREATION DE LOGICIELS ROBUSTES OU ECLR : PRINCIPES DE CONCEPTION

4.1. INTRODUCTION

Nous nous sommes intéressés aux erreurs[31] parce qu'elles peuvent constituer des symptômes de l'existence d'une conception dont on peut montrer qu'elle a par ailleurs, un domaine de validité attesté par le succès à résoudre une certaine classe de problèmes. Dans ce sens, une conception, même fausse est une véritable connaissance ; c'est l'affirmation qui porte à dire : l'élève connaît, mais il connaît autrement [Balacheff & al, 93].

Or, un concepteur de programmes exprimera en général ce qu'il désire non pas en termes de sous-programmes, variables internes et autres structures informatiques, mais plutôt en des termes appartenant à son monde [Aubord, 91]. Ce principe est valable quel que soit l'outil utilisé pour réaliser une application et entièrement vrai pour les individus se basant sur un SAM pour concevoir.

Mais ce problème ne vient pas seulement de la distinction entre le langage homme et le langage machine, mais aussi de la méthode appliquée. Les logiciels utilisent très souvent des méthodes éloignées du raisonnement humain [Nicaud, 89]. En effet, les problèmes liés à la connaissance sont souvent complexes et globaux pour pouvoir être définis rigoureusement ou spécifiés complètement lors de la conception de tout système, dont le SAM.

Nous pensons alors à un environnement permettant de résoudre une partie de ces problèmes que nous appellerons ECLR ou Environnement de Création de Logiciel Robuste. En effet, notre étude ne porte pas sur la correction ou la modification du SAM sachant que le SAM est très souvent un logiciel propriétaire, donc compilé. Il ne s'agit pas non plus d'un outil de certification de la robustesse sachant que pour nous, il n'y a pas lieu de normaliser la robustesse qui dépend uniquement du contexte et du souhait de l'auteur de niveau 1. Il s'agit

[31] Voir chapitre 3.2.1 « Problématique vis-à-vis de l'erreur"

pour nous, d'effectuer des recherches afin de créer une application que l'auteur de niveau 1 fera tourner simultanément avec le SAM telles que les macro-instructions inclus dans un logiciel de traitement de texte comme Word. Cet outil essaie de couvrir les différentes évaluations d'un logiciel [Delozanne, 99] :

- détection et correction des défauts, contrôle de qualité et test de performance.
- test de déverminage et d'acceptabilité.
- support utilisateur et maintenance.

Dans cette partie de notre étude, nous allons commencer par définir ce qu'est un ECLR. Sa mise en œuvre fait intervenir plusieurs acteurs à savoir l'auteur de niveau 1, le SAM, l'ECLR et l'utilisateur final. Ensuite, nous allons justifier l'intérêt d'une solution multi-agents car l'ECLR sera constitué de plusieurs agents dont nous donnerons les définitions. L'intégration de la notion de système multi-agents ou SMA permet d'avoir un environnement modulaire et interactif. Enfin, nous allons développer les approches possibles vers un ECLR d'un auteur de niveau 1 pour réaliser son logiciel.

4.2. DEFINITION D'UN ENVIRONNEMENT DE CREATION DE LOGICIEL ROBUSTE

La robustesse concerne l'implémentation à partir d'un système auteur du contenu du cahier des charges, en plus du nombre et de la gravité des problèmes rencontrés lors de son utilisation.

Robustesse implique également compensateur assurant la stabilité du système. Dans ce sens, il y a deux types de robustesse [Rozenberg, 98] :

- Robustesse en stabilité : le système demeure stable en présence d'incertitudes comme les erreurs de conception et des mauvaises manipulations de l'utilisateur. Un logiciel robuste doit avoir une bonne intégration avec le système pour minimiser les risques de plantage et pour être «insensible» à l'environnement matériel. Il doit également avoir des capacités de restauration d'environnements en cas de problèmes.
- Robustesse en performance : la performance est conservée en dépit des perturbations ou anomalies.

Ainsi, un environnement qui permet à l'auteur de niveau 1 de concevoir un logiciel robuste et dépourvu d'erreurs[32], est un environnement qui reste stable quel que soit l'action de l'utilisateur : intentionné ou non, prévu par le logiciel ou non, requis pour le fonctionnement du logiciel ou non. Mais, comme il est quasiment impossible d'intervenir dans le contenu du SAM lui-même dû au fait que notre rôle n'est pas d'apporter des modifications au SAM qui est par ailleurs souvent un logiciel propriétaire compilé donc non modifiable, nous allons nous aider d'une application supplémentaire et complémentaire qui procède en parallèle du SAM lors de la réalisation d'un nouveau logiciel.

Ces deux points de vue enrichis par la notion de robustesse décrite dans le chapitre 3.3, nous amènent alors à notre définition d'un ECLR :
Un environnement de création de logiciels robustes (ECLR) est une application qui offre à l'utilisateur d'un SAM la possibilité de concevoir un logiciel dans un environnement fiable et résistant à toutes actions possibles, plausibles et probables de l'utilisateur afin de rester cohérent par rapport aux spécifications décrites dans le cahier des charges.

4.3. MISE EN ŒUVRE D'UN ECLR

Le fait de mettre en œuvre une application supplémentaire en sus du SAM ne doit pas créer la moindre gêne à un auteur de niveau 1, qui rappelons-le, peut être néophyte en informatique. Le mot gêne couvre plusieurs champs :

- Problèmes supplémentaires au niveau de son environnement de travail : compatibilité avec le SAM, le système d'exploitation, les périphériques, etc.
- Blocage ou approche négative vis-à-vis d'un logiciel supplémentaire à utiliser.

Au contraire, la présence d'un ECLR doit rassurer l'utilisateur sur la qualité de son logiciel final quelles que soient les actions intentionnelles ou non de l'utilisateur final. Cela peut se faire grâce à deux approches :

- L'approche prévisionnelle : elle met en œuvre les moyens de prévoir ce que l'utilisateur est capable de faire
- L'approche préventive : elle vise à combler l'écart entre ce qu'il estime être capable de faire et ce qui est spécifié dans les cahiers des charges.

[32] Selon la définition de l'erreur que nous avons développée dans le chapitre 3.2

Ainsi, nous allons intégrer différentes techniques de développement pour créer l'ECLR :

⇒ La méthode de conception modulaire offre une souplesse d'utilisation à l'utilisateur mais également la facilité pour nous de garantir un bon fonctionnement contextuel avec le SAM et avec l'ensemble des techniques et technologies utilisées.

⇒ L'intégration des agents peut être considérée comme un vecteur incontournable dans la conception. En effet, les agents permettent d'introduire la problématique de l'intelligence collective et de l'émergence de structures par interactions [Ferber, 95].

⇒ L'interactivité permet à l'auteur de niveau 1 de jumeler l'informatique avec le côté qu'il apporte personnellement en sus des bases de connaissances relatives à son logiciel final : la pédagogie.

La pluridisciplinarité incontestable des SAM justifie ces choix qui nous ont amené à développer ECLR qui tient compte des approches informatiques et personnelles de l'utilisateur de niveau1.

4.3.1. Conception modulaire

Apprendre à programmer ressemble à apprendre à utiliser un kit de jeu de construction. Il faut alors commencer à construire quelques bouts de la construction à partir de quelques morceaux du jeu. Les quelques fragments ne peuvent être réalisés sans que l'individu ait un minimum d'images du résultat final. Il en est de même avec un programmeur qui ne peut commencer à écrire une ligne d'instruction telle la déclaration d'une variable sans savoir pourquoi. Les différents fragments seront ensuite rassemblés pour constituer l'ensemble. Le concepteur du programme doit alors connaître les relations qui peuvent avoir lieu entre les différents fragments, donc avoir une vision d'ensemble du programme.

Chaque fragment de ce jeu de construction correspond à une des fonctions qui composent l'ECLR. Sachant que l'objectif principal d'un ECLR est de minimiser les difficultés et incidents de conception de l'auteur de niveau 1, chaque fonction doit être suffisamment fiable et bien écrite afin d'être indépendante des autres. Comme dans un kit, un élément doit être transparent par rapport aux autres pour obtenir une convivialité et une facilité dans l'utilisation. Enfin, l'ECLR doit être construit de telle manière que chaque élément peut être associé à un ou plusieurs autres éléments sans difficulté. Dans cette partie de notre étude, nous allons présenter l'intégration dans un ECLR des principes relatifs à la conception

modulaire : l'indépendance, la transparence et la généralité de fonctions octroyées dans chaque module.

4.3.1.1.Indépendance des modules

La résistance d'un environnement de développement aux diverses actions de l'utilisateur peut être garantie en partie par la résolution de dysfonctionnements et de difficultés inter-modules, sachant qu'un ***module représente pour nous un ensemble de fonctions et opérations nécessaires pour effectuer une tâche de l'ECLR***. En effet, se trouver bloqué pour une action imprévue par un logiciel est très vite pénible même décourageant surtout si l'auteur de niveau 1 est un néophyte en informatique. Pour éviter qu'un problème ou un dysfonctionnement d'un module ne se répande sur les autres modules, il faut que les modules de l'ECLR soient les plus autonomes et les plus indépendants que possible.

Pour assurer l'indépendance des modules, on peut intégrer le principe de décomposition en couche des modules et de la hiérarchie des composants de chaque module[33]. Cela offre une meilleure visibilité et contrôle des actions faites par les composants de chaque module et de l'interdépendance entre les modules. Pour ce faire nous allons nous inspirer des réseaux de télécommunication où une couche est constituée d'un module. Chaque couche ne communique avec une autre qu'à travers des services tel que le décrit la figure 4.1. Selon le contexte, il peut s'agir d'un service demandé à la couche supérieure ou inférieure exemple : demande de données ; ou d'un service rendu qui résulte soit d'une demande venant d'une autre couche soit du déroulement normal du programme ECLR.

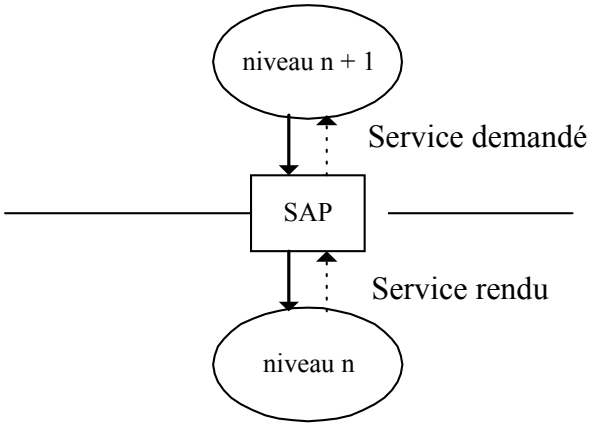


Figure 4-1 : Echange de services

[33] La hiérarchie des composants sera détaillée dans le chapitre 4.3.1.2.

Afin de valider et de contrôler ces échanges entre les couches, nous allons introduire la notion de SAP ou Service Access Point[34]. Les SAP sont gérés par une couche transversale globale et commune à tous les modules.

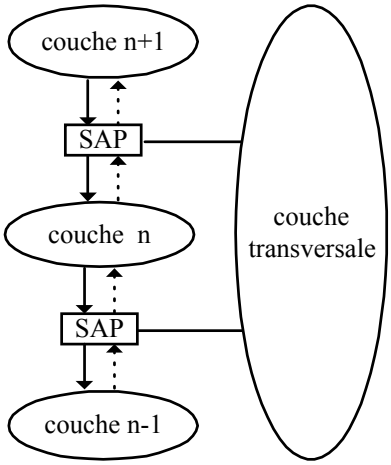


Figure 4-2 : Couche transversale et SAP

Ainsi, comme le montre la figure 4-2, l'indépendance des couches est bien évidente d'autant plus que toute communication entre elles est contrôlée par la couche transversale. Le module transversal sert également à respecter la structure de l'ECLR à travers la gestion de l'interdépendance des modules. Cette transversalité est le garant de la séparation fonctionnelle des différentes couches. Aussi la gestion et la prise de décision se positionnent dans un niveau hiérarchique indépendant offrant ainsi une meilleure lisibilité et un bon niveau de contrôle des fonctions assurées par l'ECLR.

4.3.1.2. Transparence des modules

Lors de l'exécution d'une commande pendant l'utilisation du SAM, le concepteur du produit final n'a pas à savoir qu'à un instant donné, le contrôle est passé de l'interpréteur (le SAM) au système d'exploitation. Ainsi, tout signe d'erreur ou de problème provenant du SAM et envoyé au système doit être contrôlé et résolu en arrière plan (background).
Deux solutions peuvent être appliquées :

1. Etudier les différentes étapes parcourues par la commande dans le système en repérant le cheminement et en étudiant les conséquences éventuelles. Cette solution peut être très complexe.

[34] Terme repris du monde des télécommunications et des réseaux informatiques.

2. L'appel système envoyé par une action dans le SAM doit être traité auparavant par un autre programme qui analysera la cohérence des commandes engendrées par l'action. Dès qu'un danger par rapport au système se pose, le programme envoie un signal au SAM qui proposera éventuellement par exemple à l'utilisateur de revenir en arrière dans ses démarches. Il y aura donc un lien de retour entre le SAM et le programme d'analyse d'erreurs.

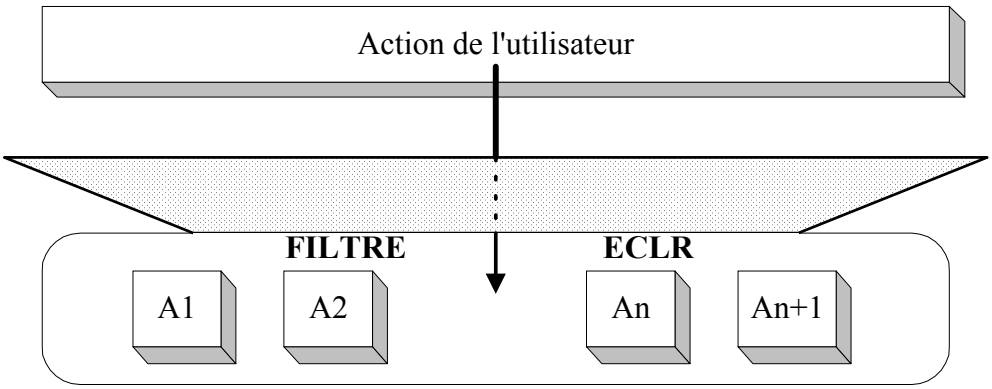


Figure 4-3 : Rôle de filtre de l'ECLR

Dans l'ECLR nous allons opter pour la seconde solution. L'ECLR doit alors jouer le rôle de **filtre** : seules les commandes syntaxiquement correctes et les actions conformes au cahier des charges ainsi qu'aux possibilités du SAM sont envoyées au système. Mais toutes ses actions attribuées à l'ECLR doivent être transparentes pour l'utilisateur, d'où la transparence des modules.

Cette notion de transparence rejoint également celle de l'indépendance dans le sens où les activités d'un module ne doivent avoir un impact direct sur les autres. Eventuellement, un module fait appel à un autre pour acquérir des données, en revanche s'il y a un dysfonctionnement interne, les autres couches n'en sauront rien, même si des résultats sont attendus de la part de celui-ci. En effet, grâce au SAP, la couche transversale prend directement en charge la situation, suite à un time-out ou un signal du module défaillant.

4.3.1.3.Modules génériques et standardisés

L'arrivée et le succès de l'Internet démontrent l'importance d'utiliser des fonctions standards c'est-à-dire courantes et normalisées et des fonctions génériques c'est-à-dire communes ; pour toucher un maximum d'utilisateurs. Ces derniers peuvent s'y retrouver plus rapidement et s'habituer aux résultats d'autant plus que les fonctions sont conviviales et faciles à utiliser. A titre d'exemple, nous pouvons citer l'intégration d'un lien hypertexte comme fonction

standard et le bouton «retour en arrière» comme fonction générique. En effet, certains principes sont incontournables pour laisser l'utilisateur «libre» de ses actes, tel que stopper l'action en cours pour effectuer un autre choix. Procurer à l'utilisateur des fonctions génériques et standards devient alors une nécessité voire une obligation pour lui assurer confort, confiance et réussite.
L'un des rôles majeurs de l'ECLR consiste également à assurer à l'utilisateur ces qualités. Pour le concepteur de l'ECLR comme pour l'utilisateur d'un SAM avec un ECLR, avoir un maximum de modules génériques et/ou standards minimise la variété d'erreurs possibles et simplifie le développement en terme de programmation résolvant ainsi une partie des blocages et de mauvaise performance pour atteindre un résultat satisfaisant.

Le savoir-faire de l'ECLR signifie alors un mode d'optimisation de *l'attribution de sens à des données* (en acquisition ou en génération), «court-circuitant» le passage par les connaissances nécessaires qui auraient abouti au même résultat [Prince, 96]. L'ECLR doit garantir la cohérence entre les données utilisateurs et le SAM. Pour ce faire, nous allons intégrer la notion d'agent qui détient le savoir-faire pour manipuler toutes les données. Un agent représente dans ce sens un élément actif du module ou du SAP (figure 4.2), sachant que chaque module ou couche et chaque SAP peuvent être composés en un ou plusieurs agents.

4.3.2. Intégration de la notion d'agent

Voici la définition minimale que donne Ferber à un agent : « *on appelle agent une entité réelle ou abstraite qui est capable d'agir sur elle-même et son environnement, qui dispose d'une représentation partielle de cet environnement, qui dans un univers multiagent, 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* » [Ferber, 89] .

Nous allons fortement nous inspirer et nous baser sur cette définition pour déterminer ce qu'est un agent. Aussi, dans le cadre de notre étude, un agent est-il défini comme une entité virtuelle[35] [Ferber, 95] ayant la capacité de :

- Agir dans un environnement,
- Communiquer avec d'autres agents,

[35] Par distinction avec une entité physique qui existe et agit dans le monde réel.

- Mettre en exerce ses ressources propres,
- Offrir des services grâce à ses compétences ou connaissances,
- Se reproduire éventuellement,
- Avoir un comportement pour satisfaire ses objectifs.

Il existe principalement deux notions d'agents [Woolridge, Jennings, 95] :

- La vision simple «weak notion of agency» où les agents respectent des propriétés standards telles que l'autonomie, l'habileté sociale et la réactivité.
- La vision profonde «stronger notion of agency» des agents, adoptée par les chercheurs en IA qui cherchent à rapprocher le comportement des agents de celui des humains en utilisant des concepts souvent appliqués aux humains tels que la connaissance, l'intuition, la croyance, l'émotion, etc.

Dans l'ECLR, nous allons adopter une vision anthropomorphique des agents car nous exploitons ces deux notions d'agents. Il s'agit alors d'un système composé de plusieurs agents «intelligents» capables de percevoir, de décider, d'agir, d'apprendre et de s'adapter [Gouardères & al, 96]. Par le terme intelligent, nous entendons également le fait que l'agent soit programmable, versatile, capable de comportement brillant ou extraordinaire dans l'accomplissement de la fonction initiale pour laquelle il a été conçu [Bui, 98].

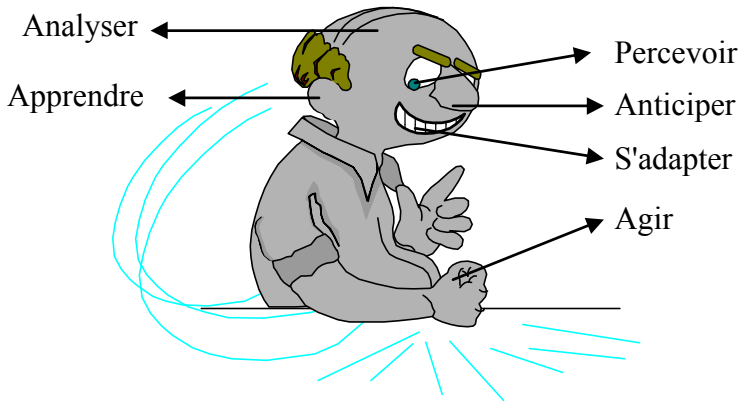


Figure 4-4 : Un agent

Plusieurs recherches, études et références existent aujourd'hui dans le domaine de prise de décision et de système multi-agents. Un système multi-agents efficace met en jeu trois critères fondamentaux [Ferber, 95] qui sont l'autonomie, l'intelligence et la mobilité de ses agents.

Cependant, l'intégration de différents agents dans un ECLR présente des caractéristiques spécifiques supplémentaires concernant la coopération et la gestion sur le plan architectural, qu'il convient d'étudier plus en profondeur.

Les agents effectuent des tâches qui peuvent être attribuées soit individuellement, soit en collaboration avec d'autres agents du même module de l'ECLR, soit en collaboration avec des agents appartenant à un autre module. Cette corrélation induit la présence dans chaque couche, de plusieurs agents ayant des caractéristiques communes. Ces caractéristiques sont inspirées de celles adaptées à un didacticiel [Woolridge, Jennings, 95] à cause de l'environnement dans lequel évolue les agents de l'ECLR : le SAM. Voici les desiderata les plus importantes :

- La *prédictibilité* : le comportement d'un agent doit être facile à anticiper pour la couche transversale (figure 4.1) afin que cette dernière puisse contrôler toute action de chaque agent d'un module et chaque agent doit aussi être en mesure d'anticiper les besoins de l'auteur de niveau 1.
- La *sociabilité* permet à un agent de déléguer des tâches ou sous-tâches, lorsqu'il ne possède pas toutes les ressources nécessaires pour réaliser une tâche en interne. Donner l'initiative et la possibilité de déléguer à un agent impliquent une connaissance maîtrisée des rôles de chaque agent car le système de gestion de la délégation peut devenir complexe. Cette sociabilité permet alors à l'agent de se situer dans son groupe.
- L'*adaptation* est une des propriétés les plus difficiles à mettre en place. Dans un ECLR, les comportements des agents doivent s'adapter à un environnement de SAM. Par conséquent, selon le niveau informatique et pédagogique de l'auteur de niveau 1, un agent de chaque module ou dans une couche constituant l'ECLR doit s'assurer que l'ensemble du module auquel il appartient est en phase avec son environnement à tout moment.
- L'*intelligence* dans le sens où l'agent est capable de résoudre certains problèmes bien définis tout seul. Cette caractéristique est particulièrement supervisée par le module transversal afin de garder le contrôle et de respecter l'ossature de l'ECLR.
- L'*anticipation* sur les événements futurs selon le contexte grâce à une mémorisation et une analyse des situations. Cette caractéristique est complémentaire à celle de l'adaptation. Il s'agit alors pour l'agent de planifier son comportement à partir des descriptions de l'environnement et de ses précédentes actions.

- Le *devoir* d'un agent par rapport à son supérieur hiérarchique est important pour la cohérence avec les autres agents mais aussi pour être en harmonie avec le contexte souhaité par l'auteur de niveau 1. Un agent doit informer l'utilisateur de ses actions et être capable de les expliquer sur demande [Billings, 91].
- La *robustesse* est exigée pour chaque agent qui doit par conséquent être prêt et réactif envers les erreurs éventuelles. Pour ce faire, tous les agents doivent être précis dans chacune de leurs actions et dans leurs rapports avec le ou les supérieurs hiérarchiques.

4.3.3. ECLR, un système multi-agents

Lors de la conception d'un ECLR, l'organisation émergente de la mise en œuvre des caractéristiques d'un agent décrit dans le sous-chapitre ci-dessus, rappelle la conception d'une fourmilière [Corbara, 91] : les fourmis (agents) se situent sur le même plan (ici un composant du module). Les actions des fourmis (agents) se coordonnent de manière à ce que la colonie (l'ECLR) survive et se protège des problèmes complexes tels que ceux posés par la recherche de nourriture (données), les soins donnés aux œufs et aux larves (pédagogie, ergonomie), la reproduction, etc. Mais, il n'existe pas de fourmis de la même catégorie ayant des pouvoirs sur les autres.
La différence avec cette vision est que, appliqués sur des agents, il faut inclure des agents décisionnaires [Sabah, 90][Prince, Nicaud, 91] pour résoudre d'éventuels conflits, résultant parfois de leurs aptitudes à «raisonner» seuls. Certains agents, que nous allons appeler agents décisionnaires, possèdent donc une autorité stricte sur les autres nommés agents simples.

Dans le SMA ECLR, chaque agent est défini par ses compétences et son expertise de coopération, c'est-à-dire par ses tâches relatives au domaine et ses aptitudes à communiquer avec les autres agents. Cela implique des contrôles sur les accointances et les connaissances des agents afin de trouver le bon consensus pour permettre le fonctionnement global du SMA.

4.3.3.1.Les agents simples

En général, les agents des SMA n'ont pas une nomenclature ou une terminologie bien définie et standardisée. En revanche, nous avons remarqué que leur composition est souvent similaire au niveau de la structure[36]. Les besoins relatifs à la conception se basant sur un SAM et un

[36] Nous développerons plus en détail le fonctionnement et la constitution de ces agents dans les chapitres 5 et 6.

ECLR, prenant en considération les objectifs d'un SAM[37] et notre souhait d'optimiser la robustesse de l'environnement de conception, nous incitent à constituer des agents simples comme suit :

1. *Agent cognitif (ACO)* qui assure l'adaptation à l'environnement du groupe d'agents auquel il appartient à l'environnement. Nous établissons à travers cet agent des relations avec la notion de conscience car l'agent est conscient de son rôle et de sa position dans le système multi-agents. Cet agent analyse les réactions successives de l'auteur de niveau 1 du SAM car il doit juger la cohérence entre les connaissances du groupe d'agents avec les données rentrées par l'utilisateur. L'ACO est alors capable de percevoir et d'agir sur l'environnement et a la capacité de cognition lui permettant de raisonner sur les autres agents et sur l'avancement de la résolution.
2. *Agent collecteur (AC)* qui collecte les informations rentrées par l'utilisateur et les événements du système. Les agents de ce type alimentent la base de données des connaissances. Chaque agent collecteur doit alors se synchroniser avec les autres agents collecteurs des autres groupes, dans le souci d'avoir une cohérence de l'ensemble et une optimisation de la capacité d'adaptation à l'apprenant.
3. *Agent contrôleur pédagogique (ACP)* dont les connaissances ne sont pas prises en temps réel. En effet, le concepteur doit fixer dès le début les règles permettant d'assurer les démarches pédagogiques du logiciel final. A titre d'exemple, dans un didacticiel proposant des exercices, l'auteur du didacticiel peut mettre la règle : l'utilisateur ne sera autorisé à regarder les réponses qu'à un moment bien déterminé. Ces démarches font partie de la qualité pédagogique de l'ensemble.
4. *Agent contrôleur ergonomique (ACE)* qui tient un rôle important dans la présentation des informations qui doit être faite d'une manière simple, claire pour éviter tout blocage et minimiser les réactions négatives de l'auteur de niveau 1 face au SAM.
5. *Agent analyseur externe (AAE)* sert à étudier la validité ou la conformité des données rentrées par rapport à la liste des exceptions[38] externes constituée par l'auteur de niveau 1 au début de la conception.
6. *Agent analyseur interne (AAI)* a pour rôle de comparer les données par rapport à la liste des exceptions internes.

[37] Voir chapitre 2.2.1.

[38] Voir chapitre 5.3.1.

La dénomination des agents reflète volontairement leurs rôles sachant que ces derniers décrivent la position des agents au sein d'une organisation ainsi que l'ensemble des activités qu'ils sont censés exercer afin que l'organisation puisse accomplir sa tâche ou atteindre ses objectifs [Ferber 95]. Les caractéristiques principales communes à tous les agents simples sont :

- Leur évolutivité à cause de leur capacité de mémoriser toutes les actions effectuées grâce à une base de connaissances. On parle alors de la capacité d'apprentissage d'un agent dont nous développerons dans les chapitres 5 et 6 les principes et la mise en œuvre.
- La communication entre agents simples ne se fait qu'à travers un agent décideur.
- Ils puisent les données qu'ils manipulent de la même base afin de maintenir une base unique et être en cohérence entre eux.

4.3.3.2.Les agents décisionnaires

Le principe selon lequel un groupe d'agents peut opérer individuellement ou en collaboration avec les autres groupes, permet d'avoir deux champs de décision : en interne au sein du groupe et en transversal à travers tous les groupes d'agents. Cette organisation donne une vision plus large que celle de l'agent superviseur de G. Sabah [Sabah, 90] qui est unique et contrôle tout le système CARAMEL.

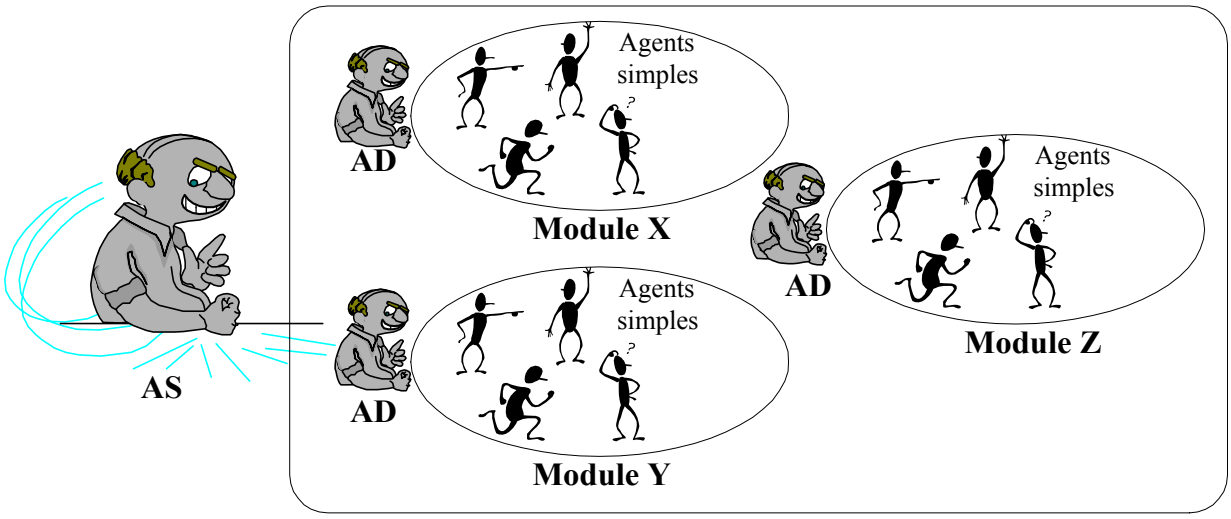


Figure 4-5 : Hiérarchie des agents

Les agents décisionnaires font partie de la catégorie d'agents dits «intelligents», car ils sont capables, en sus des capacités des agents simples, de décider parmi plusieurs alternatives, de répartir les tâches et de coordonner les actions. Ils communiquent avec les agents simples et

entre eux-mêmes. Ainsi, comme le montre la figure 4-5, les agents simples d'un module rapportent à l'agent décideur du module ou AD sachant qu'un AD est unique dans chaque couche ou module. Les AD de tous les modules rapportent ensuite à l'agent superviseur ou AD qui est unique dans tout le SMA. La communication entre les agents décisionnaires se fait selon le diagramme ci-dessous :

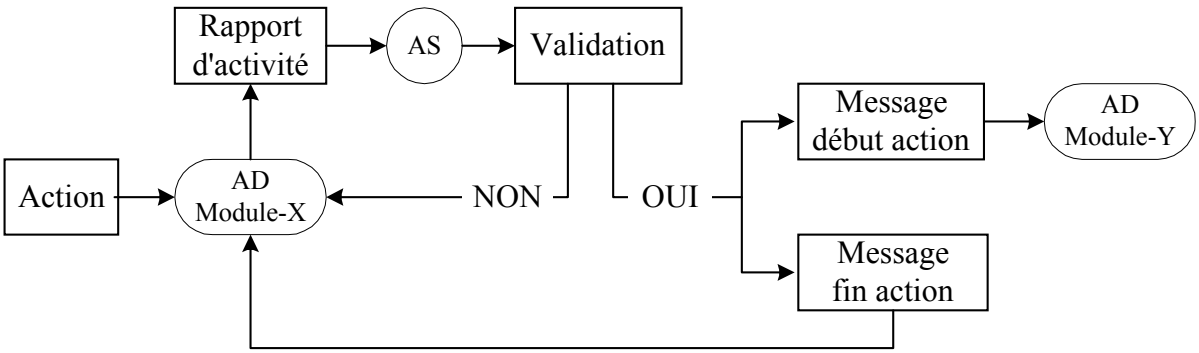


Figure 4-6 : Diagramme de communication entre agents décisionnaires

4.3.3.2.1.Agent décideur (AD)

L'activité de cet agent, est de prendre toute décision dans le groupe et d'attribuer les tâches aux agents simples au sein d'un module où il est unique. Une prise de décision est un choix sur plusieurs critères car elle ne doit causer aucune perturbation au niveau du groupe.

Il joue un rôle primordial dans la résolution de conflits. En exemple, certaines exigences de l'agent pédagogique peuvent ne pas correspondre ou être satisfaites par les agents ergonomiques. Le rôle de l'agent décideur est donc de trouver un compromis ou un arbitrage face à ce conflit.

Cet agent permet également d'assurer la pertinence et la fiabilité des informations fournies par les agents qu'il gère. De même, il surveille tous les mouvements (transmission d'informations, requêtes, délégation de tâche, etc.) effectués.

Enfin, l'implémentation d'un agent décideur unique par couche garantit la souplesse du modèle et la simplicité de la conception de l'ECLR.

4.3.3.2.2.Agent superviseur (AS)

A la différence des agents décideurs, l'agent superviseur est unique dans le modèle et possède une vision verticale de l'ensemble des modules. Son rôle est stratégique et il peut être

considéré comme le moteur du système ou comme l'épine dorsale de l'ECLR. Il se situe alors uniquement dans la couche transversale (figure 4.2).

Ses fonctions consistent à vérifier que les agents sont présents et disponibles. Cela permet de s'assurer que l'environnement technique est conforme aux objectifs de l'ECLR. Le superviseur doit également s'occuper de l'ordonnancement des tâches effectuées par les agents. En plus de la gestion des couches et la résolution de conflits entre ces couches, il garantit l'ossature de l'ensemble du système relativement à l'objectif de l'auteur de niveau 1. L'agent AS est chargé de reconnaître les actions qu'il faut analyser avec l'aide des données utilisateurs qu'il maintient également.

4.3.4. Stratégie interactive

Dans un environnement interactif, un SAM peut par exemple, permettre à l'auteur de niveau 1 de rattacher les éléments composant le logiciel qu'il veut créer, à travers différents types de liens :

- Lien hypermédia permettant de passer directement d'un fichier à un autre, de se déplacer en avant et en arrière parmi des pages ou fichiers regroupés dans un même structure ou trame, de revenir en arrière grâce à l'historisation de la consultation.
- Lien de proximité permettant de consulter un nœud parcouru précédemment.
- Lien par des descripteurs permettant d'identifier des mots clés.
- Lien calculé permettant de faire appel à un événement particulier selon certaines conditions.
- Lien de recherche permettant de lancer une recherche dans les données du logiciel.

Cet exemple montre que dans un SAM, l'ECLR peut être confronté à gérer plusieurs type d'interactivité et ce pour un même objectif. Pour que l'ECLR puisse garantir le bon fonctionnement de toute interactivité entre le système et l'auteur de niveau 1, il est nécessaire de prendre en considération certaines règles stratégiques afin de satisfaire une bonne interactivité. Les règles définies par [Schneiderman, 87] répondent à nos attentes :

Règle 1 - Lutter pour la cohérence

Règle 2 - Permettre des raccourcis

Règle 3 - Offrir un retour immédiat et affirmatif

Règle 4 - Structurer le dialogue

Règle 5 - Gérer simplement les erreurs
Règle 6 - Permettre la réversibilité des actions
Règle 7 - Donner l'initiative à l'utilisateur
Règle 8 - Améliorer la mémorisation.

L'ECLR doit alors aider de manière transparente, le concepteur de nouveau logiciel à partir du SAM à instaurer une interactivité fiable et cohérente entre le logiciel final et l'utilisateur de ce produit. En effet, l'interactivité est un élément primordial dans la robustesse d'un logiciel informatique.

4.3.4.1.Interactivité et conception

Afin de respecter les règles d'interaction au sein de l'ECLR, nous devons introduire des mécanismes supports. Pour cela, nous nous inspirons du SAM Authorware qui a su montrer la puissance de l'interactivité à travers trois concepts [Douroux, 96] :

- Une base de données contient l'état précis des icônes qui sont des opérations ou macro-instructions et des chemins de liaison de ces icônes. Cela correspond aux tâches des agents décisionnaires de l'ECLR que nous avons décrits dans le chapitre 4.3.3.2.
- La gestion des événements se fait suivant une logique de branchement des icônes. Dans l'ECLR, nous allons l'implémenter dans l'AS de la couche transversale (figure 4.1).
- Dans Authorware, les liens utilisés pour l'arborescence classique sont remplacés par des liens hypermédias afin d'obtenir une structure plus souple et évolutive. Dans le cadre de l'ECLR, ces liens peuvent être assimilés aux SAP (figure 4.2) qui établissent et contrôlent les échanges entre les modules.

L'interactivité peut se justifier également par le fait de laisser à l'utilisateur la liberté de créer des modules qu'il peut ou non réutiliser. Par exemple, Authorware offre deux options fondamentales dans la constitution de la structure de base d'un programme : «Avec réemploi» et «Sans réemploi» d'un chemin déjà parcouru. Interactivité signifie dans ce cas : suivre le déroulement de son système et le parcours de l'utilisateur final. Les questions suivantes doivent être alors prises en compte :

- Est-ce que l'utilisateur a déjà vu tel ou tel module ?
- Par quelle action a-t-il quitté telle ou telle interaction ?

- Le moteur d'exécution qui mémorise les parcours de l'utilisateur est-il facilement interrogeable pour prévoir certaines erreurs ?

Authorware a fourni deux solutions à ces questions liées au problème d'interactivité :

1. Une logique de type *répertoire* où l'utilisateur peut à tout moment abandonner la consultation ou l'utilisation d'un module. Il quitte alors définitivement l'endroit où il se trouvait.
2. Une logique de type *tableau de bord* où l'utilisateur peut appeler à tout moment en avant-plan une ressource particulière. Il quitte provisoirement alors l'endroit où il se trouvait en mettant le module à l'arrière plan.

Par rapport au développement d'un environnement de conception de logiciels robustes, il faut mettre en œuvre quasiment tous les agents simples et décideurs afin d'apporter des réponses à ces questions qui sont résolues en grande partie par la modularité complète de l'ECLR et la mise en place d'une base de données historique. La couche transversale qui supervise l'ensemble du système multi-agents, garde dans cette base de données une trace de toutes les actions, les évolutions et de toutes les modifications importantes effectuées par chaque agent. Cette pratique est tirée de la méthode RAD (Rapide Application Developpement).

4.3.4.2.Echanges entre l'ECLR et l'auteur de niveau 1

Afin de garantir la robustesse de l'ensemble des systèmes (système d'exploitation, SAM, logiciel final et ECLR), toutes les actions de l'utilisateur qui sont invalides sont signalées à l'utilisateur par l'ECLR par un message d'erreur plus ou moins concis afin que l'utilisateur soucieux de ne pas avoir un produit avec des bogues, en soit averti. Ce principe est exploité par tout type de logiciel dont les SAM, par exemple les SAM générateurs de codes HTML comme HotMetal Pro sous PC/Mac/Unix. L'ECLR fournit par conséquent ici une rétroaction[39] informative. Plusieurs types de messages d'erreurs peuvent être émis par l'ECLR dans ce cas :

- Un message catégorique disant que l'opération ou l'action qui vient d'être réalisée est incorrecte.
- Un message expliquant le problème.
- Un message plus concis expliquant la cause du problème.

[39] La rétroactivité a sa place ici dans le sens où le système agit ou réagit sur le passé.

- Un message détaillé expliquant la cause, le problème et la solution.

Nous détaillerons dans un chapitre ultérieur les moyens mis en œuvre pour gérer et décider du type de messages. Afin de mieux gérer la parution des messages, nous avons inclus les questions suivantes dans notre questionnaire[40] :

En cas de problème, vous préférez	Profanes	Débutants	Confirmés
être prévenu par un message au fur et à mesure	80%	70%	30%
être prévenu par un message avant le blocage complet	20%	30%	70%

L'ECLR capture, interprète et analyse toute action de l'utilisateur afin de détecter toute erreur éventuelle. Or, nous avons remarqué que plus l'utilisateur a des compétences informatiques, moins il veut être interrompu par des messages d'erreurs. Pour essayer d'équilibrer la parution des messages envoyés par l'ECLR, nous n'allons signaler que les erreurs majeures qui peuvent être dangereuses pour la suite des opérations du système comme celles de l'utilisateur, exemple l'appel d'un fichier qui n'existe pas dans le répertoire désigné.

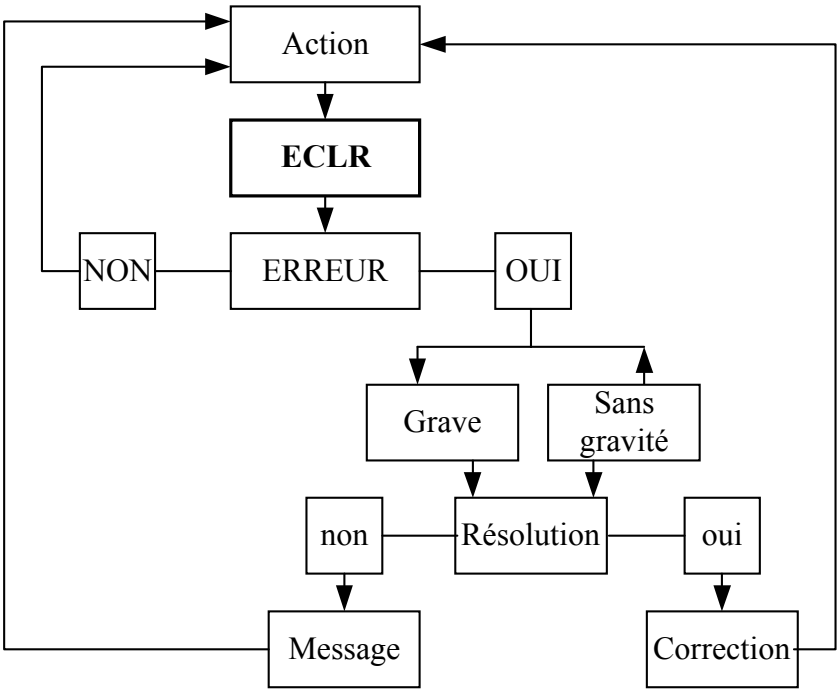


Figure 4-7 : Traitement d'une action faite par l'auteur de niveau 1

[40] Suite du questionnaire décrit dans le chapitre 3.3.

En revanche, comme nous le montre la figure 4-6 ci-dessus, certains problèmes peuvent être corrigés automatiquement par les agents de l'ECLR. Un exemple de correction automatique : si l'utilisateur crée une fenêtre ayant des coordonnées dépassant l'écran, l'ECLR et ses agents de l'ECLR peuvent modifier ces valeurs après approbation de l'auteur de niveau 1 car il est possible que le résultat c'est-à-dire le logiciel final soit utilisé sur un écran plus grand.

4.3.4.2.1.Rôle pédagogique de l'ECLR

L'expérimentation avec le système APLUSIX [Nguyen & al, 95] concernant l'effet des messages d'erreur sur l'apprentissage de l'appariement des règles de factorisation avec un environnement interactif, a permis de démontrer qu'il est important d'adapter la séquence de problèmes à résoudre à chaque apprenti. En effet, certains individus reconnaissent des actions ou des expressions avec un degré de visibilité très bas, alors que d'autres individus ne peuvent acquérir des savoir-faires qu'au moyen des expressions ou d'actions de visibilité évidentes.

D'autres expérimentations ont également montré que l'aide, dans notre cas le message de l'ECLR ne doit pas être trop détaillé [Nguyen & al, 95] : le système doit laisser à l'élève une part de responsabilité dans une action qu'il suggère. En revanche, les messages d'erreurs doivent être aussi précis que possible, de façon à ce que l'élève puisse connaître les raisons de l'échec d'une action, en déduire les conséquences et entreprendre les actions correctives.
De même, si les messages d'aides ou d'erreurs sont longs et composés de plusieurs phrases, l'apprenti a tendance à ne tenir compte, en majorité, au départ que d'une partie du message et se précipite pour effectuer l'action qu'il perçoit dans sa lecture rapide.

Il est par conséquent probable qu'il y ait une interaction entre la connaissance procédurale construite sur la base d'expériences et la connaissance conceptuelle [Nguyen & al, 95]. En effet, c'est en cherchant à ajuster un logiciel ou un «programme» qu'un individu se construit la majorité des règles et procédures correctes pour développer.

4.3.4.2.2.Ergonomie de l'ECLR

L'ergonomie des logiciels a globalement évolué pour différentes raisons :

- Exigences de plus en plus fortes des utilisateurs.
- Catégories professionnelles : les utilisateurs ne sont pas forcément des professionnels en développement informatique et peuvent être des profanes.

- Evolution de la technologie : informatique interactive, outils de prototypage, multimédia, etc.
- Réduction des coûts associés aux logiciels et aux matériels requis pour la création d'un nouveau logiciel à partir d'un SAM.

Un informaticien a souvent ses inspirations pédagogiques et ergonomiques pour créer un nouveau logiciel, à partir de la capacité du logiciel et du matériel. Il y a même une tendance à modifier les spécifications fonctionnelles définies dans le cahier des charges à cause de la faisabilité ou non de ce qui est défini.
De ce fait, l'ECLR ne doit pas gêner la créativité de l'auteur de niveau 1 à travers les échanges avec ce dernier. Il est alors primordial de limiter ces échanges et leurs présentations doivent être conformes à celles du SAM afin que l'utilisateur ne soit pas perturbé par des messages trop distincts et étrangers au SAM. Différents choix relatifs à l'apparence des échanges avec l'ECLR seront alors proposés à l'utilisateur à l'installation de l'ECLR.

4.4. DIFFERENTES APPROCHES DE L'ECLR

Un utilisateur profane qui veut développer avec un SAM doit apprendre à bien réagir face aux différents messages d'erreurs. En effet, il existe des messages d'erreurs, même dans certains systèmes réputés pour être convivial comme le Macintosh ; qui ne sont pas explicites pour un utilisateur novice ! Le dessin d'une bombe avec un texte «erreur système N° 2100» peut ne rien signifier pour un non informaticien par exemple. Cela démontre l'importance du «comportement» de l'ECLR face à l'utilisateur. C'est ce que nous allons appeler dans cette partie approche de l'ECLR.

Il est difficile de concrétiser et d'implémenter une approche mais cette étude est utile car la théorie est une profonde réflexion sur les structures, et les abstractions sous-tendent la réalité des systèmes [Ermine, 93].
Nous allons développer trois points de vue pour caractériser le comportement d'un ECLR : l'approche coopérative, l'approche systémique et l'approche cognitive.

4.4.1. L'ECLR comme un système coopératif

L'approche coopérative consiste à faire travailler simultanément différents acteurs, dans le cadre de notre étude, différents agents selon une démarche itérative et incrémentale. En effet, le fait d'avoir un certain nombre de modules ou de paramètres ne favorisent pas forcément le processus de prise de décision d'où la nécessité de faire intervenir des agents décideurs, humains ou non. De plus, la coopération entre les agents humains et logiciels pour la prise de décision offre une solution de décomposition de la complexité d'une tâche [Boy, 97]. Il est important de noter qu'un contexte s'établit automatiquement entre les agents humains et logiciels à partir du moment où il y a interaction. Cette communication contextuelle est nécessaire pour mettre les deux acteurs « sur la même longueur d'onde » et fait référence au niveau d'approximation / abstraction / agrégation avec lequel une tâche ou un objet ou un problème est perçue [Greer, 92].

Cette approche peut se concrétiser par la prévision d'une partie explicative de l'ECLR, ce qui revient à intégrer dans l'ECLR un EIAO. Cela signifie dans le cadre de notre étude à intégrer dans l'ECLR :

- Un élément d'apprentissage du rôle de l'ECLR dans le développement d'un logiciel à partir d'un SAM. Cela permet à l'utilisateur d'avoir une idée précise sur l'importance, sur les limites et sur la capacité d'un ECLR. L'expérience faite par l'équipe de psychologie cognitive de l'université Paris 8 et le laboratoire IRIN de l'université de Nantes peut être une source enrichissante pédagogiquement : la séquence des problèmes donnée à l'élève (dans le cadre de notre étude, utilisateur concepteur de logiciels à partir du SAM) est déterminée par un sélecteur de problèmes qui utilise une base de problèmes ; le principe consiste à donner un problème plus simple en cas de difficulté.

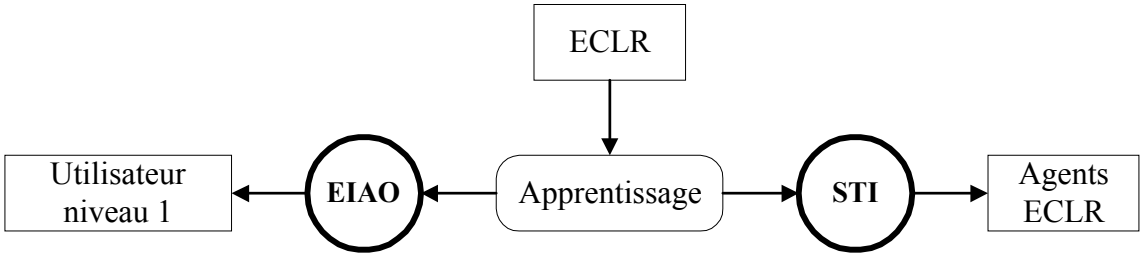


Figure 4-8 : Eléments d'apprentissage d'un ECLR

- Un élément d'apprentissage pour les agents de l'ECLR : nous allons prendre exemple sur les systèmes tuteurs intelligents ou STI qui aident les apprenants à acquérir des connaissances dans le sens où ils sont capables de s'adapter en permanence à leur niveau [Marcenac, 92]. Plusieurs solutions peuvent être envisagées :

- Créer un module répondant aux objectifs du SAM, c'est-à-dire que si le SAM a été conçu pour permettre la création de logiciels éducatifs par exemple, le contenu de ce module est chargé d'enseigner la faisabilité donc le développement de tels ou tels logiciels à partir du SAM.
- Utiliser un système expert à intégrer dans le SAM qui permet à l'utilisateur d'acquérir certaines stratégies tutorielles.

L'approche coopérative est alors particulièrement bénéfique pour l'auteur de niveau 1 et pour l'ECLR puisque les agents acquièrent au fur et à mesure du développement le «comportement» de l'utilisateur.

4.4.2. Point de vue systémique sur l'ECLR

Sachant que l'ECLR se rapporte à un système et peut affecter ce système dans son ensemble, il est important d'avoir une visibilité sur l'approche systémique de l'ECLR. Il existe deux manières pour la mettre en œuvre [Patel, 95] :

- Effectuer un contrôle adaptatif basé sur un processus d'estimation (loi d'adaptation) dont l'objet est le contrôle de systèmes stochastiques, avec une dynamique estimée. Il s'agit de prévoir ou d'anticiper la ou les perturbations agissant sur le système.
- Considérer que l'évolution du système est incertaine (évolution contingente). Des études ont été effectuées par Leitmann dans la conception des feed-back de régulation de systèmes sujets à des incertitudes bornées et des perturbations exogènes. Ces études se sont appuyées, dans les années 80, sur la théorie des systèmes à structure variable et sur la théorie de la stabilité de Liapounov[41]. Il s'agit ici d'accepter toutes perturbations du système et de ne pas en tenir compte.

[41] La méthode de Liapounov consiste à concevoir des contrôles qui garantissent que l'état du système reste borné en présence des perturbations.

Dans le cadre de l'ECLR, la première approche est la plus adaptée. En effet, ne pas tenir compte de l'environnement dans lequel évolue l'ECLR est parfois dangereux. Nous pouvons par exemple penser à la gestion des champs d'affichage ou de saisie des données selon la langue utilisée pour une date. Au cas où des calculs sur ce champ date seraient prévus dans le SAM, l'ECLR doit aider le système à vérifier que le mois ne dépasse pas «12».
L'anticipation selon la première approche consiste alors à vérifier que le contenu de chaque variable utilisée par le système est conforme et logique. En cas de doute, un message est généré à l'utilisateur.

Comme nous l'avons expliqué dans le chapitre 4.3.4.2, il y a une interaction entre la connaissance procédurale construite sur la base d'expériences et la connaissance conceptuelle [Nguyen & al, 95]. Nous pouvons en conclure qu'un individu se construit la majorité des règles et procédures à appliquer sur son logiciel en cherchant à l'ajuster suite aux différents messages générés ou engendrés par le système, l 'ECLR et le SAM. Cet enrichissement des règles et procédures est également valable pour les agents composant l'ECLR grâce à une mise à jour de la base de connaissances à chaque action prise par l'auteur de niveau 1 après un message. Il y a par conséquent un apprentissage et une maturation des agents progressivement afin d'obtenir un environnement robuste.

4.4.3. L'ECLR comme un système cognitif

Notre recherche relève un peu de la psychologie cognitive car, avant de voir s'exécuter un programme aussi élémentaire soit-il, l'utilisateur tape souvent un ensemble de commandes dont il ne saisit pas la raison au premier abord [de la Passardière, 93]. L'approche cognitive signifie pour nous étudier les processus par lequel l'ECLR acquiert la conscience des événements et des objets de son environnement. En effet, du point de vue cognitif, une fonction correspond à un groupe de connaissance qui peut être confié à l'utilisateur ou à la machine. L'ECLR exécute alors cette fonction à travers ses agents afin de la transformer en une activité.

Un agent est une entité humaine ou une machine, qui est capable d'effectuer une tâche [Boy, 95]. Un agent est conçu dans un contexte définissant ses limites. Cela justifie les questions communes qui sont généralement posées par des utilisateurs de SAM comme «*que puis-je*

faire ? » ou *«Est-ce autorisé ?*» ou «*Cela a-t-il un sens ?*». Ces questions sont également posées par les agents des modules composant l'ECLR, en particulier les agents décisionnaires. Nous pouvons assimiler les activités de l'ECLR à celui d'un acteur humain comme le décrit le schéma ci-dessous :

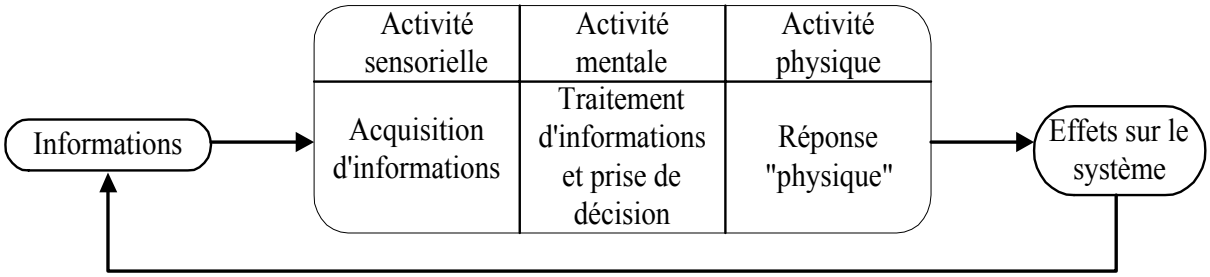


Figure 4-9 : Les activités sensorielles, mentales et physiques d'un acteur humain (inspiré de Kolski, 93)

Les activités de l'opérateur humain peuvent être regroupées en quatre classes principales [Kolski, 95] :

1. Les activités de perception qui concernent la détection et l'acquisition d'informations et l'identification de situations. Par rapport à l'ECLR, cela consiste à capturer toute action de l'utilisateur.
2. Les activités mentales pour la résolution de problèmes et la prise de décision. Cette tâche correpond à l'analyse de chaque action de l'utilisateur par l'ECLR afin de vérifier la présence ou non d'erreurs éventuelles.
3. Les communications comprenant les demandes, les réponses et les échanges d'informations avec les autres. Dans l'ECLR, les différents acteurs logiciels et humains s'échangent des informations à travers des messages.
4. Les activités physiques regroupant les actions menées par l'individu. Ce sont les résultats du travail de l'ECLR.

Cette similitude entre les activités d'un ECLR et d'un être humain confirme bien le fait que l'ECLR a pour rôle d'automatiser les activités d'un expert en informatique afin de créer des logiciels robustes et conformes aux cahiers des charges.

4.4.3.1.ECLR et l'erreur

Dans notre recherche sur l'approche cognitive, il convient de s'interroger sur les explications pédagogiques et psychologiques relatives à une situation de doute et à une erreur. Pour ce, nous allons nous appuyer sur l'observation du comportement d'un individu à faire un choix ou prendre une initiative ou essayer de résoudre un problème technique.

Les erreurs commises pendant un apprentissage indiquent le niveau de connaissance de l'apprenti et la manière dont il apprend, sachant que les circonstances et contextes d'apprentissage dépendent de plusieurs facteurs tels que l'âge et la situation socio-linguistique [Richards, 74]. Cette théorie est également valable pour les situations de prise de décision.
La première réaction d'un individu plutôt calme est de chercher à prendre connaissance de l'environnement dans lequel la situation s'est produite donc à analyser le contexte au moment de l'erreur ou du problème. Pour ce faire, plusieurs cas peuvent avoir se présenter :

- Il analyse le message d'erreur signalant un problème du système. Cela correspond à la question *«Qu'est ce que je constate ? »*
- Il revoit ses dernières actions sur la machine. Cela correspond à la question *«Qu'ai-je fait?»*
- Il attend que le système résoud le problème ou envoie un message qui lui indique les causes de ces erreurs. Cela correspond à la question *«Le système est-il capable de résoudre l'erreur ? »*
- Il cherche dans un menu d'aide ou dans le manuel d'utilisation la solution. Cela correspond à la question *«Quelle solution le SAM me propose dans ce cas précis ? »*
- Il revoit la structure appliquée à son application. Cela correspond à la question *«qu'ai-je voulu faire ? »*
- Il quitte et relance le système plus ou moins brutalement. Cela correspond au caractère d'un individu lassé par le SAM.

Cette analyse sur l 'erreur nous permet d'étudier le comportement d'un agent de l'ECLR et de bien définir les périmètres d'action de cet agent. Les questions cognitives qui correspondent à un agent de l'ECLR sont celles qui portent sur son action c'est-à-dire :

- *«Qu'ai-je fait ? »* : l'agent peut faire un rapport de son action à son supérieur hiérarchique et mettre à jour l'historique de ses actions.

- *« Qu'ai-je voulu faire ? »* : si l'action d'un agent ne convient pas, son supérieur hiérarchique peut lui retourner une requête pour revoir ou re-analyser le problème.
- *«Qu'est ce que je constate ? »* : cette question correspond à l'analyse effectuée par l'agent.

De ces questions découlent plusieurs interrogations relatives à la satisfaction des agents décideurs et de l'auteur de niveau 1.

4.4.3.2.ECLR et le SAM

Un système auteur, dans le cadre spécifique de l'EIAO, utilisant un système expert doit satisfaire les points suivants [Laurière, 84] :

- Etre le plus intelligent possible, ce qui signifie dans notre cas, être capable de comprendre et réparer les erreurs de l'utilisateur du SAM.
- Etre facilement modifiable, ce qui signifie ici, ne pas limiter l'utilisateur dans ses démarches (retour en arrière, suppression, etc.).
- Etre capable à tout moment de fournir des explications sur le bon raisonnement ou sur les erreurs commises.

Ces critères offrent plutôt une vue globale de comportement. Cela nous suggère de construire une architecture permettant aux agents de l'ECLR de répondre à ces critères. Nous développerons cette architecture dans le chapitre 5.

4.4.3.3.ECLR et auteur de niveau 1

Du point de vue cognitif, l'utilisateur d'un SAM comme tout individu est limité plus ou moins par les deux classes de mémoire dont il dispose [Feneuille, 92] :

- La mémoire à court terme appelée également mémoire de travail contient un nombre limité d'informations (visuelles, auditives, etc.). Cette mémoire fonctionne comme une file d'attente et l'accès y est plus rapide que celui de la mémoire à long terme. Elle sert à regrouper des informations utiles temporairement pour la réalisation d'une tâche.
- La mémoire à long terme appelée également mémoire permanente a une capacité beaucoup plus grande que la mémoire de travail. C'est elle qui est chargée d'organiser les informations, de vérifier la conformité des actions de l'utilisateur face au SAM.

Les aspects métacognitifs sont aussi à prendre en compte. Ceci implique pour le SAM de laisser le contrôle à l'utilisateur en le guidant de temps à autre par exemple en lui donnant la possibilité de s'informer sur ce qu'il a fait et sur ce qui lui reste à faire dans une partie du module. Cela améliore la motivation de l'utilisateur du SAM et le met en confiance surtout si le cadre de travail est agréable et facile à maîtriser. Pour illustrer cette idée, nous pouvons penser à la mise en place d'un document hypermédia, après avoir précisé les liens, il est souhaitable que le SAM préviennent l'utilisateur des éventuelles absences de fichiers relatifs à ces liens.

Il est important d'adapter les aides ou les messages auxquels un utilisateur peu expérimenté aura droit. En effet, si l'utilisateur commence à faire une erreur évidente dès la première manipulation, il est inutile de lui envoyer des messages techniques compréhensibles uniquement par les experts en développement de logiciels sur SAM !

4.5. CONTRIBUTIONS DE L'ECLR DANS UN ENVIRONNEMENT DIDACTIQUE

Pour mieux positionner l'ECLR par rapport au SAM et à l'utilisateur, nous avons décomposé un environnement didactique en trois entités selon le schéma ci-dessous :

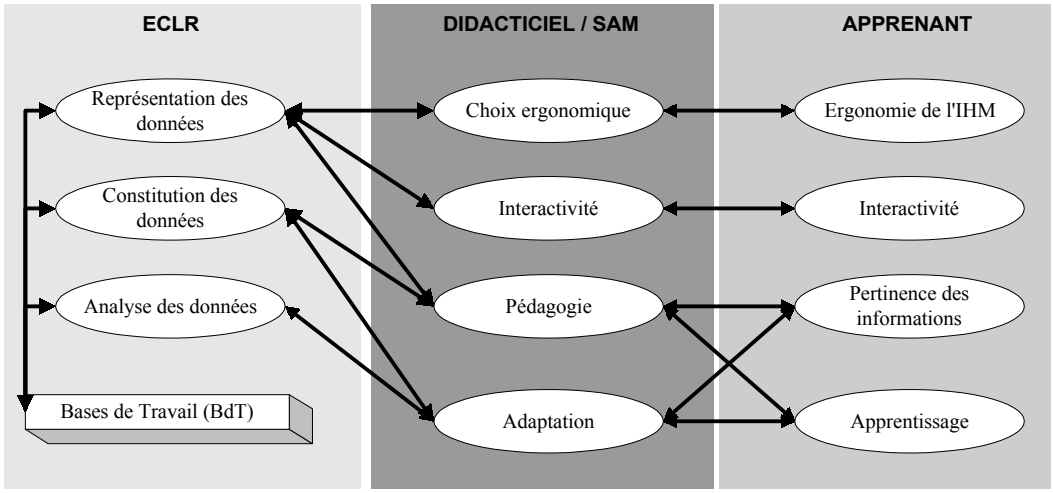


Figure 4-10 : Corrélation entre les composants d'un environnement didactique

- L'entité ***ECLR*** rassemble les connaissances ou les informations pertinentes pour assurer la robustesse de l'environnement didactique. Il s'agit des données reçues de l'utilisateur et du SAM que les différents agents font évoluer au fur et à mesure de la conception de son

logiciel final. Nous allons appeler ces données une base de travail[42] ou BdT. Deux types de bases de données peuvent y coexister :

1. *Base de données* regroupant les données relatives au domaine traité et aux fonctionnements de base (règles).
2. *Base de connaissances* qui est évolutive selon le fonctionnement et le comportement des agents. Elle résulte d'une association des modèles épistémiques[43], comportementaux et cognitifs[44] de l'apprenant. Ce type de base de travail est mis à jour en temps réel.

- L'entité ***Didacticiel/SAM*** met en avant les côtés ergonomiques et pédagogiques du logiciel, et rappelle qu'un didacticiel est un programme d'enseignement c'est-à-dire que l'interactivité avec l'utilisateur (l'apprenant) et la capacité d'adaptation d'un tel logiciel vis-à-vis de l'utilisateur sont primordiales.
- L'entité ***Apprenant*** regroupe les caractéristiques requises et utilisées par l'utilisateur final pour créer son logiciel.

Prenons par exemple un utilisateur qui souhaite intégrer une séquence d'images animées dans son logiciel final. Il utilise la fonction appropriée fournie par le SAM. Dans ce cas, l'ECLR prend en considération l'action de l'utilisateur et les possibilités du SAM. Les agents de l'ECLR analysent alors les données grâce à leurs bases de travail. Ils vérifient à partir des règles contenues dans ces mêmes bases de travail que l'ensemble est valide par rapport à ce qui a été déjà construit auparavant, toujours dans l'objectif de garantir la robustesse de l'ensemble des trois entités.

Grâce à ces analyses et éventuellement à une modification de la représentation des données, l'ECLR offre au SAM différents avantages telle l'ergonomie ou la capacité d'adaptation à l'utilisateur final. Cela entraîne une pertinence des informations ainsi qu'un environnement convivial et fiable à l'utilisateur final.

[42] Nous détaillerons les bases de travail dans le chapitre 5.

[43] Modèle de représentation du niveau de connaissance de l'apprenant.

[44] Modèle de simulation des capacités cognitives de l'apprenant.

Les interactions entre les trois entités, signalées par les flèches en gras, symbolisent alors leur forte interdépendance. Le fait d'omettre une des entités risque alors de déséquilibrer l'environnement conceptuel.

4.6. CONCLUSION

Cette partie de notre recherche nous amène à prévoir un ECLR qui permet une grande modularité afin de répondre aux exigences du SAM et de l'auteur de niveau 1 à travers ses composants : les agents. Nous avons vu que les relations ou communications entre les différents composants permettent d'évaluer l'indépendance des modules.
L'ECLR doit être doté d'une certaine capacité d'adaptation au contexte et à l'environnement. Cette aptitude à s'adapter résulte de la capacité de prendre des décisions pertinentes. Elle implique également une capacité d'anticipation par rapport aux événements et une maîtrise des échanges avec l'utilisateur. Ainsi, l'ECLR offre une sûreté de fonctionnement car il permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il délivre. Cette confiance de l'utilisateur facilite son interaction avec l'ensemble du système matériel et logiciel informatique.

Sachant que la tolérance aux fautes recouvre les diverses manières pour pallier les conséquences des fautes [LAPRIE, 1994], l'ECLR doit être lui-même tolérant aux fautes car il doit remplir sa fonction en dépit des fautes. Cela est dû à la récursivité du concept de tolérance aux fautes car les mécanismes destinés à mettre en œuvre la tolérance aux fautes doivent être protégés contre les erreurs susceptibles de les affecter eux-mêmes. Ainsi, pour garantir la robustesse de l'ensemble SAM et ECLR, il faut contrôler :

- Les données entrées pour éviter les erreurs dues à des évènements externes,
- Les résultats et les traitements pour éviter les erreurs dues à des évènements internes.

Pour répondre à ces qualités, le concepteur de l'ECLR doit construire une architecture fiable et modulable, permettant aux agents de prendre des décisions pertinentes car il doit en plus tenir compte de l'évolution de l'information, de la connaissance contextuel et de la complexité des interactions entre les entités.

5. MODELISATION D'UN ECLR

L'ECLR, programme parallèle au SAM, dont l'objectif principal est de contrôler toutes les actions d'un utilisateur de SAM pour créer un autre logiciel grâce à différents agents, ne peut réaliser cet objectif sans être «en harmonie » avec lui-même. Cela signifie que l'ECLR doit avoir le minimum de faiblesses possibles pour assurer la robustesse de l'ensemble du système. Par conséquent, des études sur différents niveaux d'organisation comme dans les SMA [Rocher, 1968] s'imposent :

- Le niveau micro-social où l'on s'intéresse à chaque élément, aux interactions entre les agents.
- Le niveau des groupes où l'on s'intéresse aux structures intermédiaires qui interviennent dans la composition d'une organisation plus complète. Il s'agit de différencier les rôles et activités des agents dans chaque module pour faire ressortir l'émergence de structures organisatrices des agents.
- Le niveau des sociétés globales où l'intérêt porte sur la dynamique d'un grand nombre d'agents et de plusieurs groupes.

La concrétisation de l'ECLR commence par une modélisation du fonctionnement de ses composants, c'est-à-dire la formalisation des agents qui composent chaque module, le modèle de fonctionnement interne et externe de chaque module et enfin le fonctionnement global de tous les modules. Chaque module peut être comparé à une couche, d'ailleurs ces deux termes ont la même signification dans notre étude. Pour ce dernier, nous allons adopter une architecture en couches ou en modules permettant d'intégrer les différents agents et donnant une importance particulière aux actions de l'utilisateur. En effet, la décomposition du modèle en plusieurs couches offre une meilleure visibilité de l'ensemble des actions faites individuellement par les agents et par les groupes d'agents. Cette architecture en couche garantit la modularité de la conception de l'ECLR et facilite l'implémentation de l'hétérogénéité des fonctions des agents dans les modules.

L'ECLR obtenu à partir de cette modélisation doit avoir un caractère anthropomorphique[45] face aux actions de l'utilisateur, ce qui signifie que l'ECLR garde en permanence la notion de

[45] Un système anthropomorphique est pour nous un système dont le comportement est à l'image de celui de l'être humain.

contexte en permettant de personnaliser et d'adapter chaque agent aux exigences du système et de l'utilisateur.

Par conséquent, l'ensemble des actions des agents qu'il s'agisse d'agents simples ou d'agents décideurs repose sur le respect de règles ou normes ou procédures et méthodes définies, afin de garantir une bonne communication et une bonne intégration de l'ensemble.

Dans ce chapitre, nous allons déterminer les rôles des agents dans l'ECLR. Les fonctionnements procéduraux et cognitifs de ces acteurs imposent une structure hiérarchique de chaque module. Nous ferons la distinction entre organisation et interaction des agents. Pour établir les actions cohérentes entre les agents et les modules afin d'obtenir un haut niveau de robustesse, nous allons contrôler les données externes et les conditions d'acceptation de ces données par rapport à des listes de référence, à des règles de vraisemblance, au caractère facultatif ou obligatoire des données et aux contraintes établies par l'architecture globale de l'ECLR.

5.1. FORMALISATION DES ACTEURS INFORMATIQUES D'UN ECLR

D'après la définition d'un agent que nous avons adopté dans le chapitre 4.3.2, nous allons considérer un agent non comme une entité isolée mais comme une pièce évolutive dans une société nommée ECLR, elle-même en évolution [Gouardères, 98]. Dans notre SMA, les agents ont plusieurs activités communes telles que :

- le contrôle des informations reçues,
- le traitement d'une requête,
- l'estimation des ressources,
- l'application des changements,
- la validation des actions.

Nous notons une distinction entre un agent et un programme. En effet, un agent est un acteur informatique en soi car il possède une attitude, et a des devoirs envers son environnement contextuel et son supérieur hiérarchique. L'organisation de la hiérarchie est représentée par la figure ci-dessous :

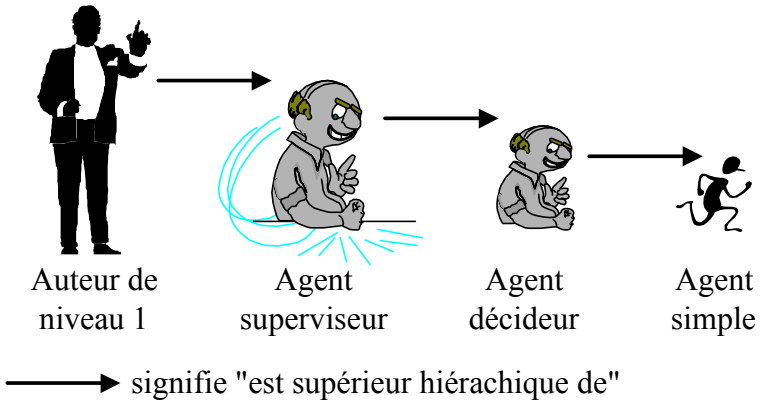


Figure 5-1 : Hiérarchie des acteurs d'un ECLR

La démarche générale de validation effectuée par un supérieur hiérarchique a pour objet d'accompagner en continu l'exploitation de l'application et d'en valider la réalisation sur le plan qualité.

5.1.1. Modèle procédural de fonctionnement d'un agent

Chaque action d'un agent simple est supervisée par l'agent décideur (AD) du module auquel il appartient. Chaque action d'un AD est supervisée par l'AS. Nous appelons « événement », le point de départ de chaque activité d'un agent quel qu'il soit. Ensuite, chaque agent respecte le même modèle de fonctionnement que nous allons appeler MPFA pour Modèle Procédural de Fonctionnement d'un Agent. Le MPFA est un modèle que nous avons spécifiquement développé et adapté pour l'ECLR.

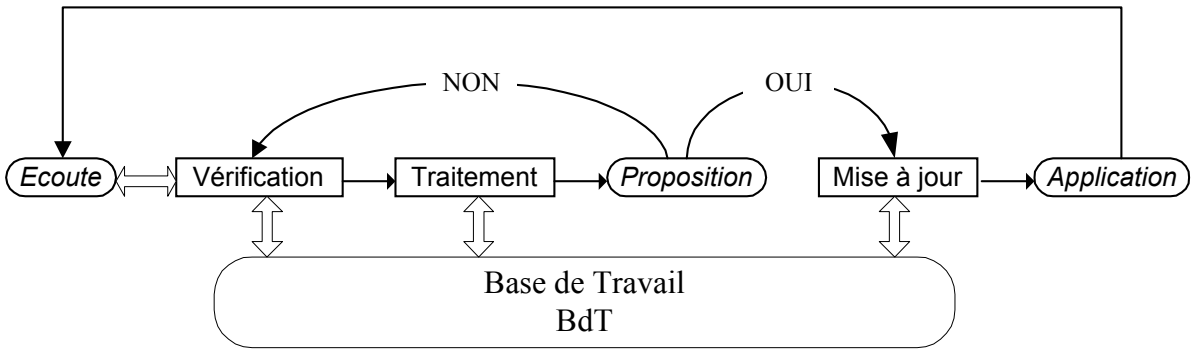


Figure 5-2 : Modèle MPFA d'un ECLR

D'après ce schéma, nous pouvons distinguer deux types de rôles :

- les rôles externes, en caractères italique, correspondent à des activités faisant appel à une entité extérieure. Il s'agit en premier lieu de **l'écoute** d'un événement qui est plutôt un rôle passif. Le second rôle actif d'un agent concerne la **proposition** d'une action à son

supérieur hiérarchique c'est-à-dire l'AD pour un agent simple, l'AS pour un AD et l'auteur de niveau 1 pour l'AS. En cas d'accord de ce supérieur hiérarchique, le dernier rôle externe d'un agent relatif à l'événement consiste en **l'application** de l'action.

- les rôles internes, en caractère normal sur la figure 5-2, montre les activités de l'agent qui ne font intervenir que l'agent et la base de travail. Il s'agit de la *vérification* qui consiste à tester si l'événement le concerne ou non et si l'agent possède les ressources nécessaires pour le *traitement*. Si c'est le cas, l'agent traite alors l'information en y ajoutant ou en ôtant des données. Après la proposition du résultat à son supérieur et la validation par ce dernier, l'agent s'occupe de la *mise à jour* de la base de travail.

A chaque fonction, un agent de l'ECLR combine les deux types de rôles (interne et externe), même si le *traitement* fait en interne ne donne aucun résultat et qu'il n'y ait pas de mise à jour de la BdT.

Ainsi, le modèle MPFA respecte la notion d'agent [Ferber,95] car chaque agent ***perçoit, décide, agit et apprend***. La principale difficulté par rapport à ce modèle concerne l'autonomie de l'agent car ce dernier doit respecter des règles de conduite afin de rester dans le contexte tel un conducteur qui respecte le code de la route.

Afin d'éviter tout débordement sur les rôles de chaque agent, nous allons réaliser chaque agent en jumelant deux formes de « pensée » :

- La pensée impérative : « Que doit faire l'agent ? ». La programmation algorithmique et/ou procédurale est la plus adaptée à cet objectif car ces méthodes de programmation offrent un suivi chronologique de l'évolution des événements.
- La pensée applicative : « Sur quoi l'agent doit-il agir ? ». Pour ce cas, la programmation par objet est nécessaire car elle permet de bien distinguer le champ d'action. L'intérêt de l'approche objet vient également de la possibilité de modéliser librement la réalité des choses, et de disposer d'une démarche canonique.

De même le modèle MPFA offre à chaque agent deux types d'apprentissage : l'apprentissage constructiviste qui est différent de l'apprentissage instructiviste où l'on apprend grâce à autrui. Le principe du « learning by doing » de chaque agent est garanti par la mise à jour systématique de la base de travail à chaque fonctionnement.

5.1.2. Modèle cognitif de fonctionnement d'un agent

La base de travail qui représente les connaissances d'un agent ainsi que la capacité d'analyse d'un agent constituent les arguments cognitifs d'un agent. Un modèle cognitif est défini dans deux champs duaux :

- l'analyse sémantique qui donne un sens au comportement de l'agent. L'agent analyse alors méthodologiquement chaque action afin de respecter son rôle et controler son effet sur l'environnement. Cette analyse est inspirée des catégories de démarches de Forner [Forner, 90] que nous avons développées dans le chapitre 3.1.1. traitant de l'erreur humaine. Elle correspond alors au comportement rationnel et contrôlé d'un homme pour prendre une décision.
- l'analyse empirique qui ne tient pas compte de la logique à suivre pour garder le contexte environnemental. L'agent ne s'occupe pas de son entourage et évolue ainsi indépendamment de ses supérieurs hiérarchiques. Cela donne toute liberté d'action et de réaction à l'agent. Si nous faisons l'analogie avec l'attitude humaine développé dans le chapitre 3.2.1.1., l'analyse empirique correspond au comportement intuitif et spontané de l'homme.

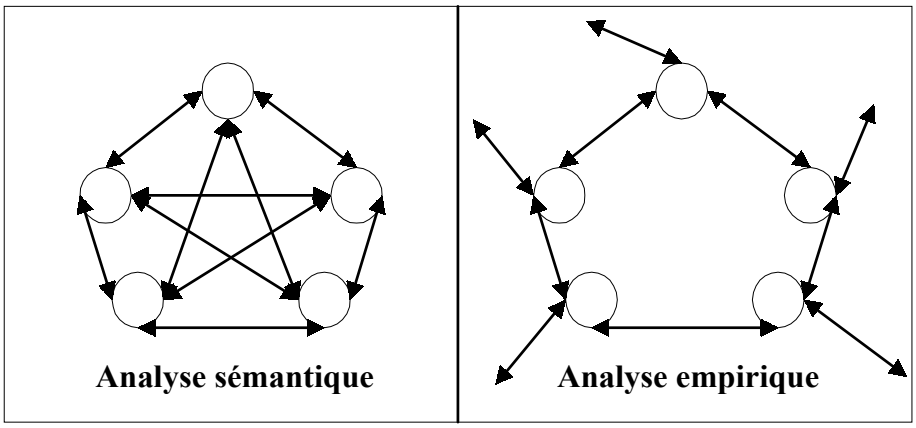


Figure 5-3 : Dualité comportementale d'un agent

Les entités sur le schéma ci-dessus (petits cercles) représentent les éléments de la base de travail d'un agent. Rappelons[46] que la base de travail est constituée de bases de données pour les informations et de bases de connaissances pour le fonctionnement de l'agent. Pour l'analyse sémantique, une entité effectue des échanges avec l'extérieur et avec les entités qui

[46] Voir chapitre 4.5

lui sont rattachées directement. Cela rend compte alors des processus de fonctionnement d'un agent [Prince, 98] à travers :

- la dynamique externe, c'est-à-dire comment l'agent échange avec son environnement et le modifie,
- la dynamique interne, c'est-à-dire comment l'agent est modifié et évolue par ce qu'il a traité et reçu de son environnement et de son propre fonctionnement.

Tous ces échanges entraînent un enrichissement de la base de travail pour l'agent et pour les autres agents du même module ou d'un autre module. En revanche, l'analyse empirique entraîne une intériorisation de l'agent car les entités de la base de travail s'échangent des informations uniquement entre elles. Cela donnera à l'agent plus de capacité d'analyse et l'aidera à rester dans le contexte. Ainsi, il y a une responsabilisation de l'agent sur ses fonctions [Boy, 98], d'autant plus qu'il peut développer des processus de compétences cognitives grâce à ce maintien du contexte.

Dans un ECLR, les résultats peuvent être interprétés différemment selon le contexte. Cela implique de mettre en oeuvre une approche linguistique dans notre modèle cognitif. Lors de la formulation des besoins, on se trouve face à des problèmes linguistiques tels que la métaphore, la polysémie (multiplicité des sens pour une même forme), la paraphrase (équivalence de sens dans plusieurs propositions). Pendant la conception de l'ECLR, pour limiter les ambiguïtés, nous allons adopter la formule « Si … alors ... » car elle fait référence à une structure condition-action mais également à une forme causale, une forme chronologique, une instruction qui rappelle celle de la programmation « if .... then .... ». [Prince, 96].

Selon le résultat de l'interprétation, l'agent réagit face au comportement du système et construit ses propres attitudes pour ne pas se retrouver en dissonance cognitive [Boy, 97]. Pour éviter cela, nous pouvons constituer plusieurs listes :

- une liste des tâches autorisées ou « do-list »
- une liste des tâches interdites ou « forbidden-list »
- une liste des vérifications à effectuer ou « check-list »

De ces trois listes, la plus critique et la plus intéressante est celle des tâches interdites car il s'agit d'assurer la robustesse de l'ECLR. Il s'agit alors d'une véritable **gestion des exceptions**

qui nous permet de contrôler les différentes évolutions et l'émergence de nouveau contexte selon les résultats des actions de l'agent. Nous détaillerons ces listes dans le chapitre 5.3.1.
De cette gestion des exceptions, nous pouvons déduire deux niveaux cognitifs pour un agent ECLR : réactif et décisionnel.

5.1.3. Structure d'un agent

Généralement, les agents expriment en terme de programmation, une méthode ou une technique de réalisation d'une tâche. Ainsi, un agent est une procédure, variable par ses connaissances et ses règles. Cette vision sur les agents, enrichies par les définitions par [Woolridge, Jennings, 95], [Ferber, 95] et [Gouardères & al, 96] justifient le comportement évolutif des agents.
En déterminant sa tâche et ses paramètres, un agent met en œuvre un mécanisme de résolution de la tâche définie dans ses procédures internes. Péninou [Péninou-93], l'un des pionniers de la structure de base d'un agent qui a inspiré plusieurs chercheurs par la suite, a représenté un agent comme suit :

Agent	Nom de l'agent.
Tâche	Tâche que l'agent assure.
But Agent	Objectif de l'agent.
Condition Activation	Les conditions nécessaires pour l'activation de la tâche.
Condition Terminaison	Conditions générales de réalisation de la tâche.
Corps	Le système permettant d'atteindre le But Agent.

Cette représentation mentionne les éléments de bases nécessaires à la définition d'un agent à savoir : le problème, le contexte et la réalisation.

5.2. PROCESSUS FONCTIONNEL D'UN MODULE ECLR

Les agents dont nous venons de présenter les modes de fonctionnement dans le chapitre précédent ne sont utiles qu'en ayant un support commun. En effet, ils ont besoin de communiquer entre eux et ce de manière précise et perspicace. Ce support est le module auquel ils appartiennent. Ainsi, l'ECLR respecte la définition et la composition d'un système multi-agents ou SMA [Ferber, 1995]. En effet, chaque module comme un SMA, est constitué de :

- Un environnement,
- Un ensemble d'agents représentant les entités actives,
- Un ensemble d'objets passifs que l'on peut situer dans le temps et l'espace, que les agents peuvent manipuler voire détruire,
- Un ensemble de relations qui unissent les agents,
- Un ensemble d'opérations permettant aux agents de percevoir, de produire, et de manipuler les objets.

Chaque module de l'ECLR fonctionne de la même manière, suivant le même modèle pour des raisons de simplicité d'intégration et de fonctionnement. Dans cette partie de notre étude, nous allons développer la modélisation de ces modules.

5.2.1. Structure organisationnelle hiérarchique d'un module

Comme nous l'avons vu dans le chapitre 4.3.2, chaque tâche ou fonction est traduite dans notre modèle par un module composé de plusieurs agents. Pour qu'un module soit valable, il doit être composé d'un agent décideur (AD) et de plusieurs agents simples. La structure organisationnelle d'un module correspond alors aux associations dynamiques d'intérêts divergents, parfois conflictuels des agents qui le composent.

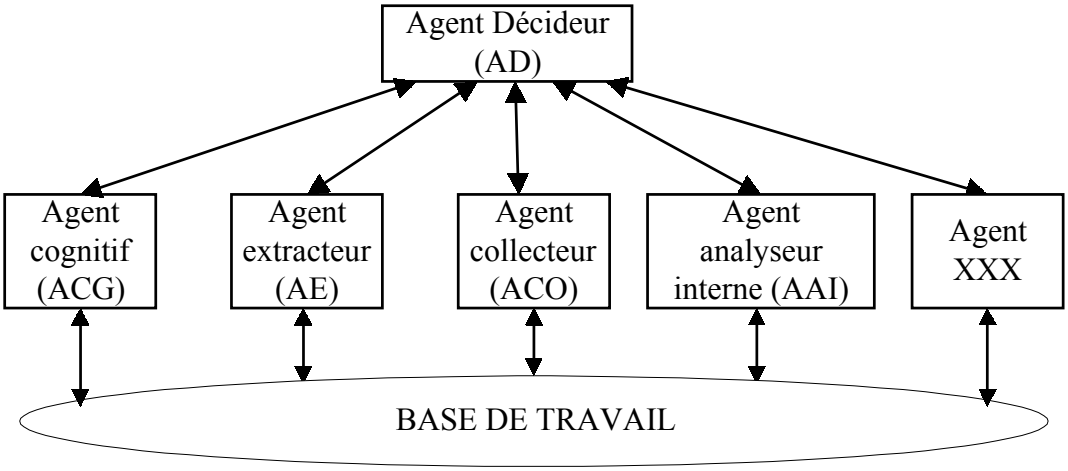


Figure 5-4 : Structure fonctionnelle d'un module ECLR

La décomposition de la complexité d'une tâche se fait par une coopération entre agents [Boy, 1995]. Aussi, faut-il éviter le cas de la compétition où deux agents ne sont pas informés au bon moment de l'existence de l'autre provoquant ainsi un ou plusieurs conflits. Il faut éviter les conflits entre les deux agents car ils peuvent détecter la même erreur mais proposer deux

solutions différentes. Par exemple, pour la barre de défilement d'une zone texte que l'utilisateur souhaite rétrécir : un agent propose de passer le reste du texte à une page différente tandis que un autre propose de diminuer la fonte.

5.2.2. Communication inter-agents

Les agents doivent communiquer entre eux pour la simple raison qu'ils doivent résoudre un problème commun qu'ils ne peuvent résoudre chacun séparément de manière satisfaisante. En effet, chaque agent a un rôle bien particulier et cette spécificité ne lui offre pas une connaissance complète du problème. Le facteur de risques dans la communication inter-agents peut être présenté comme la combinaison de deux métriques [Rozenberg, 1998] :

- la probabilité d'apparition d'un dysfonctionnement,
- l'impact de ce dysfonctionnement sur le module provoquant des problèmes par rapport à l'utilisateur et/ou sur l'environnement.

Les agents doivent par conséquent respecter un certain nombre de règles de synchronisation et de compréhension réciproque des autres agents composants le même module ou couche[47]. Cela se traduit dans notre modèle par une communication inter-agents contrôlée selon le schéma ci-dessous.

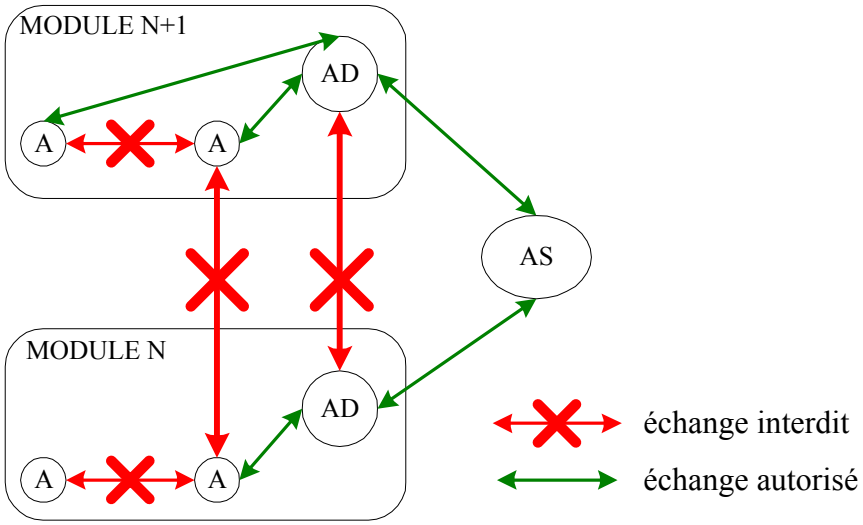


Figure 5-5 : Communication entre les agents

La figure ci-dessus représente les échanges possibles entre les agents simples et décisionnaires (AD et AS). D'après la structure fonctionnelle hiérarchique présentée sur la

[47] Voir figures 4.1 et 4.2

figure 5-2, les agents simples ne peuvent dialoguer qu'avec l'AD ce qui assure à ce dernier le contrôle total du module. Aucun échange n'est autorisé entre les modules sans passer par l'agent superviseur AS. De même un agent simple ne peut jamais s'adresser directement à l'AS. L'agent superviseur garde ainsi le contrôle suprême de l'ECLR.

Ainsi, la communication inter-agents correspond à l'interactivité entre ces agents. Le dialogue inter-agents est régi par certaines règles donnant le droit d'ouvrir une session bidirectionnelle de communication. Elle se fait à travers des messages qui peuvent être transmis avec l'une des trois méthodes suivantes [Cukier & al, 99] :

- Monocast : le message est destiné uniquement à un seul agent. Tous les agents de l'ECLR peuvent envoyer des monocast.
- Multicast : le message est destiné à un groupe d'agents. Les agents décideurs peuvent appliquer cette méthodes de communication avec les agents simples qu'ils contrôlent.
- Broadcast : le message est envoyé à tous les agents de l'ECLR. Ce mode de communication est réservé à l'AS.

5.2.3. Gestion des agents dans un module

Nous avons vu dans le chapitre 5.1 que chaque agent possède deux types de fonctionnement interne : procédural et cognitif. Sachant qu'il existe plusieurs agents dans un module, il est important d'étudier comment gérer l'activité de tous ces agents afin de minimiser les risques de conflits entre les agents simples et d'optimiser les prises de décisions faites par l'AD. Il est à noter que cette tâche de l'AD ne peut s'exprimer qu'en terme de probabilité ce qui induit un risque de se tromper quant aux décisions à prendre.

5.2.3.1.Priorité des agents

Les figures 5-3 et 5-4, nous montrent que les agents décideurs sont prioritaires dans les modules et l'agent superviseur prioritaire sur tout l'ECLR. En revanche, en cas de requêtes simultanées de plusieurs agents simples à l'AD, il faut classer ou ordonner les requêtes par rapport à l'objectif principal d'un ECLR qui est de fournir un environnement robuste.

Nous allons classer les agents simples en deux niveaux de priorité :

- Priorité haute

Les agents ACO, ACP et ACE sont les agents les plus prioritaires car leurs actions peuvent affecter le résultat final. L'ACO est primordial dans un module car il maintient le contexte dans lequel évolue le module et l'ECLR. De même, il assure la cohérence entre les bases de travail mises en œuvre. L'ACP et l'ACE peuvent avoir des conséquences directes sur l'auteur de niveau 1. Ils permettent de suivre l'évolution pédagogique du logiciel que l'auteur de niveau 1 est en train de concevoir, et de l'aider et le stimuler dans la représentation des données. Il est alors primordial de vérifier et de suivre systématiquement leurs fonctionnements et leurs actions.

- Priorité basse

Les agents AC, AAI et AAE ont une priorité inférieure aux autres agents simples parce que leurs actions n'ont pas de conséquences directes sur le résultat attendu par rapport au rôle de l'ECLR : assurer un environnement de création de logiciel robuste. Ils s'occupent des informations reçues de l'auteur de niveau 1 et des données dans les BdT. En effet, ils s'occupent des données venant de l'extérieur et de l'intérieur de l'ECLR. L'AC regroupe les données venant de l'auteur de niveau 1 et les classe dans une base de données que l'AAE va traiter pour différencier les données utiles aux différents agents. Selon le besoin, l'AAE ajoute ou enlève une partie de ces données avant de les proposer aux agents ACP et ACO. L'AAI s'occupe des données internes qu'il attribuera aux agents de priorité haute.

De ces différentes priorités des agents simples, nous obtenons les circuits des données dans le schéma ci-dessous :

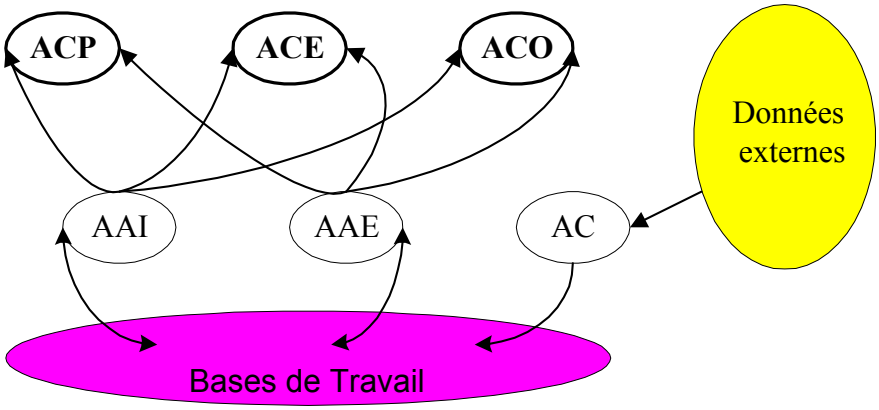


Figure 5-6 : Circuits des données dans un module

Enfin, il est important de noter que cette classification est destinée à faciliter la tâche de l'AD et ne doit pas affecter l'existence et l'utilisation de ces agents dans les différents modules.

5.2.3.2.Synchronisation des agents

Chaque module possède plusieurs agents simples ayant chacun leur rôle[48] et leur caractéristique. Lorsqu'une tâche doit être exécutée par un module, tous les agents de ce module doivent contribuer à la réalisation dans le meilleur délai de cette tâche et pour donner un résultat optimisé c'est-à-dire avec le minimum d'erreurs possible.

Le fait d'attribuer la tâche séquentiellement à chaque agent est facile à implémenter et à surveiller. En revanche, le délai risque de pénaliser l'intérêt d'utiliser un ECLR en sus du SAM pour concevoir un nouveau didacticiel. Nous allons alors faire fonctionner tous les agents de priorité haute simultanément.

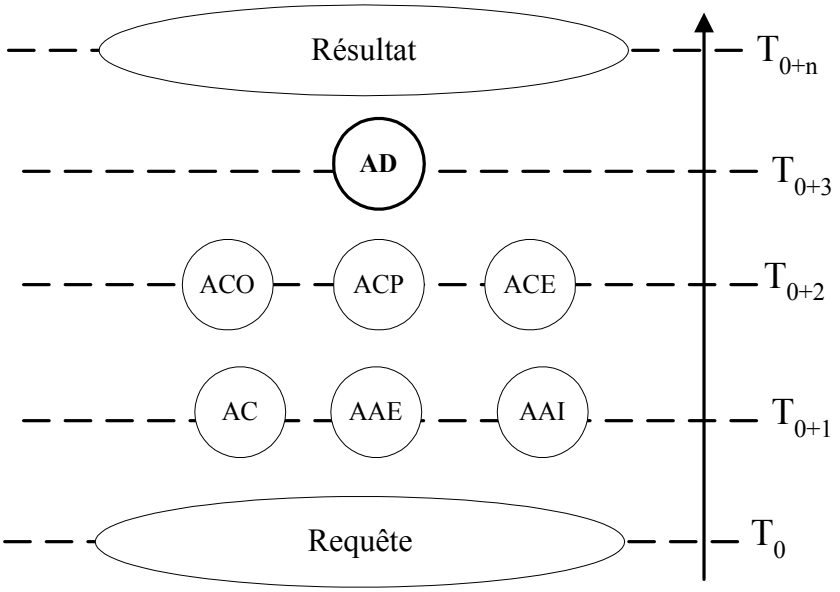


Figure 5-7 : Ordonnancement des activités des agents dans un module

La synchronisation des agents correspond à l'ordonnancement de leurs activités dans le temps. Considérons le temps T_0 le moment où la requête arrive au niveau du module.
Au temps T_{0+1} les agents de traitement de données s'activent en parallèle pour offrir des données fiables et correspondant aux fonctions des agents de priorité haute et responsable du principal travail du module.
A T_{0+2} les agents ACO, ACP et ACE manipulent les données afin de vérifier si l'objet requête donnée au module est conforme au cahier des charges et n'affecte pas la robustesse de l'ensemble.

[48] Les rôles attribués à chaque agent ont été définis dans le chapitre 4.3.3

Les résultats arrivent ensuite à T_{0+3} au niveau de l'AD qui prend alors des décisions pour soit donner le résultat, soit demander aux agents simples de re-traiter la requête suite à un conflit entre les résultats ou à la non-fiabilité du résultat. Par conséquent, le résultat des diverses activités menées en parallèles par les agents suite à une requête ne peut être donné qu'après n traitements d'où $T_{0+n.}$

5.3. MODELISATION GLOBALE DE L'ECLR

Un cahier des charges est un document structuré, rédigé en langage naturel dont la plus grande partie est consacrée aux besoins fonctionnels [Acm, 82]. Il s'agit alors d'un outil de référence aux autres phases du cycle de vie d'un logiciel. Les besoins fonctionnels décrivent les services fournis à l'utilisateur par le système et se traduisent en informatique par un programme structuré en une ou plusieurs fonctions également appelées tâches.
En génie logiciel, une spécification fonctionnelle est une référence pour le développement et un support de contrôle pour les vérifications. C'est alors la base de discussion commune entre les intervenants du projet.
Dans la méthode de développement « Cycle en V », chaque étape du développement est bien définie, donc l'application peut être découpée en plusieurs modules, ce qui permet d'attribuer chaque tâche à différentes entités, c'est-à-dire dans notre cas des agents différents. De plus, cette méthode implique des tests pour valider chaque étape.

La phase de conception est le découpage du système en sous-systèmes, eux-mêmes re-découpés en composants logiciels. Souvent le modèle qui en résulte évolue en même temps que le logiciel.
L'architecture globale d'un ECLR doit répondre à tous ces besoins en terme de données et de fonctionnalités.

5.3.1. Elaboration et catégories de bases de travail

Chaque composant d'un ECLR doit se ressourcer d'informations pour fonctionner dans le but d'assurer la robustesse de l'environnement de conception et d'utilisation du logiciel créé à l'aide d'un SAM. Ces informations proviennent du concepteur de l'ECLR, du SAM et de l'auteur de niveau 1.

Nous avons vu dans les chapitres précédents que les agents peuvent modifier certaines bases de données grâce à leur attitude empirique. De même, certaines bases évoluent à cause des données entrées au fur et à mesure du développement par l'auteur de niveau 1. Par ailleurs, le concepteur de l'ECLR et l'auteur de niveau 1 peuvent fixer dès le départ des informations qui sont valables et non modifiables quel que soit le contexte. Par conséquent deux catégories de bases de travail coexistent dans un ECLR : la base de travail interne et la base de travail externe.
Ces deux catégories de bases de travail sont collectives et accessibles par tous les agents de tous les modules de l'ECLR.

5.3.1.1.Base de travail interne

Dans le chapitre 4.5 nous avons précisé que les agents d'un ECLR possèdent une base de travail ou BdT qu'ils utilisent pour réaliser les tâches qui leur sont attribuées. Nous pouvons distinguer deux types de bases de travail interne :

- *Base de données statiques* regroupant les données relatives au domaine traité et aux fonctionnements de base (règles). Ce type de base de travail peut être qualifié d'ontologique car elle est fixée dès le départ. En effet, l'ECLR est conçu avec des variables et des paramètres dont les contenus sont fixés à l'avance telle la valeur de retour « vrai » ou « faux » d'un test d'existence d'un périphérique. Ces variables et paramètres peuvent être également renseignées par l'auteur de niveau 1. Nous pouvons prendre en exemple la nécessité d'avoir une carte son pour le fonctionnement du logiciel final.
- *Base de connaissances dynamiques* qui est évolutive selon le fonctionnement interne des agents. Elle résulte d'une association des modèles épistémiques[49], comportementaux et cognitifs[50] de l'apprenant. Ce type de base de travail est mis à jour en temps réel et représente le résultat de l'apprentissage de l'ECLR afin d'évoluer avec le contexte environnemental. Cette base est surtout dédiée au fonctionnement des agents de priorité haute telle que l'ACO.

La base de travail interne est complétée ou mise à jour par les agents simples AC et AAE et AAI de chaque module après son activité. Les éléments d'apprentissage, l'historique des

[49] Modèle de représentation du niveau de connaissance de l'apprenant.

[50] Modèle de simulation des capacités cognitives de l'apprenant.

actions des agents et les contenus des messages y sont décrits. Cela donne alors une ontogenèse des actions des acteurs (agents et l'auteur de niveau 1) permettant de vérifier le contexte et la structure du logiciel final.

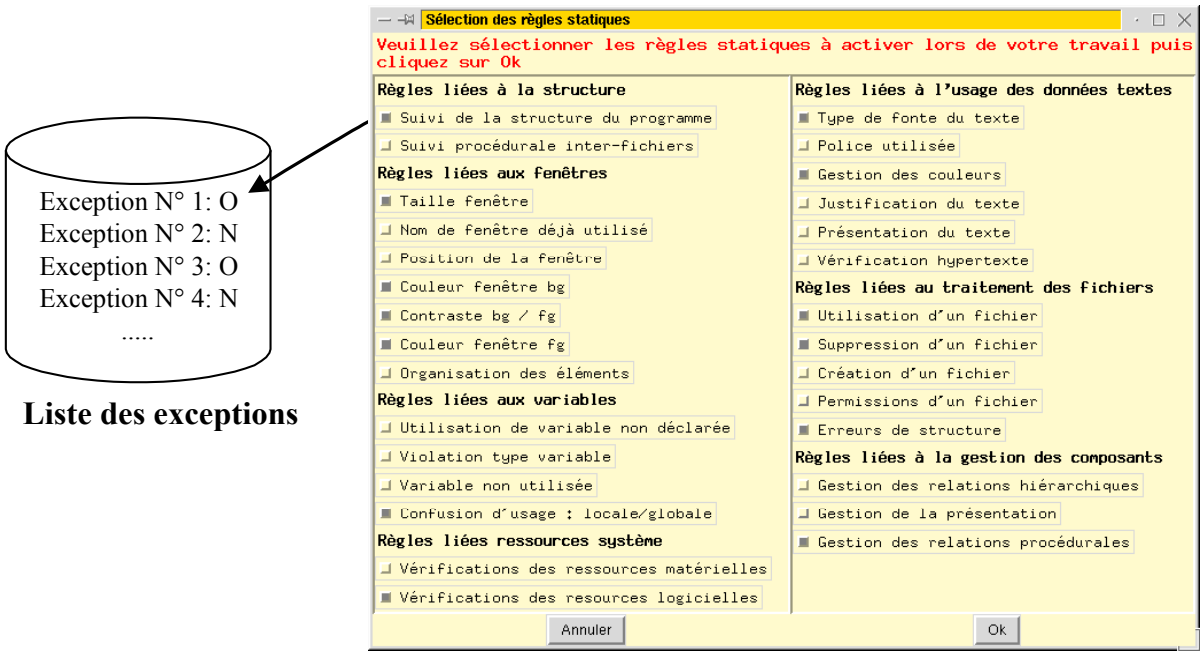


Figure 5-8 : La liste des exceptions

Parmi les informations contenues dans la base de travail interne, qu'il s'agisse d'information fixe donc constante ou d'information mobile donc variable, nous attachons un intérêt particulier pour les données constituant **la liste des exceptions**. En effet, cette liste des exceptions garantit le respect de l'objectif défini par l'auteur de niveau 1 ainsi que le respect de l'objectif de l'ECLR c'est-à-dire fournir un environnement de création de logiciel robuste. Nous pouvons scinder cette liste des exceptions en deux parties :

- une liste générale que l'ECLR doit respecter en terme de règles liées aux présentations des fenêtres et des données textuelles, sonores et graphiques. Par exemple, la gestion des polices de caractères doit être associée à celle de la taille des fenêtres.
- une liste contextuelle que l'utilisateur aura définie au préalable par rapport à son cahier des charges : dans la figure 5-8, cela correspond aux règles liées aux variables, à la structure, aux ressources système, au traitement de fichier et à la gestion des composants. Nous pouvons par exemple penser à l'impossibilité d'utiliser le bouton « résultat » du didacticiel tant que l'utilisateur n'a pas choisi sa réponse dans un questionnaire. Cela correspond à la gestion de la présentation et à la gestion des relations procédurales contenues dans les règles liées à la gestion des composants.

Cette liste peut être enrichie par les anomalies rencontrées lors de la conception du logiciel final.

5.3.1.2.Base de travail externe

Nous appelons base de travail externe la base qui regroupe toutes les actions faites par l'auteur de niveau 1. Cette base constitue un historique de tout ce que fait l'utilisateur permettant ainsi à l'ECLR d'apprendre son comportement et de vérifier si les actions sont valables par rapport à la liste des exceptions. Pour que cette BdT externe ne soit pas une simple duplication du programme et afin d'éviter une base volumineuse et difficile à gérer, nous n'y ajoutons que les actions signifiantes et nécessaires pour l'évolution future du programme, par exemple les dernières valeurs des variables et paramètres, les appels de fonctions.

5.3.1.3.Plan de gestion des BdT

L'agent superviseur a un rôle fondamental pour la gestion des données. En effet, aucune information ne parvient aux modules de l'ECLR sans son accord. Cela signifie que l'AS est l'interface unique face au SAM et face à l'auteur de niveau 1.

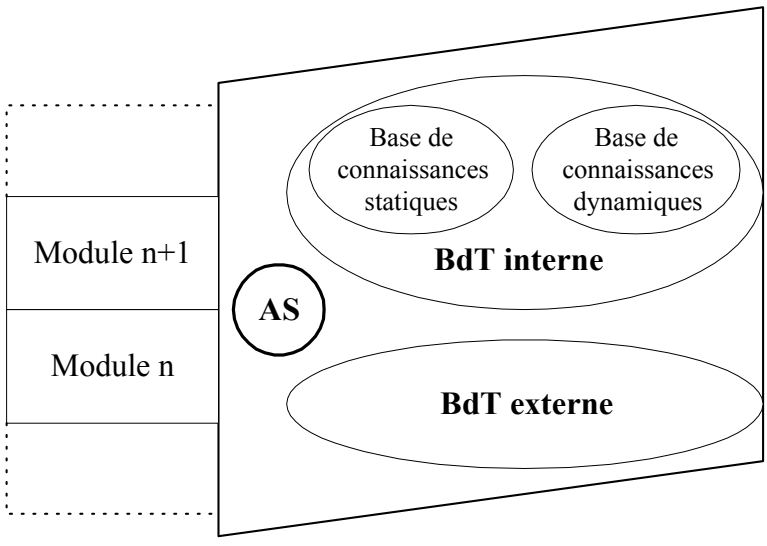


Figure 5-9 : Plan de Supervision d'un ECLR

Comme l'AS possède un rôle capital dans la gestion des BdT externes et internes, nous allons les regrouper, comme le montre la figure 5-9, dans un même plan dont les périmètres sont juxtaposés à tous les modules de l'ECLR. Ce plan, appelé plan de *Supervision* correspond à la couche transversale présentée figure 4-2, et qui sert à contrôler les échanges entre les modules

ainsi que le résultat livré par chaque module, à gérer, maintenir et enrichir les bases de travail de l'ECLR.

Les BdT internes ou externes sont communes à tous les agents et à tous les modules pour des raisons surtout pratiques. Selon les tâches effectuées, les agents peuvent utiliser une base plutôt qu'une autre. Cet accès illimité et libre aux deux types de BdT permettent aux agents de mieux évaluer le contexte global c'est-à-dire qu'ils peuvent mieux estimer les souhaits de l'auteur de niveau 1 et faire un rapprochement avec les anciennes actions faites par les deux catégories d'acteurs (les agents et l'individu) pour s'assurer de la structure globale.

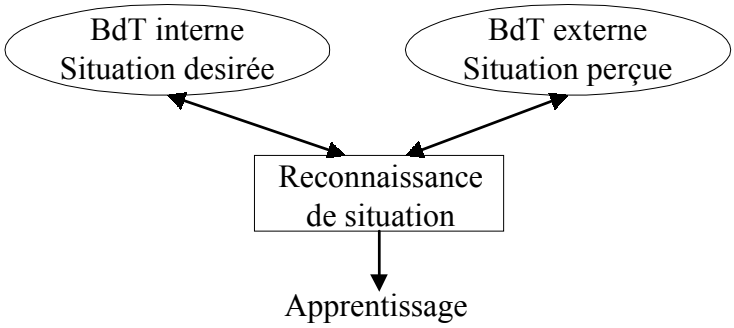


Figure 5-10 : Statut opérationnel-cognitif de l'ECLR

La BdT interne regroupe les informations fixées (interdites ou non) dès le départ dans l'ECLR et par l'auteur de niveau 1, avec les anciennes informations apprises dans le temps et qui reflètent les souhaits de ce dernier pour la création de son nouveau logiciel. La BdT externe exprime la situation ou le contexte que veut mettre en œuvre l'individu à un moment donné, d'où l'expression situation perçue dans la figure ci-dessus. En effet, les agents du SMA ECLR perçoivent à travers l'analyse de l'action en cours, une situation voulue et résultante de l'action de l'auteur de niveau 1. L'accès aux deux BdT permet à chaque agent de mieux évaluer le contexte et par conséquent de retenir les informations utiles et donc d'optimiser l'apprentissage.

Cette distinction des bases de travail offre également l'avantage d'être facile à implémenter lors de la conception de l'ECLR lui-même.

5.3.2. Architecture en couche de l'ECLR

Afin de mieux exploiter les différents agents et modules, nous proposons pour l'ECLR un modèle en couche qui respecte les relations entre les différents composants de l'architecture générale d'un environnement didactique tel que nous l'avons défini dans le chapitre 4.5. Comme nous l'avons développé dans le chapitre 3.2.4, l'ECLR a un rôle permanent d'apprentissage. Il faut qu'il incite l'auteur de niveau 1 à progresser à travers l'EIAO. Il faut qu'il apprenne aux agents à évoluer à travers le STI. L'architecture de l'ECLR doit alors lui permettre de trouver, de sélectionner et de composer les éléments pour ces apprentissages.

Comme nous l'avons présenté dans le chapitre 5.3.1, les trois modules de l'ECLR partagent les mêmes BdT administrées par l'AS et qui se situent dans le plan de *Supervision*. Le fait d'avoir des bases communes améliorent la reconnaissance de situation par l'ECLR comme le décrit la figure 5-9. En effet, grâce à l'analyse des différentes situations désirées ou recherchées par l'auteur de niveau 1, et des situations perçues au fil du temps, les agents ECLR peuvent mieux estimer ses activités (perception, anticipation, analyse et action) et optimiser leurs apprentissages.

Les trois modules partagent également un seul agent superviseur qui a le rôle d'un chef de projet car l'AS contrôle, prend des décisions et qualifie les tâches à exécuter par chaque module. L'AS gère également la synchronisation des activités de tous les modules pour contrôler et maintenir le contexte.

Pour représenter l'ensemble de l'ECLR, nous adoptons une architecture en couche en trois dimensions selon le schéma suivant :

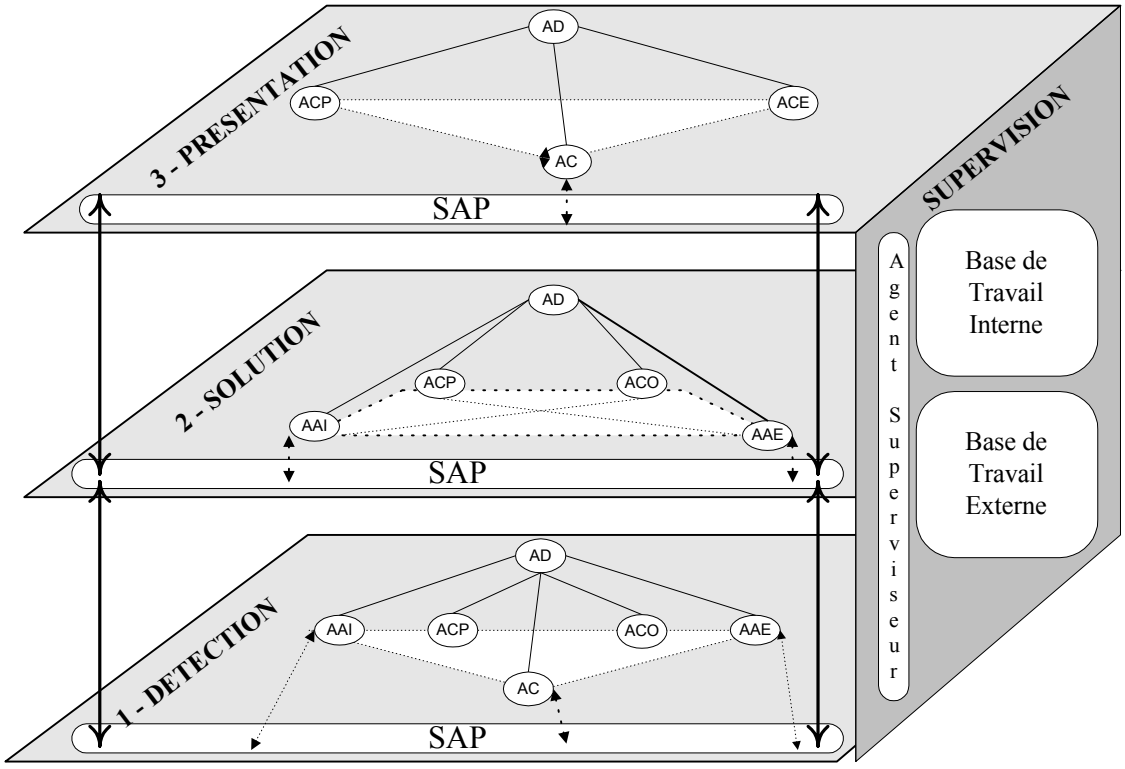


Figure 5-11 : Modèle d'intégration en couche

Comme le montre la figure ci-dessus, notre modèle global d'ECLR possède uniquement trois couches bien définies en sus du module ou de la couche de *Supervision* (figure 5-8). Ce choix provient des raisons suivantes :

- Sachant que l'ECLR est un programme actif en parallèle du SAM, nous ne souhaitons pas affecter la performance de l'ensemble des deux applications l'ECLR et le SAM. Par expérience, nous constatons qu'un informaticien ou non est impatient devant le temps d'exécution d'une tâche par un ordinateur.
- Par souci de clarté et de simplicité de gestion, nous limitons le nombre de **modules ou couches**. En effet, les expériences dans les autres domaines de l'informatique utilisant des modèles en couche comme les réseaux de télécommunication, ont démontré qu'un nombre excessif de couches complique l'ossature et par conséquent la gestion du modèle global.
- En dehors de la gestion, comme nous mettons en œuvre des agents ayant des capacités empiriques et cognitifs, il est alors important de contrôler chaque action et chaque composant de l'ECLR.
- Respect de la modularité de l'ECLR : ce dernier doit être facile à implémenter et à faire évoluer.

Le modèle d'intégration de l'ECLR est fortement inspiré des modèles en couches des réseaux informatiques. Il résulte d'un mélange de deux types de modèles de réseaux : TCP/IP pour la simplicité de fonctionnement et le nombre limité de couches, et ATM pour la gestion transversale des couches. Ces deux types de réseaux ont démontré leur performance et leur efficacité à travers Internet par exemple.
Le fonctionnement global de l'ECLR est en partie une adaptation du modèle de fonctionnement d'un opérateur humain [Rasmussen, 80]. Ce modèle comprend 4 étapes : la détection d'événement, l'évaluation de la situation, la prise de décision et l'exécution des actions. Bien que souvent critiqué par ses aspects réducteurs, ce modèle permet d'analyser le comportement cognitif de l'opérateur humain dans la mesure où il rend compte des multiples catégories de comportement couramment recensés dans la littérature.

5.3.2.1.Analogies entre les modules

Bien que chaque module ait un rôle bien défini et soit constitué de différents agents, l'ECLR propose des couches ayant des points communs ou des ressemblances dans leurs fonctions. Nous allons décrire dans ce sous-chapitre ces analogies.

Chaque module de notre modèle ECLR raisonne avec ses savoirs propres et possède parmi ses objectifs, un objectif local particulier. Ce dernier est géré ou supervisé par l'agent décideur de la couche sachant que tous les agents simples, quel que soit le module sont reliés directement à l'AD. La prise de décision globale est assurée par l'agent AS dont le champ d'action est étendu sur toutes les couches. Il prend ses décisions à partir de la synthèse des bases de travail en attachant un intérêt particulier à la liste des exceptions. L'AS possède alors un comportement procédural fondé sur des règles, appelé également « Rule based Behaviour » en langage des sciences cognitives [Rasmussen 1986] [Reason 1993].

Le fait d'avoir à utiliser les mêmes BdT internes et externes optimise ce que l'on appelle « skill based behaviour » c'est-à-dire le comportement basé sur l'habilité et l'expérience où l'acteur effectue de façon quasi-réflexe des actions en réponse à des informations reçues.

Les trois modules ou couches communiquent entre eux uniquement à travers les interfaces d'accès SAP que nous avons représentées figure 4-1. Ils utilisent également le SAP pour accéder aux BdT qui se situent dans le plan de *Supervision*.

Un bon système multi-agents met en jeu trois critères fondamentaux [Ferber, 95] qui sont l'autonomie, l'intelligence et la mobilité de ses agents. Le modèle d'intégration de l'ECLR répond à ces critères grâce à la mise en œuvre de plusieurs dichotomies en psychologie comme dans la conception de systèmes à base de connaissances en IA [Van Daele, 92] à savoir :

- Le savoir et le savoir-faire
- L'image cognitive et l'image opérative
- La structure et la procédure
- La représentation et le traitement
- La connaissance profonde et la connaissance de surface.

Le transfert de ce savoir-faire entre les couches se fait à travers les interfaces d'accès SAP que nous avons représentées figure 4-1, grâce aux protocoles et primitives[51] qui régissent la communication entre les couches. Ils utilisent également le SAP pour accéder aux BdT qui se situent dans le plan de *Supervision*.

5.3.2.2.La couche Détection

Cette couche assure la fonction d'analyse syntaxique, lexicale et sémantique des données provenant de l'utilisateur. Son rôle est primordial car elle est la base de l'ECLR. De plus, elle permet de vérifier la compatibilité ou la conformité de chaque action de l'utilisateur par rapport à la définition du logiciel, préalablement rentrée par l'utilisateur et conformément au cahier des charges.

Par exemple, si un utilisateur a demandé à avoir au maximum 3 fenêtres affichées simultanément à l'écran pour son didacticiel final. A une certaine période du développement, l'utilisateur par oubli veut créer une fenêtre supplémentaire alors qu'il en a déjà créé trois. La couche Détection permet à l'ECLR de détecter ce problème, de l'identifier et de le communiquer au plan supervision ainsi qu'à la couche Solution.

[51] Voir chapitre 5.4

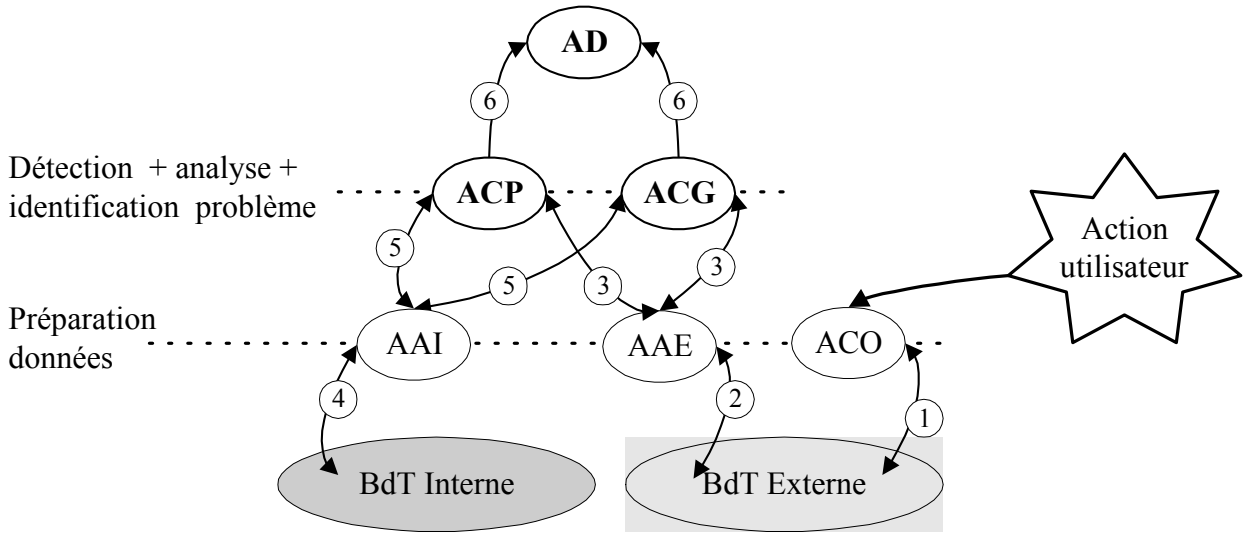


Figure 5-12 : Fonctionnement de la couche *Détection*

La couche *Détection* est constituée des agents AAI, AAE, AC, ACO, ACP, AD et fonctionne comme suit :

1. L'ACO collecte les actions de l'auteur de niveau 1, venant de l'extérieur et les traduit sous forme de données dans la base de travail externe. Il s'agit alors d'une codification de l'information reçue afin que les agents puissent la traiter. La fin de l'extraction des informations utiles dans la BdT externe déclenche l'activité des agents ACO et ACP.
2. L'AAE extrait des informations de la BdT interne et de la BdT externe et les envoie vers les agents de priorité haute afin d'être analysées.
3. Les données utiles sont reçues par l'agent ACP qui atteste la conformité pédagogique et par l'agent ACO qui contrôle si tout correspond à l'objectif et au contexte. Ces agents de priorité hautes font ensuite appel aux agents AAI et AAE pour vérifier si les informations ne sont pas dans les listes des exceptions afin de détecter s'il y a anomalie dans l'action de l'utilisateur.
4. L'AAI utilise alors la liste des exceptions pour éliminer les erreurs graves, par comparaison des variables ou paramètres de la BdT interne. L'AAI renvoie aux agents ACO et ACP le résultat des comparaisons.
5. Les agents de priorité haute adressent ensuite le diagnostic à l'agent décideur afin que ce dernier les valide.

Si aucun problème particulier n'est détecté par la couche *Détection* , l'AD de celle-ci envoie un signal à l'AS et ne communique rien à la couche supérieure. Dans ce cas, l'AS mettra à

jour la BdT interne et la BdT externe, et ensuite attend le prochain événement ou action de l'auteur de niveau 1.

5.3.2.3.La couche Solution

La couche *Solution* prépare les données qui seront rendues à l'auteur de niveau 1, à partir des bases de travail internes et externes. L'extraction par l'AAE et par l'AAI de ces informations est relative aux problèmes détectés et identifiés par l'analyse de la couche 1.

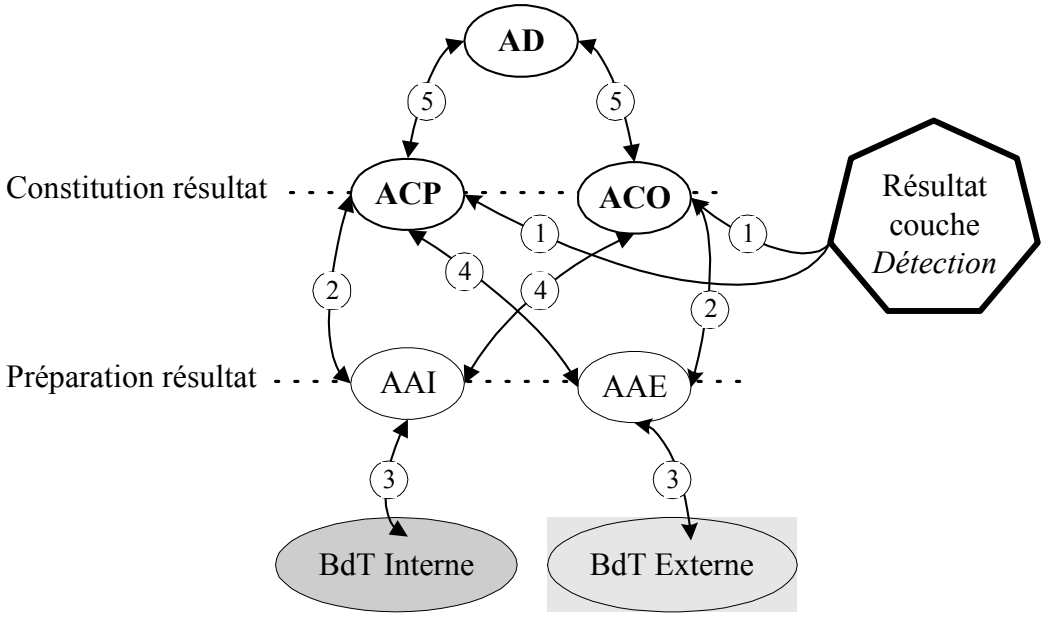


Figure 5-13 : : Fonctionnement de la couche *Solution*

La couche *Solution* est composée d'un agent décideur et des agents ACP, ACO, AAI et AAE. Elle fonctionne comme suit :

1. Les agents ACP et ACO reçoivent des informations venant de la couche *Détection*. Les résultats reçus indiquent les données qui sont critiques voir interdites, et vont être analysées par ces agents.
2. Les mêmes agents demandent ensuite aux agents de priorité basse les informations nécessaires pour construire le message d'erreur qui sera pédagogique et qui aidera l'utilisateur à corriger le problème.
3. Les agents AAE et AAI viennent rechercher les informations requises dans les BdT externe et interne respectivement.

4. Les agents ACP et ACO mixent alors les résultats venant de la couche précédente avec chaque information reçue des BdT, et compose le contenu du message qui peut être uniquement une alerte ou un message d'erreur avec proposition de résolution.
5. Les agents de priorité haute adressent ensuite à l'AD leurs propositions :
    - Si la solution venant de l'ACO ne contredit pas celle de l'ACP, l'AD les communique à l'AS pour que ce dernier enrichisse les BdT, et complète les primitives qui véhiculent les données à la couche *Présentation*.
    - S'il y a conflit de résolution, l'AD renvoie les deux propositions simultanément à l'ACP et ACO afin qu'ils recommencent leurs analyses.

5.3.2.4.La couche Présentation

La couche Présentation garantit la présentation des messages d'erreurs à l'apprenant sous forme pédagogique et ergonomique et assure l'interactivité entre l'apprenant et le didacticiel. Elle représente alors l'interface qui a pour rôle d'établir la communication entre deux mondes : celui de l'homme et celui de la machine, et d'assurer la correspondance des concepts propres à chaque partenaire en s'appuyant sur un ensemble de représentations qui doivent avoir un double aspect : formel pour la machine et perceptible pour l'homme.

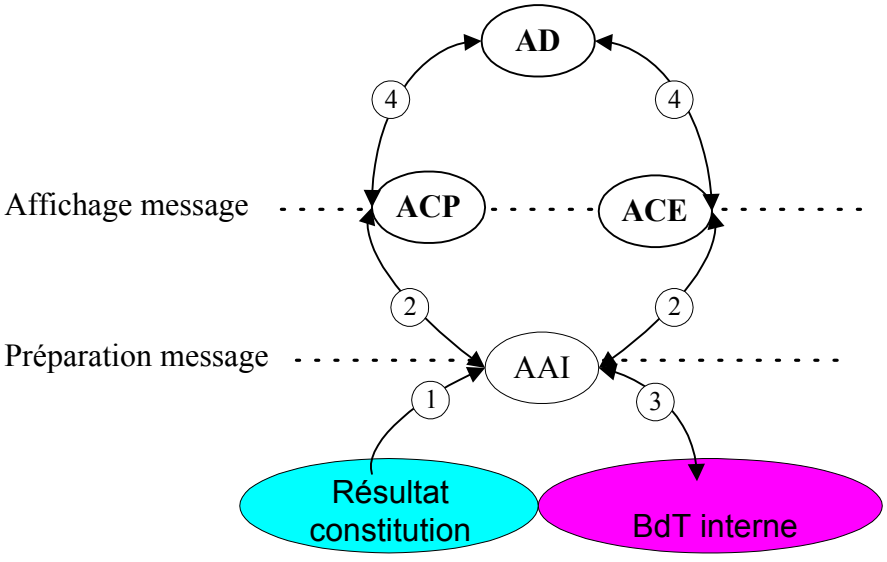


Figure 5-14 : Fonctionnement de la couche *Présentation*

Le rôle des agents ACP et ACE est important mais d'autres agents composent la couche *Présentation*. En effet, ce module ne peut fonctionner sans l'AD et l'AAI.

1. L'AAI extrait le message d'erreur provenant de la couche *Solution*.

2. Les agents ACP et ACE travaillent pour la présentation du message d'erreur à l'auteur de niveau 1. Rappelons que cet utilisateur peut être néophyte et que l'ECLR doit l'aider à ne pas se trouver bloquer face à un problème.
3. Des données venant de la base de travail interne peut être utile afin d'être en harmonie avec l'interface utilisateur. Il est inutile d'afficher un message avec une police de caractères qui est en contradiction avec le choix de l'utilisateur.
4. Enfin, l'AD communique le résultat de l'ensemble à l'AS qui affichera en dernier le message à l'écran, mettra à jour la BdT interne, et réinitialisera la BdT externe.

5.4. PROTOCOLES ET PRIMITIVES

Pour optimiser la gestion des couches, nous allons proposer deux groupes de protocoles de gestion, inspirés de la méthode MERISE :

- Les protocoles de traitement ou MT pour Modèle de Traitement, décrivent le cycle parcouru par les données à travers le modèle en couche de l'ECLR.
- Les protocoles de communication ou MC pour Modèle de Communication, décrivent les modalités d'échanges d'informations entre les modules.

Notre modèle d'intégration en couche possède alors des caractéristiques heuristiques car le traitement de l'information suit des règles bien définies et respecte les sept activités d'un processus cognitif défini par [Norman et Drapper, 86] :

- l'établissement d'un but,
- la formation d'une intention,
- la spécification d'une suite d'actions,
- l'exécution des actions,
- la perception de l'état du système,
- l'interprétation de l'état du système,
- l'évaluation.

5.4.1. Protocole de traitement

Au niveau du traitement, la gestion est assurée par l'agent décideur de chaque couche, et par un traçage de chaque événement et action portés par un agent jusqu'au concensus de l'agent superviseur pour donner le résultat au module suivant.

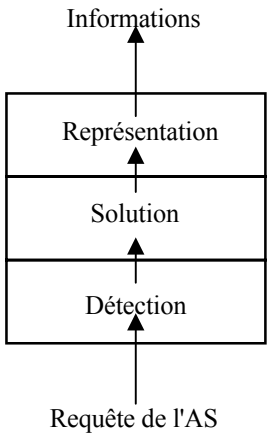


Figure 5-15 : Processus de traitement

Les informations rentrées par l'utilisateur passent d'abord à l'AS qui en gardent une trace et les communiquent au module *Détection* pour être traitées. L'AS peut également demander aux modules de traiter une action indépendamment de l'entrée ou de l'intervention de l'utilisateur. Par exemple, l'agent superviseur peut demander aux 3 modules de lui retourner l'état des différents agents qui les composent afin de vérifier le bon fonctionnement de l'ECLR.

Sachant que l'AS est la seule interface de l'ECLR avec l'extérieur, en particulier avec le SAM et l'auteur de niveau 1, toutes les actions des modules commencent par la couche *Détection* . En effet, par rapport à notre modèle, lorsque le plan de *Supervision* soumet une requête au plan de traitement, le processus est initié au niveau de la couche *Détection* . Celle-ci développe un service qui une fois validée par l'AS, est transféré à la couche *Solution*. Cette dernière travaille à son tour sous le contrôle de l'AD qui lui-même est supervisé par l'AS, et envoie son résultat à la couche suivante. La dernière couche *Présentation* formate dans le sens structurer et adapter, les données et les communique à l'utilisateur.

5.4.2. Protocole de communication

Au niveau du plan de communication, la gestion se traduit par la gestion globale de l'information et la mise en place d'un dialogue entre les couches. Ainsi, les modules sont autonomes mais adoptent une stratégie de communication commune.
Bien que le circuit des informations suive uniquement le sens module *Détection* vers module *Présentation*, les couches et le plan de supervision communiquent entre eux, rien qu'à travers l'accès aux bases de travail qui sont localisées dans le plan de supervision.

Un protocole de communication est l'ensemble des règles et des formats déterminant la communication des entités et des modules lorsqu'ils effectuent des fonctions appelées également services.

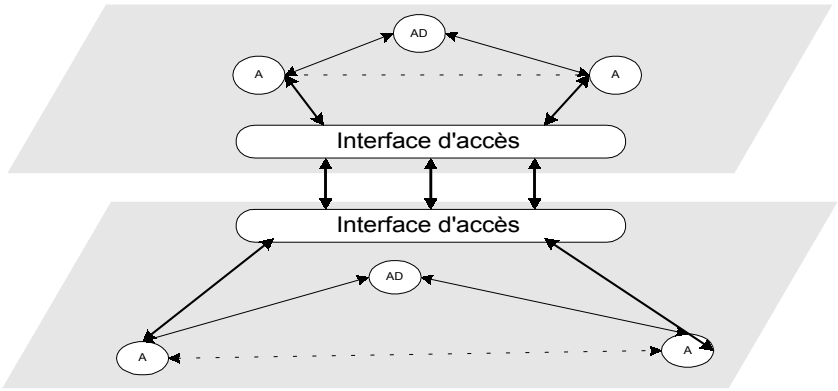


Figure 5-16 : Gestion de dialogue entre les couches

Le dialogue entre les interfaces d'accès se fait par le biais de primitives[52] [Tanenbaum, 1996][Stevens, 1994].

Les primitives constituent la base du dialogue entre les couches adjacentes de notre modèle. Elles peuvent être de plusieurs types : des demandes, des indications, des réponses, des confirmations, des transferts, etc.

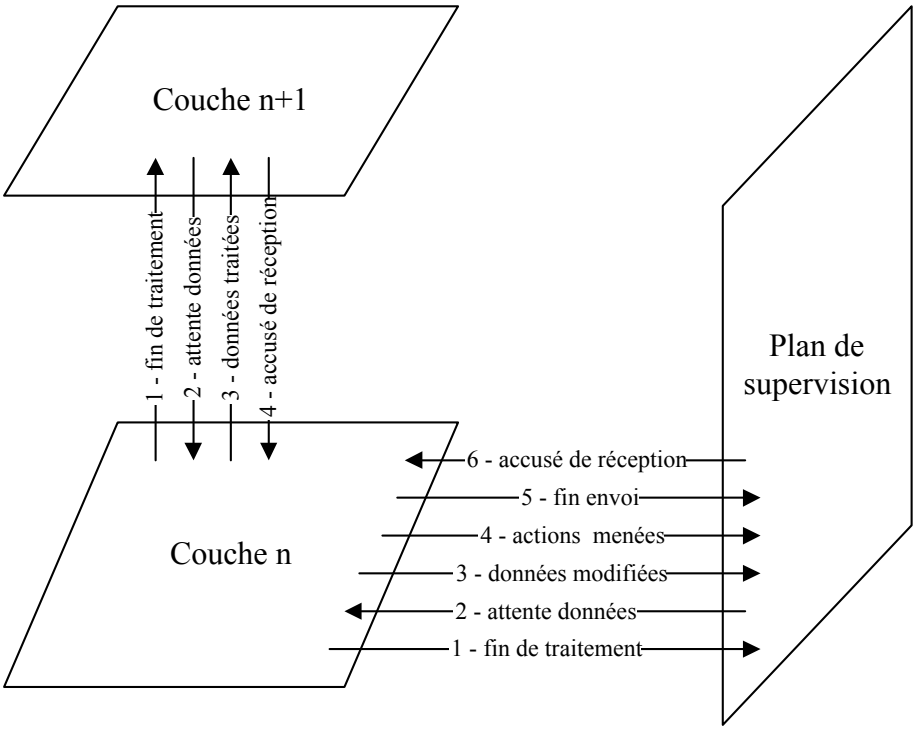


Figure 5-17 : Les primitives d'un ECLR

[52] Par analogie aux modèles en couche dans les réseaux protocolaires informatiques.

Les primitives peuvent utiliser des paramètres pour donner un sens aux différentes requêtes entre les couches. Les communications entre les modules de l'ECLR sont :

1. Fin de traitement de la couche n
2. Attente de données à traiter par la couche n+1
3. Envoi de données vers la couche n+1
4. Accusé de réception à la couche n

Voici un exemple de dialogue pour signaler un transfert de données entre la couche *Solution* et la couche P*résentation*.

1. La couche 2 indique à la couche 3 que le contenu des messages est prêt à être mis en forme via la primitive «Fin de traitement »
2. La couche 3 confirme qu'elle est prête à recevoir le message par un avis «Attente données ». Cette confirmation est nécessaire pour éviter de saturer l'ECLR au cas où la couche *Présentation* n'a pas fini un autre traitement. Cette couche peut éventuellement rajouter une demande de paramètres supplémentaires à cet avis.
3. La couche *Solution* envoie alors le message.
4. La couche 3 doit confirmer la bonne réception des données afin de libérer définitivement la couche 2 par rapport à l'action en cours.

En ce qui concerne la gestion globale, l'AS évalue et valide en permanence le déroulement des échanges entre les modules. Nous notons ici une différence entre évaluation et validation : l'évaluation est centrée sur le module lui-même en contrôlant sa qualité technique alors que la validation se préoccupe des effets sur l'ensemble de l'ECLR mais également sur l'auteur de niveau 1.

Les primitives échangées entre chaque module et l'AS sont :

1. Fin de traitement : le module signale à l'AS qu'il a fini sa partie de travail. Cela correspond à demander à l'AS l'autorisation de passer à l'état inactif.
2. Attente de données : l'AS accuse réception de cette demande changement d'état mais ne l'accorde que si les données correspondantes aux actions effectuées ne lui soient communiquées.
3. Données modifiées : le module envoie au plan de supervision les informations relatives à son action tel que le changement des valeurs d'une base de travail.
4. Actions menées : le module envoie l'historique de ses actions.

5. Fin envoi : le module signale à l'AS qu'il n'y a plus rien à envoyer.
6. Accusé de réception : l'AS après avoir mis à jour les bases de travail, et après avoir vérifié la validité des actions à travers une analyse de l'historique accorde au module de se mettre en état inactif ou en attente d'un événement comme l'indique la figure 4-6.

5.5. CONCLUSION

Cette partie de notre étude présente une architecture basée sur un système multi-agents et dédiée à la conception de nouveaux logiciels robustes à partir d'un SAM. Cette architecture possède de nombreux avantages :

- La décomposition en couche des principales fonctionnalités des didacticiels permet une modularité dans la conception qui se traduit par une simplicité dans l'implémentation, une décomposition de la problématique et la possibilité de faire des retours en arrière partiels en amont ou en aval de l'implémentation sans modifier pour autant tout le système. Ce modèle en couche permet également l'ajout simplifié de nouveaux agents éventuels.
- Les propriétés cognitives des agents, ainsi que leur traçage, garantissent l'adaptation de l'ECLR au contexte.
- La présence des agents extracteurs et collecteurs d'informations, assure la pertinence et la fiabilité des données traitées.
- Basée sur une coopération entre agents à deux niveaux différents (plan de traitement et plan de supervision), cette architecture offre souplesse et simplicité dans la conception.
- Grâce aux agents décideurs, le système est sécurisé et garantit une vérification qualitative et systématique des actions des agents simples et des modules.

La granularité des tâches, la gestion des bases de travail, la mise en place d'une liste des exceptions, le contrôle des actions effectuées par chaque module et les primitives échangées entre les couches, sont les vecteurs de garantie de la robustesse de l'environnement de création de logiciels et de la qualité de ces logiciels finaux.

6. CONCRETISATION ET MISE EN ŒUVRE D'UN ECLR

L'ECLR est un outil qui aide l'utilisateur à créer un logiciel robuste par construction c'est-à-dire qu'il intervient pendant la conception d'un nouveau logiciel. Il s'agit de détecter tout type d'erreur de conception en mettant en œuvre des agents intelligents dans le sens où ils ont la capacité d'apprendre, de décider et d'agir. Les agents possèdent également des connaissances d'accointance car ils ont des informations explicites sur leurs compétences et les connaissances d'autrui. Ainsi, l'ECLR offre un environnement d'apprentissage actif et évolutif à ses acteurs : les agents eux-mêmes et l'auteur de niveau 1.

Par ailleurs, cet outil doit s'adapter aux changements dynamiques c'est-à-dire aux évolutions de la conception du nouveau logiciel [Aknine &al, 98]. Gérer des changements dynamiques implique dans notre cas, vérifier que les activités réalisées par l'auteur de niveau 1 sont consistantes avant et après l'occurrence de chaque changement. Cela nous amène à définir les trois activités principales suivantes pour l'ECLR :

- Reconnaître les événements quel que soit leur environnement,
- Réagir en temps opportun à ces événements en cas d'incompatibilité ou d'incohérence,
- Constituer dynamiquement les informations utiles pour les tâches à venir.

Dans ce chapitre, nous allons détailler les moyens et méthodes mis en œuvre pour réaliser chacune de ses activités. Nous commencerons par décrire les processus d'analyse de l'ECLR. Ce processus commence par l'identification d'un événement, c'est-à-dire en terme de programmation à partir d'un SAM, la reconnaissance d'une instruction de ce SAM. Cette dernière sera ensuite analysée et vérifiée par notre outil afin de valider sa cohérence avec l'objectif de l'auteur de niveau 1. **La reconnaissance d'une instruction suppose la connaissance du fonctionnement du SAM. Aussi, un ECLR ne peut s'appliquer et être associé qu'à un SAM particulier.**
Ensuite, nous exposerons les structures, les relations et l'implémentation des composants actifs qui constituent l'ECLR. Les bases de travail de ces composants seront également présentées à travers leurs références, leurs constitutions et leurs formats. Nous verrons que la détection d'erreurs est assurée par des assertions exécutables portant sur les données traitées et les données obtenues. La mise en œuvre de ces assertions nous incite à appliquer *une programmation défensive* se basant sur un traitement des exceptions. La constitution par

l'auteur de niveau 1 d'une liste des exceptions en sus de celle intégrée à l'outil ECLR offre une approche analytique et comparative.

Nous décrivons ensuite la constitution de chaque module qui compose l'ECLR. Avant de donner une synthèse globale sur notre outil, nous décrirons les comportements opérationnels de l'ECLR par rapport aux traitements standards d'un auteur de niveau 1 comme la saisie d'un texte ou la constitution d'un scénario d'images animées.

6.1. PROCESSUS D'ANALYSE DE L'ECLR

Tout programme informatique est constitué d'un chaînage logique d'instructions. Cela implique dans le cadre de l'ECLR l'étude de deux concepts de base : la pertinence d'une instruction et la validité des relations entre les instructions. Il est alors important de définir ce qu'est une instruction car il s'agit pour l'ECLR de l'identifier le plus tôt possible pour minimiser le nombre et la gravité des erreurs éventuelles.

Pour déterminer la cause d'un problème, deux démarches sont proposées : vue locale c'est-à-dire par étape et vue globale du système [Saxena & al, 99]. Cela correspond à analyser chaque instruction pour contrôler localement sa validité, ensuite à étudier la corrélation avec les autres instructions pour vérifier l'ossature du logiciel final à travers la vue globale.

Le processus global d'analyse se déroule comme suit :

```
Proc processus-global {
Délimitation-instruction
Analyse-locale
Analyse-globale
Ajout-instruction $liste-instruction
} ;
```

Au point de vue informatique « standard », l'échantillonnage du temps de prise en considération de nouvelles entrées utilisateur est en fonction du MTTF ou Mean Time to Failure, c'est-à-dire le temps moyen jusqu'à la prochaine défaillance donné par la formule [Fournier, 93] :

$$MTTF = E(t) = \int_0^{\infty} t.f(t)dt$$

Cette estimation n'est pas applicable dans le cadre de l'ECLR car notre outil prend en considération des facteurs humains et contextuels. La prise en considération de nouvelles

entrées est uniquement en fonction de l'objectif, du comportement et du niveau de connaissance informatique de l'utilisateur. Ces particularités donnent un aspect asynchrone et aléatoire [Bui, 98] à la détection d'erreur dans la conception d'un logiciel car le moment où une action du concepteur se manifeste est parfaitement imprévisible (aspect aléatoire) et ne peut être correlé avec aucune grandeur physique du système (aspect asynchrone).

6.1.1. Reconnaissance d'une instruction

L'outil ECLR suit l'évolution de programmation de l'auteur de niveau 1 pendant la conception de son logiciel. Il doit par conséquent suivre des points d'interventions bien définis afin de ne pas déranger l'évolution de ce processus. Ces points d'interventions sont représentés par les fins d'instructions sachant qu'une instruction est pour nous une action de l'auteur de niveau 1.

Une action-événement, donc une instruction est une suite d'informations introduites par l'auteur de niveau 1 avec un objectif particulier. Cela signifie qu'une instruction résulte d'une entrée d'informations à travers les entrées standards de la console de l'ordinateur et donne obligatoirement un résultat ou un effet direct ou indirect sur le système. Par exemple, le glissement de la souris sur un menu n'est pas considéré comme une instruction ou une action tant que l'auteur de niveau 1 ne clique pas sur une ligne de ce menu sachant qu'un clic implique l'appel d'une fonction du SAM. De même, le déplacement d'une fenêtre est une action car cela peut avoir des conséquences environnementales. Le suivi des actions de l'utilisateur se traduit alors par la boucle suivante :

```
while true {
set étatAS 0
action_utilisateur
if resultat-action_utilisateur {
set étatAS 1
action_agents
} ;
} ;
```

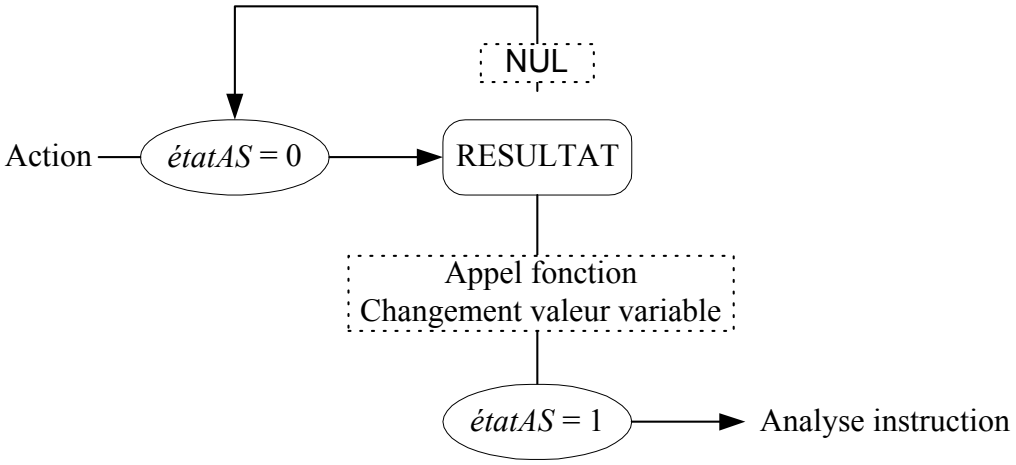


Figure 6-1 : Scrutation d'une instruction

La scrutation des événements-actions se traduit par un changement d'état de l'AS : 1 si actif ou 0 en attente. La variable *étatAS* est un booléen qui indique aux agents de l'ECLR, en particulier à l'AS qu'ils doivent s'activer dès qu'une instruction est identifiée. A la fin de l'analyse de cette instruction, ce booléen revient à son état initial. L'identification de l'instruction est faite alors par l'agent superviseur AS.

La variable *résultat* comme le montre la figure 6-1, correspond à un appel de fonction ou à un changement de valeur d'une variable. En effet, ce sont les seuls cas de figures où la résultante d'une action peut avoir des conséquences sur l'environnement et sur l'évolution de la conception d'un logiciel.

6.1.2. Analyse d'une instruction

L'analyse d'une instruction consiste à vérifier la validité et la logique de l'action de l'utilisateur. Une illustration d'un problème de logique sur une action de l'auteur de niveau 1 est la création d'un bouton qui n'a pas de nom dans une fenêtre. Cette exemple montre que l'analyse peut parfois aider à l'anticipation des actions de l'utilisateur car s'il crée un bouton, c'est qu'il a l'intention de faire appel à celui-ci et attribuer aucun nom pour l'identifier est illogique.

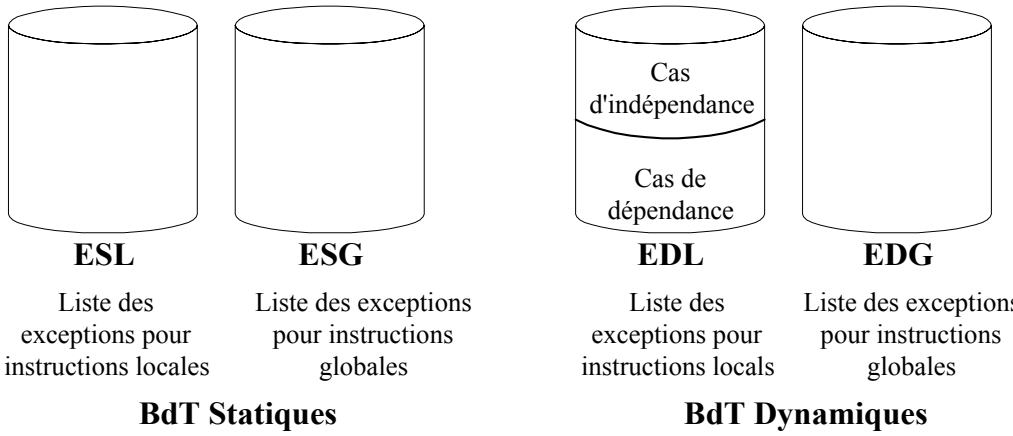


Figure 6-2 : Les BdT constituant la liste des exceptions

Nous reposons notre analyse sur deux dimensions :

- Niveau local pour vérifier la validité de l'instruction par rapport à la liste des exceptions statiques et dynamiques locaux.
- Niveau global pour vérifier la cohérence de l'instruction par rapport aux instructions précédentes et par rapport à la liste des exceptions statiques et dynamiques globaux.

Nous obtenons ainsi les vues verticales et horizontales décrites dans le chapitre 3.2 pour analyser chaque erreur éventuelle.

6.1.2.1.Analyse locale d'une instruction

Pendant cette phase, l'ECLR vérifie si le résultat de l'action de l'auteur de niveau 1 existe dans les listes statiques et dynamiques des exceptions locales. Au cas où il y analogie entre ce résultat et l'une des exceptions, l'ECLR cherche à travers la fonction *Assistance* un moyen de le corriger et émet un message sur la correction faite ou à défaut un message qui aide et oriente l'auteur de niveau 1 grâce à la mémorisation des actions correctrices faites précédemment. Sinon, l'ECLR mémorise grâce à la fonction *Apprentissage-correction*, les actions associées à l'erreur à travers la *fonction Apprentissage-erreur* et affiche un message détaillant la nouvelle erreur. Le processus est décrit de la manière suivante :

```
proc Analyse-locale (résultat) {
if ($etatAS == 1) {
foreach exception $ESL && foreach exception $EDL
if (compare $exception $résultat == true) {
Assistance $exception $résultat
puts stdout « Message erreur + correction»
if (appliquer-correction == false)
```

```
Apprentissage-correction
} ;
else {
puts stdout « Message erreur »
Apprentissage-erreur
Apprentissage-correction
} ;
} ;
Analyse-globale $résultat
} ;
}
```

Chaque action correctrice faite par l'auteur de niveau 1 suite au message de l'ECLR est mémorisée dans la BdT interne. Au cas où la même erreur se reproduit, l'outil rappelle dans son message en plus de la description de l'erreur détectée la correction associée. Comme nous l'avons précisé dans les chapitres précédents, l'auteur de niveau 1 est maître d'œuvre de son logiciel ce qui signifie qu'il peut appliquer ce qui est décrit dans le message ou adopter une autre action correctrice. Dans ce dernier cas, l'ECLR mémorise de nouveau cette action en plus de l'ancienne. Ainsi, la base d'apprentissage évolue et enrichit les choix proposés par l'ECLR. Au niveau opérationnel, cette partie d'analyse de notre ECLR se présente sous la forme suivante :

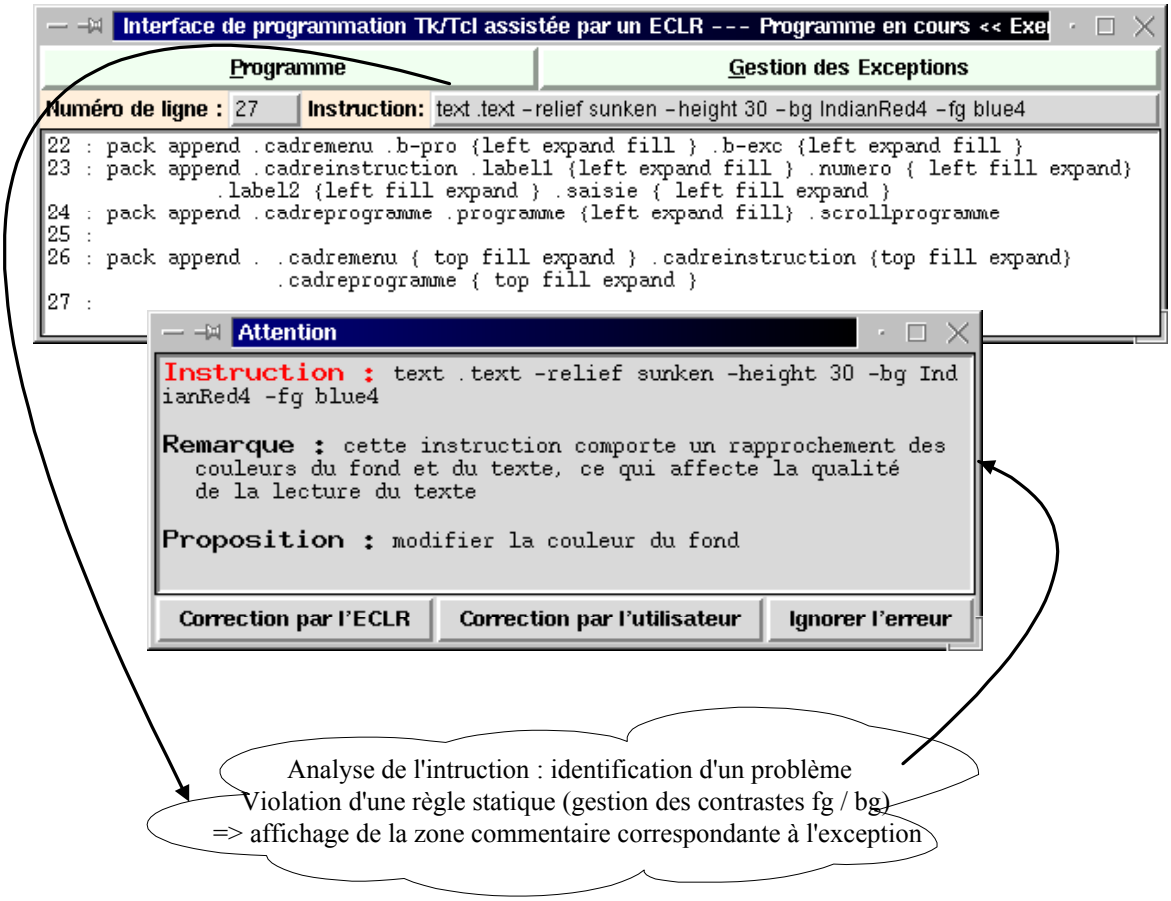


Figure 6-3 : Application de la partie analyse locale d'une instruction

6.1.2.2.Suivi relationnel des instructions

Pour effectuer un contrôle de cohérence d'une nouvelle instruction par rapport aux anciennes instructions, il est primordial de déterminer les relations de dépendance et d'indépendance entre elles. Cela permet de valider l'usage des exceptions par rapport au contexte mais également de vérifier la cohérence de l'environnement contextuel. Ces relations sont définies de la manière suivante :

- Relation de dépendance entre les instructions :

Soient I_x et I_y deux instructions, et R_d la relation de dépendance entre les instructions.

I_x R_d I_y signifie que l'instruction I_x est fonctionnellement dépendante de I_y. En d'autres termes, les résultats en sortie de l'instruction I_x sont des entrées ou des ressources requises pour l'instruction I_y.

- Relation d'indépendance entre les instructions :

Soient I_x et I_y deux instructions, et R_i la relation d'indépendance entre les instructions.

I_x R_i I_y signifie que l'instruction I_x est fonctionnellement indépendante de I_y. En d'autres termes, les résultats en sortie de l'instruction I_x n'ont aucune relation ni influence sur le fonctionnement de l'instruction I_y. Il existe alors un paramètre commun aux deux instructions. Voici un exemple de cas de dépendance :

set x 12

set y x

Pour que y existe, il faut que x ait une valeur auparavant.

Une fois les relations de dépendance vérifiées, il faut procéder aux contrôles des exceptions globales.

```
proc Analyse-globale (résultat) {
foreach instruction $liste-instruction {
if (Ix Rd Iy)
foreach exception-dépendante $ESG && foreach exception-dépendante $EDG
if (compare $exception-dépendante $résultat == true) {
Assistance $exception-dépendance $résultat
puts stdout « Message erreur + correction »
if (appliquer-correction == false)
Apprentissage-correction
} ;
else {
puts stdout « Message erreur »
Apprentissage-erreur
Apprentissage-correction
} ;
if (Ix Ri Iy)
foreach exception-indépendante $ESG && foreach exception-indépendante $EDG
if (compare $exception-indépendante $résultat == true) {
Assistance $exception-indépendance $résultat
puts stdout « Message erreur + correction »
if (appliquer-correction == false)
Apprentissage-correction
} ;
else {
puts stdout « Message erreur »
Apprentissage-erreur
Apprentissage-correction
} ;

} ;
} ;
```

Le script de la fonction *Analyse-globale* ci-dessus peut être illustré par l'exemple qui suit.
Soient les deux exceptions suivantes :

- Exception-1 : Il est interdit de créer deux fenêtres qui se superposent.
- Exception-2 : Une sous-fenêtre doit être affichée dans le périmètre de la fenêtre mère.

Nous remarquons que ces deux exceptions sont contradictoires et dans les deux cas, la création d'une seconde fenêtre entraînera une erreur. En y associant des relations de dépendance aux exceptions pour définir le contexte d'application, une seule exception est applicable. La liste ci-dessus devient alors :

- Exception-1 applicable si deux instructions sont indépendantes -> Exception-indépendance-1
- Exception-2 applicable si deux instructions sont dépendantes. La seconde fenêtre est en relation directe et n'existe qu'à travers la première.

Ainsi, l'ambiguïté est levée et le contexte reste cohérent.

6.2. MISE EN ŒUVRE DES COMPOSANTS

Dans le développement des composants, nous nous gardons de faire une conception trop mécaniste et systémique [Gouardères, 98] car l'ECLR est un système socio-technique complexe où il faut faire face à des facteurs d'erreurs humaines et techniques et où l'acteur humain doit être le suprême décideur dans toutes actions et prises de décisions. Les composants actifs de l'ECLR sont les agents simples et les agents décisionnaires. Mais quelle que soit sa qualification, un agent est une entité autonome dans ses fonctions et éventuellement ses prises de décision. Cette autonomie offre à l'agent un certain degré de liberté dans ses actions mais introduit également une dynamique qui complexifie le maintien de la cohérence et de l'efficacité du SMA [Foisel, 98].

Un agent de l'ECLR ne possède pas certaines caractéristiques développées dans différents SMA telles que [Huang &al, 1999] :

- La notion de durée de vie d'un agent : dans l'ECLR un agent est créé et opérationnel définitivement et ne peut s'autodétruire.
- Le transfert d'un agent d'un module vers un autre n'existe pas dans le sens où nous ne voyons pas un intérêt quelconque pour implémenter la notion de mobilité des agents dans l'ECLR.

6.2.1. Description des agents simples

L'agent est l'entité de base de toute approche multi-agents. Il est caractérisé par son autonomie de décision et sa recherche de satisfaction [Ferber, 97]. Un SMA comporte de nombreux agents ayant des interactions mutuelles ou environnementales [Camps &al, 98]. Ces définitions que nous adoptons pour l'outil ECLR nous amènent à décrire un agent simple par la structure suivante :

Paramètres	**Définitions**
Id-agent	Le nom identifiant chaque agent simple
Compétences-individuelles	Rôle et fonction de l'agent
Compétences-sociales	Relation entre l'agent simple et les autres agents du même module
Id-module	Nom du module d'appartenance
Info-in	Informations utilisées par l'agent
Info-out	Résultats émanant de son activité
Priorité	Niveau de priorité de l'agent simple (haute/basse)

Figure 6-4 : Structure d'un agent simple

Comme le montre la structure d'un agent simple ci-dessus, il est constitué de plusieurs paramètres définissant les services qu'il offre. On distingue deux types de services : externes et internes. Comme nous l'avons vu dans le chapitre 5.1.2 sur le modèle cognitif de fonctionnement d'un agent, ces services sont la concrétisation des comportements empiriques et sémantiques. Nous considérons alors les compétences-individuelles comme étant des services internes et les compétences-sociales comme représentant les services externes. Ces derniers sont les résultats des différentes visions de la coopération [Abchiche, 99] où coopérer signifie : adopter une attitude, contrôler, planifier, conceptualiser, engager, partager des résultats et des tâches, partager les modèles mentaux. La différence de notre modèle avec les SMA existants vient surtout du fait que la coopération n'a lieu qu'entre agents simples et agents décisionnaires comme nous l'avons décrit dans le chapitre 5.2.2.

Exemple d'application de cette structure générique des agents simples au cas d'un agent collecteur :

Id-agent	AC
Compétences-individuelles	Collecte d'informations
Compétences-sociales	Rapport d'activité à l'AD et mise à jour BdT

Id-module	Détection
Info-in	Données entrées par l'utilisateur
Info-out	Informations utiles résultant du filtre des données utilisateur
Priorité	Basse

6.2.2. Implémentation des agents décisionnaires

Les agents décideurs possèdent différentes activités :

- Activité coopérative : ils favorisent la poursuite des actions des agents simples. Il existe alors des échanges mutuels bénéfiques entre les agents.
- Activité antinomique : les agents décisionnaires empêchent la poursuite de l'activité suite à une incompatibilité quelconque.
- Activité indifférente : les agents décisionnaires laissent libres les agents simples si leurs activités n'ont aucune conséquence sur les autres.

Etant donné la faculté des agents décisionnaires à opter pour une solution par rapport à d'autres, nous allons les décrire un par un afin de mettre en évidence leurs activités coopératives, antinomiques et indifférentes.

6.2.2.1.Agents décideurs

Dans l'architecture en couche de la figure 5-10, nous avons montré qu'il y a un agent décideur pour gérer et contrôler chaque couche ou module. Cela s'explique par le fait que chaque couche a une fonction bien déterminée et est composée d'agents simples qui peuvent exister ou non dans les autres couches. Les fonctions à assurer par l'AD nous amène à définir la structure suivante :

Paramètres	**Définitions**
Id-agent	AD
Id-module	Nom du module d'appartenance
Info-in	Informations reçues des agents simples du module
Info-out	Informations envoyées vers les agents simples ou l'AS
Rôle	Description des fonctions assurées

Figure 6-5 : Structure d'un agent décideur

En ce qui concerne la couche *Détection*, le rôle de l'AD est de vérifier dans les propositions des agents simples de priorité hautes, en l'occurrence l'ACO et l'ACP s'ils ont détecté une ou plusieurs erreurs, et ensuite d'informer l'AS de ses conclusions. En cas de présence d'erreurs, l'AD fournit à la couche au dessus les informations relatives à ces erreurs et envoye ces mêmes informations à l'AS afin que ce dernier mette à jour les BdT.

```
Proc AD-Détection {
set detection-erreur résultats-ACO-ACP
if ($detection-erreur == null) {
information AS
break
} ;
else {
information AS
envoi-infos AS
echange-infos Solution $propositions
} ;
} ;
```

L'AD de la couche *Solution* possède en sus de ses fonctions d'échanges d'informations avec la couche *Présentation* et ses obligations d'envoyer des informations pour mettre à jour les BdT, un rôle particulier : la prise de décision en cas de conflit de résolution.

```
Proc AD-Solution {
set proposition1 résultats-ACO
set proposition2 résultats-ACP
if ($proposition1 $proposition2 compatibles) {
action-corrective $proposition1 $proposition2
consitution-message $proposition1 $proposition2
information AS
} ;
else {
if (AD-trancher == true) {
action-corrective $proposition1 $proposition2
consitution-message $proposition1 $proposition2
information AS
} ;
else {
retour-infos $proposition1 $proposition2 ACO
retour-infos $proposition1 $proposition2 ACP
information AS
```

```
} ;
} ;
} ;
```

Dans la couche *Présentation*, l'AD peut également avoir à décider entre les messages que proposent les agents ACP et ACE. Comme l'AD de la couche *Solution*, s'il y a conflit entre les propositions des agents simples et que l'AD ne peut trancher, ce dernier renvoie toutes les informations reçues aux agents de priorité hautes afin d'avoir un retraitement croisé, une nouvelle analyse des propositions.

```
Proc AD-Présentation {
set message1 proposition-ACP
set message2 proposition-ACE
if ($message1 $message2 compatible) {
information AS
puts stdout $message1 $message2
} ;
else {
if (AD-trancher == true) {
message = choix $message1 $message2
information AS
puts stdout $message
} ;
else {
retour-infos $message1 $message2 ACE
retour-infos $message1 $message2 ACP
information AS
} ;
} ;
}
```

6.2.2.2.Agent Superviseur

Nous allons décrire ici la constitution des fonctions qui sont prises en charge par l'AS.

```
Proc AS {
identification-instruction
envoi-Détection $instruction
attendre-réponse-Détection
if {no-erreur) {
break
```

```
} ;
else {
MAJ-BdT $détection-erreur
autorisation-suite-action
MAJ-BdT $propositions
autorisation-suite-action
MAJ-BdT $message
} :
} ;
```

La fonction autorisation-suite-action consiste à valider ou à interdire la suite des actions de l'ECLR. En effet, l'AS vérifie la cohérence des informations reçues de chaque couche avec les BdT et les résultats des actions relatives aux instructions précédentes. Grâce à l'apprentissage de l'ECLR, l'AS peut décider de demander à une couche de revoir ses actions si elles sont en contradiction avec les anciennes propositions.

6.2.3. Structure des bases de travail

L'élément perçu, émis ou reçu par l'agent est la donnée. Le résultat de l'interprétation de la donnée est l'information utile par rapport à l'agent. Ainsi, donnée et information sont les deux composantes émergentes du processus cognitif d'un agent [Prince, 98]. Nous pouvons déterminer cinq processus pour gérer les bases de travail [Kolski, 93] :

- L'*encodage* consistant à capter l'information de l'extérieur et à la transférer vers la BdT externe.
- Le *stockage* consistant à créer des représentations des informations dans la BdT externe afin d'être utilisables par les différents agents des différents modules.
- La *récupération* des données de la BdT externe vers la BdT interne.
- L'appariement ou « *matching* » consistant à comparer les BdT afin de respecter la liste des exceptions.
- La *remise à zéro* de la BdT externe pour marquer la fin d'un traitement par l'ECLR.

Rappelons qu'il existe deux types de liste des exceptions :

- Une liste statique qui regroupe les exceptions définies et fixées dans l'ECLR. Elle constitue alors les premières données de la base de travail interne.
- Une liste dynamique qui regroupe les exceptions définies par l'auteur de niveau 1, et les exceptions apprises par les agents de l'ECLR durant la conception.

Dans chaque liste, il y deux niveaux :

- Niveau local où sont stockées les informations nécessaires au fonctionnement interne de l'agent.
- Niveau global où sont regroupées les informations utiles pour les interactions ou échanges entre agents. Généralement les données non techniques permettant aux agents de se positionner socialement et d'affirmer leurs rôles dans le système. Ces données structurent en grande partie le fonctionnement de l'ECLR.

Pour prendre en considération toutes ces informations, nous allons définir une structure générique applicable à toutes les listes des exceptions :

Exception	Nom identifiant l'exception et définissant sa sémantique ou son rôle
But	Description informelle de l'objectif de l'exception
Niveau-Expertise	Niveau d'application de cette exception : • Locale ou globale • Dépendant ou Independant
Classe-exception	Prédicat de sélection des exceptions par rapport aux informations utiles [53]
Paramètres	Paramètres associés à l'exécution et/ou résultats de la requête utilisateur
Pré-conditions	Prédicat exprimant les conditions avant l'exécution de la requête
Post-conditions	Prédicat exprimant les contraintes à la sortie
Action-correctrice	Actions entreprises pour corriger l'exception
Agents	Liste des agents pouvant utiliser cette exception

Figure 6-6 : Structure générique des listes des exceptions dans la BdT Interne

Exemple sur une exception qui gère le nombre de fenêtres proposées dans le logiciel final :

Exception	Nombre-fenêtre
But	Description informelle de l'objectif de l'exception
Expertise	Globale – Dépendant
Classe-exception	Fenêtre
Paramètres	Sous-fenêtre (taille plus petite que la fenêtre parent)
Pré-conditions	Nombre-fenêtre > 0

[53] Voir chapitre 5-3-1

Post-conditions	Nombre-fenêtre < 6
Action-correctrice	
Agents	AC, AAI, AAE

Pour l'analyse de la demande l'agent X doit reconstruire le contexte à partir des données stockées dans sa base de travail. Ce contexte lui permet de mieux comprendre l'événement et par conséquent de maîtriser ses réactions et ses propositions au superviseur. La base de connaissance correspond alors à un environnement cognitif.

6.3. IMPLEMENTATION DES MODULES

L'ECLR est un outil où chaque acteur apporte et reçoit des connaissances et des compétences. Le transfert de savoir-faire s'applique entre les agents d'une même couche et signifie également une communication contrôlée entre les couches. Cela est particulièrement valable lorsqu'une couche ne sait pas réaliser une action. Dans ce cas, le contrôle est fait par l'AS qui doit prendre une décision face à ce problème. A titre d'exemple, l'utilisateur de niveau 1 demande de lancer un fichier « mpeg » dans une partie de son logiciel final. La couche *Solution* reçoit des données relatives à cette requête depuis la couche *Détection* . Or le système ne dispose pas des ressources matérielles requises. L'AD du module qui constate cette anomalie délègue alors cette tâche à l'AS. Le savoir-faire de ce dernier peut se traduire à ce moment-là, par une prise de décision pour afficher un message explicatif de la situation à l'utilisateur, ou par une nouvelle requête à la couche *Détection* pour effectuer une nouvelle démarche.

Ainsi, la mise en œuvre de l'ECLR nécessite plusieurs niveaux de cohérence à respecter :

1. Avec l'environnement de l'utilisateur,
2. Avec les autres applications actives,
3. Entre les couches de l'ECLR.

Le point commun en terme de fonctionnement entre les trois couches vient du côté pragmatique de chaque couche car il y a systématiquement un retour de chaque action effectuée auprès de l'AS.

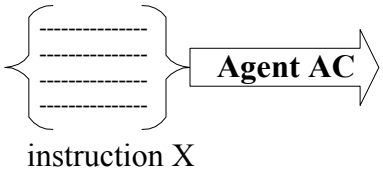
6.3.1. Implémentation de la couche *Détection*

La couche *Détection* est la base de l'outil ECLR. En effet, comme nous l'avons décrit dans le chapitre 5, la fonction principale de cette couche consiste à analyser chaque instruction de l'auteur de niveau 1 pour minimiser la gravité et le nombre de failles possibles dans la conception de son logiciel.

Le changement d'état du booléen *EtatAS* déclenche l'activité de la couche *Détection*. Cette dernière commence par l'extraction des informations utiles à partir de l'instruction en cours de traitement. Cette tâche est primordiale car les agents des différentes couches de l'ECLR utiliseront par la suite ces informations en sus de celles prédéfinies dans la BdT interne, en l'occurrence les listes des exceptions, pour effectuer leurs fonctions.

L'extraction des informations utiles est réalisée par l'agent collecteur AC (voir figure 5-11) immédiatement après l'identification d'une instruction. L'agent AC copie les informations utiles extraites d'une instruction dans la BdT externe afin que l'agent analyseur externe AAE puisse les utiliser en cas de besoin.

Notre recherche sur la composition finale d'un logiciel multimédia nous conduit à fixer les informations utiles suivantes :



Classes	Attributs
Variable	Nom, type, valeur
Fenêtre	Nom, position, taille, bg, fg, scroller
Texte	Fonte, police, couleur
Fichier	Nom, type, path, taille
Ressource-matériel	Nom, type, paramètres
Bouton	Nom, action, taille, position
Menu	Nom, liste-procédure
Procédure	Nom, paramètres, valeur-retour

Figure 6-7 : Informations utiles constituant la BdT externe

Une instruction est alors composée d'une ou plusieurs informations utiles ayant le format :

<classe-info-utile, attributs-info-utile>

Afin de différencier les anciennes instructions de celle en cours de traitement, nous allons utiliser le format <classe-info-utile-T, attributs-info-utile-T> avec le sigle T pour temporaire.

Quand l'AC a fini de collecter les informations utiles, ces dernières sont comparées aux contenus des listes des exceptions de la BdT interne. Cette comparaison est initiée par les agents de priorité haute qui demandent aux agents AAI et AAE de sélectionner les classes qui sont similaires à celles recensées dans l'instruction en cours de traitement.

```
Proc AAI-Détection (classe-info-utile-T) {
foreach $exception
if $classe-exception == $classe-info-utile-T
return-to-ACO $exception
} ;
```

Le rôle de l'agent AAI consiste à fournir aux agents ACO et ACP la liste de toutes les exceptions de la BdT interne dont les classes sont similaires à celles de l'instruction en cours de traitement.

L'agent AAE peut également être amené à fournir des données supplémentaires venant d'une autre instruction aux agents ACO et ACP. En effet, dans la BdT externe sont regroupées toutes les classes qui ont été extraites de toutes les instructions qui composent le logiciel final. Ainsi, les valeurs des attributs d'une classe peuvent être mises à jour par ceux d'une nouvelle instruction après sa validation. Nous pouvons penser par exemple aux changements résultant d'une nouvelle action de l'auteur de niveau 1 et entrâinant les modifications des coordonnées d'une fenêtre ayant le même attribut « nom ».

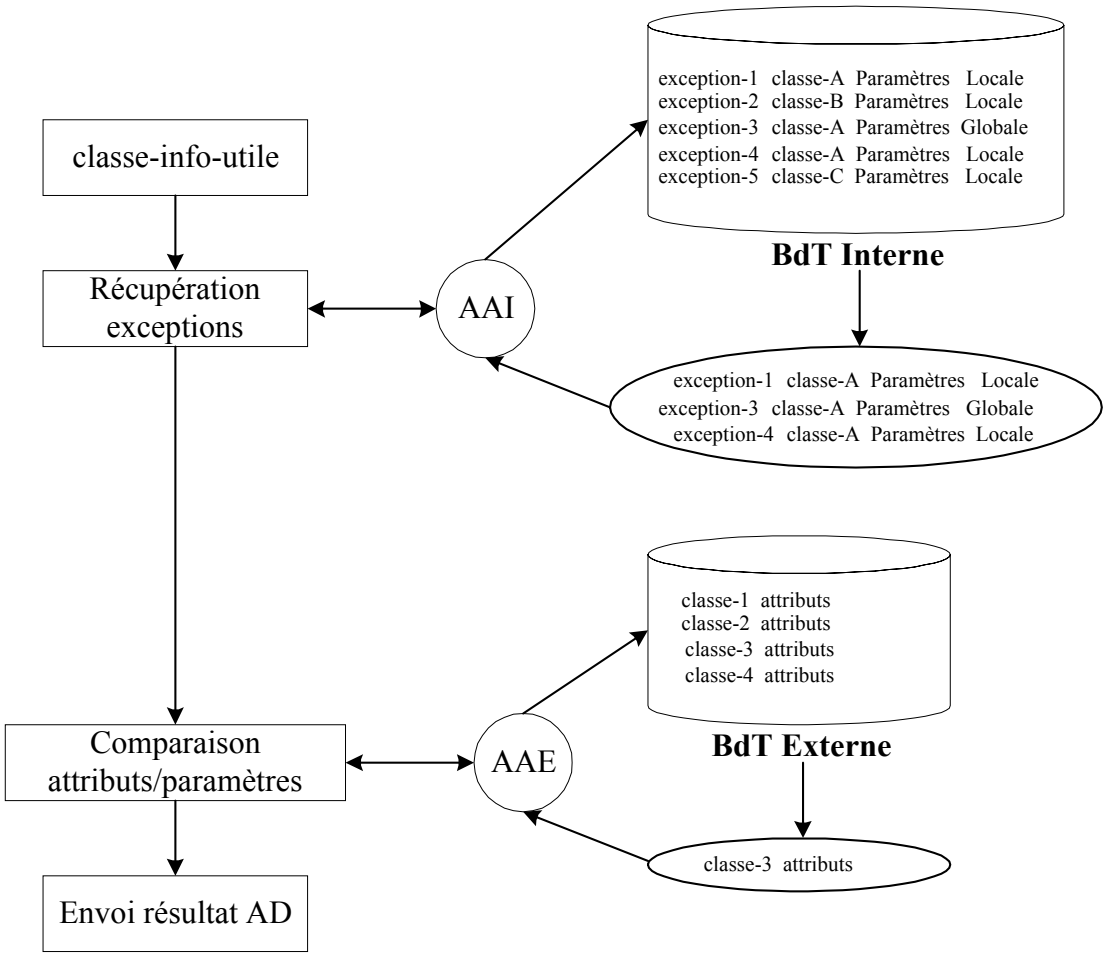


Figure 6-8 : Fonctions agents ACO / ACP au niveau de la couche *Détection*

Les procédures ACO et ACP apparaissent à première vue identiques. En réalité, l'ACO et l'ACP n'utilisent pas les mêmes attributs lors de la comparaison étant donné la différence de leurs objectifs. En effet, l'ACO compare uniquement les valeurs pouvant avoir des conséquences sur l'environnement et l'évolution du logiciel tandis que l'ACP rapproche le contenu de chaque information utile aux données pouvant influer sur la qualité pédagogique et ergonomique du logiciel final.

```
Proc ACO-Détection (classe-info-utile-T){
set liste (AAI-Détection $classe-info-utile-T)
foreach exception $liste {
compare(paramètres, attributs-info-utile-T
return-resultat AD-Détection
} ;
} ;
```

Pour illustrer la différence entre les procédures *ACP-Détection et ACO-Détection*, nous allons continuer sur l'exemple de création de fenêtre. Lorsque l'auteur de niveau 1 crée une fenêtre,

l'ECLR considère cette action comme une instruction. L'AC regroupe alors les données relatives à cette fenêtre dans la BdT externe sous la forme <fenêtre, attributs>. Parmi les attributs de la classe fenêtre, l'ACO se focalisera plus sur le fait que la taille de la fenêtre ne dépasse pas de l'écran, donc sur les effets sur l'environnement alors que l'ACP vérifiera la disposition et les couleurs du fond et les autres repères ergonomiques de la fenêtre.

6.3.2. Concrétisation de la couche *Solution*

La couche *Solution* récupère de l'AD de la couche *Détection* les résultats de l'analyse de l'instruction uniquement lorsqu'une erreur est détectée. Deux cas peuvent se présenter :

- Si l'erreur est connue par l'ECLR, la couche *Solution* extraits de l'exception le paramètre Action-correctrice.
- Si l'erreur peut être corrigée automatiquement, tel le dépassement des coordonnées d'une fenêtre par rapport à la taille de l'écran, la couche *Solution* envoie à la couche suivante un message d'information. Si ce dernier est validé par l'auteur de niveau 1, l'ECLR effectue automatiquement la correction. En revanche si le concepteur accomplit une correction différente de celle proposée dans le message, l'outil met à jour le contenu du paramètre Action-correctrice avec la nouvelle correction.
- Si la correction nécessite une validation ou une action supplémentaire du concepteur du nouveau logiciel, la couche *Solution* envoie seulement à la couche *Présentation* le contenu du paramètre Action-correctrice. Il s'agit alors d'une recommandation ou suggestion pour guider l'individu.
- Si aucune correction n'est connue par l'ECLR, un message descriptif sur l'erreur est envoyé à la couche *Présentation*. L'auteur de niveau 1 effectuera ensuite les corrections nécessaires à sa guise. Dans ce cas, l'ECLR accomplit sa fonction d'apprentissage à travers la mise à jour du paramètre Action-correctrice de l'exception concernée.

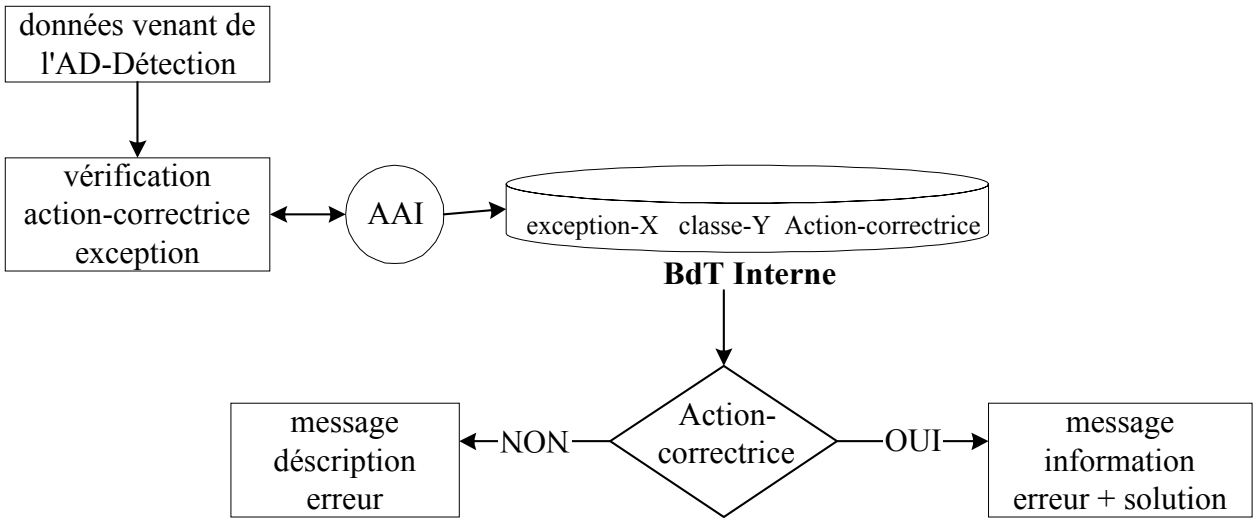


Figure 6-9 : Fonctions agents ACO / ACP au niveau de la couche *Solution*

En cas d'absence de solution sur une erreur détectée, seul l'ACP prend en charge la composition du message qui doit être pédagogique. Pour ce faire, l'ACP inclut dans le message la différence de valeurs entre les attributs de la classe qui pose problème.

Si une correction est proposée dans l'exception, l'ACO intervient afin de vérifier que la modification présentée n'ait aucun effet négatif sur l'environnement. Pour cela, l'ACO demande à l'AAE de vérifier si l'information utile est existante. Si tel est le cas, l'ACO compare qu'il n'y ait pas de contradiction dans les valeurs des paramètres. Si les valeurs sont différentes, l'ECLR met à jour l'information utile après la correction de l'utilisateur et supprime l'information utile temporaire (en cours de traitement).

```
Proc Solution (exception){
If($action-correctrice != NULL)
ACO-Solution $action-correctrice
else
ACP-Solution $exception $attributs-info-utile-T
} ;
Proc ACP-Solution (exception, attributs-info-utile) {
envoi-message AD-Solution $classe-info-utile-T $attributs-info-utile-T $paramètres
} ;
Proc ACO-Solution $action-correctrice {
If(AAE-Solution $classe-info-utile-T == true) {
If((comparer $attributs-info-utile $attributs-info-utile-T) != true)
envoi-message AD-Solution $attributs-info-utile-T $attributs-info-utile
else
envoi-message AD-Solution
} ;
} ;
```

6.3.3. Mise en œuvre de la couche *Présentation*

Dans le but de favoriser l'identification, la compréhension, la mémorisation ou encore la hiérarchisation des informations, il est utile de structurer l'écran et d'organiser les données selon des formats d'affichage. Ce sont les objectifs de la couche *Présentation.* Elle met en œuvre les agents ACP et ACE pour valider et afficher chaque message venant de la couche *Solution* à l'auteur de niveau 1. L'agent contrôleur ergonomique ACE est important car il uniformisera la présentation d'un message selon qu'il y ait confirmation (OUI sur l'organigramme de la figure 6-9) ou infirmation (NON sur l'organigramme de la figure 6-9) sur une action ou sur une solution.

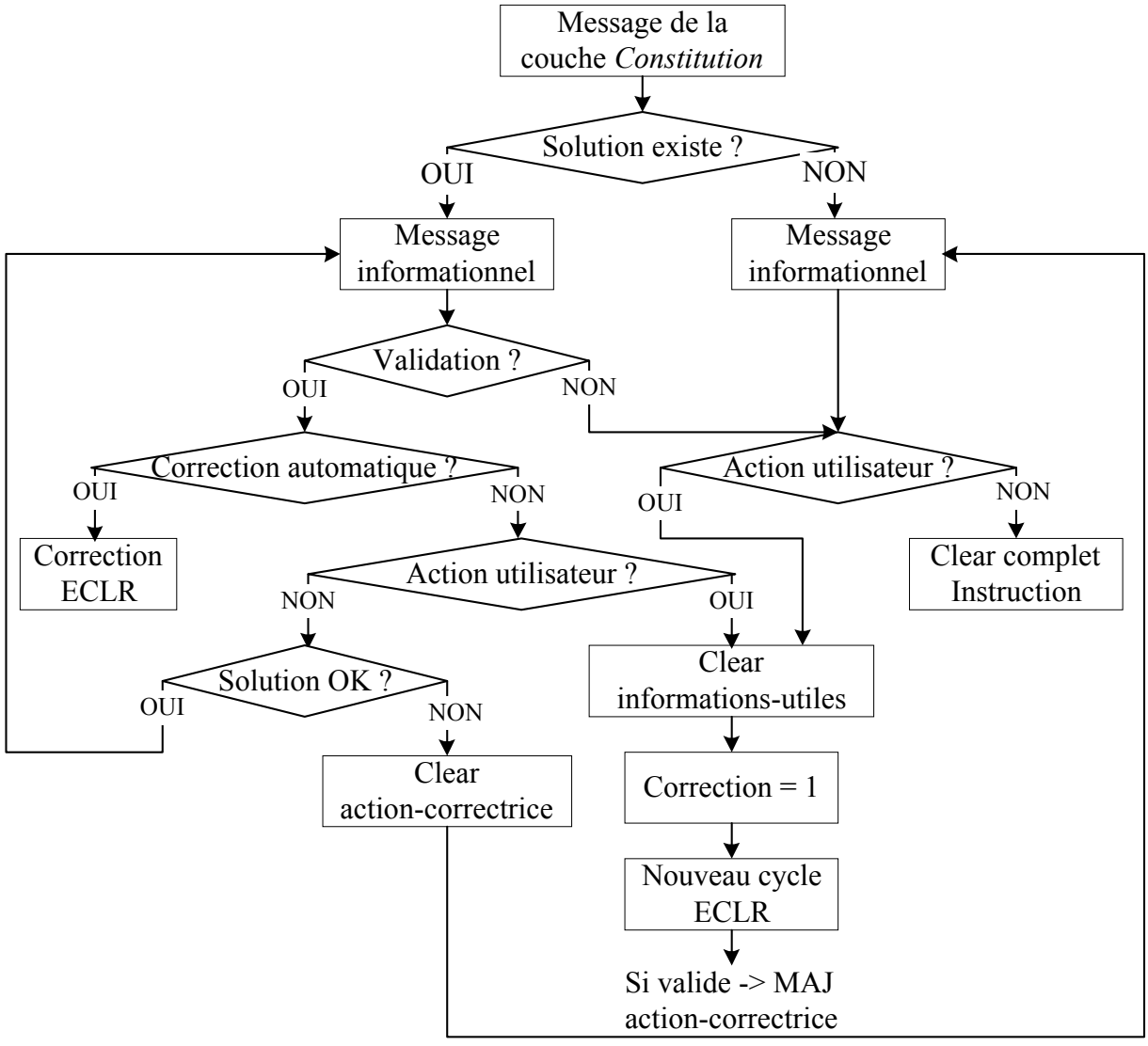


Figure 6-10 : Fonctions agents ACO / ACP au niveau de la couche *Présentation*

L'agent AD de la couche *Présentation* reçoit de la couche *Solution* un message qui lui indique l'existence ou non d'une solution à une erreur détectée par le module *Détection* de notre outil.
Si la solution est connue par l'outil, c'est-à-dire que le champs *Action-correctrice* de l'exception est non vide, alors l'agent contrôleur ergonomique ACE et l'agent contrôleur pédagogique ACP constituent le message pour informer l'auteur de niveau 1. Comme décrit dans le chapitre précédent, ce message présentera le détail de l'action correctrice que l'auteur de niveau 1 approuve par la validation du message ou non.
Si ce dernier est accepté par l'utilisateur et si la correction peut être automatiquement effectuée par l'ECLR, tel le redimensionnement d'une fenêtre par rapport à la taille de l'écran, alors un message est envoyé à l'AS. Ce dernier réalise alors la correction, réinitialise les informations utiles de l'instruction en cours et remet la variable *Etat-AS* à 0 afin de signaler que l'outil est prêt à recevoir une nouvelle instruction.
En revanche, si la solution présentée et acceptée par l'auteur de niveau 1 nécessite une action supplémentaire de ce dernier, alors les informations utiles remises en cause sont réinitialisées, le booléen *Correction* est mis à 1 afin que la nouvelle action qui sera considérée comme une instruction par l'ECLR soit traitée en suivant le cycle normal : *Détection –> Solution –> Présentation*.

Le fait d'avoir le booléen *Correction* à 1 permet à la couche *Détection* de demander à l'agent AS de mettre à jour ou de compléter le champs *Action-Correctrice* si l'information utile ne présente aucun problème. Il y a alors apprentissage de l'ECLR de cette action correctrice.
Si la solution proposée n'est pas acceptée par l'auteur de niveau 1, alors l'ECLR se comporte comme si le champ *Action-correctrice* est vide donc pas de solution. Dans ce cas, si aucune action n'est faite par l'utilisateur, l'ECLR réinitialise les informations utiles relatives à l'instruction en cours ainsi que le booléen *Etat-AS*. En revanche, si l'utilisateur entame une action pour corriger le problème détecté, les informations utiles remises en cause sont réinitialisées, le booléen *Correction* est mis à 1 afin que la nouvelle action qui sera considérée comme une instruction par l'ECLR soit traitée en suivant le cycle normal : *Détection –> Solution –> Présentation*.

6.4. IMPLEMENTATION ET MISE EN ŒUVRE DE L'ECLR

6.4.1. Réalisation de l'ECLR

Selon les définitions et concepts décrits dans les chapitres précédents sur l'ECLR, la composante de l'outil que dont nous avons conçu une maquette, doit être considérée comme un environnement de simulation au sens large c'est-à-dire un outil qui intègre des évènements de toutes sorte : le multimédia, les systèmes de menus et de boutons etc.
Pour comprendre comment un outil convient à un ensemble de système (système d'exploitation, SAM, réseaux …), il faut implémenter une infrastructure qui offre des services fiables [Bagchi & al, 98]. Cette infrastructure est composée entre autres, dans notre cas, du SAM utilisé pour développer l'outil.
Nous avons choisi d'implémenter l'ECLR en utilisant le langage Tk/Tcl pour les raisons suivantes :

- Portabilité de cet environnement : les codes Tk/Tcl tournent sur les deux systèmes d'exploitation les plus utilisés actuellement à savoir Unix (toutes les versions dont Linux) et Ms-windows (toutes les versions dont NT).
- Il permet de regrouper différentes applications de natures diverses en une seule, d'ailleurs il est désigné comme étant un langage liant ou « glue language ».
- Tk/Tcl est un langage créé dans le monde universitaire, gratuit et l'évolution des codes est garanti par plusieurs milliers d'informaticiens dans le monde. Les capacités de ce langage ne cessent de progresser depuis sa création par John Ousterhout en 1988.
- Ce langage permet un affichage simplifié des messages et une rentrée conviviale des données externes pour la constitution de la liste des exceptions par l'utilisateur.

Enfin, en optant pour ce langage, nous pouvons expérimenter l'application de l'ECLR sur un SAM qui est le Tk/TCL lui-même. Comme nous l'avons présenté dans le chapitre sur l'état de l'art, TK/TCL fait partie des systèmes auteurs à base de langage générateur de codes. Le fait que ce SAM permette l'intégration des dernières technologies comme le développement HTML ou le traitement d'images animées, nous a motivé davantage à le choisir.

Pendant toute la phase de programmation de notre maquette, nous avons essayé de mettre en valeur le côté utilité et utilisabilité. En effet, dans le cadre de l'architecture globale d'un

système interactif : l'utilité du système est représentée par le noyau fonctionnel du logiciel tandis que l'utilisabilité est véhiculée par les composants de l'interface Homme-Machine [Balbo, 92].

6.4.2. Mise en œuvre et comportement de l'ECLR

Par abus de langage, il arrive souvent que l'on qualifie un problème comme une tâche, exemple le problème de la tour de Hanoï. Dans le cadre de notre étude, nous proposons qu'un problème qualifie une situation, c'est-à-dire la confrontation d'un système cognitif (sujet humain ou agent artificiel) à une tâche donc à une fonction [Hoc, 92]. Dans cette perspective, la construction de la représentation de la tâche dans l'ECLR est appelée *compréhension*, la construction de la procédure, *stratégie de résolution*.

L'attitude des concepteurs d'interfaces homme-machine face à l'erreur humaine qui ne doit pas être vue comme un comportement totalement imprévisible, décrite dans les chapitres 2 et 3, peut être résumée par [Jambon, 96] :

1. L'évaluation du risque d'erreur.
2. Une fois ce risque évalué, le concepteur peut se tourner vers deux approches complémentaires :
   - ➔ *Prévenir*, c'est-à-dire éviter l'occurrence de l'erreur par une conception adéquate de l'interface.
   - ➔ *Guérir*, c'est-à-dire minimiser les conséquences d'une erreur. Pour cela, le concepteur doit permettre à l'utilisateur de détecter son erreur, afin de pouvoir ensuite la corriger.

Dans notre maquette, nous mettons en œuvre ces points : évaluation, prévention et correction de l'erreur.

La figure 6-11 montre la partie évaluation de l'ECLR. En effet, chaque nouvelle action rentrée par l''tilisateur est identifiée en tant que nouvelle instruction. Celle-ci sera ensuite examinée par la couche *Détection* et selon le résultat de l'analyse, les autres modules se chargent de la valider en mettant à jour les bases de travail et en insérant l'instruction dans le programme, ou bien d'assister l'utilisateur à corriger ses erreurs.

En cas de modification d'une instruction déjà existante par l'utilisateur comme le montre la figure 6-12, l'ECLR est capable de vérifier la conformité des changements par rapport aux

autres instructions. Les bases de travail sont mis à jour en conséquence. Ainsi, l'ECLR prévient des erreurs éventuelles d'une part et étudie l'impact du changement sur la suite du programme.

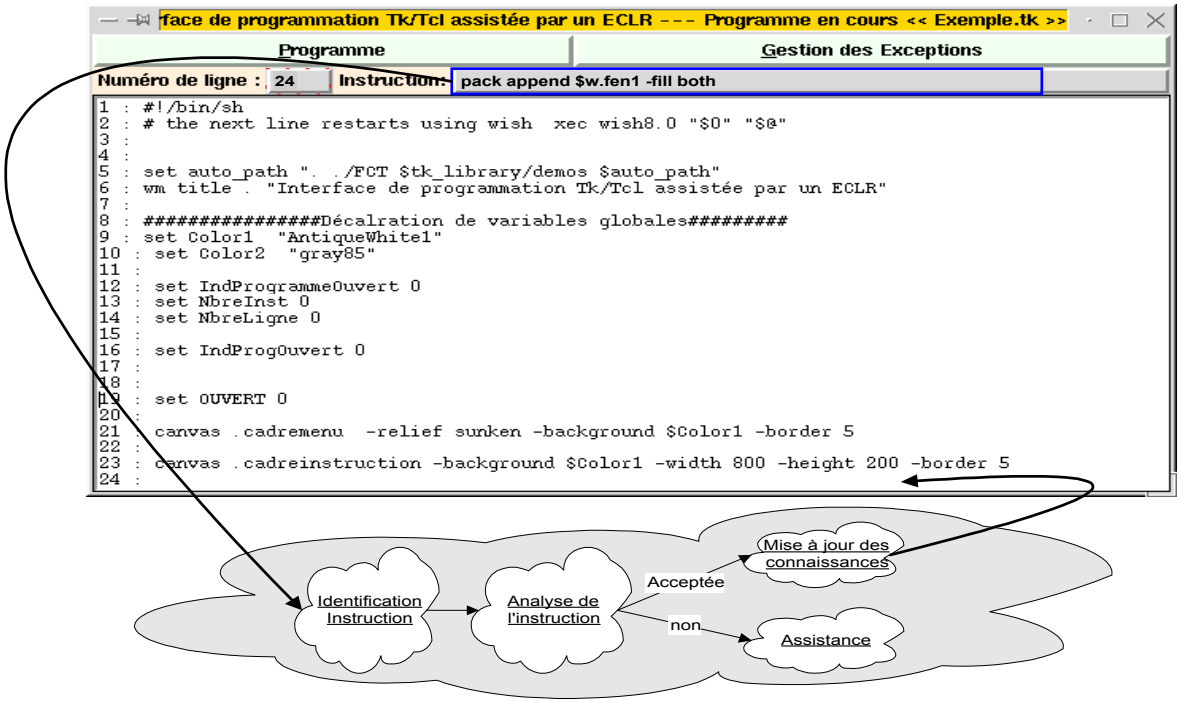


Figure 6-11 : Evaluation et mise à jour des BdT

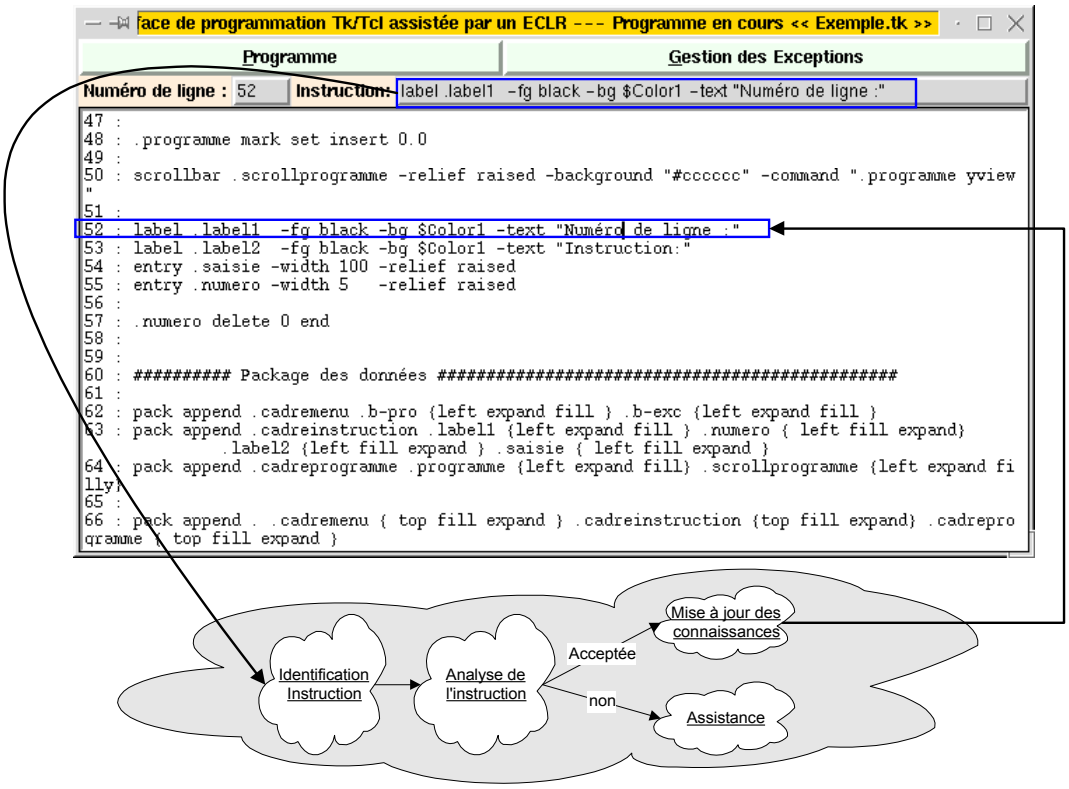


Figure 6-12 : Assistance et prévention de l'erreur

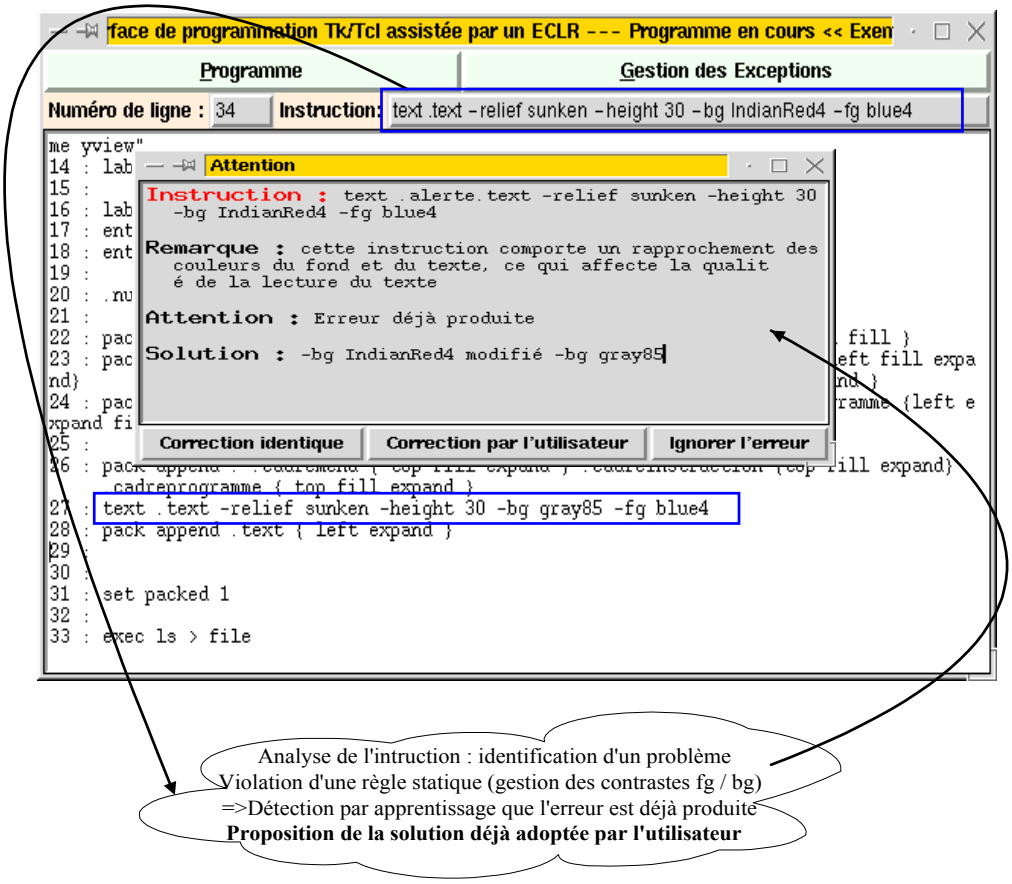


Figure 6-13 : Correction et apprentissage

L'ECLR comme le montre la figure 6-13, nous présente un exemple la violation d'une règle d'exception par l'utilisateur. Après analyse et vérification de la base de connaissances que ce problème a déjà eu lieu, l'ECLR l'ECLR signale l'erreur, la commente, propose la solution déjà retenue et appliquée par l'utilisateur. La décision finale pour appliquer ou non la solution proposée revient toujours à l'utilisateur.
Si l'utilisateur adopte la solution suggerée, l'ECLR corrige automatiquement l'erreur, sinon, l'ECLR apprend la nouvelle solution en mettant à jour sa base de connaissance.

6.4.3. Bilan opérationnel

La résistance aux perturbations de notre ECLR, vient de sa capacité à résoudre un problème par exploration d'un espace d'états en tenant compte des changements qui se produisent et qui entraînent une modification du problème à résoudre.

Nous avons remarqué le caractère opportuniste de notre outil dans le sens où il est en mesure de tirer profit des circonstances opportunes ou imprévues en tout temps, grâce à la mémorisation des changements donc, au maintien de la BdT dynamique. Mais ce côté positif est à nuancer car le contexte d'apprentissage pour une connaissance regroupe les conditions dans lesquelles s'est déroulé cet apprentissage [Frasson et Ramazani, 92].

Grâce à cette méthode d'apprentissage qui permet de mieux prendre en compte la dualité individuelle/collective [Gouardères, 97], du point de vue applicative, nous pouvons remarquer que l'ECLR répond à cinq des huit critères ergonomiques de [Scapin et Bastien, 93] :

- L'adaptabilité qui se réfère à la capacité de l'ECLR à réagir selon le contexte et selon les besoins et préférences de l'utilisateur.
- La gestion des erreurs qui doit réduire les occasions d'erreur car l'ECLR doit détecter le maximum d'erreurs grâce à ses BdT statiques et dynamiques.
- L'homogénéité qui s'apparente à la notion de cohérence (par exemple, séquence de commandes identiques pour un même résultat). L'ECLR grâce à sa faculté d'apprentissage, conserve et propose les mêmes résultats pour des contextes identiques.

- Le guidage qui offre une information rapide et pertinente sur l'état du système. C'est l'ensemble des moyens mis en œuvre par l'ECLR pour conseiller, orienter, informer et guider l'utilisateur lors de ses interactions avec le SAM.
- La compatibilité qui suppose un faible re-codage des informations entre le savoir de l'utilisateur et le format imposé par le logiciel.

Dans la mise en pratique de notre maquette, nous remarquons également que l'ECLR adopte les deux types de stratégie de résolutions décrites par [Hoc, 92] :

- la stratégie prospective qui consiste à générer la procédure en partant des données pour aller vers les résultats, pendant la détection d'une erreur,
- la stratégie rétrospective qui se traduit par une remontée des résultats vers les données, pendant la phase de correction de cette erreur.

Nous ne pouvons prétendre à la complétude avec notre outil, surtout qu'il propose des actions correctrices. En effet, *« l'erreur est commune à tous les hommes »* (Sophocle), et notre maquette d'ECLR n'est que le résultat de notre recherche pour minimiser ces erreurs dans la conception à partir d'un SAM.

6.5. SYNTHESE

Nous avons proposé un outil capable de s'adapter quasi-immédiatement aux changements pouvant surgir lors de la conception grâce à un système à base d'agents réactifs pour la résolution de problèmes. Contrairement au compilateur qui s'active uniquement à l'initiative du programmeur, l'ECLR a l'initiative de vérifier, d'évaluer, de corriger et d'anticiper en temps réel pendant la phase de programmation. Cela est possible grâce aux activités des agents qui le constituent.

Ce SMA est un outil intelligent capable de s'adapter de façon réactive aux évolutions de l'activité d'un collectif de travail en facilitant le parallélisme et l'ordonnancement de certains travaux. La disposition des agents dans notre modèle permet d'assurer les qualités principales d'un didacticiel grâce à la présence dans chaque couche des agents ACP, ACO, AAI et AD[54] avec leurs propres connaissances qu'ils doivent exploiter et enrichir au fur et à mesure. Nous adoptons une vision isomorphe dans le sens où les mêmes agents se retrouvent dans les

[54] Voir leurs définitions dans la partie Descriptif des groupes d'agents.

différents modules de l'ECLR. L'ensemble {agents, environnement, interactions} définit un système complexe qui se traduit par un cycle permanent d'action/réaction [Aknine &al, 98].
Nous constatons que de nombreuses informations ne se transforment pas nécessairement en connaissances pour les agents du SMA ECLR. Tout dépend de la pertinence de leurs qualités cognitives et systémiques. Aussi, l'ECLR est un outil capable d'apprendre par sélection des informations utiles à sa propre évolution.

La méthode de programmation adoptée pour l'ECLR est non bloquante car on n'est pas forcé à chaque action-événement de retourner un résultat ou un message à l'auteur de niveau 1. Rappelons qu'une action-événement ou une instruction est l'actionneur de l'ECLR. Ce dernier est alors actif par induction [Bui, 98] c'est-à-dire qu'il n'est actif que sous l'influence d'un autre acteur actif, en l'occurrence ici l'auteur de niveau 1.
Chaque transition dans l'ECLR provoque une série d'actions à l'entrée de l'état suivant et le démarrage de plusieurs activités. Les actions peuvent être alors classées en deux catégories : celles exécutées pour rentrées dans un nouvel état et celles exécutées avant de quitter l'état courant.

L'hétérogénéité des fonctions des agents rend notre tâche d'implémentation complexe. La simplicité de notre programmation défensive optimise la performance et l'efficacité du résultat attendu d'un ECLR c'est à dire assurer en temps réel la robustesse d'un environnement de création de logiciels. Ainsi, l'ECLR est un système réactif contenant plusieurs processus maintenant une interaction constante avec l'environnement. Il nous offre les avantages suivants :

- bonne lisibilité grâce à la clarté et la simplicité de son fonctionnement
- apprentissage facile grâce au nombre restreint des concepts de base
- aptitude à communiquer dans un langage non informatique
- structure hiérarchique et modulaire du modèle.

Par ailleurs, il laisse l'auteur de niveau 1 complètement libre de ses actes comme revenir sur ses actions (feedback), corriger une erreur à sa convenance, etc. L'ECLR offre par conséquent une ouverture et une dynamique de réflexions et d'actions.

Notre recherche montre également trois principes convergents : la nécessaire aptitude à l'adaptation d'un outil d'aide à la robustesse lorsque son environnement est dynamique,

l'importance de l'organisation et de l'architecture de ses composants, et l'intérêt d'une coopération structurée entre les acteurs humains et artificiels.
A travers cette implémentation de l'outil ECLR, nous espérons avoir aidé tout utilisateur d'un SAM pour écrire un logiciel sans autre souci que son objectif.

7. CONCLUSION GENERALE

La détection et la correction des erreurs ne doivent pas intervenir après l'écriture du programme mais doivent au contraire jouer dans le processus même de sa création. Tel est l'axiome de notre recherche pour trouver un moyen d'assurer la robustesse de l'environnement de conception d'un logiciel à partir d'un système auteur multimédia. Nous avons été motivé tout au long de notre étude par l'aspiration à redonner à l'informatique sa vocation première dans le sens où, cette discipline doit aider l'homme dans ses tâches à travers l'automatisation de certaines fonctions complexes en l'occurrence dans notre cas le « cautionnement » de la robustesse afin d'obtenir un logiciel fiable et robuste. Ce souhait permanent peut être ressenti dans le comportement de l'outil que nous avons développé car l'individu reste maître de ses choix et de ses actes quelle que soit la situation. L'outil se limite à aider de manière interactive le concepteur au niveau de son raisonnement et de son approche.

Pour implémenter la robustesse, nous avons mis en œuvre un outil appelé ECLR ou Environnement de Création de Logiciels Robustes afin d'assister l'individu à concevoir un logiciel final dépourvu d'un maximum d'incohérence syntaxique, sémantique et d'un maximum d'erreurs de conception. L'ECLR maintient un contexte cohérent et fiable grâce à l'analyse systématique de chaque action effectué par le concepteur, et à l'apprentissage des corrections apportées suite à chaque erreur détectée.

Notre outil est basé sur une architecture en couche composée de trois modules de traitement et d'un module qui supervise l'ensemble et qui gère les bases de données nécessaires au fonctionnement de l'outil. Chaque module appelé également couche est doté d'un ou de plusieurs agents de différentes catégories et ayant chacun sa spécificité et son rôle à jouer. Toute communication entre les composants de l'outil (couche, agents, bases de données) est contrôlée suivant des règles bien définies toujours dans le souci d'optimiser la qualité de l'environnement de conception.

Notre outil met alors en œuvre une intelligence collective et fait ressortir une émergence de structure par interactions grâce aux acteurs qui le constituent et son architecture . Ces

principes de base sont : la prédictibilité grâce à l'apprentissage, la simplicité, la cohérence, l'homogénéité et la réversibilité.
Ce document décrit notre recherche pour formaliser l'expertise sur les erreurs de conception pendant la phase de développement ou de la programmation, en se basant sur un système multi-agents que nous avons défini nous-mêmes. L'originalité de notre approche vient du fait que le fonctionnement de l'outil évolue dynamiquement avec le contexte et l'avancée de la conception du nouveau logiciel à partir d'un SAM. Cela est possible grâce à la combinaison des actions et apprentissages faits par chaque acteur. Ainsi, le modèle ECLR est précurseur dans l'intégration de détection et de correction d'erreurs de conception.

7.1. BILAN

Le mélange de différentes technologies à travers le multimédia génère presque systématiquement une dépendance entre les modules applicatifs et l'environnement. L'ECLR est un outil qui évolue grâce à son apprentissage par rétroaction de son environnement. En effet, il résulte d'un modèle proche et extrait du réel grâce aux comportements des agents qui le composent. L'ECLR est le résultat d'une interaction organisationnelle entre plusieurs acteurs. Il s'inscrit ainsi dans la longue lignée de systèmes d'intégration de modèles de raisonnement hétérogènes [Abchiche, 99]. Pour son intégration, nous avons adopté une démarche bi-directionnelle :

- Depuis le côté utilisateur où l'auteur de niveau 1 donne les spécificités des tâches à éviter,
- Depuis les agents où les connaissances de ces derniers permettent de vérifier la robustesse.

L'orientation multi-agents de notre recherche a été motivée par la puissance potentielle qu'offre une communauté d'agents dans un système coopératif lors de la résolution « à vif » d'un problème complexe [Aknine &al, 98]. De même, l'approche SMA offre une forte potentialité pour s'adapter aux changements dynamiques de contexte, donc aux changements intervenant dans une activité en l'occurrence le développement d'un nouveau logiciel dans le cadre de notre étude.

Suite à la concrétisation de notre ECLR, nous constatons que chaque action d'un agent peut donner l'émergence d'un nouvel état du système. De même lorsque l'auteur de niveau 1 va agir sur le système, il y a émergence de nouvelles idées. Cela est en phase avec notre recherche car favoriser l'émergence, c'est favoriser l'apparition d'indicateurs de conception qui permettent les interactions. Tous ces points positifs nous amènent à une nouvelle forme

d'organisation et de comportement des acteurs humains et logiciels. En effet, l'homme et l'outil sont interdépendants dans leur comportement et ont le même objectif en ce qui concerne la robustesse. Le modèle ECLR distingue le concept dynamique d'organisation de celui d'interaction entre les agents. Ce qui permet le fonctionnement de ce système socio-technique complexe.
A travers notre modèle, nous mettons en avant la notion d'autonomie qui implique une certaine liberté à tous les acteurs de l'ECLR à savoir :

- Le concepteur a la liberté de définir une base de travail à travers la liste des exceptions et a la liberté de réagir face à un problème.
- Chaque agent a la liberté d'agir, de décider, et d'évoluer dans le périmètre d'actions qui lui sont autorisées.

Notre modèle en couche ECLR est inspiré de celui des réseaux de télécommunications sachant que les réseaux sont eux-mêmes modélisés à partir d'inspiration sur la communication humaine. L'architecture en couche, la définition logique de l'ECLR ainsi que la gestion des données manipulées permettent de simplifier la complexité de sa mise en œuvre qui consiste en un jumelage de deux applications : le système auteur multimédia et l'ECLR lui-même.
Ce jumelage consiste à activer en parallèle au SAM l'ECLR qui lui est associé spécifiquement afin que ce dernier récupère chaque action de l'utilisateur sur le SAM, l'analyse et le valide en temps réel selon le contexte environnemental.

La simplicité de l'architecture et la flexibilité de ses composants permettent une modification ou une adaptation dynamique du modèle sans faire recours à une réécriture même partielle de son noyau comme c'est le cas de la plupart des programmes de la même catégorie.

Notre étude sur le création d'un environnement de création de logiciels robustes nous a permis de soulever diverses problématiques :

- La diversité des langages et des environnements de conception ne facilitent pas la concrétisation de l'ECLR qui est lié fortement au langage de programmation utilisé.
- La prise de décision lors de conflits par les agents décideurs est parfois longue, ce qui peut affecter la performance de l'outil.
- La gestion dynamique des exceptions rend complexe le maintien du contexte et par conséquent la prévision du comportement de l'ECLR.

- La correction apportée par l'utilisateur suite au message d'alerte qu'une erreur a été detectée peut être relatif à un contexte particulier qui est différent du contexte initial.
- L'extraction des informations utiles suite à une action donnée est complexe et dépend du langage ou du SAM utilisé. En effet, pour que l'ECLR capture et interprète toute action d'un utilisateur sur un SAM, il faut qu'il dispose d'un modèle de fonctionnement du SAM. Or chaque SAM est particulier. Cela implique la nécessité de developper un ECLR spécifique à chaque SAM.

7.2. PERSPECTIVES

L'ECLR est un outil destiné à être mis en œuvre avec un système auteur dans sa définition initiale. Mais son champ d'application peut être étendu à tout type de programme informatique. De même, l'architecture en couche adoptée peut être utilisée par tout domaine nécessitant l'intégration de modèles de raisonnement par apprentissage.
Chaque couche de notre outil raisonne avec ses savoirs propres (agents) et possède des objectifs globaux et locaux selon l'activité qui lui est attribuée. Ces objectifs peuvent être enrichis par différentes techniques. Nous pensons en particulier à la couche *Détection* où l'on peut intégrer un raisonnement hypothético-déductif pour diagnostiquer les problèmes et dysfonctionnements éventuellement commis par l'utilisateur. De même à la couche *Présentation*, nous pouvons enrichir la communication des messages à l'utilisateur par voie orale ou sonore en sus ou en remplacement des messages affichés sous forme textuel ou graphique actuellement.

Par ailleurs, il est possible de décomposer chacune des couches en des sous-couches. Bien que les expériences dans les réseaux protocolaires en télécommunication aient montré que la multiplication des couches peut être un handicap pour un système[55], il est intéressant d'explorer cette voie. Nous pensons que le nombre réduit de couches permet d'obtenir une homogénéité de réaction et de comportement de l'ECLR. Mais une étude dans cette direction pourra mieux faire ressortir les avantages et les inconvénients de notre modèle et permettra d'améliorer l'architecture donc le fonctionnement et la performance de l'ECLR.

[55] Le modèle ISO est critiqué aujourd'hui à cause des redondances et lourdeur fonctionnels de ses 7 couches.

L'inéluctabilité actuelle de l'Internet offre également la perspective d'appliquer le concept de système distribué sur l'ECLR. Ainsi, plusieurs utilisateurs à travers le monde, peuvent simultanément enrichir les bases de travail optimisant ainsi la capacité d'analyse et d'apprentissage de cet outil. Ce nouvel archétype est particulièrement intéressant pour la conception d'une application à grande échelle telle le développement d'un système comme Linux.

Les agents mis en œuvre dans l'ECLR représentent malgré notre effort de leurs attribuer des comportements empiriques, cognitifs et évolutifs, des acteurs réducteurs du réel et du prédit dans le sens où l'analogie entre un automate aussi performant soit-il avec l'être humain est toujours limitée. En effet, un modèle implique toujours un rétrécissement ou un appauvrissement du réel. Il sera alors judicieux d'intégrer l'ingénierie des connaissances afin de mettre en œuvre une cognition socialisée et une coopération active entre les acteurs. Une application de cet angle de projection peut être l'intégration de la reconnaissance automatique du langage ou bien du dialogisme [Ricordel &al, 98] dans l'ECLR. Cela revient à instaurer entre les agents composant l'outil des conversations similaires à celles de hommes. Il s'agit alors d'enrichir et révolutionner les processus de communication entre les agents d'un SMA.

Enfin, nous espérons que nos travaux auront apporté une contribution à la concrétisation de tout type de logiciel selon les attentes des concepteurs et des utilisateurs par rapport à leur domaine d'activité. Bref, nous souhaitons simplement que chaque concepteur puisse cultiver son jardin au mieux dans le meilleur des mondes possibles[56] !

Voltaire « Candide », 1759 France

BIBLIOGRAPHIE

[Abchiche, 99] Nadia Abchiche
« *Elaboration, implémentation et validation d'une approche distribuée pour l'intégration de modèles de raisonnement hétérogènes : application au diagnostic de pannes éléctriques* », Thèse Université Paris 8, janvier 1999.

[Ackerman, 95] E. Ackerman
« *Environnements interactifs : culture de zappeurs ou culture d'auteurs* », Environnements Interactifs d'Apprentissage avec Ordinateur, 4ème Journées EIAO-ENS de Cachan, Tome 2, Editions Eyrolles, Mars 95.

[Aknine &al, 98] Samir Aknine, Suzanne Pinson, Manuel Zacklad
« *Un système d'agents recursifs pour l'aide au travail coopératif* », Journées Francophones d'Intelligence Artificielle et Système Multi-Agents, France 1998.

[Aubord, 91] A. Aubord, B. Ibrahim
« *Interprétation des intentions des concepteurs de didacticiels* », Rapport dépt Informatique, Université de Genève, 1991.

[Bagchi & al, 98] S Baghki, K Whisnant, R. Iyer
« *Chameleon : a software infrastructure for adaptative fault tolerance* », Conference FTCS-28, Munich 1998.

[Bailey, 83] Robert Bailey
« *Human error in computing systems* », Prentice-Hall, New Jersey, 1983.

[Balacheff & al, 93] N. Balacheff, M. Baron, P. Dillenbourg, M. Grandbastien, R. Gras, F. Madaule, P. Mendelsohn, A. Nguyen-Xuan, J-F. Nicaud
« *EIAO : points de vue des disciplines* », Environnements Interactifs d'Apprentissage avec Ordinateur, 3ème Journée EIAO Cachan, Editions Eyrolles, 1993.

[Balacheff, 92] Nicolas Balacheff
« *Exigences épistémologiques des recherches en EIAO* », Génie éducatif N°4&5, p 4-14, Déc 1992.

[Balbo, 94] Sandrine Balbo
« *Evaluation ergonomique des interfaces utilisateurs : un pas vers l'automatisation* », Thèse de doctorat, Université Joseph Fourier, Grenoble, 1994.

[Bastien et Scapin, 93] Bastien C.J. & Scapin D.L
« *Ergonomic criteria for the evaluation of human-computer interfaces* », Rapport technique N°156, INRIA Rocquencourt, Juin 1993.

[Bessière, 88] Christian Bessière
« *Outil logiciel pour l'EAO multimédia - EDP, un éditeur pédagogique pour l'interactivité* », CNRS, 1988.

[Billings, 91] C. E. Billings
« *Human-centered aircraft automation : A concept and guidelines* », NASA, Technical Memorandum 103885, Août 1991.

[Blanc, 96] F. Blanc
« *Des mécanismes pour la modélisation d'environnements interactifs d'apprentissage avec ordinateurs basée sur la simulation qualitative* », Thèse de doctorat de l'Université Paul Sabatier, Toulouse III, 1996.

[Boulay, 86] Benedict du Boulay
« *Some difficulties of learning to program* », Educational computing research, Hypermédias et apprentissages Vol. 2(1), 1986.

[Bourron & al, 95] Yves Bourron, J-P. Chapuis, J-L. Ruby
« *Pédagogie de l'audiovisuel et du multimédia* », Editions Organisations, Paris, 1995.

[Boy, 95] Guy A. Boy
« *Coopération Homme-machine dans les systèmes complexes : agents et aide à la décision* », Conférence Nationale IHM, Toulouse, 1995.

[Boy, 98] Guy A. Boy
« *Cognitive function analysis for human centered automation of safety-critical systems* », 2ème journée Acteurs, Agents et apprentissages, Bayonne, Septembre 1998.

[Boy, 97] Guy A. Boy
« *Documents actifs de conception* », Actes de la Conférence Nationale sur l'Interaction Homme-Machine (IHM'97), Poitiers, Septembre 1997.

[Camps &al, 98] Valérie Camps, Marie-Pierre Gleizes, Pierre Glize
« *Une théorie des phénomènes globaux fondée sur des interactions locales* », JFIADSMA'98, Editions Hermès, France, 1998.

[Bui, 98] Minh Duc Bui
« *Conception et modélisation objet des systèmes temps réel* », Editions Eyrolles, Paris, 1998.

[Chebili, 2000] Mourad Chebili
« *Une interface ergonomique et intelligente d'assistance à l'utilisation d'un système d'exploitation* », Université Paris 8, 2000.

[Corbara, 91] B. Corbara
« *L'organisation sociale et sa genèse chez la fourmi* », Thèse de l'Université Paris XIII, 1991.

[Coulette & al, 90] Bernard Coulette, Ridha Medallel
« *Conception d'outils didactiques intelligents en méthodologie de développement de projets logiciels* », Génie logiciel & systèmes experts, Projet Iphigénie, 1990.

[Coutaz-95] Joëlle Coutaz

« Interaction Homme-machine : Points d'encrage entre ergonomie et génie logiciel », Actes 01Design'95, Aspects communicatifs en conception, 4ème table ronde Francophone sur la conception, 11-13 Janvier 1995.

[Cukier & al, 99] Michel Cukier, Jennifer Ren, Paul Rubel, David Bakke, David Karr
« *Building dependable distributed objects with the Aqua Architecture* », Electrical and Computer Engineering Departement, University of Illinois, USA 1999.

[De La Passardière, 93] Brigitte de la Passardière
« *Hypermédia et apprentissages* », Actes des deuxièmes journées scientifiques, Institut National de Recherche Pédagogique, Paris, 1993.

[De La Passardière, 85] Brigitte de la Passardière
« *De la conception à l'expérimentation d'outils logiciels pour l'initiation à la micro-informatique* », Thèse Université Paris 6, 1985.

[Douroux, 96] Eric Douroux
« *Authorware : résolument interactif* », revue CD-RAMA N°15, p 8-10, Avril 1996.

[Dubrovsky, 97] Ben Dubrovsky
« *Creating and designing multimedia with Director 5.0* », Prentice Hall, 1997.

[Eloi, 96] Murielle Eloi
« Mise en œuvre d'un noyau de générateur d'environnements interactifs d'apprentissage avec ordinateur pour l'atelier de génie didacticiel intégré », Thèse de doctorat, Université Paul Sabatier, Toulouse III, 1996.

[Ferber, 95] J. Ferber
« *Les systèmes multi-agents : vers une intelligence collective* » , InterEditions, 1995.

[Ferber, 95] J. Ferber
« *Les systèmes multi-agents : un aperçu général* », Techniques et science informatiques, Volume 16(8), page 979-1012, 1997.

[Ferber, 89] J. Ferber

« *Objets et agents : une étude des structures de représentation et de communication en intelligence artificielle* », Thèse de doctorat, Université de Pierre et Marie Curie, Paris 6, juin 89.

[Foisel, 98], Rémi Foisel
« *Modèle de réorganisation de système multi-agents : une approche descriptive et opérationnelle* », Thèse de doctorat de l'Université Nancy I, novembre 1998.

[Forner, 90] Yann Forner
« *La psychologie différentielle de la décision dans les logiciels de guidance* », Revue Orientation scolaire et professionnelle de l'INETOP, Editions OSP, Paris, 1990.

[Fournier, 93] Jean-Pierre Fournier
« *Fiabilité du logiciel : concepts, modélisations, perspectives* », Editions Hermès, Paris, 1993.

[Frasson, Ramazani, 92] Claude Frasson et D. Ramazani
« *Prédiction du niveau d'acquisition des connaissances dans la modélisation de l'étudiant* », ITS'92, Montréal Canada, Juin 1992.

[Ghosh & al, 98] Anup Gosh, Tom O'Connor, Gary McGraw
« *An automated approch for identifying potential vulnerabilities in software* », RST Corporation, 1998 IEEE Symposium on Security and Privacy, USA 1998.

[Gouardères, 90] Guy Gouardères
« *Le projet AGDI* », Revue Techniques et Sciences de l'Informatique N°5, vol. 9, Octobre 1990.

[Gouardères & al, 96] Guy Gouardères, Claude Frasson, Thierry Mengelle, E. Aimeur
« *An actor-based architecture for intelligent tutoring systems* », Third International Conference, ITS '96, Montréal, 1996.

[Gouardères, 97] Guy Gouardères
« Compte rendu de la table ronde de la journée Recherche du 26/07/97», Actes de communications, 2ème Journée Acteurs, Agents & Apprentissage, Bayonne, 97.

[Gouardères, 98] Guy Gouardères
« *Des agents pour l'étude et la conception des systèmes socio-techniques complexes - interface* », 2ème journée Acteurs, Agents et apprentissages, Bayonne, Septembre 1998.

[Gouyet & al, 95] Jean-Noël Gouyet, Thierry Najean, Jacques Renard, Sandra Enlart-Michel
« *Facteurs clés de succès des produits multimédia interactifs* », rapport de l'Institut National de l'Audiovisuel, Étude Création Multimédia, 1995.

[Greer &al, 92] Jim E. Greer, S. Bhuiyan, Gordon I McCalla
« Learning recursion through the use of a mental model-based programming environnement », ITS'92, Montréal Canada, June 92.

[Guignier, 91] Daniel Guignier
« *Sécurité et Qualité des systèmes d'information* », Editions Masson, Paris, 1991.

[Hoc, 92] Jean-Michel Hoc
« *Psychologie cognitive dans la planification* », Presse Universitaire de Grenoble, 1992.

[Hoc, 79] Jean-Michel Hoc
« *Le problème de la planification dans la construction d'un programme informatique* », Le travail humain, Grenoble, 1979.

[Huang & al, 99] Yanjun Huang, Shivakant Mishra, Harshavardhan Kuntur
« DaAgent : a dependable mobile agent system », Proceedings of the 29th IEEE International Symposium on Fault-tolerant Computing, Madison, WI (June 1999)..

[Ibrahim & al, 96] B. Ibrahim, B. Laustsen, A. Aubord, M. Tepper
« *Techniques de génie logiciel pour l'EAO* », Université de Genève, 1996.

[Jambon, 96] Francis Jambon
« *Erreurs et interruptions du point de vue de l'ingénierie de l'interaction homme-machine* », Thèse de doctorat, Université Joseph Fourier, Grenoble, 1996.

[Jaulent, 94] Patrick Jaulent
« *Génie logiciel - les méthodes* », Editions Armand Colin, Paris 1994.

[Kolski, 93] Christophe Kolski
« Ingénierie des interfaces homme-machine : concepetion et évaluation », Editions Hermès, 1993.

[Labat, 91] Labat
« *Projet de recherche EMI : EAO multimédia intelligent* », Hay Space & Laforia, Juillet 1991.

[Laprie, 94] Jean-Claude Laprie
« *Informatique tolérante aux fautes* », p29 à 47, Editions Masson, 1994.

[Latour, 99] Bruno Latour
« *Prothée : une méthode d'évaluation en continu des projets d'innovation*», Journée Projectique, Bayonne 1999.

[Laurière, 84] Jean-Louis Laurière
« *Système expert et enseignement* », Journal de la formation continue et de l'EAO N° 176, 1984.

[Madaule & al, 87] Françoise Madaule, Brigitte de la Passardière, Patrick Barril, Françoise Le Calvez
« *Systèmes d'enseignement assisté par ordinateur : Étude comparative* », Technique et Science Informatiques, Vol. 6, 1987.

[Marcenac, 92] Pierre Marcenac
« *EDDI : un prototype de système auteur pour des didacticiels intelligents* », Génie éducatif, No 4&5, 1992.

[Marquesuzaa, Nodenot, 96] Christophe Marquesuzaa, Thierry Nodenot
« *Spécification pédagogique des logiciels éducatifs* », EdMedia 1996.

[Masini & al, 89] Gérald Masini , Amedeo Napoli, Domnique Colnet, Daniel Léonard, Karl Tombre
« *Les langages à objets* », InterEditions, Paris 1989.

[Meinadier, 91] J.P. Meinadier,
« L'interface utilisateur : pour une informatique plus conviviale », Editions Dunod, 1991 Paris.

[Morejon, 89] José Morejon
« *Merise par l'exemple* », Editions Organisation, 1989.

[Nanard, 96], Marc Nanard
« *Le multimédia, de l'age artisanal à l'age industriel* », LIRMM Montpellier, 1996.

[Nanard, 90] Jocelyne Nanard,
« *La manipulation directe en interface homme-machine »*, thèse de doctorat, université des Sciences et Techniques du Languedoc, Décembre 1990.

[Nguyen, Hoc, 87] Anh Xuan Nguyen, Jean-Michel Hoc
« *Learning to use a command device* », , Vol. 7, p 5-31, 1987.

[Nguyen, Nicaud, 95] Anh Xuan Nguyen, Jean-François Nicaud
« *Effet des messages d'erreur sur l'apprentissage de l'appariement des règles de factorisation avec un environnement interactif intelligent* », Sciences et techniques éducatives, Vol. 2, p 145-172, 1995.

[Nicaud, 89] Jean-François Nicaud
« *APLUSIX : un système expert pédagogique et un environnement d'apprentissage dans le domaine du raisonnement algébrique* », Technique et Science Informatique, vol 8, N° 2, p 146-147, 1989.

[Norman et Drapper, 86] D.A. Norman, S.W. Drapper
« User Centered System Design », Lawrence Erlbaum Associates, Publishers, 1986.

[Norman, 91] K.L. Norman
« The Psychology of Menu Selection : Designing Cognitive Control at the Human/Computer Interface », Ablex Publishing Corporation, 1991.

[Péninou, 93] A. Péninou
« MACT : un modèle d'agents centrés tâches pour la production des systèmes tuteurs intelligents par l'atelier de génie didacticiel intégré », Thèse de doctorat de l'université Paul Sabatier, Toulouse III, Novembre 1993.

[Prince, 98] Violaine Prince
« *Un cahier des charges pour la modélisation d'agents cognitifs au sein d'un système socio-technique complexe* », 2ème journée Acteurs, Agents et apprentissages, Bayonne, Septembre 1998.

[Prince, 96] Violaine Prince
« *Vers une informatique cognitive dans les organisations* », Editions Massons, Paris 1996.

[Quere, 83] M. Quere
« *Langage d'auteurs d'hier et d'aujourd'hui* », Journal de la formation continue et de l'EAO N° 166, 1983.

[Rasmussen, 80] J.Rasmussen
« *The human as a system component* », Editions Human Interaction with Computer, London Academic Press, 1980.

[Rasmussen 1986] J . Rasmussen
« Information processing and Human-Machine Interaction : An approach to cognitive engineering », North-Holland, 1986.

[Reason 1993] J. Reason
« L'erreur humaine », Presses Universitaires de France, Paris 1993.

[Richards, 74] Jack C. Richards

« *Error analysis : Perspectives on Second Language Acquisition* », Longman, London 1974.

[Ricordel &al, 98] PM Ricordel, G. Chicoisne, S. Pesty, Y Demazeau
« *Outils et pistes pour la pratique du dialogisme entre agents* », JFIADSMA'98, Editions Hermès, 1998, France.
[Rocher, 1968] G. Rocher
« *Introduction à la sociologie générale* », Le Seuil, 1968.

[Roulin, 90] Jean-Luc Roulin
« *Psychométrie et informatique* », Université de Savoie, Chambéry France, 1990.

[Rousseau, 96] Michel Rousseau
« *Les systèmes auteurs* », CD-RAMA N°16 - Mai 1996.

[Rozenberg, 98] Maurice Rozenberg
« *Test logiciel* », Editions Eyrolles, Paris 1998.

[Sabah, 90] G. Sabah
« *CARAMEL : A Computational Model of Natural Language Understanding using a Parallel Implementation* », ECAI-90. Stockholm. Pp. 563-565, 1990.

[Salamone, 96] Salvatore Salamone
« *Storyboarding speeds development by organizing everything from concept to finish product* », Byte p 67-70 - Mars 1996.

[Saxena & al, 99], Nina Saxena, Jacob Abraham, Avijit Saha
« A divide and conquer strategy for consistency and troubleshooting in specifications of large design », The 29th International Symposium on Fault-Tolerant Computing, Madison, Wisconsin, USA, June 15-18, 1999.

[Scapin et Bastien, 93] Dominique Scapin, J.M. Bastien
« *Preliminart findings on the effectiveness of ergonomic criteria for the evaluation of human-computer interfaces* », InterCHI'93, Amsterdam, April 1993.

[Strevens, 69] Peter Strevens
« Two ways of looking at error analisis », Washington, 1969.

[Stroustroup, 90] Bjarne Stroustrup
« *The annotated C++ Référence Manual* », Margaret Ellis, Addison Wesley, 1990.

[Surcin & al, 95] S. Surcin, C. Choppy, M-G. Séré
« *Analyse didactique d'un cours de génie logiciel basé sur l'approche orientée objet* », Laboratoire LIMSI, Université Orsay, 1995.

[Stevens, 1994] W. Richard Stevens
« *TCP/IP Illustrated, Volume 1 : The Protocols* », Addison-Wesley, 1994.

[Tanenbaum, 1996], Andrew S. Tanenbaum
« *Computer Networks* », Prentice Hall, 1996.

[Tiberghien, 88] G. Tiberghien
« Informatique et Société : Psychologie cognitive, science de la cognition et technologie de la connaissance », Les Editions de la Presse, 1988.

[Vacherand-Revel, Bessière, 90] Jacqueline Vacherand-Revel, Christian Bessière
« *L'interaction graphique et la prise en compte des différences entre usagers dans la formation d'adulte assistée par ordinateur* », LEACM - Université Lyon 2, CNRS-IRPEACS - Lyon, 1990.

[Van-Eylen, Hiraclides, 96] Hugues Van-Eylen, Georges Hiraclides
« *La démarche GRAAL* », Editions Angkor, Paris 1996.

[Van Daele, 92] A. Van Daele
« *La réduction de la complexité par les opérateurs dans le contrôle de processus continus, contribution à l'étude du contrôle par anticipation et de ses conditions de mise en œuvre* » Thèse de doctorat en Psychologie, Université de Liège, Octobre 92.

[Vergnaud, 89] Gérard Vergnaud

« *La théorie des champs conceptuels* », Revue, Recherches en Didactiques des Mathématiques RDM, Vol 10, 1989.

[Woolridge, Jennings, 95] M. Woolridge, N. R. Jennings
« *Intelligent Agents : Theory and Practice* », Knowledge Engineering Review, Janvier 1995.

ANNEXE A : Liste des figures

ANNEXE B : Lexique des abréviations

AAE	Agent Analyseur Interne
AAI	Agent Analyseur Externe
ACE	Agent Contrôleur Êrgonomique
ACG	Agent Cognitif
ACO	Agent Collecteur
ACP	Agent Contrôleur Pédagogique
AD	Agent Décideur
AGD	Atelier de Génie Didacticiel
API	Application Programming Interface
AS	Agent Superviseur
CBL	Computer Based Learning
CBT	Computer Based Training
EAO	Enseignement Assisté par Ordinateur
ECLR	Environnement de Création de Logiciels Robustes
EDG	Exceptions Dynamiques Globales
EDL	Exceptions Dynamiques Locales
EIAO	Environnement Interactif d'Apprentissage par Ordinateur
EDL	Exceptions Statiques Globales
ESL	Exceptions Statiques Locales
GUI	General User Interface
LP	Langage de Programmation
MC	Modèle de Communication
MT	Modèle de Traitement
POO	Programmation Orientée Objet
SAM	Système Auteur Multimédia
SAP	Service Access Point
SMA	Système Multi-Agents
WYSIWYG	what you see is what you get

Annexe C : Implémentation d'un ECLR pour le SAM TK/TCL

Dans cette annexe, nous allons présenter les différentes procédures composant l'ECLR que nous avons développé pour le système auteur multimédia TK/TCL.

Robustesse des systèmes auteurs multimédias : contribution théorique et mise en œuvre

Résumé :

Les systèmes auteurs multimédias ou SAM sont des outils de développement permettant à tout individu de concevoir un didacticiel ou tout type de logiciel pour partager leurs connaissances avec le minimum d'effort en programmation. Mais le résultat de ces concepts n'est pas proportionnel aux efforts et aux mérites des chercheurs et des professionnels qui s'y sont investis.
La complexité des fonctions et des problèmes rencontrés par les personnes initiées ou non à la conception, la distribution physique et fonctionnelle des erreurs et l'hétérogénéité des moyens et des médias mis en œuvre ne facilitent pas l'adaptation des SAM à des changements de structures et à des évolutions de contexte. La robustesse incluant les concepts de fiabilité, de pérennité et de sûreté de fonctionnement est alors un des piliers manquant pour la réussite des SAM.

Dans cette thèse, nous défendons notre point de vue sur la nécessité d'assister l'utilisateur de SAM lors de la conception, de l'évaluation et de l'interaction homme-machine. Nous allons proposer une solution théorique basée sur un système multi-agents et une architecture en couches, qui conduit à la création d'un outil appelé Environnement de Création de Logiciels Robustes ou ECLR. Contrairement aux autres outils, l'ECLR possède une architecture simple et performante qui s'adapte à l'évolution du système, du contexte et de l'environnement.
Après une synthèse sur les SAM actuels, nous orientons notre recherche sur les notions d'erreurs et de robustesse dans la conception de logiciel. Nous mettons en œuvre ensuite un système multi-agents où nous ferons la distinction entre l'organisation et l'interaction des agents afin d'intégrer les qualités d'analyse, de décision, d'anticipation, d'apprentissage présentes chez un humain.

Mots-clés :
Système auteur multimédia | Système multi-agents | Architecture en couches
Interaction homme-ordinateur | Apprentissage automatique | Environnement de conception robuste
Conception assistée par ordinateur en génie logiciel

==========

Multimedia authoring systems robustness : theoretical contribution and implementation.

Abstract :

Multimedia authoring systems or MAS are development tools allowing people to design a CBT or any other type of software in order to share their knowledge with the minimum of effort on programming. But the result of these concepts is not proportional to the efforts and to the merits of some researchers and some professionals who invested in.
Functions complexity and problems met by people initiated or not in designing, the physical and functional distribution of mistakes and the heterogeneity of means and of media don't ease MAS adaptation to structure changes and to contextual or environment evolutions. The notion of robustness, including the concepts of reliability, of perenniality and of working safety ; is then a missing pillar for MAS concept's success.

In this thesis, we defend our opinion to assist MAS users in designing, in assessment and in human computer interaction. We submit a theoretical solution founded on a multi-agents system with a layers architecture. This solution opens out onto the creation of one tool called Robust Software Design Environment or RSDE. Unlike the other tools, RSDE has one simple and efficient architecture who fits automatically to the system, context and environment evolutions.
After a synthesis on current MAS, we orientate our research towards the notions of mistakes and of robustness in software design. Then, we make concrete a multi-agents system where we show differences between agents organization and interaction in order to integrate a human qualities of analysis, of decision, of anticipation, of learning.

Keywords :
Multimedia authoring system | Multi-agents system | Layers architecture
Human Computer Interaction | Machine learning | Robust design environment
Computer aided software engineering

Laboratoire de Recherche Informatique Avancée
15, rue Catulienne – 93200 Saint-Denis