

THESE

présentée par

Karine DUPREZ

pour obtenir

**Le titre de Docteur
en spécialité Informatique : Systèmes et Communications
de l'Institut National Polytechnique de Grenoble
(Arrêté ministériel du 30 mars 1992)**

EXPLICATIONS ET CONSEILS CONTEXTUELS POUR L'UTILISATION D'UN PROGICIEL DE CONCEPTION

Application à un simulateur de circuits électroniques

Soutenue le 27 mai 1998,

devant le jury composé de :

M.	Jean-Pierre	VERJUS	Président
Mme.	Noëlle	CARBONELL	Rapporteurs
M.	Amedeo	NAPOLI	
Mme	Joëlle	COUTAZ	Examineurs
M.	Michel	DEPEYROT	
M.	François	RECHENMANN	Directeur

Thèse préparée au sein de

l'Unité de Recherche INRIA Rhône-Alpes de
l'Institut National de Recherche en Informatique et en Automatique.

Remerciements

Je remercie chaleureusement madame Noëlle Carbonell et monsieur Amedeo Napoli d'avoir accepté de rapporter sur ce travail. Leurs conseils et leurs encouragements ont largement contribué à améliorer la présentation de cette synthèse.

Par son cours et ses travaux, Joëlle Coutaz a suscité de riches discussions et je tiens à la remercier de manifester son intérêt pour mon travail en participant au jury de ma soutenance.

Je remercie particulièrement Monsieur Jean-Pierre Verjus, Directeur de l'INRIA Rhône-Alpes, de me faire l'honneur de présider le jury de ma soutenance de thèse. J'y suis d'autant plus sensible que l'institut est le fil conducteur de mes travaux de recherche menés dans différents projets, depuis mon DEA jusqu'à la soutenance de mes travaux réalisés dans le cadre d'une convention CIFRE.

Je tiens à remercier l'INRIA pour toutes les personnes rencontrées dans ce bouillon de recherche. Un grand merci à Fethi Bounaas, pour les week-end consacrés à l'explication du fonctionnement de la plate-forme Shood, base du développement de la maquette.

L'organisation de l'institut facilite les rencontres entre les projets, et j'ai ainsi pu bénéficier de l'expérience d'André Bisseret qui a eu la gentillesse de commenter les interfaces avec l'utilisateur du premier prototype.

Merci aux membres du projet Sherpa, qui ont eu la gentillesse de m'accueillir et ont su m'encourager dans les moments difficiles, en dépit de ma présence erratique dans les locaux de l'institut. La vie d'équipe n'est bien sûr pas partagée de la même façon quand le lieu de travail est distant, ne serait-ce que de quelques kilomètres. Cela induit une autre organisation pour discuter des avancées et des difficultés rencontrées : les échanges de points de vue se font au cours des réunions de travail et des réunions d'équipe. Les instants privilégiés des discussions informelles autour du café ont été trop rares.

Ce document est le fruit d'un travail de recherche de plus de trois ans que j'ai mené avec mon directeur de thèse : François Rechenmann. Il m'a fait bénéficier de ses conseils avisés pour la mise en forme de ce mémoire. Ces heures de travail en commun m'ont montré toute l'importance de cette « forme » qui seule permet de communiquer et de faire partager les idées.

Ce travail n'aurait pas pu être mené à bien sans la collaboration de nombreux utilisateurs. Merci à toutes les personnes ayant apporté une critique constructive :

- Autour de la maquette, mes premières discussions avec M. Ballane furent un grand encouragement. Ce premier contact avec des utilisateurs m'a convaincue de m'ouvrir à leur critique, pour définir ensemble ce que le module d'aide devait leur apporter en priorité. Aujourd'hui ce module d'aide est concrétisé par Advisor, implémenté dans

le simulateur Smash™. Ce module d'aide n'est encore qu'à la version 1.0, pour l'instant...

- Merci à Etienne Gheeraert pour toute son énergie consacrée à l'introduction de Smash dans l'enseignement de l'école d'ingénieurs 3I. C'est avec plaisir que j'ai pu voir des étudiants découvrir le module d'aide et m'apporter des idées pour leur faciliter la tâche.
- Je tiens à remercier les clients de Dolphin Integration qui ont pris le temps d'étudier en site expérimental le prototype d'Advisor et nous ont aidés à l'améliorer par leurs judicieuses suggestions. Le résultat est aussi le fruit de leur investissement. Thanks Haakan !
- De nombreuses discussions avec les concepteurs, travaillant à Dolphin Integration, accompagnées de patientes explications sur leurs façons de procéder dans les situations délicates ont largement contribué à la définition des priorités entre les différents conseils apportés par Advisor.

Michel Depeyrot, PDG de Dolphin Integration, est l'initiateur de cette collaboration entre la recherche en laboratoire et le développement d'une application dans le cadre de son entreprise. Le statut de StartUp Inria de cette société, est un cadre privilégié pour établir ce type de collaboration. Au quotidien, c'est avec énergie qu'il encourage une recherche motivée par des besoins. La recherche appliquée est valorisée pour peu qu'elle respecte les contraintes stimulantes que sont un temps raisonnable pour la voir concrétiser, et une écoute permanente des requêtes clients. Stimulantes, car ce sont les conditions de base pour réussir le pari d'une collaboration recherche/entreprise. Je lui dois l'envie de concrétiser cet enthousiasme dans un outil innovant et performant.

Au cours de ces trois années, mon responsable, Cyril Desclèves m'a permis de découvrir le monde de l'entreprise. Il a encouragé la validation de mes travaux à chaque étape (maquette, prototype et module d'aide), en m'invitant à les confronter avec les utilisateurs et leurs besoins parfois si difficiles à spécifier. Je remercie Cyril de m'avoir laissé carte blanche pour envisager différentes voies (raisonnement à partir de cas, heuristique, apprentissage), et par ailleurs de m'avoir incité à valider les idées retenues aussi tôt que possible. J'ai particulièrement apprécié l'autonomie qu'il m'a accordée, dans la répartition de mon activité entre la recherche et son application, et dans les contacts avec les clients lors des évaluations de prototypes.

Ce travail s'est nourri du terreau de mon éducation, enrichi jour après jour par l'affection de mes grands-parents et de mes parents. Je remercie chaleureusement Ramon de m'avoir aidée à trouver confiance en moi, et de m'avoir encouragée à voir le travail comme le vecteur d'une découverte permanente. Merci à Elise et Ernest, pour leur gentillesse et toute l'attention qu'ils m'ont accordée.

Un mot enfin pour leur merveilleux soutien à Charlotte, pour les instants privilégiés de la tétée, les joies simples et essentielles de la vie qu'un petit bout fait redécouvrir et à Robert, pour tout.

Table des matières

0. INTRODUCTION	9
1. LA PROBLÉMATIQUE ET LE CADRE DE TRAVAIL	11
1.1. EXEMPLE DE PROBLÉMATIQUE : LE SIMULATEUR SMASH	11
1.1.1. LA DESCRIPTION DU PROCÉDÉ À ÉTUDIER	14
1.1.2. LES DIRECTIVES DE SIMULATION	17
1.1.3. LES SITUATIONS D'ÉCHEC	19
1.1.4. LES CONFLITS	23
1.2. LES BESOINS DES UTILISATEURS	24
1.2.1. COMMENT DÉFINIR LES BESOINS DES UTILISATEURS ?	25
1.2.2. COMMENT DÉTERMINER LE TYPE D'AIDE SATISFAISANT ?	25
1.2.3. QUEL SUPPORT POUR UNE CRITIQUE CONSTRUCTIVE ?	26
1.2.4. QUEL TYPE D'INTERACTION ?	28
1.2.5. Y A-T-IL UN BESOIN SPÉCIFIQUE POUR L'UTILISATEUR NÉOPHYTE ?	30
1.3. L'ENVIRONNEMENT DU SYSTÈME D'AIDE	32
1.3.1. LES INTERFACES	32
1.3.2. LES CONNAISSANCES DU SYSTÈME D'AIDE	34
1.4. LES AUTRES CANDIDATS POUR UNE APPLICATION DE LA MÉTHODE	36
1.5. LE CADRE DE CE TRAVAIL	38
1.6. LES APPORTS DE CE TRAVAIL	39
2. LES ÉLÉMENTS DE SOLUTION	41
2.1. UNE DÉFINITION DE L'AIDE CONTEXTUELLE	41
2.1.1. LA MODÉLISATION DE L'UTILISATEUR	42
2.1.2. LES CONNAISSANCES DU DOMAINE D'APPLICATION	43
2.1.3. LE SUIVI DE L'UTILISATEUR	45
2.2. L'OBSERVATION DES ACTIONS DE L'UTILISATEUR	46
2.2.1. LE MODÈLE DE TÂCHES	46
2.2.2. LA MODÉLISATION DU FONCTIONNEMENT DU PROGICIEL	47
2.3. UNE AIDE PILOTÉE PAR L'UTILISATEUR	48
2.3.1. LIMITER LE RECOURS AU SUPPORT TECHNIQUE	48
2.3.2. IMPLIQUER L'UTILISATEUR	49
2.4. LA MODÉLISATION DU SYSTÈME D'AIDE	51
2.4.1. L'INTERFACE AVEC LE PROGICIEL	52
2.4.2. L'EXPLOITATION DES CONNAISSANCES	54
2.5. L'ENRICHISSEMENT DES CONNAISSANCES DU SYSTÈME D'AIDE	55

2.5.1. L'INTÉGRATION DE NOUVELLES CONNAISSANCES	56
2.5.2. LA REMÉMORATION EFFICACE PAR RÀPC	57
3. L'INTERACTION AVEC L'UTILISATEUR	61
3.1. POURQUOI SE DISPENSER DU PROFIL UTILISATEUR ?	61
3.2. LA VALORISATION DE L'UTILISATEUR	62
3.3. LE MODE D'INTERACTION	63
3.3.1. UNE INTERACTION SOUPLE.....	63
3.3.2. INCITER L'ACTIVITÉ DE L'UTILISATEUR	64
3.3.3. PROFITER DES SITUATIONS D'ÉCHEC.....	65
3.4. L'INTERACTION RETENUE.....	66
4. LES FONDEMENTS DU MODULE D'AIDE	71
4.1. LA DÉFINITION DU CONTEXTE.....	71
4.2. LE SCHÉMA DE PHASES POUR LE SUIVI DE L'UTILISATEUR.....	72
4.3. LA DISTINCTION ENTRE LES CONNAISSANCES STRUCTURELLES ET LES CONNAISSANCES CONTEXTUELLES.....	74
4.4. LA REPRÉSENTATION PAR OBJETS.....	74
4.5. LES MÉTACONNAISSANCES	77
4.6. LA MAQUETTE VALIDE LE SYSTÈME RÉACTIF	79
5. LES CONNAISSANCES SUR LE FONCTIONNEMENT DU PROGICIEL	81
5.1. L'ORGANISATION DES CONNAISSANCES	81
5.1.1. L'ORGANISATION AUTOUR DU SCHÉMA DE PHASES	82
5.1.2. LES CONNAISSANCES DU NOYAU SÉMANTIQUE.....	83
5.1.2.1. Les conditions d'utilisation	85
5.1.2.2. Les règles de bonne utilisation	85
5.1.2.3. Les vérifications avant une confirmation	87
5.1.3. LES RÈGLES RÉFLEXES POUR TENIR COMPTE DU CONTEXTE.....	88
5.1.3.1. Les pré-conditions d'un contrôle : un exemple de règle.....	88
5.1.3.2. La formalisation des règles réflexes	90
5.1.3.3. L'intervention des règles réflexes.....	91
5.2. LES EXPLICATIONS	91
5.2.1. LE DESCRIPTIF DES OBJETS DU NOYAU SÉMANTIQUE.....	93
5.2.2. LES NOTIONS	95
5.2.3. LES CONSEILS	97
6. LE FONCTIONNEMENT DU SYSTÈME D'AIDE	99
6.1. RÉPERCUTER LES ACTIONS DANS LE MODULE D'AIDE	99
6.1.1. L'OBSERVATION DES ACTIONS	101
6.1.2. ORGANISATION DU SUIVI POUR UNE MISE À JOUR DU CONTEXTE	102
6.1.3. L'HISTORIQUE MÉMORISE LES ACTIONS.....	105
6.2. UNE AIDE PRÉVENTIVE	109
6.2.1. INTERVENTION AVANT LA CONFIRMATION D'UNE ACTION	109

6.2.2. EXEMPLE DE PRÉVENTION	110
6.3. LA MODIFICATION DU CONTEXTE DANS LE MODULE D'AIDE.....	113
6.3.1. RÉPERCUTER LA CONFIRMATION D'UNE COMMANDE	114
6.3.2. EXEMPLE DE SUIVI LORS DE LA CONFIRMATION D'UNE COMMANDE.....	115
6.4. LES CAPACITÉS DU MODULE D'AIDE DANS UNE SITUATION D'ÉCHEC	117
7. L'AIDE À LA PRISE DE DÉCISION	119
<hr/>	
7.1. LES CONSEILS.....	120
7.1.1. LA DÉFINITION DES CONSEILS FORMALISÉE AUTOUR DES ÉCHECS	120
7.1.2. LE RAISONNEMENT LOCAL POUR LES CONSEILS.....	123
7.1.3. L'EXPLOITATION DES CONSEILS DANS LE FONCTIONNEMENT	124
7.2. LA MÉMORISATION DES EXPÉRIENCES.....	126
7.2.1. LA DÉFINITION D'UNE EXPÉRIENCE	127
7.2.2. COMMENT LES EXPÉRIENCES COMPLÈTENT LES HEURISTIQUES	128
8. CONCLUSION	131
<hr/>	
9. BIBLIOGRAPHIE	133
<hr/>	
10. ANNEXE A : MANUEL ADVISOR	139
<hr/>	
11. ANNEXE B : FORMULAIRE POUR L'ÉVALUATION DU PROTOTYPE	147
<hr/>	
11.1. VOTRE PROFIL	147
11.1.1. L'ÉVALUATION DE VOTRE EXPÉRIENCE EN SIMULATION DE CIRCUITS ÉLECTRIQUES	147
11.1.2. VOS TYPES DE CIRCUITS SIMULÉS AVEC SMASH™ :	148
11.2. VOS COMMENTAIRES :	148
11.2.1. LA FACILITÉ D'UTILISATION D'ADVISOR	148
11.2.2. LES CARACTÉRISTIQUES D'ADVISOR.....	148
11.2.3. L'UTILISATION D'ADVISOR	149
11.2.4. CE QUI VOUS A MANQUÉ DANS LA RECHERCHE D'UNE EXPLICATION.....	149
11.2.5. CE QUE VOUS AVEZ APPRIS PAR L'UTILISATION D'ADVISOR.....	150
11.3. VOS SUGGESTIONS VIS-À-VIS DU SYSTÈME D'AIDE ADVISOR :	150

0. INTRODUCTION

Etant donné un progiciel¹ de conception existant, comment introduire un système qui conseille les utilisateurs de manière pertinente sur son utilisation?

Il y a une vingtaine d'années, les outils professionnels de conception s'attachaient plus à l'efficacité des algorithmes numériques qu'à leur facilité d'utilisation. La priorité n'était pas de rendre l'outil informatique accessible à tous les utilisateurs, d'autant plus que le public concerné était restreint. Dans la mesure où le coût de la formation des utilisateurs était négligeable par rapport à celui du matériel informatique, la valeur de l'outil était étroitement liée à la performance des algorithmes, même si leur exploitation nécessitait une certaine expérience.

Aujourd'hui cette préférence est moins marquée. Les avancées dans la conception des interfaces homme-machine en sont la preuve. Elles répondent à la nécessité de performance d'un concepteur qui travaille avec de multiples environnements, imposés par des projets distincts. Si les outils conçus aujourd'hui offrent une plus grande facilité d'utilisation, les outils de conception performants développés auparavant sont toujours en usage. Comme il est coûteux de réécrire ces derniers, il apparaît intéressant de les compléter avec un système d'aide pour valoriser toutes leurs fonctionnalités.

Il s'agit alors de mettre en œuvre un système d'aide capable d'expliquer et de conseiller l'utilisateur pour le conduire à une bonne utilisation des capacités de l'outil existant.

Une bonne utilisation du progiciel n'est pas obtenue en limitant les interactions entre l'utilisateur et le progiciel. Limiter le nombre de paramètres que l'utilisateur doit systématiquement spécifier simplifie l'emploi de l'outil, mais il ne faut pas pour autant tomber dans une automatisation exagérée. Bien au contraire, les concepteurs plébiscitent un meilleur contrôle plutôt qu'une automatisation dont ils ne pourraient pas vérifier les conséquences. Notre approche développe ce contrôle en donnant des informations pertinentes, aux moments opportuns. De plus, pour être acceptées par le concepteur, ces informations devront être étayées par une justification.

Ce mémoire propose une méthode pour intégrer un système d'aide dans un progiciel de conception existant. Le rôle principal du système d'aide est d'encourager l'utilisation des capacités du progiciel, qui restent en partie méconnues des concepteurs. Le deuxième objectif du système d'aide est de conseiller l'utilisateur, en justifiant ses propositions, de manière à permettre une appropriation du progiciel. Cette méthode est aujourd'hui appliquée à un outil de simulation, c'est-à-dire un outil de vérification en conception.

De manière concrète, les utilisateurs ont été particulièrement sollicités pour évaluer les trois étapes du développement du module d'aide :

- La maquette, en novembre 1995, pour éprouver une modélisation du système et une ébauche d'interface avec l'utilisateur. Le développement de la maquette est fait sur une plate-forme *a priori* indépendante de celle du progiciel. Dans notre cas, elle a été développée sur la plate-forme Shood en langage Le-Lisp, alors que le progiciel est écrit en langage C.

¹ Produit logiciel, par opposition à un logiciel à la demande.

- Le prototype, fini en octobre 1996 et écrit en langage de programmation par objets C++, est une première implémentation des idées validées par la maquette. Il est totalement intégré au progiciel, c'est-à-dire qu'il lui est possible de connaître instantanément les actions de l'utilisateur. C'est seulement à cette étape que le suivi des actions est effectif : auparavant, il était émulé.
- La première version commerciale du module d'aide, nommé alors Advisor, est disponible depuis juillet 1997. C'est le résultat de l'évolution du prototype pour tenir compte des réactions de ses utilisateurs dans le cadre expérimental de sites beta test, qui a duré six mois.

La thèse défendue dans ce mémoire est qu'il faut donner au concepteur un statut d'expert, c'est-à-dire que le système d'aide doit être pensé pour les méconnaissances du progiciel de la part de l'utilisateur, mais en aucun cas pour son ignorance du domaine d'application. Cette approche garantit le respect de ses initiatives et de son savoir-faire, fruit de son expérience professionnelle. Le système d'aide donne avant tout des conseils, en aidant l'utilisateur à évaluer la pertinence des propositions. Il offre une aide proactive, qui devance les besoins de l'utilisateur et qui traite avec lui les problèmes qui surviennent.

Ce mémoire comprend sept chapitres pour présenter l'introduction d'un système d'aide contextuel dans un progiciel existant. Ils abordent les démarches successivement, bien que, en pratique, elles aient été menées en parallèle plutôt que séquentiellement. En effet, les réflexions de chaque démarche sont enrichies par les problèmes que soulèvent les autres aspects, pour ensuite converger vers les propositions ici rapportées. Dans un souci de clarté, les problèmes sont groupés en sept points qui traitent respectivement :

- Les besoins de l'utilisateur, et en particulier comment ils ont été identifiés, sont le sujet du chapitre 1.
- Les éléments de réponses, fruits de travaux de recherche qui ont contribué à définir les choix pour notre système d'aide, sont présentés dans le chapitre 2. Ce chapitre n'est pas une synthèse complète de mes lectures dans un état de l'art sur les systèmes d'aide, mais une discussion sur les références qui ont particulièrement orienté mon travail.
- Le type d'interaction retenue pour encourager l'appel au système d'aide est discuté dans le chapitre 3.
- Les bases du fonctionnement du système d'aide pour fournir cette interaction sont exposées au chapitre 4.
- La modélisation des connaissances sur l'utilisation du progiciel fait l'objet du chapitre 5.
- Le fonctionnement du système d'aide est présenté dans le chapitre 6, qui expose comment les connaissances du système d'aide permettent de fournir des informations contextuelles à l'utilisateur.
- L'aide à l'utilisateur pour prendre une décision est introduite par le chapitre 7, qui expose par ailleurs comment est envisagé l'ajout de connaissance dans le système d'aide.

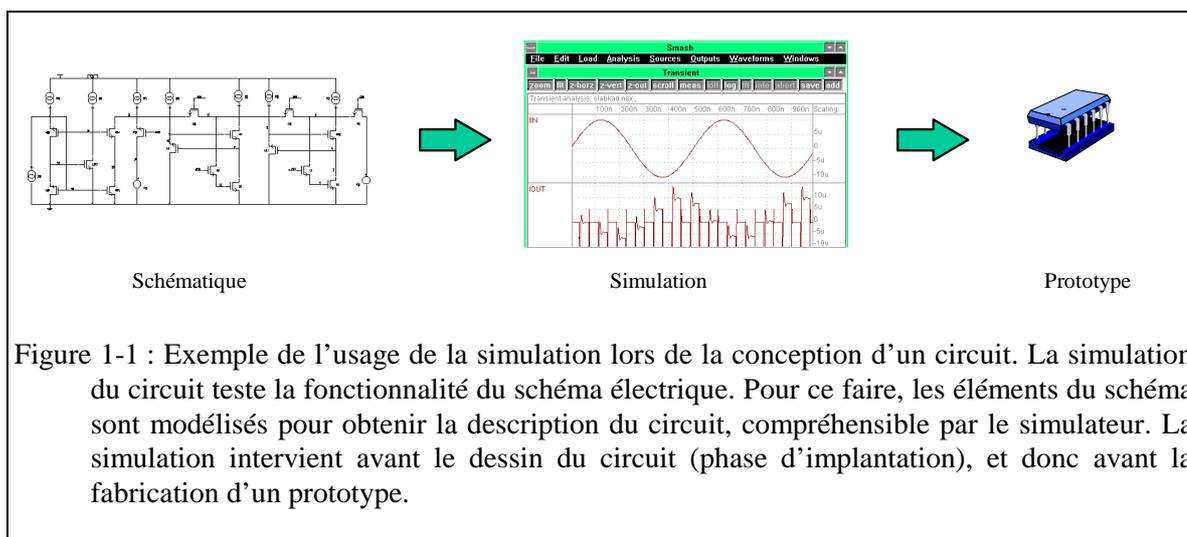
Chapitre 1

1. La problématique et le cadre de travail

Avant d'aborder l'analyse des besoins et les propositions pour y répondre, nous montrons quelles sont les difficultés rencontrées par l'utilisateur dans un cas réel. Pour ce faire, nous présentons le progiciel Smash de simulation de circuits électriques, qui va illustrer nos propos tout au long du mémoire. La méthode pour introduire un système d'aide, qui fait l'objet de ce travail, a été appliquée à ce progiciel. Elle se concrétise par l'implémentation du module d'aide, baptisé Advisor.

1.1. Exemple de problématique : le simulateur Smash

Dans le processus global de la conception de circuits, la simulation est une première phase de validation, chargée de détecter les problèmes avant de passer à une phase de prototype, où la découverte d'une erreur est plus coûteuse à réparer.



Son but est de simuler le comportement d'un circuit, soumis à différents stimuli. Pour ce faire, comme tous les outils de conception, le simulateur exploite le modèle du processus étudié, décrit par une formalisation mathématique. Ce point est important, car il est une cause

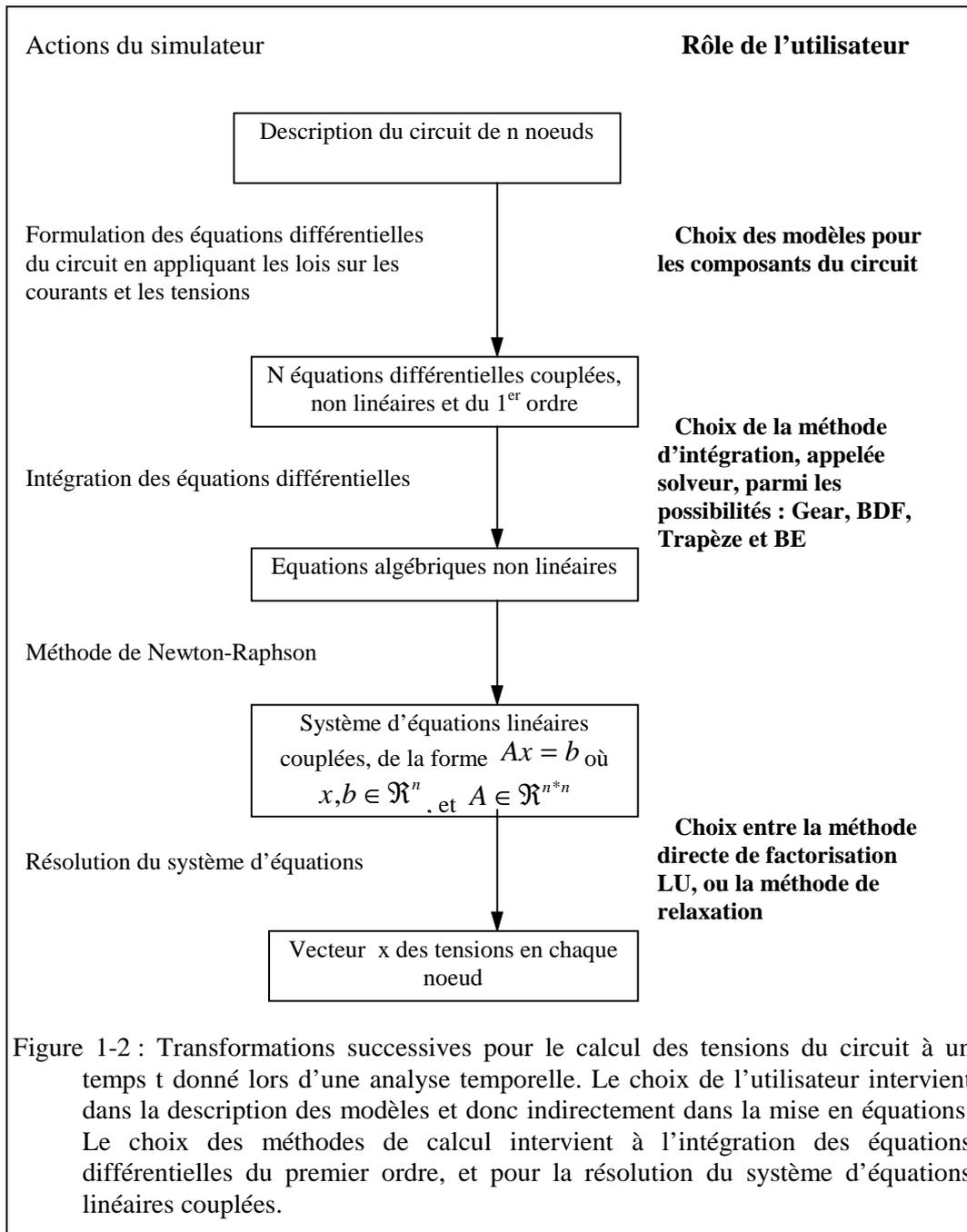
inévitables de l'introduction d'approximations, tant un modèle a ses limites pour reproduire le processus physique réel.

Le terme de simulation est souvent adopté pour désigner des réalités distinctes, illustrées par notre domaine d'application :

- l'étude générale de la description d'un circuit, qui correspond à l'ensemble des vérifications à faire au cours d'une session de travail,
- une analyse qui vérifie un aspect du circuit, en appliquant de manière itérative une analyse numérique qui modélise le comportement à tester,
- une itération de l'analyse numérique, qui est une étape lors du calcul d'une solution, qui n'est atteinte que si cette analyse numérique converge. Un exemple de divergence est donné ultérieurement avec la Figure 1-7 qui illustre comment les conditions initiales peuvent intervenir dans la recherche d'une solution.

Pour clarifier notre discours, nous réservons le terme de « simulation » à l'étude du circuit, et plus généralement à l'étude complète d'un processus. En adoptant ce vocabulaire, la session est étroitement liée à la simulation du circuit, puisqu'elle correspond à la simulation menée avec le progiciel, sur un circuit particulier. Dans notre application, le progiciel ne permet de travailler que sur un circuit à la fois, c'est-à-dire qu'il ne permet pas d'ouvrir plus d'une session à la fois. Au cours d'une session, plusieurs analyses peuvent être successivement appliquées à la description du circuit. Nous nommons phases de travail ces différentes analyses du circuit.

Considérons maintenant une analyse d'un circuit électrique. C'est la transformation successive d'une mise en équations de son modèle, qui est initialement obtenu en appliquant les lois de Kirchoff sur les courants et les tensions. Il en résulte un système d'équations différentielles couplées, non linéaires et du premier ordre. La figure suivante (Figure 1-2) donne un exemple des transformations successives pour une analyse temporelle, c'est-à-dire une étude du comportement dans le temps du circuit, et indique où le choix de l'utilisateur intervient pour le calcul à un temps donné de l'état du circuit.



La première étape est de choisir la méthode d'intégration, parmi les « solveurs » que sont *trapeze*, *backward Euler*, *Gear* et *BDF*, pour obtenir des équations algébriques non linéaires et couplées. La seconde étape est l'application de la méthode itérative de Newton-Raphson pour obtenir le système d'équations linéaires couplées. Ce dernier est ensuite résolu par la méthode de la factorisation LU, dite aussi méthode directe. Dans certains cas, l'application de la méthode dite de relaxation conduira à une simulation beaucoup plus rapide.

La difficulté est non seulement de faire ces choix, mais aussi de correctement fixer les paramètres qui contrôlent ces différentes méthodes, pour que l'analyse temporelle converge.

Nous détaillons maintenant les différents points où le choix de l'utilisateur est déterminant dans l'étude du circuit, qui comprend en général plusieurs analyses, comme l'analyse temporelle ou l'analyse fréquentielle. Chacune de ces analyses est une étude particulière, avec ses transformations successives. Le résultat d'une analyse est l'ensemble des états atteints par le circuit, c'est-à-dire l'ensemble des tensions aux noeuds du circuit et des courants dans ses branches, calculés par l'itération des transformations successives. Dans la pratique, l'utilisateur est intéressé par le résultat de l'analyse. C'est en fonction de la qualité des résultats attendus, qu'il faut contrôler les paramètres de chaque analyse.

En particulier, quelques exemples simples montrent que le choix de l'utilisateur pour fixer des paramètres doit tenir compte des caractéristiques du circuit et des simulations appliquées.

1.1.1. La description du procédé à étudier

Même si la modélisation n'est pas l'objet du système d'aide, elle reste omniprésente dans les difficultés d'utilisation du progiciel de conception. Il est donc intéressant de préciser son importance et de l'illustrer dans le cadre de notre application. Dans la suite du mémoire, nous abrégeons le terme *modèle de circuit* par le nom *circuit*, en sachant que le simulateur ne connaît que le modèle.

La description du circuit comprend :

- l'ensemble des noeuds (ou connexions) du circuit, avec les composants qui y sont connectés : c'est la *netlist*. La description est hiérarchique car un composant est éventuellement un sous-circuit ;
- les modèles des composants présents dans le circuit. Le modèle d'un composant est l'expression mathématique de la loi physique que suit le comportement du composant comme l'illustre la Figure 1-3. Par analogie, un modèle peut être identifié à une fonction, qui décrit le comportement général d'un type de composant. Certains composants de base, comme un élément capacitif, sont par défaut modélisés dans le simulateur : ce sont des primitives. Dans le cas général, le concepteur doit fournir le modèle, soit en le créant de toutes pièces, soit en utilisant un modèle défini dans une librairie de modèles ;

$$\frac{\partial I_D}{\partial V_D} = g_{md} = -\mu \frac{W}{L} \left(\frac{Q'_r}{1 + \frac{Q'_r - Q'_f}{Q'_o}} + \frac{Q'_r{}^2 - Q'_f{}^2}{\left(1 + \frac{Q'_r - Q'_f}{Q'_o}\right)^2} \frac{1}{2Q'_o} \right) \frac{Q'_{ID}}{Q'_r}$$

Quand $V_D = V_S$, les densités de charges Q'_r and Q'_f sont égales et

$$\frac{\partial I_D}{\partial V_D} = \frac{\partial I_D}{\partial V_S} = g_{ms} = -\mu \frac{W}{L} Q'_{IS}$$

Figure 1-3 : Extrait des équations du modèle ACM de mosfet, pour le simulateur Smash™. L'exemple est le calcul de la tension de saturation. La description complète du modèle est disponible sur simple demande à Dolphin Integration.

- un composant est décrit par son modèle et ses paramètres physiques, à l'image de l'instanciation d'une fonction, avec les paramètres d'appel. En effet, une résistance de 1ohm suit les mêmes règles qu'une résistance de 1Kohm, mais les effets sont bien entendu fonction de ces valeurs;
- de surcroît, certaines primitives sont des modèles paramétrables, pour affiner le comportement d'un composant complexe. Par exemple, la description du comportement d'un transistor *MOS* pose d'une part le problème de choisir le modèle le plus adapté, et d'autre part de le paramétrer pour obtenir un comportement satisfaisant. Pour paramétrer un tel modèle, il faut choisir pas moins de 30 paramètres de manière à approcher au mieux le composant réel utilisé. Cet exemple illustre la difficulté pour modéliser avec une certaine précision un composant. Ci-dessous la Figure 1-4 explicite les paramètres du modèle ACM d'un transistor mos.

NAME	DESCRIPTION	UNITS	DEFAULT
CREC	gate-source/drain overlap capacitance	F/m	0.0
CGSO	gate-source overlap capacitance	F/m	0.0
CGDO	gate-drain overlap capacitance	F/m	0.0
PB	junction potential	V	0.8
CGBO	gate-bulk overlap capacitance	F/m	0.0
CJ	bottom junction capacitance / area	F/m ²	0.0
CJSW	side-wall junction capacitance	F/m	0.0
MJ	exponent for bottom capacitance formula	-	0.5
MJSW	exponent for side-wall capacitance formula	-	0.33
IS	junction saturation current	A	0.0
JS	junction saturation current density	A/m ²	0.0
LDIFF	lateral diffusion width	m	0.0
FC	coefficient for reverse formula in junction capacitance	-	0.5
RD	drain ohmic resistance	ohm	0.0
RS	source ohmic resistance	ohm	0.0
RDC	drain contact resistance	ohm	0.0
RSC	source contact resistance	ohm	0.0
RSH	diffusion resistance	ohm/square	0.0
UO	mobility	cm ² /V.s	550E-4
TOX	oxide thickness	m	1.5E-8
VTO	threshold voltage ($V_{DB}=V_{SB}=0V$)	V	0.77
GAMMA	body effect parameter	sqrt(V)	0.77
PHI	bulk Fermi potential	V	0.61
LAMBDA	depletion length coefficient	-	0.25
WETA	narrow channel effect coefficient	-	0.26
LETA	short channel effect coefficient	-	0.44
DW	channel narrowing	m	-0.1E-6
DL	channel shortening	m	-0.4E-6
EPSILON	ratio for drain saturation voltage definition	-	0.05
UCRIT	longitudinal critical field	V/m	2.6E6
THETA	mobility reduction coefficient	1/V	0.08**

Figure 1-4 : Ensemble des paramètres pour déterminer les caractéristiques d'un mosfet du modèle ACM. Les valeurs par défaut correspondent rarement à une réalité physique, c'est-à-dire qu'il faut adapter cette trentaine de paramètres qui interviennent dans les équations qui définissent le comportement du composant. La description complète comprend en outre les paramètres physiques du transistor, comme sa taille.

Nous avons indiqué que cette étape de modélisation qui précède l'étude du circuit n'est pas traitée par le système d'aide. Néanmoins, nous verrons qu'une analyse peut être arrêtée, ou freinée, par une modélisation inadaptée, et le système d'aide gagne à vérifier un minimum de données. Ainsi, quand l'analyse est trop lente, il faut envisager des modèles plus simples, et donc plus rapides à calculer, dans la mesure où la précision peut être relâchée.

1.1.2. Les directives de simulation

Notre travail se concentre sur la simulation à partir d'un modèle de circuit donné et supposé satisfaisant. Le simulateur est un outil de vérification dont l'utilisation est similaire aux successifs appliqués à un prototype physique pour le tester :

- Mettre le prototype sous-tension, ce qui en anglais est nommée une analyse de « powerup » (Figure 1-5). Physiquement cela correspond au branchement de l'alimentation électrique du circuit.
- Vérifier le point de fonctionnement (en anglais « Operating Point », abrégé par OP dans la Figure 1-5) du circuit, qui correspond à l'état d'équilibre lorsque le circuit est alimenté. Dans cet état, les composants doivent être correctement connectés, ou polarisés, avant d'être soumis à des stimuli. Si tel n'est pas le cas, le prototype physique, voir même le banc d'essai, peut-être déjà endommagé. Dans la pratique, il arrive que le prototype fonde. Avec une simulation, outre les tensions et courants, la puissance est calculée, ce qui facilite le repérage des zones trop chaudes.
- Appliquer des stimuli sur le prototype, avec des sources de tensions et/ou de courant.
- Vérifier l'état du circuit en plaçant des sondes pour visualiser les signaux sur un oscilloscope. Cette étude temporelle des signaux est nommée en anglais « transient analysis », et la Figure 1-6 illustre les résultats graphiques.

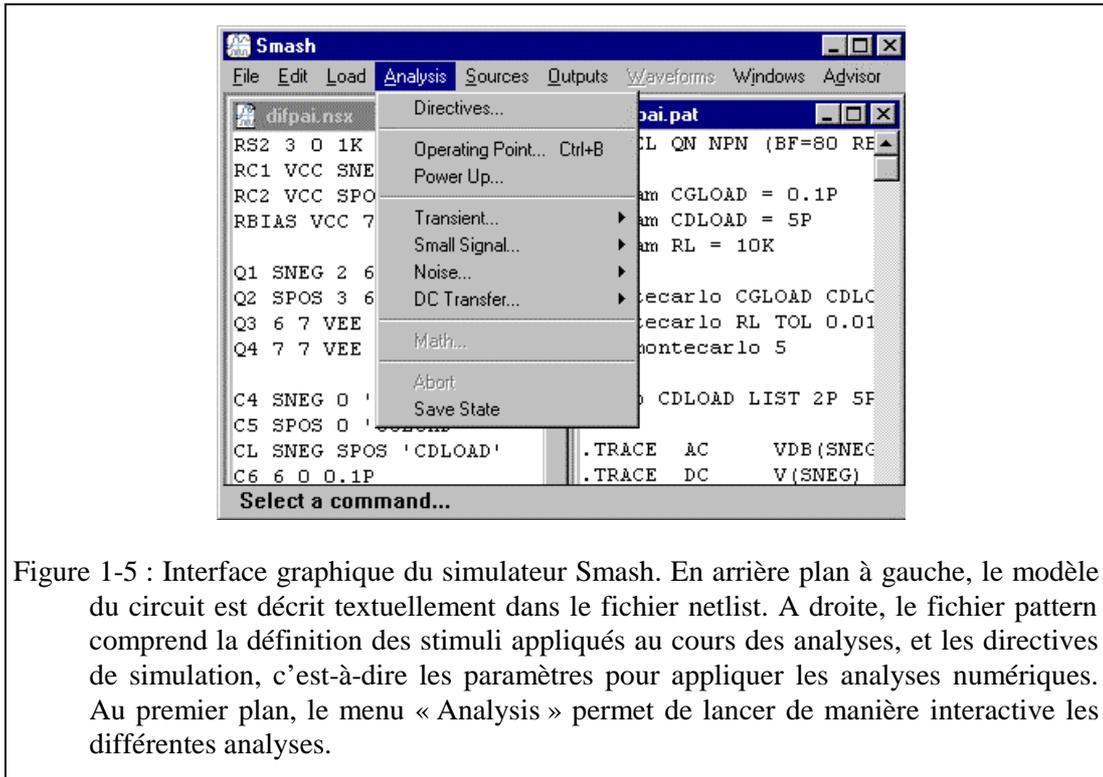


Figure 1-5 : Interface graphique du simulateur Smash. En arrière plan à gauche, le modèle du circuit est décrit textuellement dans le fichier netlist. A droite, le fichier pattern comprend la définition des stimuli appliqués au cours des analyses, et les directives de simulation, c'est-à-dire les paramètres pour appliquer les analyses numériques. Au premier plan, le menu « Analysis » permet de lancer de manière interactive les différentes analyses.

Cependant, l'étude du processus n'est pas une procédure simple et linéaire, comme cet exemple peut le laisser croire. Elle repose sur une méthode d'essais et erreurs, pour affiner les analyses numériques appliquées au modèle du circuit. Dans le cas du prototype, il est manifeste que quelques allers retours sont nécessaires pour atteindre un prototype validé, en corrigeant les erreurs ou en améliorant les performances du circuit.

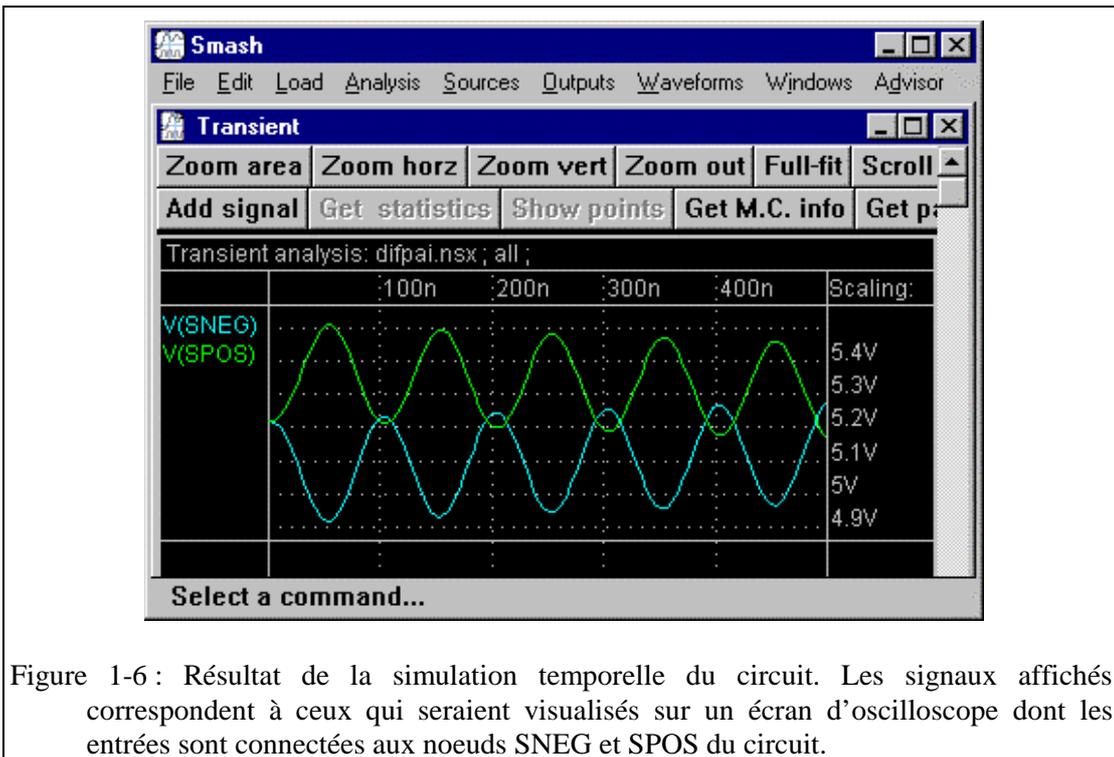


Figure 1-6 : Résultat de la simulation temporelle du circuit. Les signaux affichés correspondent à ceux qui seraient visualisés sur un écran d'oscilloscope dont les entrées sont connectées aux noeuds SNEG et SPOS du circuit.

Voyons maintenant ce qu'il en est dans le cadre de la simulation, où les phénomènes physiques de la mise sous tension, de l'état d'équilibre du point de fonctionnement, de l'analyse temporelle sont modélisés par autant d'analyses numériques appliquées au modèle du circuit. De la maîtrise de ces analyses dépend la qualité de la simulation, qui sera comparée avec les tests faits sur le prototype réel en fin de processus de conception.

1.1.3. Les situations d'échec

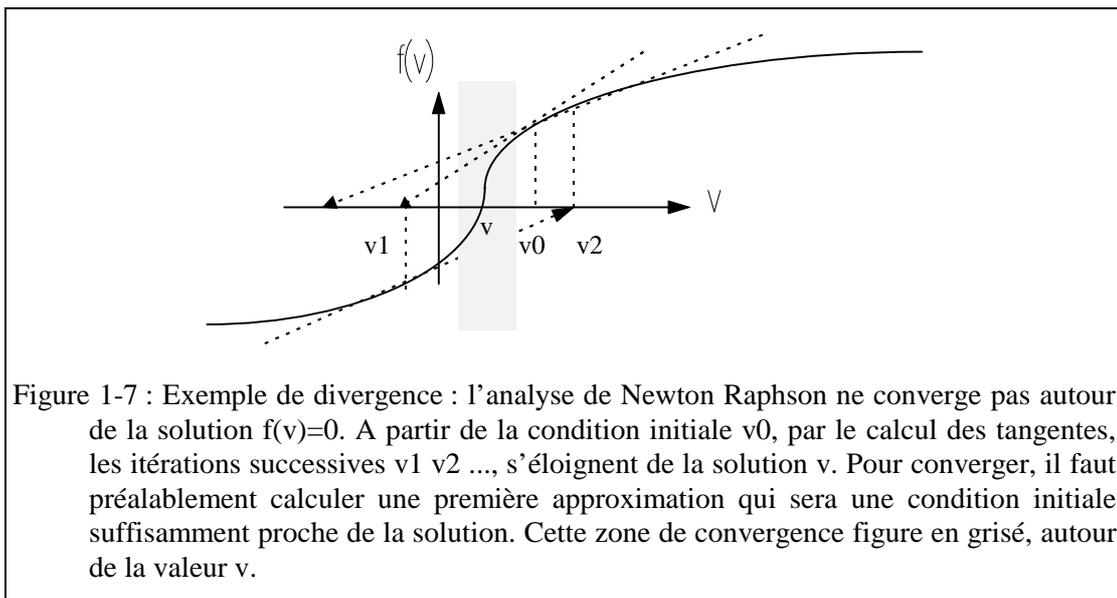
Une analyse numérique est contrôlée par le concepteur qui fixe les conditions pour l'arrêter en déterminant les paramètres de précision au-delà desquels une solution est retenue. Si l'analyse ne converge pas sous les contraintes de précision imposées, elle est arrêtée après un nombre maximum d'itérations par exemple. Dans ce dernier cas, le simulateur précise qu'il n'a pas pu calculer une solution respectant les critères.

Le concepteur est donc confronté aux limites de la modélisation que représentent les analyses numériques. Dans la suite du mémoire c'est ce que nous nommons une *situation d'échec*, où l'analyse numérique ne peut pas converger sous les conditions fixées.

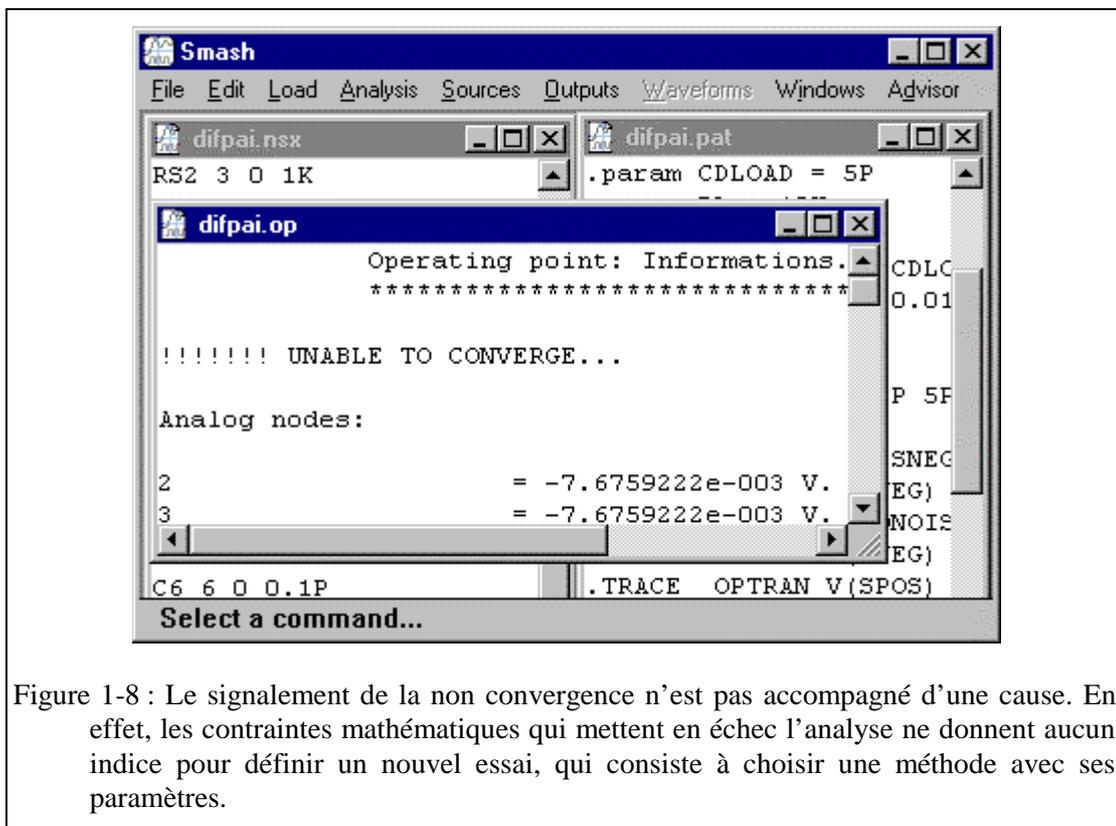
Dans cette situation, plusieurs causes sont à envisager :

- la précision est trop sévère pour que l'analyse trouve une solution ;
- les paramètres qui définissent l'analyse sont incohérents entre eux et/ou avec le circuit étudié ;
- pour mener l'étude, l'analyse numérique choisie n'est pas adaptée, et il faut envisager une autre méthode de calcul ;
- le modèle du circuit introduit des artifices non physiques, ou néglige des composants parasites, ce qui induit des problèmes algorithmiques.

Les analyses numériques sont parfois des pièges, dont le concepteur ne sait pas sortir. En effet, elles introduisent la notion de convergence autour de la solution illustrée par la Figure 1-7. Ces problèmes sont algorithmiques, et ne correspondent pas nécessairement à une impossibilité physique. Ainsi, un prototype physique sous tension peut être dans un état d'équilibre satisfaisant, sans que l'analyse qui calcule le point de fonctionnement converge. De plus, l'algorithme de calcul du point de fonctionnement n'a aucune réalité physique au cours du calcul : seule la solution résultante du calcul a une signification physique. De ce fait, il est encore plus difficile de déterminer ce qui peut être fait pour corriger la divergence de l'algorithme.

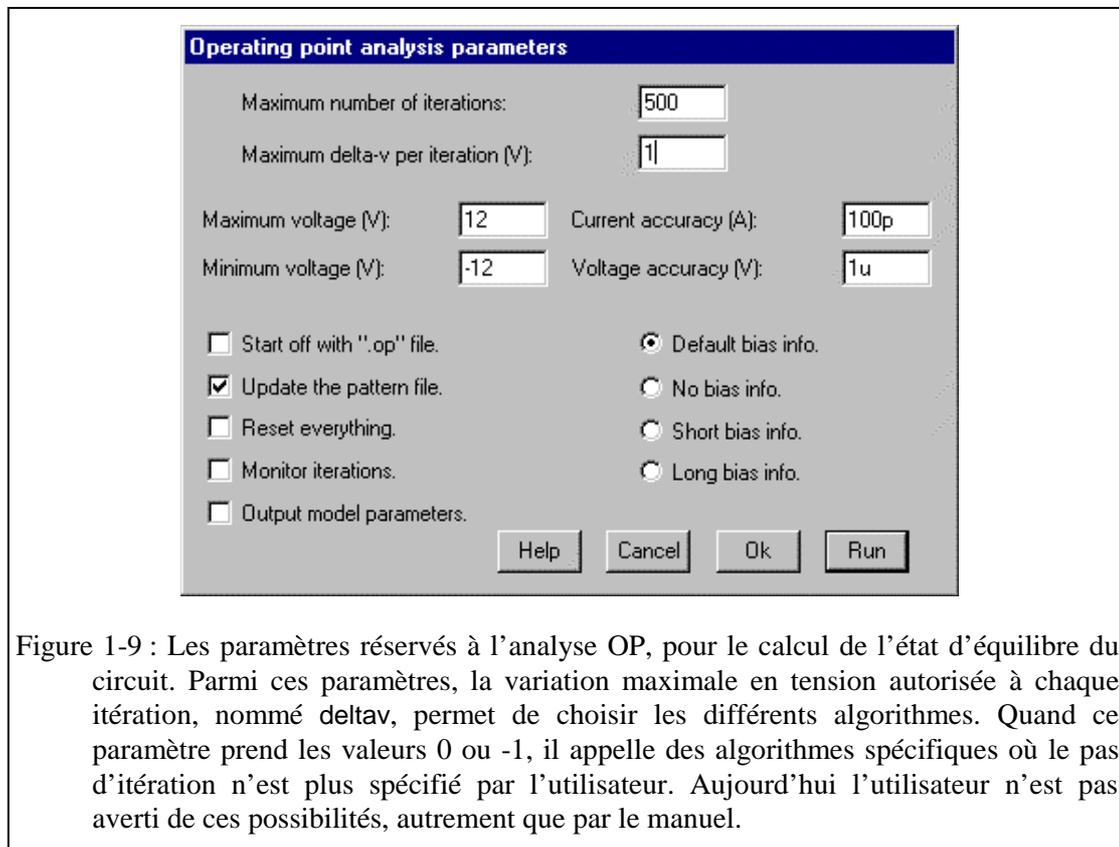


L'exemple suivant (Figure 1-8) illustre les difficultés que rencontrent les concepteurs lorsque l'analyse ne converge pas. Il concerne le calcul de l'état d'équilibre du circuit mis sous tension, qui peut être mené par différents algorithmes.



Dans cette situation d'échec, le concepteur reconsidère les paramètres de contrôle, regroupés dans la boîte de dialogue (Figure 1-9). Il faut noter que les capacités du simulateur ne sont

pas toujours mises en valeur dans l'interface avec l'utilisateur. Ainsi dans la boîte de dialogue suivante se dissimule le choix des algorithmes de calcul, et la possibilité de définir des conditions initiales à partir desquelles l'analyse est appliquée en cochant la case « start with .op file ». Mais comment faire pour obtenir cette première approximation ?



Bien entendu, la boîte de dialogue pourrait mettre en évidence les différentes méthodes, comme cela est fait pour l'analyse temporelle. Néanmoins, les difficultés rencontrées en analyse temporelle montrent que le problème pour le concepteur est non seulement de connaître les alternatives, mais ensuite de savoir si elles sont applicables, et enfin de comprendre dans quelle limite elles le sont. C'est avec ces informations que l'utilisateur pourra choisir de manière autonome ses essais.

Cependant, la validité des méthodes ne peut pas être définie *a priori* en étudiant le contexte. Autant il est possible d'écarter une méthode dont les conditions d'application ne sont pas réunies, autant une méthode susceptible d'être appliquée ne mène pas forcément à une solution : la résolution se fait par un processus d'essais et erreurs. De plus, les méthodes ne sont pas exclusives. Quand plusieurs méthodes sont candidates, c'est-à-dire que leurs conditions d'application sont respectées, c'est l'expertise de l'utilisateur du simulateur qui guide le choix de la première méthode à essayer.

L'expertise de l'utilisateur concerne non seulement le choix de la méthode, mais aussi la façon de la paramétrer. Ces paramètres dépendent étroitement des caractéristiques du circuit étudié. Par exemple, une analyse de recherche du point de fonctionnement aboutit à une situation d'échec si elle est appliquée avec un grand pas d'itération sur un circuit qui comprend des transistors bipolaires. Par expérience, les utilisateurs savent qu'une diminution du pas de

calcul à une valeur de l'ordre de 100mV facilite la convergence de l'algorithme, mais cette indication n'est pas nécessairement suffisante. De telles règles d'utilisation sont nombreuses, mais restent méconnues. Leur formalisation est un aspect du travail mené avec les utilisateurs. Elles donnent au moins une indication, si ce n'est une solution. En effet, ces règles ne sont pas rigides : la convergence sera certainement préservée avec une valeur légèrement différente. Dans d'autres cas, en dépit d'une valeur proche de 100mV, la situation d'échec ne sera pas résolue.

En résumé, le concepteur confronté à une situation d'échec doit adapter son essai infructueux, soit en corrigeant les seuls paramètres de la méthode retenue, soit en remettant en cause la méthode choisie dans la mesure où il existe d'autres alternatives. C'est donc par une succession d'essais et erreurs qu'il procède. Pour faciliter la résolution de la situation d'échec, il faudrait que le concepteur puisse connaître les méthodes qui s'appliquent à son problème.

Si l'analyse de la recherche du point de fonctionnement aboutit à une situation d'échec, plusieurs propositions peuvent être envisagées. L'ordre d'essais des propositions privilégie celles qui ne remettent en cause ni la méthode numérique, ni le circuit. Si la méthode numérique ne peut pas s'appliquer en ajustant les différents paramètres, il faudra en envisager une autre. D'une manière générale, les adaptations portent avant tout sur le progiciel, mais il n'est pas exclu de modifier la modélisation du circuit pour faciliter la convergence.

La liste non exhaustive de propositions est la suivante pour la situation d'échec précédente :

- P1 : diminuer le pas d'itération pour tenir compte de la présence de transistors bipolaires, en augmentant le nombre d'itérations maximum.
- P2 : choisir une méthode numérique qui incrémente lentement le pas d'itération.
- P3 : rechercher une première approximation de l'état d'équilibre, qui servira de conditions initiales pour l'analyse. Pour faciliter la convergence, la conductance des transistors bipolaires est modifiée.

L'essai de la proposition P1 donne une solution, puisqu'un point de fonctionnement est calculé et accepté par le concepteur. Dans l'hypothèse où il fallait adapter les conductances des transistors, ce qui correspond à la proposition P3, le concepteur aurait du ouvrir la boîte de dialogue « Directives » (Figure 1-10), réservé aux paramètres à modifier avec prudence. Ce dialogue illustre un autre problème rencontrés par les concepteurs, que nous nommons les conflits.

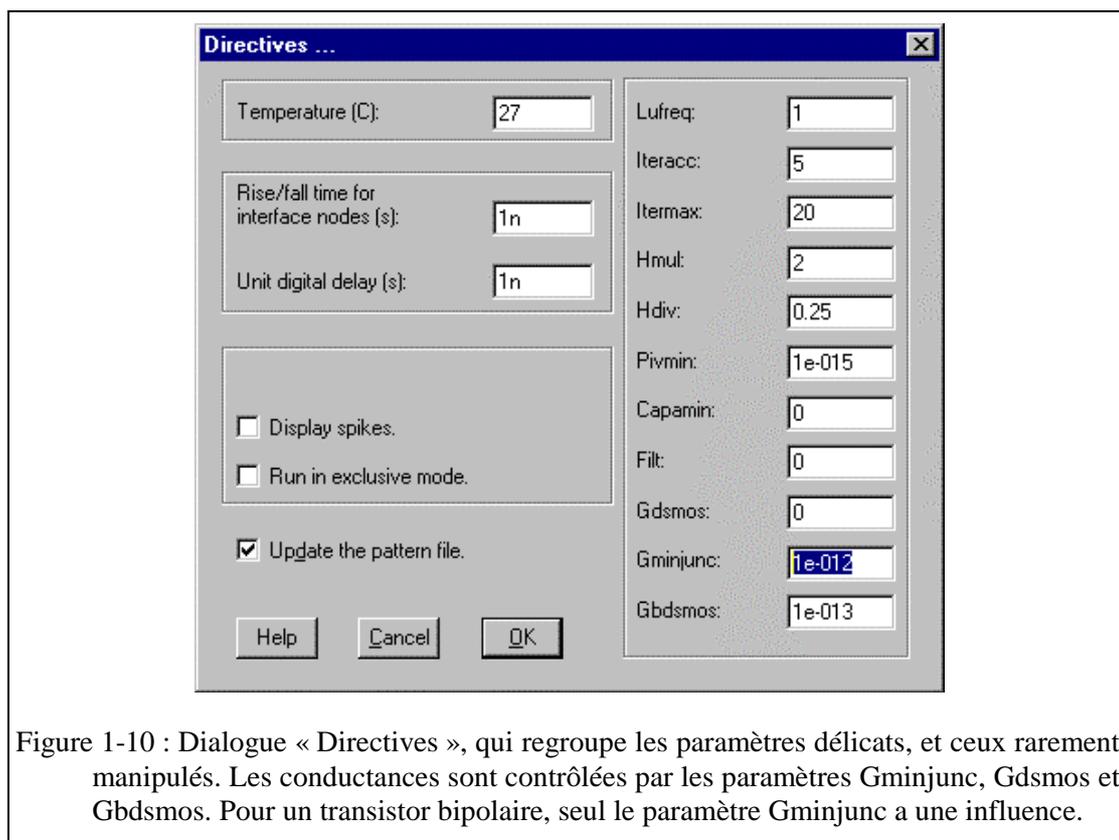


Figure 1-10 : Dialogue « Directives », qui regroupe les paramètres délicats, et ceux rarement manipulés. Les conductances sont contrôlées par les paramètres Gminjunc, Gdsmos et Gbdsmos. Pour un transistor bipolaire, seul le paramètre Gminjunc a une influence.

1.1.4. Les conflits

Nous avons montré dans l'exemple précédent l'influence du type de circuit sur le choix des paramètres de l'analyse numérique, où le pas d'itération est modifié en fonction de la présence des transistors bipolaires. Il existe en fait un ensemble de paramètres pour définir la simulation, dont la pertinence est définie par les caractéristiques du circuit, et par le contexte d'utilisation.

Nous nommons *conflit* la violation d'une règle de bonne utilisation du progiciel, dictée par le manuel du progiciel, ou par l'expertise d'utilisation. Le conflit est indépendant d'une situation d'échec : il peut intervenir à tout moment de l'utilisation du progiciel.

Par exemple, quand le circuit présente uniquement des transistors bipolaires, seul le paramètre Gminjunc a une influence dans le contrôle des conductances. Dès lors, modifier un autre paramètre (Gdsmos ou Gbdsmos) est inutile, même si cela ne crée pas d'erreur. Le concepteur n'a cependant aucune indication dans le dialogue « Directives » pour déterminer si un paramètre est influent ou non.

Laisser manipuler des paramètres inutiles ne présente que des inconvénients pour l'utilisateur :

- il perd du temps, même s'il ne s'en rend pas compte ;
- il ne distingue pas les conditions d'application des paramètres.

La cause du conflit peut être une étourderie ou une incompréhension. Dans les deux cas, l'important est d'attirer l'attention sur le problème et de savoir expliciter sa cause du point de vue du fonctionnement du progiciel (pourquoi tel paramètre n'a pas d'influence dans le contexte courant par exemple).

Le signalement d'un conflit correspond donc à la détection d'un « mauvais emploi » du progiciel, par rapport aux règles de bonne utilisation. Un exemple simple de règle porte sur une température excessive pour un composant électronique. Il est en effet irréaliste de faire fonctionner un composant dans un milieu à 500°C.

Le paramètre Température doit être compris entre -50°C et +200°C. En dehors de ces valeurs raisonnables, les composants risquent d'avoir un comportement étrange, dans la mesure où leurs modèles ne prennent pas en compte des valeurs extravagantes.

Figure 1-11 : Exemple de règle de bonne utilisation sur le paramètre qui fixe la température ambiante.

Ce bon sens évite de pousser les modèles mathématiques, qui modélisent le comportement du composant, dans des domaines où ils ne sont plus réalistes. Les paramètres manipulés ne sont pas toujours aussi concrets que la notion de température. Le signalement d'un conflit est donc un garde-fou pour prévenir une utilisation déraisonnable.

Les règles d'une bonne utilisation du progiciel portent sur :

- les conditions d'application d'une méthode numérique ou d'un paramètre ;
- la plage de valeurs raisonnables que peut prendre un paramètre ;
- la cohérence entre les valeurs de paramètres liés.

Le signalement du conflit doit aider le concepteur à s'approprier le progiciel, en insistant sur les utilisations qui ne respectent pas les règles. Toutefois, hormis celles qui définissent les conditions d'application, les règles ne sont pas rigides et elles doivent être perçues comme des indications et non pas comme des interdictions.

1.2. Les besoins des utilisateurs

L'exemple du simulateur Smash illustre la problématique d'une utilisation d'un progiciel de conception : la maîtrise des analyses numériques est le problème principal. La difficulté pour le concepteur est de connaître les méthodes disponibles dans le simulateur, de distinguer leurs spécificités pour les choisir dans les situations adaptées, et enfin de savoir les appliquer avec le contrôle qui sied.

L'introduction d'une aide est souhaitée pour répondre aux besoins suivants :

- dans une situation d'échec, sélectionner les méthodes et les règles formalisées qui s'appliquent au problème courant d'utilisation du progiciel, en tenant compte du circuit étudié ;
- signaler les actions de l'utilisateur qui n'ont aucune conséquence sur le circuit étudié, afin d'éviter les actions inutiles ou redondantes ;
- donner des indications pour fixer les paramètres de contrôle des analyses numériques.

Nous décrivons maintenant les principales questions qui nous ont conduit à proposer une méthode pour définir avec les différents utilisateurs leurs besoins. Le critère retenu au début de l'étude était le niveau global de connaissance de chaque concepteur interrogé sur l'utilisation du progiciel : il était néophyte, avisé ou expert. Nous montrons que cette distinction est trop superficielle pour apporter des conseils adaptés.

1.2.1. Comment définir les besoins des utilisateurs ?

La définition des besoins de l'utilisateur se fait par le constat des difficultés que rencontrent les concepteurs. Néanmoins la méthode qui se limite à l'observation d'un groupe d'utilisateurs ne révèle pas toutes les difficultés :

- d'une part, toutes les options du progiciel ne sont pas exploitées. Ainsi faute d'observer les problèmes attachés à leurs emplois, il ne sera pas possible de déduire l'aide qui serait utile pour ces options moins exploitées;
- d'autre part, les concepteurs avisés évitent des maladresses, souvent rencontrées par des utilisateurs qui ne connaissent pas bien le progiciel, ou qui sont simplement distraits.

En conséquence il faut trouver une autre approche complémentaire. Dans le cadre de notre application sur le simulateur Smash, l'implication personnelle comme utilisateur, d'abord néophyte, puis avisé, a établi un terrain d'échange avec les concepteurs de circuits. Par ailleurs, l'étroite collaboration avec un développeur du simulateur, à la fois informaticien et électronicien, a contribué à expliciter les incompréhensions liées au domaine de la conception de circuits, et aux problèmes de convergence posés par les analyses numériques. La principale contrainte de cette pratique est le niveau de connaissances du domaine de conception à acquérir avant de prétendre exploiter le progiciel. Cette démarche, que nous nommons « méthode de la pratique », s'est révélée fructueuse : elle est particulièrement efficace pour avoir une idée précise des difficultés et pour mieux comprendre les attentes des utilisateurs.

1.2.2. Comment déterminer le type d'aide satisfaisant ?

Au fur et à mesure que les besoins d'aide se précisaient, la discussion avec les concepteurs s'est aussi portée sur la manière dont l'aide pouvait répondre aux besoins. C'est-à-dire que les concepteurs étaient invités à formaliser leurs règles de bonne utilisation du progiciel. Dans la mesure où ces discussions avaient lieu pendant leur travail, les concepteurs ne prenaient pas le temps de formaliser par écrit les exemples. Prendre le temps de les expliquer n'était déjà pas facile. Cette contrainte de temps devenait impérieuse lorsque, placée derrière un concepteur, je lui demandais de tester et de commenter diverses propositions élaborées par d'autres concepteurs. En sus de l'interruption que la discussion nécessitait, certains concepteurs n'appréciaient pas d'être observés et questionnés sur leurs méthodes de travail.

Deux enseignements sont tirés des discussions sur la formalisation des règles :

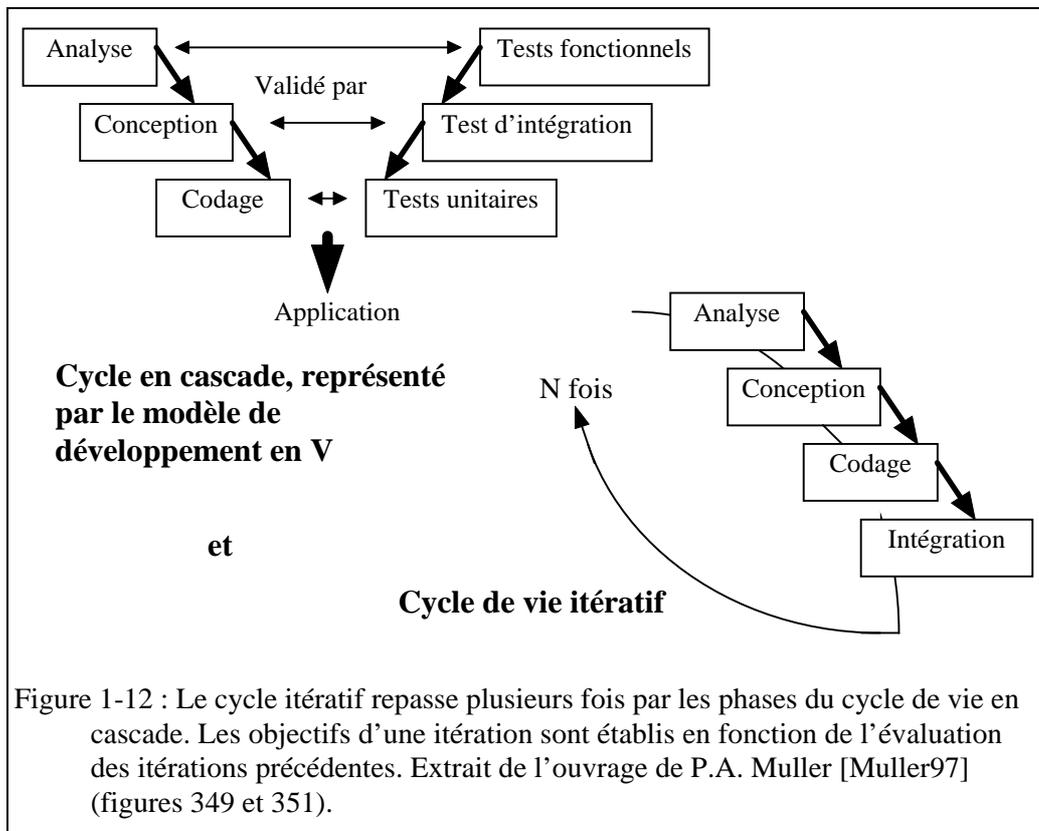
1. il est plus simple pour un concepteur de critiquer une proposition, que de faire l'inventaire des solutions qu'il envisagerait ;
 2. quand un concepteur résout une situation d'échec, il n'apprécie pas d'appliquer strictement une proposition, dictée par un observateur. Si de plus, cette proposition lui est inconnue, le concepteur demande quand est ce qu'elle est applicable, et quelles sont ses spécificités.
-

La discussion devenait donc trop limitée pour définir plus précisément les règles et la manière dont l'aide devait intervenir dans le déroulement de la session. Pour poursuivre la collaboration avec les concepteurs, futurs utilisateurs du système d'aide, il fallait un nouveau terrain d'échanges. La propension des concepteurs à défendre leurs alternatives, et à étudier de près les nouvelles propositions, a privilégié un travail de critique de leur part.

1.2.3. Quel support pour une critique constructive ?

Pour observer les réactions des utilisateurs et encourager leurs propositions, il fallait soumettre une proposition à la critique des concepteurs. L'étape suivante fut donc le développement d'une maquette de système d'aide, qui intègre les règles établies par les concepteurs. La maquette valide les points essentiels du système d'aide. Dans notre cas, elle n'est pas la base d'une réalisation de prototype : les idées bien sûr seront exploitées pour la suite, mais le code n'est pas repris.

La maquette amorce le développement itératif (Figure 1-12) visant à impliquer le plus tôt possible des utilisateurs. P.A. Muller met en évidence les avantages d'un développement itératif, par rapport au développement linéaire [Muller97]. Il discute les idées fausses sur le cycle itératif, notamment en précisant que ce dernier reprend plusieurs fois le cycle en cascade. Il ne s'oppose pas au « cycle en V », mais ouvre l'évolution aux remarques des utilisateurs, dans la mesure où ils sont consultés. Le but de la maquette est de valider l'organisation des connaissances qui doit permettre de produire des conseils, en fonction du contexte.



Ce choix d'une maquette « jetable » donne les avantages suivants :

- un développement rapide, car les bases du système d'aide seront repensées après l'évaluation de la maquette ;
- une implication précoce des utilisateurs pour avoir un retour sur les grandes lignes du système d'aide ;
- la constitution d'une communauté d'experts et d'utilisateurs moins avisés pour établir un échange tout au long du développement ;
- la consolidation de la modélisation par une validation expérimentale des idées, qui sont confirmées ou corrigées pour construire une base éprouvée de travail ;

Une maquette pensée comme un premier prototype aurait posé, au même moment, des problèmes essentiels de choix de type d'aide et des problèmes d'interface avec le progiciel. L'évaluation et la définition du type d'aide auraient été influencées par les autres aspects, qui étaient secondaires à ce stade de la réflexion. De plus, la question du temps n'est pas négligeable. Dans notre cas, la maquette a été développée en six mois, formalisation de la connaissance incluse, quand le premier prototype a demandé dix mois, en exploitant les acquis de la maquette.

Son développement s'attache aux problèmes de fond, en faisant abstraction des échanges avec le progiciel choisi pour l'application : la maquette communique exclusivement avec le développeur. Le contexte est donc simulé à partir de cas réels, mais il est transmis par l'utilisateur de la maquette qui remplit le rôle d'interface fonctionnelle. Cette charge était beaucoup trop lourde pour être imposée aux clients sur un site beta test, ce qui explique que seul un nombre restreint de concepteurs ait participé à la validation. Cette pseudo interface a toutefois privilégié les informations indispensables au fonctionnement de la maquette. Quand les opérations sont manuelles, on cherche naturellement à diminuer leur nombre en ne retenant que les données essentielles.

Le choix de la communauté de concepteurs vise à retenir des profils d'utilisateurs représentatifs, avec des personnes qui souhaitent établir un dialogue constructif. En effet, les critiques sont intéressantes, mais les propositions restent le moteur de l'amélioration de l'existant. Pour l'évaluation de la maquette, quatre types d'utilisateurs ont été mis à contribution, dans des conditions différentes :

- l'équipe de développement du progiciel, qui comprenait deux personnes ;
 - les concepteurs, employés par la société Dolphin Integration, experts dans la manipulation du progiciel. Quatre d'entre eux ont contribué à la formalisation des règles de bon emploi et à la critique de leur implémentation dans la maquette ;
 - deux clients, enseignants en électronique, qui exploitaient le progiciel pour leurs travaux de recherche. La maquette leur a été présentée au cours d'une journée de présentation des produits commercialisés par Dolphin Integration. Dans la mesure où ils n'avaient pas défini les règles implémentées dans la maquette, il leur était plus facile de les critiquer. Par ailleurs, ils voyaient des étudiants travailler avec le progiciel, le temps d'un projet. Les enseignants ont fait part de leurs propres difficultés, et de celles rencontrées par des étudiants ;
 - deux commerciaux représentaient les nouveaux utilisateurs du progiciel, qui n'avaient pas nécessairement utilisé un simulateur auparavant.
-

L'objectif de la maquette était de faire réagir les utilisateurs potentiels du système d'aide, et ce le plus tôt possible pour valider des hypothèses et une modélisation.

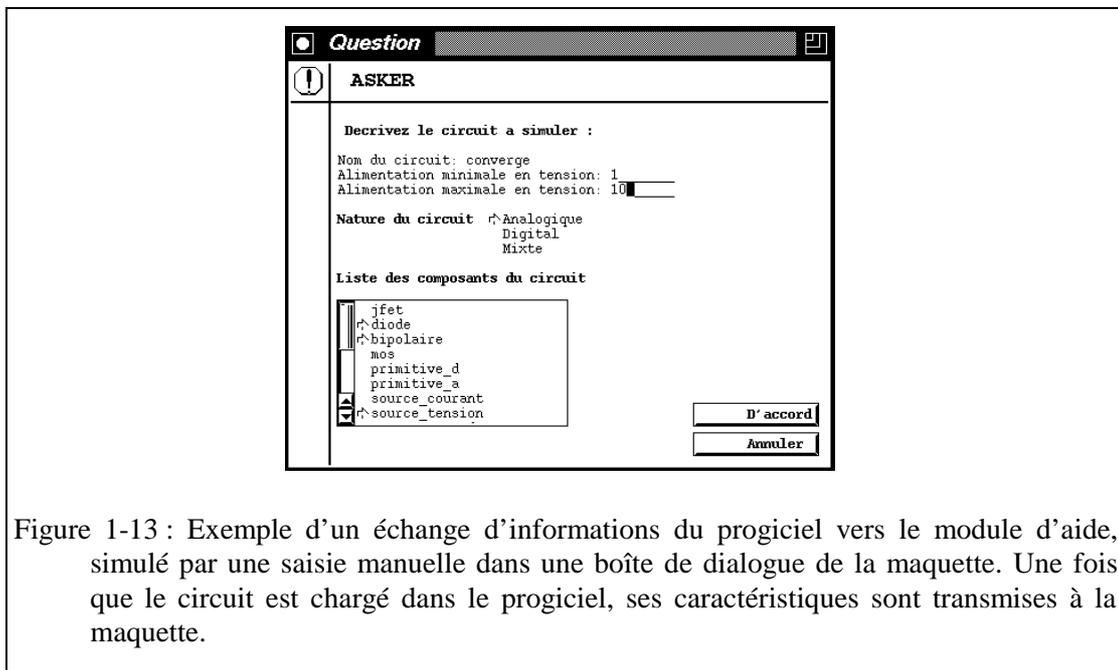


Figure 1-13 : Exemple d'un échange d'informations du progiciel vers le module d'aide, simulé par une saisie manuelle dans une boîte de dialogue de la maquette. Une fois que le circuit est chargé dans le progiciel, ses caractéristiques sont transmises à la maquette.

Pour concentrer l'attention des observateurs sur les conseils du système d'aide, il était indispensable que j'assure moi-même les échanges visibles entre le progiciel et le système d'aide, comme le montre la Figure 1-13. C'est-à-dire que je donnais, au fur et à mesure du déroulement de la session, les informations telles qu'elles auraient du être transmises au système s'il avait été intégré. Ma présence introduit certainement un biais dans l'évaluation de la maquette. En effet, les utilisateurs, se sachant observés, réfléchissaient à deux fois pour résoudre une situation d'échec. D'un autre côté, la discussion autour des conseils du système d'aide dans les situations concrètes était stimulée par l'échange avec des utilisateurs, parfois surpris de découvrir des options méconnues, alors même qu'ils étaient retenus à titre d'experts.

Le prototype, totalement intégré au progiciel, était affranchi de ma nécessaire présence pour son évaluation. En conséquence, il a été plus largement diffusé, et adressé non seulement aux précédents experts qui avaient contribué à la maquette, mais aussi à des clients sur des sites expérimentaux et à des utilisateurs néophytes.

1.2.4. Quel type d'interaction ?

Le type d'interaction souhaité par les utilisateurs a été discuté autour de la maquette. Cette dernière a permis non seulement de valider des choix, mais aussi de révéler des préoccupations qui n'avaient jusque là pas été clairement exprimées. En particulier, elle a mis en avant le refus d'une interaction qui perturbe le déroulement d'une session. Par contre, elle a privilégié une interaction discrète, pilotée par l'utilisateur.

Le bilan de l'évaluation de la maquette comprend plusieurs enseignements, qui cernent le type d'interaction attendu par les concepteurs :

- La surprise devant la diversité de l'expertise des utilisateurs. A partir du moment où le progiciel offre un grand nombre de fonctionnalités, l'expertise n'est pas synonyme d'une connaissance exhaustive, mais bien plus d'un savoir-faire attaché à chaque fonctionnalité. Cette situation est illustrée par un concepteur expert, qui utilise intensément le progiciel de manière satisfaisante, mais qui néanmoins connaît mal les fonctionnalités auxquelles il n'a pas souvent recours, du fait du type de circuit qu'il étudie. **L'expertise n'est pas globale, mais attachée à chaque fonctionnalité.** Il n'est donc pas satisfaisant de travailler sur le profil global d'utilisateur, en classifiant ce dernier comme néophyte ou expérimenté, avec des degrés divers.
- La confirmation d'une hypothèse forte, qui stipule que les utilisateurs d'un progiciel de conception sont des *concepteurs* : **ils partagent une connaissance du domaine d'application** et possèdent chacun leur savoir-faire en conception et leur expérience dans la manipulation de divers progiciels de simulation. Leur besoin consiste effectivement en une aide à l'utilisation du progiciel, et non pas une aide à la conception de manière plus générale. Cette base commune aux utilisateurs nous permet de ne pas travailler sur le profil de l'utilisateur, dont le principal intérêt est de dispenser des explications adaptées à leur niveau de compréhension.
- La priorité est donnée au travail de simulation. La présentation peu conviviale de la maquette a probablement accentué **la demande de ne pas interrompre le travail pour les besoins du système d'aide.** Les concepteurs ont clairement indiqué que l'aide était un plus qui ne devait pas leur coûter d'effort, ni de temps. En conséquence, le système d'aide peut les observer, mais pas les interrompre.
- les concepteurs revendiquent **une grande initiative**, qui ne doit pas être restreinte par le système d'aide. Si des conflits apparaissent, ils ne doivent pas pour autant bloquer la session : le signalement d'un conflit doit être discret et seul l'utilisateur décide de le résoudre ou d'ignorer ce signalement. La relation à instaurer est celle préconisée dans l'auto-contrôle en qualité : il faut que l'utilisateur devienne demandeur du contrôle pour en bénéficier.
- Les concepteurs sont réceptifs aux indications qui peuvent être appliquées immédiatement, dans le cours de leur conception. **Le contexte de travail est déterminant** pour donner des indications pertinentes, et donc efficaces. Les informations doivent être concises pour être mémorisées, tout en laissant la possibilité d'une recherche complémentaire.
- Enfin, le dernier point important sera le souci de comprendre la motivation des indications du système d'aide, et d'évaluer la pertinence de ces conseils. La volonté de s'approprier le progiciel se traduit par **une demande de justifications des indications**, pour garder l'esprit critique nécessaire à la bonne utilisation du progiciel. L'esprit critique est peut-être renforcé quand il s'agit d'une étape de vérification. A partir du moment où le concepteur remet en cause sa conception si la validation par la simulation n'est pas satisfaisante, il faut lui donner les moyens de vérifier les conseils du système d'aide. Ainsi son regard critique porte sur son travail, mais s'exerce aussi sur le progiciel. A titre d'exemple, le signalement d'un conflit doit pouvoir être justifié par le système d'aide.

En résumé, les concepteurs ne maîtrisent pas toujours bien l’outil informatique qui leur est imposé par les clients pour accomplir leur travail. Ils ont donc besoin d’une aide qui n’entrave pas leur activité, mais à laquelle ils puissent faire appel pour exploiter complètement les capacités du progiciel. L’aide doit être adaptée au travail en cours, pour donner des conseils pertinents. Le système d’aide est donc propre à un progiciel de conception et il n’a aucun rôle de formation sur le domaine de la conception.

1.2.5. Y a-t-il un besoin spécifique pour l'utilisateur néophyte ?

Le terme « utilisateur néophyte » recouvre les concepteurs qui partagent des connaissances sur le domaine d’application, mais qui ne maîtrisent pas suffisamment le progiciel pour mener à bien leur travail de manière autonome. Quand nous parlons ultérieurement d’utilisateur néophyte, il faut entendre « concepteur expert mais néophyte dans l’exploitation de l’outil ».

L’absence d’utilisateur néophyte lors de la première évaluation est avant tout liée à la finition graphique trop grossière de la maquette. Il était déjà difficile, avec les concepteurs, de faire abstraction de la forme pour discuter du mode d’interaction et du contenu, pour espérer mener une étude concluante avec les étudiants. En effet, un étudiant en électronique est un utilisateur néophyte intéressant, qui n’a pas ou peu manipulé de simulateur auparavant. Il imagine donc difficilement ce que sera l’interface, à partir de la maquette.

Dès le début de notre travail, l’introduction d’un système d’aide dans le progiciel est aussi motivé par la volonté d’ouvrir l’accès à une communauté plus large que les seuls concepteurs expérimentés en simulation. Pour cette raison, avant l’étape de la maquette, le progiciel a été introduit dans une école d’ingénieurs pour observer les difficultés rencontrées par les étudiants. Cette initiative correspondait dans le calendrier scolaire à une période de projet en électronique, où les étudiants de seconde année en équipe de 4 devaient concevoir un circuit. Cette première observation a conduit aux conclusions suivantes :

- L’observation est facilitée quand au moins deux étudiants utilisent ensemble le progiciel car ils s’expriment à voix haute : ils discutent ce qu’ils doivent vérifier, comment ils doivent procéder, et ils commentent les résultats.
- Les étudiants n’ont pas le souci de s’appropriier le progiciel, ni la préoccupation de la qualité des analyses. Dans la mesure où le projet est organisé par les étudiants, l’étude du circuit avec le simulateur n’est pas imposée. Si elle est appliquée, ce n’est que pour une validation approximative.
- La diversité des projets couvre une utilisation variée du progiciel. L’observation et la discussion avec les étudiants affinent la définition et l’enchaînement des phases de travail, dont la synthèse est faite par le schéma de phases, exposé dans le chapitre 4.

Du point de vue de la méthode de travail, cette première observation a permis de prévoir de meilleures conditions pour collaborer ultérieurement avec les étudiants. Il est en particulier apparu nécessaire de pallier la connaissance encore imparfaite du domaine de l’électronique, pour ne pas perturber l’évaluation. L’hypothèse d’une connaissance du domaine n’est vérifiée que lorsque le travail avec le progiciel porte sur des notions correctement assimilées.

Le bilan de la méthode est plus mitigé pour son apport dans l’identification des besoins des néophytes. En effet, les difficultés majeures n’étaient pas liées au progiciel en particulier, mais à l’étape de la modélisation du circuit, qui existe dans tous les simulateurs. Cette observation a néanmoins confirmé l’intérêt d’un schéma de phases pour le suivi de l’utilisateur,

structure qui sera en effet validée avec la maquette. En outre, l'observation a permis de corriger quelques difficultés liées à l'interface du progiciel, qui conduisaient à une incompréhension. Ces points noirs étaient évités par les concepteurs, qui prenaient les dispositions nécessaires. Un exemple simple est la visualisation des signaux, qui correspond physiquement à mettre un capteur sur la zone à observer. Faute de signaux précisés avant de lancer la simulation, les étudiants obtenaient un écran noir, vide de toute trace de résultats : le processus est simulé mais il n'est pas observé. Depuis, le progiciel suggère de spécifier les signaux, si aucun n'est défini.

L'évaluation du prototype, c'est-à-dire du simulateur couplé avec le système d'aide, a été menée dans des conditions favorables pour l'observation. Les étudiants, pendant deux séances de quatre heures, ont utilisé le simulateur dans le cadre de travaux pratiques de simulation, sur des sujets qu'ils avaient, en toute logique, préparé sur le plan théorique. Ayant personnellement déroulé les travaux pratiques auparavant, il était plus facile de distinguer les incompréhensions liées au simulateur, ou à la théorie mise en évidence par les simulations. Les conditions, appliquées pour l'évaluation du prototype étaient :

- Les étudiants travaillent en binôme, ce qui évite l'intrusion systématique de l'observateur et privilégie l'écoute des discussions entre les étudiants.
- Les travaux pratiques mettent l'accent sur l'utilisation du simulateur, avec les différents réglages, et leurs influences sur les résultats de simulation. Les étudiants sont toujours limités dans le temps, mais le but n'est pas de valider leur circuit.
- L'ordre des deux séances est respecté pour comparer les difficultés rencontrées. De fait, sur la seconde séance, les problèmes de prise en main sont rares : charger le circuit et commencer les analyses sont des manipulations menées plus rapidement. Les étudiants se posent alors les questions du contrôle des analyses.

Pour compléter l'observation des étudiants, un questionnaire leur a été remis à la première séance de travaux pratiques. Ils devaient le remplir en fin de deuxième séance, et le rendre avec le compte rendu des travaux pratiques. Le lecteur peut consulter le questionnaire en annexe.

Globalement, les étudiants se prêtent bien à l'évaluation car ils sont éminemment critiques. Même s'ils sont agréablement surpris par l'interface, et s'ils apprécient la manipulation interactive pour le contrôle du simulateur, ils restent très exigeants. En fait, lors de la première séance, leur regret est que le simulateur ne fait pas tout, tout seul. Cette requête est moins forte de la part de ceux qui ont fait un peu de simulation auparavant. Elle est aussi plus nuancée lors de la seconde séance, quand les étudiants découvrent l'importance du contrôle des analyses, qui est en fait le sujet des travaux pratiques. Le bilan de l'évaluation du prototype par les étudiants est essentiellement contre l'existant. Toutefois ils sont trop souvent incapables de formuler des propositions par manque d'expérience, contrairement aux concepteurs qui suggèrent des améliorations : ils s'emploient davantage à formuler des réserves. Les points essentiels de l'évaluation par les étudiants ont été :

- De soigner la cohérence des explications textuelles, en utilisant le même vocabulaire que le progiciel, et dans le cas général le vocabulaire partagé par l'ensemble des concepteurs de circuits.
- D'enrichir les explications du système d'aide, notamment sur les modèles de composants, existant sous forme de primitives dans le simulateur.

- De modifier l'intervention du système d'aide lorsqu'il existait des conflits. Ils n'appréciaient guère l'apparition systématique des boîtes de dialogue qui détaillaient le problème. Ces dialogues, chargés de trop d'informations, les perturbaient plus qu'il ne les aidaient. Depuis, le signalement d'un conflit est plus discret, et c'est l'utilisateur qui pilote sa recherche d'informations.

Leur demande de compléter le module d'aide, plutôt que de le refuser nous amène à penser qu'ils adoptent facilement ce système d'aide, et dans tous les cas ne le rejettent pas. Au final, les difficultés rencontrées par les étudiants diffèrent peu de celles des concepteurs. Néanmoins, les néophytes attendent une aide sur tous les aspects du progiciel, quand les concepteurs s'attachent essentiellement au problème du contrôle des analyses.

1.3. L'environnement du système d'aide

Clairement le manuel n'est pas une solution pour améliorer l'utilisation du progiciel, tant il est peu consulté. Le problème reste entier avec une aide en ligne, construite avec un fichier bien structuré par des liens hyper-texte. Même si cet accès est plus rapide, il ne répond pas à l'interaction attendue. D'un côté les utilisateurs veulent être avertis des capacités du progiciel. D'un autre côté, ils ne veulent pas être arrêtés dans leur travail inutilement. En conséquence, ils souhaitent n'avoir que les informations pertinentes dans le contexte courant. Par ailleurs, le système d'aide doit être disponible tout au long d'une session, qui correspond à l'étude d'un même circuit avec le progiciel.

Pour que le système d'aide intervienne de manière pertinente, il faut :

- un échange d'informations entre les trois intervenants que sont l'utilisateur, le progiciel et le système d'aide, ce qui amène la discussion sur les interfaces ;
- une exploitation de ces informations, pour sélectionner les indications qui doivent être présentées à l'utilisateur.

1.3.1. Les interfaces

Introduire un système d'aide pose le problème d'une triple interface (au minimum) entre l'utilisateur, le système d'aide et le progiciel existant.

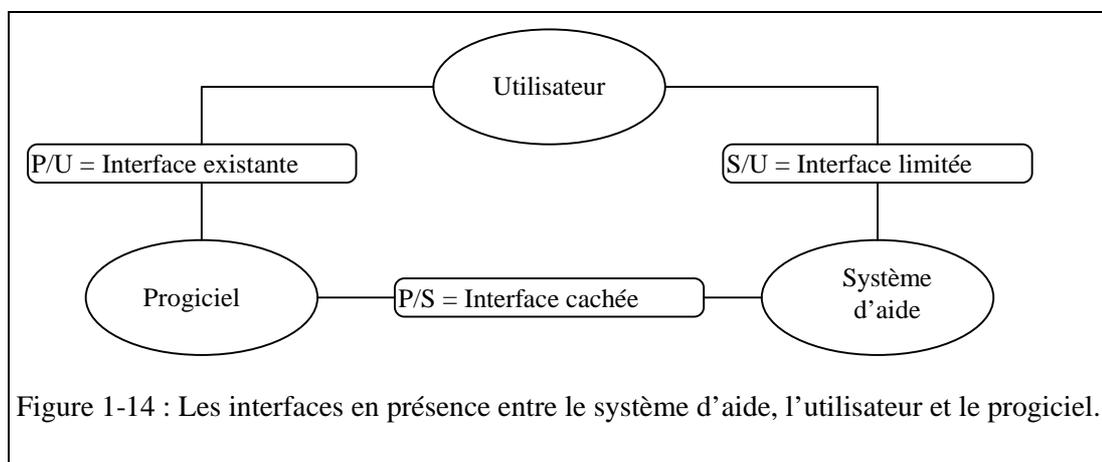


Figure 1-14 : Les interfaces en présence entre le système d'aide, l'utilisateur et le progiciel.

Dans le cadre de notre application, il nous était demandé de minimiser les modifications au niveau de la perception du simulateur par les utilisateurs. Dès lors, l'interface physique entre le système d'aide et l'utilisateur, S/U sur la figure précédente (Figure 1-14), est fondue dans l'interface entre le progiciel et l'utilisateur (interface P/U). En fait, cette contrainte va dans le même sens que la demande des concepteurs qui préfèrent une aide discrète, disponible à tout moment au cours de la session. De plus, dans la mesure où les concepteurs refusent de consacrer du temps pour indiquer au module d'aide leurs intentions et leurs problèmes, l'interface S/U est dédiée aux signalements du système d'aide vers l'utilisateur. Ces contraintes excluent la perspective d'un tuteur d'enseignement, qui imposerait un dialogue trop contraignant.

D'une manière générale, modifier une interface P/U familière n'est pas satisfaisant *a priori*. Une modification ne doit être réalisée que si l'intérêt est certain, par exemple si l'interface provoque des erreurs d'utilisation. En effet, les utilisateurs se font une idée, un modèle, du fonctionnement du progiciel, qui est basé sur l'interface P/U connue, et non pas sur le fonctionnement réel. Le risque est qu'une interface plus proche du fonctionnement réel introduise une complexité inutile.

La solution retenue dans notre application est donc de compléter l'interface P/U existante. Cependant ces modifications se limitent à l'ajout d'un menu dédié au système d'aide, et à des signalements de conflits discrets, comme le montre la Figure 1-15. Ainsi, paradoxalement, le module d'aide intervient peu dans l'interface graphique avec l'utilisateur.

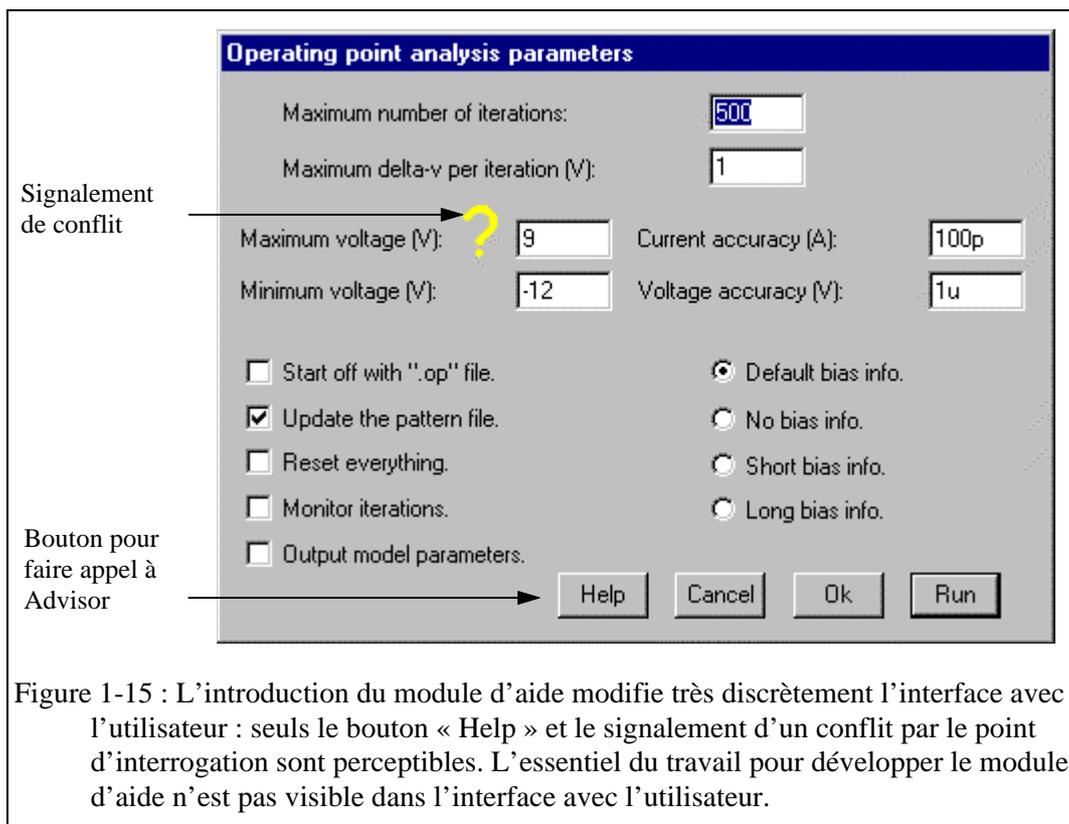


Figure 1-15 : L'introduction du module d'aide modifie très discrètement l'interface avec l'utilisateur : seuls le bouton « Help » et le signalement d'un conflit par le point d'interrogation sont perceptibles. L'essentiel du travail pour développer le module d'aide n'est pas visible dans l'interface avec l'utilisateur.

Par contre, le module d'aide dépend de la qualité de l'interface fonctionnelle P/S, qui est son seul vecteur pour recevoir des informations sur le déroulement de la session. En effet, l'hypothèse de travail est qu'il n'est pas acceptable de solliciter le concepteur en cours de tra-

vail. En conséquence, il n'est pas possible de lui adresser directement des requêtes pour satisfaire les besoins d'informations du système d'aide. A défaut d'être explicitement fournies par le concepteur, les informations devront être déduites de l'utilisation qu'il fait du progiciel, pour le suivre sans le gêner. Cette interface P/S devra être cachée, car elle n'apporte rien au travail du concepteur : c'est un enseignement inattendu de l'évaluation de la maquette, où l'interface était visible.

La méthode pour définir les données nécessaires au module d'aide consiste :

1. A définir les données indispensables pour déterminer le contexte, ensemble noté D_c ;
2. Identifier dans l'ensemble D_c , le sous-ensemble D_f des données manipulées dans le noyau fonctionnel du progiciel. Pour que le module d'aide détermine le contexte courant, il est indispensable que les données soient effectivement une copie de celles du noyau fonctionnel. A charge à l'interface P/S d'actualiser les données à partir de la référence qu'est le noyau fonctionnel ;
3. Vérifier que toutes les données de D_c peuvent être déduites à partir de D_f . Dans le cas contraire, il faudra modifier le noyau fonctionnel pour en extraire l'information voulue, et l'inclure dans l'API, « Application Programming Interface », du progiciel. Si le noyau fonctionnel n'est pas accessible, il faudra supprimer les conseils qui dépendent de cette information. Ceci illustre combien il est important de travailler avec l'équipe de développement du progiciel, pour ne pas être limité par un manque d'information.

La modélisation du module d'aide détermine les données du noyau fonctionnel utiles à son fonctionnement. Le rôle de l'interface P/S est alors de mettre ces données à disposition du système d'aide, et d'assurer leur mise à jour lorsqu'elles sont modifiées. L'interface P/S consiste donc à élargir l'API du progiciel, souvent restreinte aux besoins de l'interface avec l'utilisateur.

Il faut distinguer la présentation des éléments faite dans l'interface avec l'utilisateur, et leur abstraction dans le noyau fonctionnel. En effet, dans le cas du simulateur disponible sur les plates-formes Macintosh, Windows et Motif sous Unix, il apparaît que le noyau fonctionnel représente une proportion du code très importante par rapport à la couche d'interface. Seul 3% du code est dédié à chacune des plates-formes, le reste étant rigoureusement le même.

1.3.2. Les connaissances du système d'aide

Toutefois, il ne suffit pas de dupliquer l'information, du noyau fonctionnel vers le module d'aide via l'interface fonctionnelle P/S, pour que le module d'aide soit satisfaisant. Cette duplication de l'information est coûteuse, car elle nécessite un mécanisme de mise à jour du noyau fonctionnel vers le module d'aide. Cependant, elle présente des avantages attrayants :

- elle préserve le fonctionnement initial du noyau fonctionnel du progiciel. Les modifications consistant au maximum à compléter l'API du progiciel ;
- mais surtout, elle permet d'organiser la connaissance autour de chacune des données qui définissent le contexte, et cela dans le module d'aide.

En effet, pour exploiter correctement une donnée du noyau fonctionnel, il faut connaître non seulement sa valeur, mais aussi ses influences et ses contraintes avec les autres données.

<p>C'est précisément le rôle du noyau sémantique de représenter les contraintes et les conditions d'utilisation des données du noyau fonctionnel, et il sera détaillé dans le chapitre 5 de ce mémoire.</p>
--

Le noyau sémantique regroupe pour chacune des données, les contraintes qui ont des origines diverses :

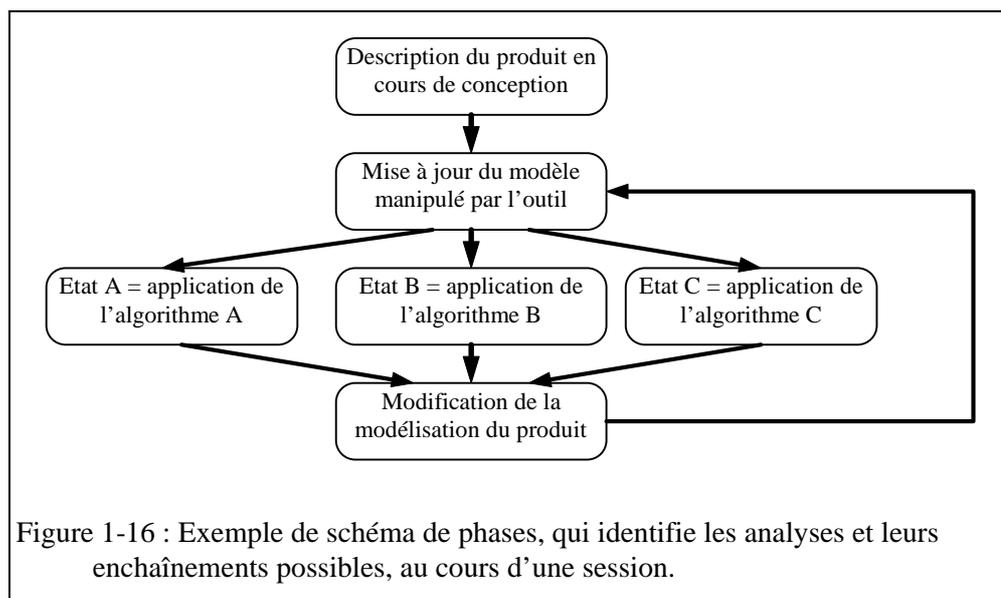
- des contraintes numériques, quand les experts estiment que la valeur du paramètre P1 doit être inférieure à celle du paramètre P2. Cette connaissance est représentée par une règle pour fixer la valeur de P1 en fonction de P2.
- des contraintes liées au processus étudié avec le progiciel, car des options dépendent de la modélisation choisie. Le module d'aide exploite donc une représentation du processus, avec ses caractéristiques ;
- des contraintes liées à la phase de travail courante, car fixer un paramètre sans influence ne peut pas résoudre une situation d'échec. Le module d'aide exploite un suivi de l'utilisateur, pour déterminer les commandes et paramètres accessibles dans chaque situation ;

Cette approche est généralisée à l'ensemble des données manipulées par le système d'aide, de manière à pouvoir vérifier les conditions d'application et le respect des règles de bonne utilisation.

Si le cœur du module d'aide est le noyau sémantique, il faut cependant apporter les informations nécessaires à son bon fonctionnement. Le flux « vital » transite par l'interface P/S, et il devra être structuré pour suivre les actions de l'utilisateur.

Le module d'aide est constitué de trois parties :

1. les explications textuelles, qui sont le reflet textuel des connaissances du noyau sémantique, destinées à l'utilisateur ;
2. le noyau sémantique, qui indique pour chaque donnée, y compris le processus étudié, l'ensemble des contraintes ;
3. un mécanisme qui doit permettre le suivi de l'utilisateur au cours d'une session. Le mécanisme proposé est nommé le schéma de phases, où une phase correspond à la mise au point d'une analyse, et à l'exploitation de ses résultats : c'est une phase de travail.



Le schéma de phases schématisé par la Figure 1-16 est un modèle du fonctionnement du progiciel. Pour que les explications et les conseils du module d'aide soient compréhensibles, ce modèle respecte la perception offerte à l'utilisateur par l'interface P/U. Ainsi, quand plusieurs phases sont accessibles, le modèle préserve ces possibilités sans contraindre un enchaînement.

Sans aborder la description du système d'aide, qui fait l'objet des chapitres suivants, son fonctionnement global est défini pour répondre aux besoins de l'utilisateur. Il consiste à déterminer la position dans le schéma de phases qui correspond à l'activité de l'utilisateur, pour évaluer les contraintes connues par le système et ainsi déterminer :

- les actions potentielles dans la phase,
- les actions candidates dont les conditions d'application sont effectivement vérifiées,
- les actions influentes susceptibles de corriger une situation d'échec.

L'utilisateur peut alors se renseigner sur les motifs du système pour considérer une action potentielle mais néanmoins pas candidate.

1.4. Les autres candidats pour une application de la méthode

La méthode appliquée au simulateur Smash pour introduire un système d'aide peut être généralisée à d'autres outils informatiques. Ce paragraphe examine les critères des progiciels susceptibles d'être complétés efficacement par un système d'aide similaire à Advisor.

- L'utilisateur reste l'acteur principal, sans déléguer au système d'aide l'expertise des choix. C'est l'utilisateur qui a le contrôle et la responsabilité des décisions. En conséquence, lors de la résolution d'un problème, l'utilisateur n'exige pas que le système fournisse la solution à appliquer. Mais plutôt, il souhaite que le système sélectionne les propositions qui pourraient être appliquées, en les justifiant.
- En effet, pour faire son choix en connaissance de cause, l'utilisateur n'a pas toujours les connaissances sur les composants de l'interface qu'il manipule. Il doit donc pouvoir les interroger, pour évaluer leur pertinence dans le contexte courant. L'objectif

du système d'aide est de permettre à l'utilisateur de s'approprier le progiciel, en exposant les possibilités satisfaisant un contexte donné.

- L'utilisateur n'a pas besoin de connaître exhaustivement le progiciel, mais il veut manipuler efficacement les fonctionnalités dont il a besoin. Cette expertise est acquise par l'expérience. Le système d'aide encourage l'utilisation des options au moment où elles sont pertinentes, facilitant ainsi une mémorisation complète, de l'option et de son contexte d'utilisation.

Le module d'aide ne vise pas à élire une solution. Il a pour objectif d'expliquer les propositions avant qu'une décision ne soit prise. Cette liberté est positive si l'utilisateur souhaite rester l'acteur principal. En conséquence, les candidats privilégiés pour une application du module d'aide sont :

- les outils de diagnostic (par exemple le diagnostic médical, où le médecin souhaite prendre *in fine* la décision),
- les outils de conception où la liberté de création est demandée pour la construction,
- les outils de vérification comme la simulation où le système d'aide prévient des limites de l'outil et suggère des voies pour résoudre un échec.

Dans ces domaines, il n'existe pas une unique bonne solution consensuelle obtenue par des experts, mais plusieurs possibilités qu'il faut prendre le temps de considérer.

Les points indispensables pour appliquer la méthode de l'introduction du module d'aide sont :

- L'observation des actions des utilisateurs. Typiquement il n'est pas difficile d'intercepter des appels de commandes dans une interface avec menus.
- Une interface fonctionnelle ouverte aux modifications nécessaires. L'interface fonctionnelle regroupe les capacités d'échanges entre le noyau fonctionnel du progiciel vers d'autres processus. La collaboration avec les développeurs du progiciel est indispensable si le noyau fonctionnel est conséquent et à plus forte raison si l'interface fonctionnelle est réduite.
- Un progiciel structuré et un fonctionnement proche de l'image donnée par l'interface avec l'utilisateur. La phase de modélisation proposée dans ce mémoire est valable pour un progiciel dont le schéma de phases représente effectivement les étapes d'une session.

En conséquence, les processus qui demandent un temps de réponse bref, ou une réponse optimale pour le contexte donné, ne sont pas susceptibles d'être concernés. Par exemple, un système de surveillance de processus markovien gagnera à s'appuyer sur le graphe d'états pour déduire avec certitude le remède optimal à appliquer : la démarche consiste alors à suivre une procédure pré-établie par les experts du processus. Dans ce cas, il n'y a pas de doute sur l'expertise, et l'utilisateur ne doit pas remettre en cause les procédures, qui fixent l'utilisation de l'outil.

Par ailleurs, les progiciels en cours de développement ne sont pas les meilleurs candidats, puisque la méthode proposée s'appuie sur l'utilisation constatée d'un progiciel existant. L'hypothèse pour appliquer la méthode est que le progiciel sera effectivement utilisé comme prévu lors de sa conception.

1.5. Le cadre de ce travail

Le besoin de l'introduction d'un système d'aide dans un progiciel de simulation de circuits électroniques a motivé ce travail. Le but était alors de répondre aux besoins des utilisateurs néophytes, expérimentés ou occasionnels, et de rendre le progiciel accessible à un ensemble plus vaste de concepteurs.

Cette étude a été réalisée en collaboration² avec l'unité de recherche de l'INRIA Rhône-Alpes (Institut National de la Recherche en Informatique et en Automatique) et la société Dolphin Integration, qui développe et commercialise le simulateur Smash. Cependant la société a aussi une activité de conception de circuits électroniques mixtes (analogiques - logiques), où des concepteurs utilisent le simulateur pour valider leur travail. Cette recherche demandait une formation pluridisciplinaire en électronique et en informatique, pour d'une part comprendre les besoins des concepteurs en électronique, et d'autre part modéliser un système et le valider par la réalisation d'une application logicielle.

Certaines caractéristiques du progiciel Smash ont indéniablement facilité l'introduction du système d'aide. Smash était déjà ouvert à des échanges autres que ceux dédiés à l'utilisateur. Par exemple, la topologie du circuit, la *netlist*, peut être obtenue à partir d'un éditeur de schémas de circuit. Plus récemment, Smash est devenu un serveur dans le cadre de la plateforme client serveur, nommée EDA Client : le simulateur met à disposition ses fonctionnalités et il peut lui-même faire appel aux serveurs disponibles. Ainsi Smash ne gère plus lui-même ses fichiers textes mais fait appel au serveur d'édition. L'équipe de développement était donc prête à mettre à disposition les informations spécifiques au fonctionnement du système d'aide, malgré le travail supplémentaire.

En dépit de ses qualités, le simulateur Smash présente un gros noyau fonctionnel mais pas d'architecture adaptée à l'échange des informations pour un suivi au cours d'une session. Comme il n'est pas acceptable de réécrire ce type d'application pour faciliter leur emploi, il est nécessaire de créer une structure pour recueillir les informations. Ces dernières se trouvent donc dupliquées et devront être remises à jour systématiquement.

Dans le cadre de ce travail, la collaboration avec la société Dolphin Integration a apporté deux atouts majeurs.

- Les idées avancées au cours de ce doctorat ont été prototypées et confrontées aux critiques des concepteurs en électronique de Dolphin, puis aux clients ayant accepté de contribuer dans le cadre d'un site expérimental. Ce développement itératif s'est concrétisé par une maquette proposée dès novembre 1995, suivie d'un tout premier prototype en octobre 1996 installé en beta test chez les clients ayant manifesté un intérêt pour l'aide contextuelle, puis d'un produit commercialisé en juillet 1997. Ainsi, la maquette nous a permis de vérifier très tôt certaines hypothèses et d'adapter nos propositions aux attentes des utilisateurs. Il était plus facile d'obtenir une réaction devant la maquette que d'extraire par la seule discussion les besoins des utilisateurs. Les beta tests ont impliqué un plus grand nombre d'utilisateurs et ont apporté de nouvelles suggestions.

² Collaboration dans le cadre d'une Convention Industrielle de Formation par la REcherche, CIFRE, avec la contribution de l'ANRT qui dépend du Ministère de la Recherche et de l'Industrie.

- L'échange permanent avec des experts en conception électronique, et en particulier avec le créateur de Smash, a permis de formaliser plus rapidement les heuristiques adaptées au contexte.

1.6. Les apports de ce travail

La démarche suivie est le premier apport de ce travail. Elle repose sur le bon sens qui veut impliquer les futurs utilisateurs dans la définition de leurs besoins, et privilégie l'écoute de leurs propositions. L'écoute passive est néanmoins insuffisante. Les deux démarches particulièrement fructueuses mises en œuvre sont :

- de s'impliquer personnellement dans l'utilisation du progiciel, pour acquérir une expertise d'utilisation, et ainsi comprendre et mieux formaliser les connaissances empiriques des concepteurs. Ceci permet aussi de définir, en connaissance de cause, la notion d'expertise en utilisation du progiciel ;
- de confronter les utilisateurs à leurs propositions, pour valider et adapter avec eux les connaissances qu'ils ont contribué à formaliser. Le développement itératif du système d'aide a permis d'affiner par deux fois son mode d'interaction dans l'environnement de travail du progiciel. La maquette, puis le premier prototype, ont fait converger l'interaction, dans sa forme et son contenu, pour encourager les utilisateurs à faire appel au système d'aide.

Le second apport est le système d'aide qui résulte de cette démarche. D'une part le résultat concret de l'application, puisque le système d'aide est intégré au simulateur et commercialisé depuis juillet 1997. D'autre part, la modélisation du système existant et des fonctionnalités futures, en cours d'implémentation ou encore au stade de spécification. La synthèse de la modélisation, avec les choix retenus, fait l'objet de ce mémoire.

Le système d'aide ne doit pas être un système expert. Il peut être généralisé aux domaines de diagnostic ou de conception, où l'utilisateur demande un conseil, plutôt qu'une solution dans tous les cas de figures en acceptant « la moins mauvaise possible ». En effet, ces domaines présentent une connaissance non consensuelle : quand il existe une solution elle est rarement unique, car les moyens d'y accéder sont bien souvent multiples et dépendent de l'expérience de chacun. Le but du système d'aide n'est pas d'imposer une des solutions possibles, mais de proposer les solutions envisageables en les expliquant pour laisser l'utilisateur choisir celle qui lui semble la plus adaptée. Il ne suffit donc pas de proposer une solution, il faut aussi pouvoir la justifier par rapport au contexte courant. La justification est l'explication de la règle ayant mené à retenir cette solution.

Les points essentiels de la modélisation sont :

- Un système réactif, c'est-à-dire qui adapte ses propositions au contexte courant.
- Un raisonnement local, mis en valeur avec une modélisation par objets. Chaque objet modélise une donnée manipulée par le système d'aide. Les connaissances qui déterminent les conditions d'application, ou d'efficacité de cette donnée sont modélisées par les méthodes de l'objet. Par exemple, un objet qui modélise un paramètre déter-

mine, dans un contexte donné, si il a présentement une influence. Le système réactif est le résultat d'un ensemble d'objets « réactifs ».

- Un système qui expose à l'utilisateur les connaissances utilisées, et ce d'autant plus facilement que chaque objet mémorise les résultats de l'application de ces connaissances.
- Un système qui puisse dans le futur être enrichi par les concepteurs experts qui l'utilisent. Là encore, le raisonnement local facilite la modification ou l'ajout de connaissances.

Ce dernier point n'est pas encore implémenté. Néanmoins, la modélisation intègre les éléments nécessaires à son introduction. En effet, si le système d'aide n'a pas les moyens de dépasser le palier des connaissances consensuelles aujourd'hui implémentées dans le système d'aide, nous retrouvons l'inconvénient d'un système rigide, incapable de gérer des exceptions ou des spécificités. Les concepteurs souhaitent en effet un système d'aide qui prenne en compte leur savoir-faire capable de résoudre un problème. Ces nouvelles connaissances ont pour caractéristiques :

- de pouvoir être en conflit avec les heuristiques connues du système,
- de ne pas être validées par d'autres personnes que l'utilisateur qui apporte son savoir-faire,
- de ne pas avoir été formalisées par rapport au schéma de phases,
- et enfin d'être spécifiques à un type de travail.

L'ajout de ce complément de connaissances doit donc être fait dans une structure distincte de celle qui regroupe aujourd'hui les heuristiques. L'utilisation du raisonnement à partir de cas est une solution adaptée pour constituer un système hybride.

Cette perspective prolonge l'implication de l'utilisateur, qui non seulement décide des conseils dont il a besoin, mais qui pourra aussi compléter les connaissances du système d'aide pour répondre à ses besoins, ou à ceux de ses collègues.

Chapitre 2

2. Les éléments de solution

L'introduction d'un système d'aide contextuelle à l'utilisation d'un progiciel de conception se situe aux confluent de diverses activités de recherche :

- La définition du contexte, nécessaire pour apporter une aide contextuelle pertinente.
- La prise en compte de l'utilisateur avec d'une part le suivi de ses actions et d'autre part son profil pour modéliser ses connaissances. Le suivi des actions soulève l'intérêt pour les modèles de tâches et la reconnaissance d'intentions.
- L'acquisition de la connaissance sur l'emploi du progiciel et sa représentation qui doit être adaptée au type de raisonnement mis en œuvre.
- L'enrichissement des connaissances du système d'aide est ici abordé comme la mémorisation d'une nouvelle expérience, organisée dans une base de cas. Les apports du raisonnement à partir de cas (RàPC) dans notre étude sont la construction de la base de cas et la phase de remémoration. L'adaptation des cas connus pour répondre à une nouvelle situation est réalisée en collaboration avec l'utilisateur.

Des travaux ont été réalisés dans ces domaines de recherche et nous précisons ici leurs apports pour notre sujet.

2.1. Une définition de l'aide contextuelle

La conception d'un système d'aide contextuelle nécessite de définir :

- le contexte retenu,
- une interface entre le progiciel et le système d'aide pour tenir compte de la modification du contexte au cours d'une session.

Avant de déterminer les méthodes pour garder un contexte en phase avec les actions de l'utilisateur, il faut identifier les connaissances sur lesquelles repose le fonctionnement souhaité du système d'aide.

Le contexte est l'ensemble des connaissances nécessaires pour apporter des informations pertinentes au moment opportun sur le travail de conception en cours. Selon le point de vue adopté, le contexte sera défini par des connaissances concernant :

- l'utilisateur, pour identifier les explications qu'il peut appréhender,
-

- le domaine de conception, pour déterminer les actions de l'utilisateur qui sont possibles, en fonction de l'historique de la session,
- le but de l'utilisateur, déclaré ou déduit par l'étude de son comportement.

Ces points de vue ne sont pas exclusifs mais ils peuvent être abordés séparément dans un premier temps.

2.1.1. La modélisation de l'utilisateur

Comme le système d'aide est destiné à faciliter le travail de l'utilisateur, il est légitime d'impliquer ce dernier dans la définition du contexte. Le but est de tenir compte de ses connaissances afin de lui proposer des messages compréhensibles. Cette démarche est courante dans un système pour l'enseignement, où les tuteurs intelligents s'assurent de la compréhension de l'utilisateur. Ainsi le système Scent de Mc Calla et Greer [Mc Calla et al.92] construit le modèle de la stratégie de l'étudiant pour la commenter. Cependant cette approche est adaptée aux systèmes qui emploient leurs connaissances de manière jugée optimale, ou tout au moins d'une manière à laquelle l'utilisateur doit se conformer. L'hypothèse implicite est donc que le système est plus compétent que l'apprenant.

Dès 1987, Bernard Senach [Senach87] cherche à appliquer cette démarche aux systèmes d'aide à l'utilisation. Il pose le problème de la gestion de contexte dans le cadre de tâches ouvertes comme la conception, où il n'est pas souhaitable de contraindre l'utilisateur dans une démarche. En effet, l'organisation de travail du concepteur ne correspond pas nécessairement avec l'enchaînement de tâches prévues par un expert. Le concepteur n'est pas un apprenant, mais un néophyte dans l'emploi d'un progiciel. Il revendique sa propre habitude de travail. Bernard Senach propose d'introduire une méta-communication et une modélisation cognitive de l'utilisateur pour pallier les insuffisances d'une gestion de contexte, qui ne peut reposer sur le seul suivi de l'activité dans le cas de tâches ouvertes. Cette modélisation cognitive englobe le profil de l'utilisateur, l'identification de ses connaissances, la représentation mentale de la situation et le modèle de l'activité. Elle doit permettre de comprendre ce que fait l'utilisateur, et d'évaluer l'adéquation des représentations mentales afin d'expliquer les problèmes. Là encore, l'hypothèse implicite est que le système identifie la représentation mentale de chaque utilisateur à tout moment de la session de travail. Cette hypothèse est d'autant plus difficile à vérifier que la notion abstraite de représentation mentale ne peut se contenter d'une modélisation grossière. Or il n'existe pas de modélisation complète du fonctionnement complexe de l'utilisateur : un modèle ne s'attache qu'à quelques aspects, et a donc des limites inhérentes.

Les modèles existants nous apportent néanmoins leurs enseignements, notamment la théorie de l'action de Norman [Norman86], et le modèle de Rasmussen [Rasmussen86] reconnu par exemple dans les systèmes de surveillance [Kolski93]. Ce dernier présente le fonctionnement de l'utilisateur en quatre étapes séquentielles : la détection d'événements, l'évaluation de la situation, la prise de décision, et enfin l'action. La détection d'événements pose le problème du signalement par le progiciel des erreurs avec leurs différents niveaux de gravité (du simple message ou « warning » à l'erreur qui bloque le système « erreur fatale »). Le système d'aide à l'utilisation d'un progiciel est attendu pour l'évaluation de la situation et pour aider la prise de décision. L'action reste l'apanage de l'utilisateur, seul décideur final. Rasmussen identifie trois types de comportements : basé sur l'entraînement (réflexe), basé sur les règles (procédural) et basé sur la connaissance (savoir). Ces trois types de raisonnement coexistent dans un souci d'économie cognitive et d'efficacité de l'action sur le contexte courant. Ce-

pendant, il n'est pas aisé d'identifier le type du comportement de l'utilisateur avec ce modèle qui reste trop général. Il ne permet pas de dépasser la classification en types d'utilisateur. En effet, une approche grossière consiste à définir l'utilisateur comme un expert, un utilisateur intermédiaire ou un néophyte. Il s'avère néanmoins nécessaire d'affiner cette distinction sur deux plans :

- Le domaine d'expertise, qui s'entend souvent comme une connaissance du domaine d'application. Cependant pour être performant, un utilisateur doit aussi maîtriser l'outil informatique. Cela est d'autant plus difficile que les outils varient d'un projet à l'autre, sans compter qu'ils évoluent dans le temps : l'expertise n'est pas définitivement acquise.
- Le niveau d'expertise de l'utilisation du progiciel, qui n'est pas lié à l'expertise sur le domaine d'application. En effet, si le travail fait appel à des commandes rarement manipulées, l'expert n'est pas un utilisateur expert : il ne domine plus l'interface du progiciel. Cette constatation conduit à étudier l'expertise de l'utilisateur pour chaque fonctionnalité de l'outil [Berthomé et al.95], et à la construction dynamique du profil de chaque utilisateur en tenant compte de ses difficultés pour chaque fonctionnalité.

Dans la théorie de l'action, D. Norman avance l'importance de la représentation mentale, ou de l'image, que l'utilisateur élabore à propos de l'outil. C'est en effet son interface qui est le vecteur de construction de cette image. Ce point justifie à lui seul les efforts déployés pour penser l'interface homme-machine au stade du développement d'un progiciel. Notre problème étant d'introduire un système d'aide dans un progiciel existant, en limitant les modifications de l'interface existante, nous n'abordons pas cet aspect de l'ingénierie de l'interface. Cette image est cependant une notion importante que le système d'aide doit prendre en compte. Comme la modélisation de l'utilisateur ne nous permet pas d'identifier l'image qu'il peut se faire, le système d'aide repose directement sur l'interface du progiciel.

Pour la modélisation de l'utilisateur, il faut tenir compte de ce qu'il perçoit du fonctionnement du progiciel, qui ne correspond pas nécessairement à son fonctionnement réel et complexe. L'interface est aussi un filtre pour ne pas compliquer inutilement la prise en main par l'utilisateur. C'est un réel atout si le filtre donne une image cohérente, même si elle est incomplète : de cette manière les actions de l'utilisateur n'entrent pas en conflit avec le fonctionnement du progiciel.

2.1.2. Les connaissances du domaine d'application

Le système d'aide contextuelle doit être à même de déterminer les actions de l'utilisateur possibles, recommandées ou inutiles. Bien que nous limitions notre étude à l'aide à l'utilisation du progiciel, l'aide contextuelle reste étroitement liée au domaine d'application. En effet le processus étudié, comme le circuit dans notre application, fixe en partie les fonctionnalités pertinentes, à l'image d'un éditeur adapté soit à la réalisation d'un dessin, soit à l'édition d'un tableau. C'est pourquoi les caractéristiques du processus étudié avec le progiciel sont à prendre en compte dans la définition du contexte.

Pour notre domaine d'application qu'est la simulation de circuits électroniques, les connaissances du domaine ont été essentiellement obtenues par une utilisation intensive du simulateur et par un échange permanent avec des utilisateurs experts, et l'équipe de développement du progiciel de conception. Néanmoins, il ne faut pas négliger l'apport des connaissances formalisées en dehors de notre travail. La première source d'informations formalisées

sur le fonctionnement du simulateur sont le manuel de l'utilisateur et celui de référence, [Descleves95]. Trop rarement consultés par l'utilisateur, ils renferment non seulement une description détaillée des fonctionnalités, mais surtout des recommandations pour mener à bien des analyses qui se révèlent délicates. Les autres sources d'informations formalisées étaient moins spécifiques au progiciel, puisqu'il s'agit d'ouvrages dédiés à la simulation de circuits électroniques. Ils abordent la question du choix des algorithmes de simulation. Outre la description des algorithmes, ces ouvrages détaillent le rôle des paramètres qui les contrôlent. L'ouvrage de Saleh, Jou, & Newton [Saleh et al.94] traite en particulier la méthode de résolution de relaxation, qui est justement une option méconnue du simulateur, car elle n'est performante que sur un certain type de circuit. Des comparaisons et des spécificités d'autres algorithmes sont discutées par K.S. Kundert [Kundert95]. Par ailleurs, ce dernier ouvrage apporte des suggestions pour définir le déclenchement de l'aide préventive. Ainsi suggère-t-il de définir des plages de valeurs raisonnables pour certains paramètres clés. Cette idée est développée dans notre système d'aide en indiquant que les plages choisies sont adaptées au domaine de la micro-électronique. A terme, il faudrait que l'utilisateur définisse lui-même ces plages.

Les connaissances du domaine d'application interviennent aussi bien dans la logique de fonctionnement que dans la logique d'utilisation du progiciel, toutes deux définies par Richard [Richard83]. Cette distinction formalise la séparation entre le manuel de référence et le manuel de l'utilisateur livrés avec le progiciel :

- Le manuel de référence définit chaque commande, en précisant sa fonctionnalité, ainsi que les contraintes souvent liées au type de processus étudié. Il est donc intéressant de représenter dans le système d'aide les contraintes attachées à chaque fonctionnalité, afin de signaler un conflit avec le processus.
- Le manuel de l'utilisateur est lui, destiné à une prise en main du progiciel, en décrivant une procédure à suivre pour mener à bien une action. Ces procédures ne sont pas figées et peuvent nécessiter des aménagements en fonction du processus en cours. Ainsi, les exemples apportés par le manuel de l'utilisateur sont plus une indication qu'une méthode stricte à appliquer. Cette liberté dans l'application d'une procédure est une source de difficulté pour répondre aux questions du type « comment atteindre tel but ? », « quelle action appliquer ? ».

Il existe plusieurs attitudes pour répondre à de telles questions. Soit le système d'aide est souhaité expert et il peut imposer ses méthodes à l'utilisateur, soit le système d'aide se présente comme un conseiller qui suggère les possibilités qu'il connaît, sans prétendre détenir la connaissance exhaustive.

La solution d'un système d'aide expert est une démarche appropriée pour les systèmes d'enseignement qui s'adressent à des apprenants dont les connaissances sont vues comme un sous-ensemble des connaissances du système. Cette couverture des connaissances par un expert est appelée la méthode « overlay ». C'est ainsi que A. Cawsey [Cawsey92] construit le modèle de l'apprenant. Cette méthode est aussi appliquée par A. Berthomé-Montoy [Berthomé et al.95] pour définir dynamiquement le profil de l'utilisateur, selon que ce dernier tient compte ou non des contraintes d'emploi. Comme le manuel de référence donne les connaissances exhaustives sur chaque fonctionnalité, cet usage de la méthode overlay dans la limite de la logique de fonctionnement est tout à fait adapté. La logique d'utilisation quant à elle pose le problème ardu d'aider l'utilisateur sans le contraindre à suivre une procédure qui ne correspond pas nécessairement à son organisation de travail.

Quand le système d'aide se présente comme un conseiller, l'utilisateur reste le décideur final. L'intérêt du système d'aide est alors d'expliquer pourquoi il retient une suggestion en fonction du contexte, en s'appuyant donc sur des connaissances du domaine. Selon le domaine d'application et les caractéristiques des utilisateurs, il est important de cerner le type d'explications attendues comme le notent C. Amergé et O. Corby [Amergé et al.94]. Cette notion d'explication apparaît indispensable dans le domaine de la conception, afin de laisser à l'utilisateur l'initiative de demander des compléments d'information. Le mécanisme de génération d'explications pourra être une étape suivante vers le but ultime qui est d'obtenir une négociation entre l'utilisateur et le système d'aide. Le lecteur pourra consulter les travaux de B. Safar sur le système « Pourquoi-pas ? » [Safar90], ainsi que ceux du groupe GENE [GENE97], pour la production d'explications négociées avec une interface graphique dédiée. Cependant cette négociation demande à terme une interface en langage naturel et elle sort aujourd'hui de notre champ de travail.

L'importance de l'explication est renforcée par l'attitude critique du système d'aide pour signaler les conflits. Même lorsque la critique se limite à la logique du fonctionnement, elle n'est pas systématiquement pertinente. Par exemple, alors que le processus étudié est encore incomplet, l'utilisateur peut anticiper des actions nécessaires par la suite. La logique de fonctionnement détectera en conséquence un conflit momentané, dont l'utilisateur ne voudra pas tenir compte. Comme le remarque G. Fischer dans [Fischer et al.91], les concepteurs supportent mal l'intrusion des critiques dans le cours du travail, surtout quand les critiques sont exclusivement négatives car liées à des conflits. Il est donc important d'argumenter les critiques en précisant dans quelles conditions elles sont déclenchées. Le travail de conseiller du système d'aide ne doit pas être une entrave, au risque d'être refusé en bloc par l'utilisateur. Dans le cadre de la logique d'utilisation, la critique est beaucoup plus délicate. Autant les actions redondantes, ou interdites, dans un contexte donné peuvent être expliquées, autant le conseil d'une action à faire n'est pas facile à étayer. La difficulté est essentiellement liée au statut de conseil, et non d'expert, du système d'aide qui ne peut pas proposer une solution sans indiquer ses avantages et inconvénients. Il n'existe pas toujours une solution, et si elle existe, elle n'est probablement pas unique. Dans le sous-chapitre suivant, « L'observation des actions de l'utilisateur », nous discutons des travaux pour la reconnaissance d'intentions, qui repose sur le suivi de l'utilisateur.

Nous avons ainsi mis en évidence le rôle des connaissances du domaine dans la définition du contexte. Elles interviennent dans la définition de la logique de fonctionnement et de la logique d'utilisation pour tous les outils non génériques, c'est-à-dire pour lesquels les caractéristiques du processus étudié participent à la définition du contexte.

2.1.3. Le suivi de l'utilisateur

Indépendamment de la méthode adoptée pour obtenir le suivi des actions de l'utilisateur, il est naturel d'en tenir compte pour la définition du contexte, afin de détecter les actions redondantes et inutiles. Plus important, il est à la base de la reconnaissance d'intentions. Le suivi permet de savoir ce que fait l'utilisateur, et si la notion de but est introduite, peut permettre de déduire ce que souhaite faire l'utilisateur. Le suivi des actions permet en effet au système d'aide de travailler sur l'historique des actions. A condition que des informations pertinentes puissent être extraites de l'historique et de tout le contexte, le système d'aide peut alors fournir des indications à l'utilisateur. La qualité des conseils dépend ainsi de la prise en compte

dans le contexte de la succession des actions. Le paragraphe suivant est consacré à la modélisation des connaissances nécessaires au suivi.

2.2. L'observation des actions de l'utilisateur

Après avoir précisé l'importance du suivi dans la définition du contexte, nous nous penchons sur les modélisations proposées pour réaliser ce suivi. Les deux approches envisagées sont le recours au modèle de tâches qui privilégie la logique d'utilisation, et la modélisation du fonctionnement du progiciel qui se concentre sur l'image donnée par l'interface.

2.2.1. Le modèle de tâches

L'approche du modèle de tâches est préconisée en ingénierie de l'interface homme-machine lors du développement d'un outil, comme l'explique J. Coutaz dans son cours « Interaction Homme-Machine, conception, réalisation, évaluation » de 1996-1997. Le modèle de tâches doit simplifier la réalisation d'un travail précis en indiquant à l'utilisateur une procédure à suivre. Du point de vue de la modélisation, la dite procédure est présentée comme un ensemble de tâches séquentielles, où une sous-tâche est le résultat soit d'une spécialisation d'une tâche principale comme dans MAD de Scapin [Scapin et al.89], soit d'une décomposition en tâches élémentaires proposée par Jutta Willamowski [Willamowski94]. L'acquisition des connaissances consiste alors à répertorier l'ensemble des buts (logique d'utilisation) que l'utilisateur peut réaliser avec le progiciel considéré, puis à planifier ce travail avec des utilisateurs experts pour définir la procédure en termes d'actions successives. C'est cette approche que propose J.-C. Tarby [Tarby94] en s'appuyant sur la distinction entre la logique d'utilisation et la logique de fonctionnement. A chaque but correspond un plan, qui ne sera pas nécessairement unique. La planification peut en effet tenir compte des alternatives, quand plusieurs méthodes sont validées par les experts. L'approche du modèle de tâches impose une identification claire du but de l'utilisateur, et surtout impose que l'utilisateur se conforme au plan proposé.

La première étape est donc de permettre à l'utilisateur d'énoncer son but, choisi parmi ceux connus par le système. Soit l'utilisateur peut directement le sélectionner, et observer graphiquement les alternatives proposées pour l'atteindre, comme Jutta Willamowski le permet dans son système grâce à une interface dédiée [Willamowski94]. Soit une reconnaissance d'intentions efficace doit déduire le but poursuivi par l'utilisateur en observant ses actions. Cette reconnaissance d'intentions est délicate dans la pratique car, à partir des actions concrètes, il faut extraire le concept de but. Elle est d'autant plus ardue qu'il existe un nombre important d'alternatives pour réaliser un but. De surcroît, les erreurs de manipulation de l'utilisateur sont une source de mauvaise identification du but poursuivi.

Les contraintes du modèle de tâches sont plus fortes encore au niveau du respect du plan sélectionné. A moins que l'utilisateur ne revendique aucun degré d'initiative, le strict respect d'un plan est incompatible avec une activité de conception où l'esprit critique est valorisé. B. Senach parle de tâches ouvertes, où la décomposition en sous-tâches d'un expert (du domaine et de l'interface) ne correspond pas nécessairement avec l'organisation de travail de l'utilisateur. Un concepteur, expert du domaine mais pas nécessairement de l'interface, n'accepte pas de se voir contraint dans ses actions par un système d'aide. Mais il pourra cependant utiliser des directives comme des indications, ce qui rappelle l'importance des explications qui accompagnent les interventions du système d'aide. Une seconde limite pertinente

sur le modèle de tâches appliqué à une activité de conception est avancée par C. Mais [Mais88]. En s'intéressant aux utilisateurs occasionnels, C. Mais met en évidence leur souci d'opérationnalisation, devant leur souci d'optimisation : la priorité est accordée au résultat du travail (réaliser une fonction qui marche) et non pas aux moyens pour y parvenir. Les moyens pourront ultérieurement être améliorés mais l'expertise de l'interface du progiciel n'est pas prioritaire. Cette attitude est d'autant plus courante que les concepteurs sont amenés à utiliser des progiciels différents pour chaque projet. Il est donc particulièrement important de ne pas signaler comme une erreur tout écart vis-à-vis du plan retenu par le système, au risque de perturber plus que d'aider l'utilisateur. Il est ici clair que le modèle de tâches présente une rigidité mal venue pour toute activité où l'utilisateur est aussi un expert du domaine. S'il existe une solution optimale qui sort du plan figé, elle risque de ne pas être retenue. Le plan est aussi une oeillère dans ce sens qu'il empêche la découverte de la procédure optimale. Par ailleurs, lorsque l'organisation de travail du concepteur diffère des plans connus par le système, l'utilisateur doit fournir l'effort de se conformer au plan choisi par les experts de l'interface, et il ne pourra pas satisfaire son souci d'opérationnalisation.

2.2.2. La modélisation du fonctionnement du progiciel

Le suivi de l'utilisateur ne peut pas reposer sur la seule observation de ses actions. Il faut définir un cadre pour exploiter ces observations brutes, afin de les interpréter de manière satisfaisante, c'est-à-dire par rapport au contexte qui a été précédemment explicité. Nous avons souligné les limites de l'application du modèle de tâches comme support d'interprétation pour une tâche de conception. Les limites sont liées au souci d'initiative du concepteur qui ne tolère pas l'imposition d'une procédure.

Cependant, il est clair que des contraintes existent, au moins au niveau du fonctionnement du progiciel. Pour ne pas ajouter de contrainte supplémentaire tout en évitant à l'autre extrême une utilisation incohérente, G. Forslund propose de travailler sur des systèmes d'aide qui donnent des conseils [Forslund95] et respectent l'initiative de l'utilisateur. L'activité du système d'aide est envisagée comme une **intervention minimale** dans le déroulement du travail du concepteur. Le but est de prévenir des situations incohérentes, et de signaler les actions conflictuelles ou inutiles. G. Forslund s'appuie sur les travaux de G. Fischer et al [Fischer et al.91] sur l'intérêt de la critique. Pour que l'utilisateur puisse exploiter cette critique, nous répétons ici la nécessité de l'accompagner d'explications.

Le cadre que nous proposons pour mettre en application cette intervention minimale de l'aide est une description fonctionnelle du progiciel que nous nommons le schéma de phases, auparavant nommé le schéma d'états [Duprez96]. Chaque phase correspond à un type d'étude du processus avec ses actions spécifiques, à l'image du mode de travail dans un document, où les modes dessin, texte et tableur cohabitent. Cependant dans un progiciel de conception, il peut exister des enchaînements impératifs, incompatibles, ou même automatiques. Le schéma de phases permet de mettre en évidence ces enchaînements entre les phases, avec les transitions possibles. Sans préjuger des connaissances de l'utilisateur, le schéma de phases groupe des informations liées au progiciel et au processus étudié. Ainsi la modélisation du fonctionnement du progiciel repose sur la description des phases dans lesquels l'utilisateur se place selon ses actions.

L'utilisateur est donc contraint par les impératifs du progiciel, en respectant les enchaînements entre les grandes phases de travail. Dans notre application, la phase initiale est un chargement du circuit qui conditionne le début d'une session. En cours de session, il est aussi nécessaire d'obtenir un état d'équilibre du processus, nommé le point de fonctionnement, avant d'envisager une analyse fréquentielle. Mais dans chaque état, aussi bien pour la recherche du point de fonctionnement, que pour l'étude fréquentielle, il choisit sa méthode de travail. Cette approche d'intervention minimale est adaptée aux progiciels dans lesquels le résultat d'une action n'est pas prévisible et où la méthode d'essai et d'erreur prévaut. Ceci est notamment le cas pour les progiciels faisant appel à l'analyse numérique. En effet, la garantie d'une solution n'existe pas *a priori* : il faut appliquer l'algorithme pour savoir s'il converge. De plus, quand il y a convergence autour d'une solution, seul le concepteur peut la valider, selon ses critères de satisfaction comme la précision et/ou la vitesse. Dans les cas où aucune connaissance ne permet de prédire la satisfaction de l'utilisateur, soit par l'absence de prédiction sur le résultat, soit par l'incapacité d'évaluer ce résultat, le système d'aide doit impliquer le concepteur. Cette implication de l'utilisateur renverse l'approche d'un système d'aide chargé d'évaluer les connaissances de l'utilisateur et de contrôler ses actions, pour proposer un système qui observe les actions pour fournir des conseils justifiés à la demande de l'utilisateur.

2.3. Une aide pilotée par l'utilisateur

La recherche d'une aide sur l'utilisation d'un progiciel n'est pas systématique. Nous envisageons ici les possibilités pour quelle soit mieux utilisée, à partir de l'observation des besoins de l'utilisateur.

Quand un concepteur doit employer un progiciel pour réaliser son travail, la tendance est de le manipuler, plutôt que de consulter consciencieusement son manuel. Cette démarche spontanée met en avant l'initiative de l'utilisateur et l'importance de l'apprentissage par l'action : les éléments de l'interface sont d'autant mieux maîtrisés qu'ils sont manipulés. Cependant exécuter une commande n'est pas synonyme de pleine compréhension : l'utilisateur n'a pas nécessairement perçu les conditions d'application, ni même les paramètres influençant l'exécution. En dehors d'un échange avec l'utilisateur, l'évaluation de la compréhension est malaisée pour un observateur humain, et à plus forte raison pour un système informatique qui n'appréhende l'utilisateur qu'à travers un modèle. Or, dans la pratique, la solution d'un dialogue pour vérifier sa compréhension n'est pas envisageable. En effet, le concepteur tolère mal la présence d'un observateur humain, et *a fortiori* les interventions d'un système informatique. Il apparaît rapidement que les interventions orales menées pour vérifier sa compréhension excèdent le concepteur au point de compromettre une future collaboration pour définir ses besoins. Rappelons que ce comportement est naturel face à l'introduction d'un système d'aide : il ne s'agit pas pour le concepteur de perdre du temps à cause d'un module tout à fait mineur par rapport à la fonctionnalité du progiciel.

2.3.1. Limiter le recours au support technique

A défaut d'envisager un dialogue avec l'utilisateur, nous observons sa méthode de travail. Dans la pratique, le concepteur cherche de l'aide quand il se trouve dans une situation d'insatisfaction. Soit il ne parvient pas à exécuter une action et il se trouve en situation de blocage, soit il ne comprend pas les résultats du progiciel. Dès lors, il cherche des informa-

tions par une consultation du manuel, ou plus fréquemment, contacte l'assistance téléphonique, pour soumettre son problème à une personne compétente.

De nombreux systèmes ont été développés pour assister les opérateurs des supports technique et notamment en faisant appel au raisonnement à partir de cas, discuté ultérieurement. Le rôle de l'opérateur est de servir d'interface entre l'utilisateur et le processus de résolution du problème en décrivant le problème de manière compréhensible par le système. Cette étape de mise en forme est importante si la diversité d'expression d'un même problème est considérée. Les auteurs de CARET [Shimazu et al.94], un système de recherche de cas travaillant sur une base de données relationnelles, retiennent ainsi les approches suivantes: l'explication pas-à-pas des commandes à effectuer, l'explication par diagramme ou modèle interne du produit que s'est fait l'utilisateur (et qui n'est pas toujours correct), l'explication « graphique » basée sur les modifications de l'interface utilisateur consécutives à une action. Les auteurs précisent que la réponse du système, via l'opérateur, doit respecter l'approche de chaque utilisateur. Quand le système extrait une proposition, l'opérateur l'adapte si nécessaire au problème courant pour offrir une solution finale. Ce recours à une aide personnalisée est certes un confort, mais elle présente l'inconvénient de ne pas donner à l'utilisateur les informations pour résoudre par lui-même un problème similaire. Après avoir cité différents types de systèmes d'aide pour les opérateurs, Rewari [Rewari93] note qu'il serait intéressant d'offrir directement à l'utilisateur un système d'aide. Cette approche rejoint notre préoccupation de mettre à disposition du concepteur expert du domaine les éléments explicatifs pour maîtriser le progiciel. Elle répond aussi au souci de limiter le recours au support technique en favorisant des méthodes dites « proactives » [Tourniaire et al.97], qui apportent une réponse avant que l'utilisateur ne demande du support. C'est prendre le problème à sa source.

2.3.2. Impliquer l'utilisateur

Une situation d'insatisfaction n'est pas souhaitable et doit être anticipée par un système d'aide préventif. Cependant elle présente l'avantage d'amener le concepteur dans une position de recherche d'informations. Cette attitude est nettement plus positive à l'égard des propositions d'un système d'aide, à condition que celui-ci apporte des informations pertinentes. En effet, maintenant arrêté dans son travail, le concepteur cherche la solution la plus efficace pour dépasser cet état, ou au moins une possibilité même si elle n'est pas optimale. A charge au système d'aide de suggérer des possibilités pour résoudre cette insatisfaction, en les expliquant dans le contexte courant. Les conditions sont alors réunies pour bénéficier de l'apprentissage par l'action, où l'utilisateur résout un problème en exploitant les conseils du système d'aide. Cette approche est aussi présentée en enseignement assisté par ordinateur par J.F. Nicaud [Nguyen-Xuan et al.95] où il précise que « Le système doit laisser à l'élève une part de responsabilité dans une action suggérée ».

L'implication de l'utilisateur peut donc être très forte, sans faire appel à une modélisation et à une évaluation de son profil. Un bon exemple est le module d'aide du système 7.5 du Macintosh™ d'Apple, où l'utilisateur choisi lui-même les informations qu'il souhaite consulter. Les figures suivantes (Figure 2-1 et Figure 2-2) donnent un aperçu de l'interaction avec l'utilisateur.

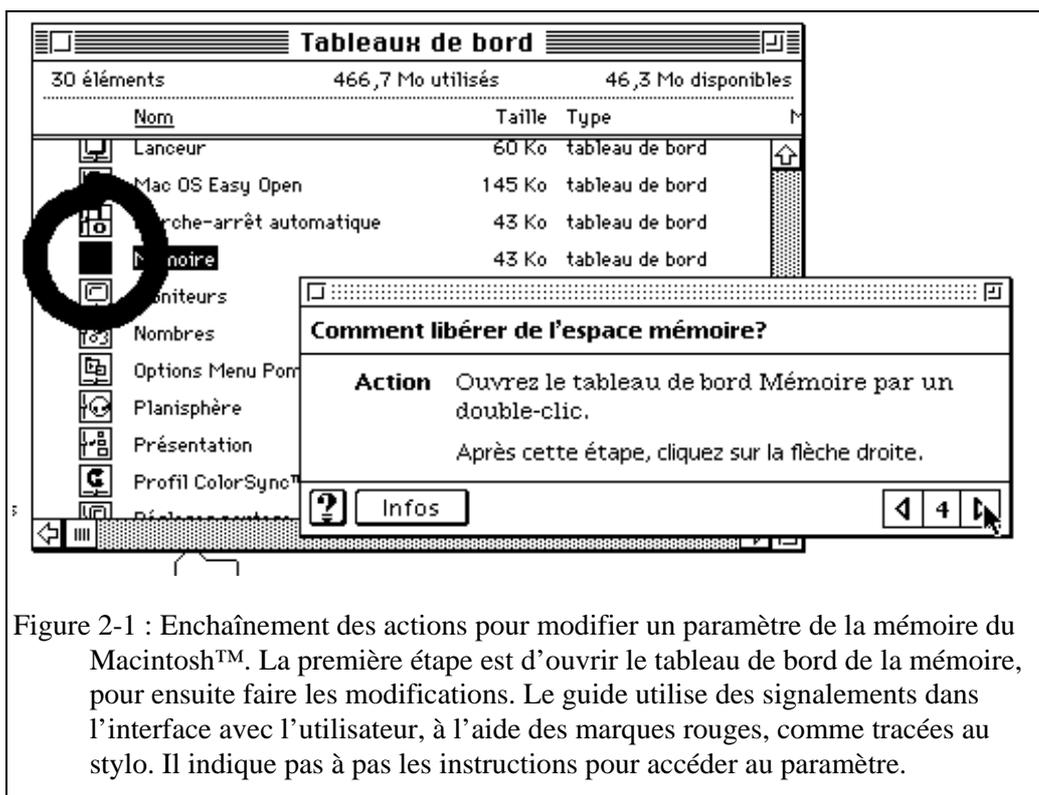


Figure 2-1 : Enchaînement des actions pour modifier un paramètre de la mémoire du Macintosh™. La première étape est d'ouvrir le tableau de bord de la mémoire, pour ensuite faire les modifications. Le guide utilise des signalements dans l'interface avec l'utilisateur, à l'aide des marques rouges, comme tracées au stylo. Il indique pas à pas les instructions pour accéder au paramètre.

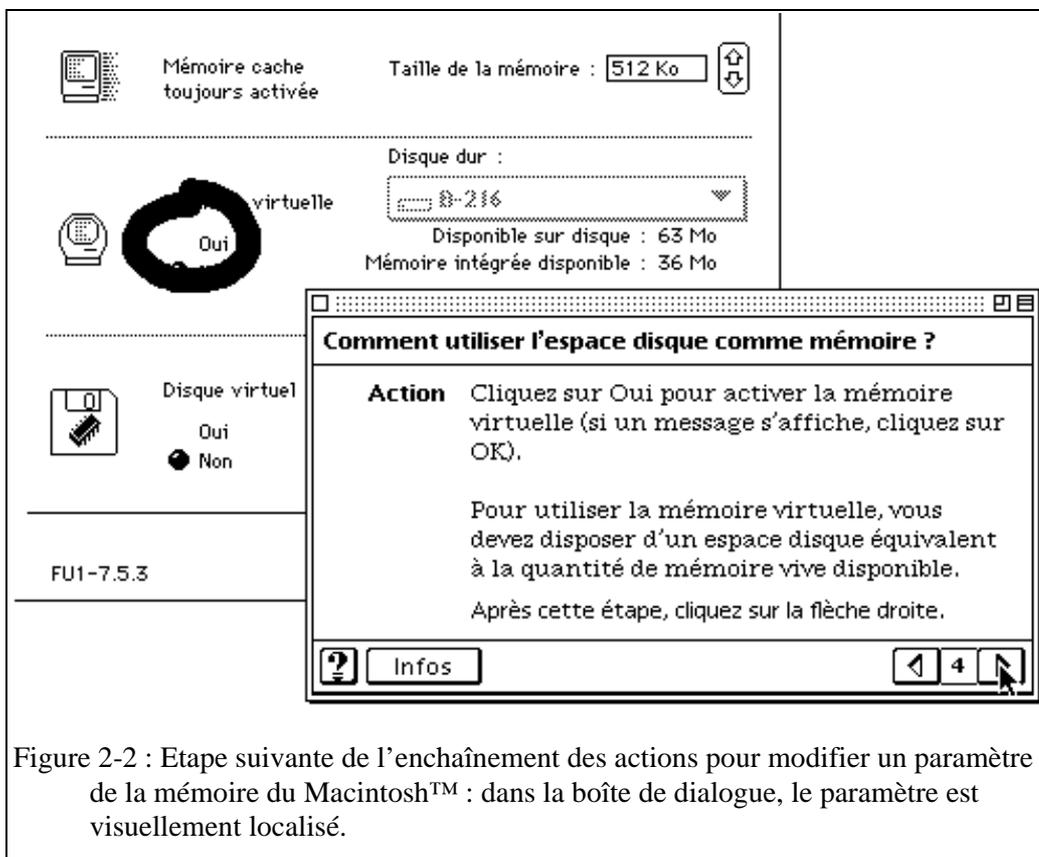


Figure 2-2 : Etape suivante de l'enchaînement des actions pour modifier un paramètre de la mémoire du Macintosh™ : dans la boîte de dialogue, le paramètre est visuellement localisé.

Privilégier l'initiative permet aussi au système d'aide de limiter ses interventions à la prévention de conflit, en laissant l'utilisateur demander des suggestions dans une situation d'insatisfaction. Cette discrétion du système d'aide est un atout pour être accepté par les concepteurs, surtout si les intrusions dans l'espace de travail restent rares.

Outre la discrétion, l'intérêt du système d'aide est lié à la justifications de ses propositions. C'est-à-dire que l'utilisateur doit comprendre pourquoi telle suggestion est retenue dans le contexte courant. La transparence correspond à la présentation des connaissances mises en jeu pour élire une suggestion, mais ne porte pas sur l'architecture du système d'aide.

Notre travail s'attache donc à exploiter l'initiative de l'utilisateur valorisée par un système qui propose des conseils, aussi bien dans une phase préventive que pour résoudre une situation d'insatisfaction. Nous discutons maintenant des connaissances nécessaires au système d'aide pour définir les besoins au niveau de l'interface entre le progiciel et le système d'aide.

2.4. La modélisation du système d'aide

Pour être efficace, le module doit suivre l'activité de l'utilisateur. Sans demander de contributions à ce dernier, il doit obtenir des informations du progiciel. Cet échange repose donc sur une interface fonctionnelle, entre le progiciel et le système d'aide, pour accéder aux données du noyau fonctionnel.

Le terme interface est entendu comme l'interface fonctionnelle, ou API « Application Programming Interface ». L'interface avec l'utilisateur n'est pas discutée dans ce mémoire, étant donnée la nécessité de respecter l'interface existante. Pour notre application, la consigne imposée est d'introduire dans le progiciel un module d'aide qui ne perturbe pas l'environnement perçu par l'utilisateur. Cette contrainte a poussé à la distinction d'une part de la présentation de l'information au niveau de l'utilisateur, et d'autre part de la représentation des connaissances pour leur exploitation par le module d'aide. Si la facette visible, qu'est la présentation, n'est pas une composante explorée dans notre travail, le corps du module d'aide a tiré les enseignements des solutions déjà proposées pour réaliser un système interactif innovant.

Son architecture repose sur les objectifs de chacune des analyses disponibles dans le progiciel, en exploitant leurs connaissances implicitement attachées. Ces dernières sont par exemple les contraintes de précédence entre les analyses, qui ne sont donc pas indépendantes les unes des autres. Par ailleurs, pour réaliser un objectif il existe *a priori* un ensemble de moyens, ou de fonctionnalités, dont la pertinence est relative au contexte de travail. La tâche du système d'aide est aussi d'évaluer leurs intérêts respectifs dans un contexte donné, en exploitant la sémantique des éléments manipulés par l'utilisateur dans l'interface.

Les connaissances manipulées par le système que nous proposons ne sont pas exhaustives. Elles ont été formalisées avec plusieurs experts, qui ont spontanément utilisé des heuristiques pour définir une connaissance consensuelle. A l'opposé d'un système expert chargé de trouver la solution à un problème, le but est d'expliquer à l'utilisateur le statut de chaque élément de l'interface, en fonction des connaissances sémantiques recueillies. Toutefois, les considérations envers la représentation des connaissances rejoignent celles d'un système expert, quand il est question d'introduire une métaconnaissance ou d'enrichir les connaissances initialement recueillies.

2.4.1. L'interface avec le progiciel

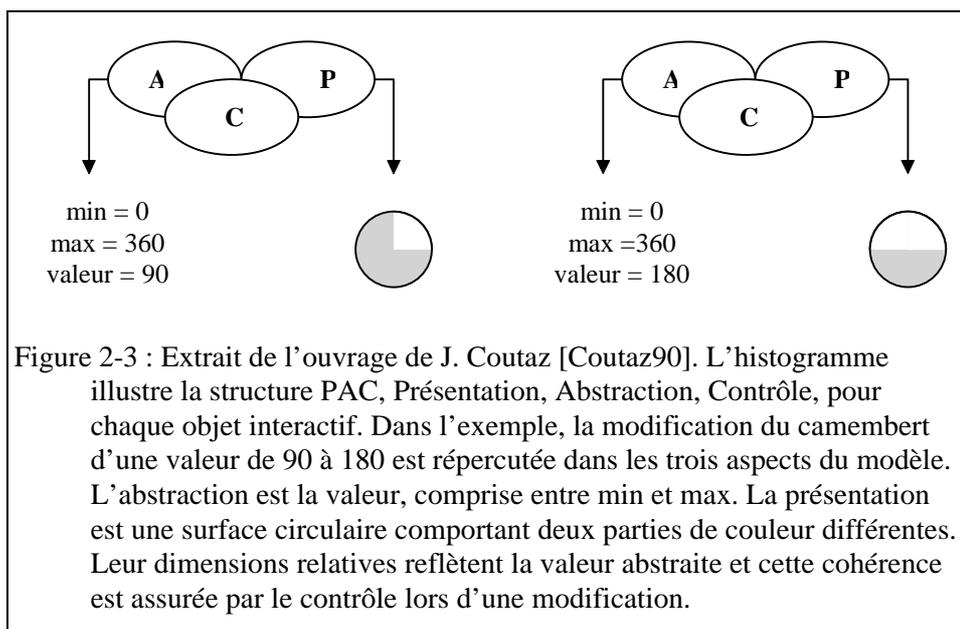
Dans le cadre de notre travail, l'interface du module d'aide est tournée vers le progiciel. En effet, l'utilisateur n'est pas sollicité et l'interface qu'il connaît déjà doit être respectée dans l'outil de conception, pour ne pas bousculer ses habitudes. Ceci n'interdit pas d'améliorer ultérieurement la présentation, en impliquant avantagusement des ergonomes expérimentés. Leur regard neuf et critique amène des suggestions pour renforcer la cohérence de l'interface avec l'utilisateur. Par exemple, les recommandations des ergonomes pour l'évaluation d'une interface [Bastien et al.94] et [Bastien et al.93] donnent des indications qui une fois énoncées semblent évidentes. Le décalage entre la pratique et leurs suggestions prouve néanmoins que ce bon sens n'est pas intuitif, et qu'il faut de la rigueur pour le concrétiser.

L'interface fonctionnelle met à disposition les données du noyau fonctionnel. C'est par son intermédiaire que ces données peuvent être modifiées à partir d'une application extérieure. Les actions de l'utilisateur peuvent aussi être vues comme une application, modifiant les données via le canal de l'interface fonctionnelle. Par conséquent, il n'est pas nécessaire de changer la présentation pour structurer l'interface fonctionnelle.

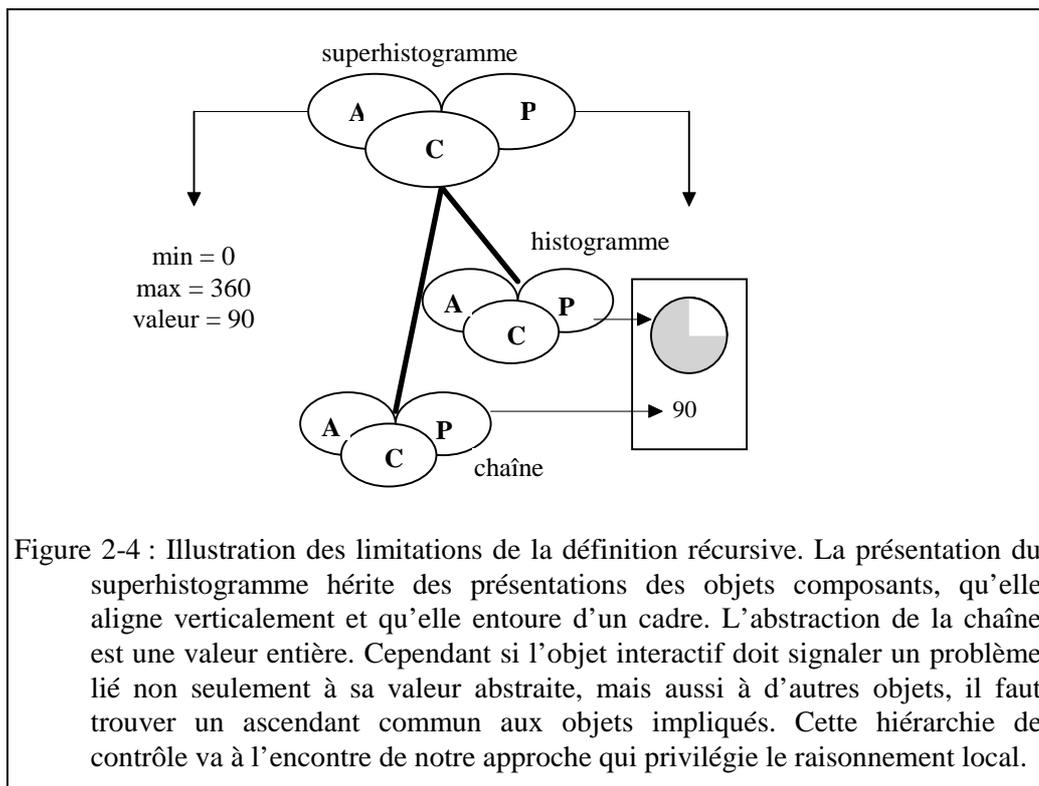
La préservation de l'aspect visible de l'interface n'est pas un frein pour la réflexion sur l'architecture de l'ensemble de l'interface. Elle présente même l'avantage de poser immédiatement la distinction entre les niveaux de représentation des connaissances. En cela nous adoptons l'approche de J. Coutaz [Coutaz90] et [Coutaz et al.91], qui propose de distinguer le niveau syntaxique, correspondant à la présentation, du niveau sémantique des données de l'objet manipulé, nommé abstraction.

Nous nommons par la suite « noyau sémantique », l'ensemble des informations du niveau sémantique qui représentent les données du noyau fonctionnel.

La cohérence et les contraintes entre la présentation et l'abstraction sont gérées par un contrôle propre à chaque objet comme le montre la Figure 2-3. Cette approche présente l'intérêt de ne pas imposer une frontière arbitraire entre l'application et l'interface, mais de considérer plusieurs niveaux qui s'intègrent dans l'interface fonctionnelle. L'indépendance de la présentation et de l'abstraction est garantie par le contrôle, intermédiaire obligatoire.



Toutefois, nous n'adoptons pas la définition récursive de l'interface proposée, qui impose cette même structure (présentation, abstraction, contrôle) à tous les objets de l'interface. En effet cette récursivité repose sur la hiérarchisation des composants de l'interface, et donc de leur contrôle (voir Figure 2-4). En simplifiant, deux boutons qui modifient mutuellement leur aspect dans une fenêtre seront contrôlés par cette dernière. Cependant, il apparaît qu'un objet exerce des contraintes sur un autre objet, très éloigné dans la hiérarchie de la présentation. C'est le cas de paramètres usuellement inchangés et regroupés dans une boîte de dialogue spécifique. Il n'est plus alors logique que le plus proche ascendant commun gère cette influence. L'organisation hiérarchique stricte se limite aux outils dont la présentation est l'image exacte du fonctionnement réel. Malheureusement, il est des progiciels avec un noyau fonctionnel trop complexe pour tout exposer. Une image simplifiée et structurée permet de ne pas noyer l'utilisateur dans des détails de fonctionnement dépourvus d'informations utiles. Dès lors, l'organisation de l'interface ne calque pas la hiérarchie de la présentation, mais elle ne peut cependant pas faire abstraction de l'image présentée à l'utilisateur.



Par ailleurs, cette hiérarchie n'intègre pas les informations de contexte, propres à chaque session. Ces données sur l'objet de la conception ne sont pas obligatoirement présentes dans l'interface visible, mais en sont éventuellement déduites. Puisque ces données influencent tous les niveaux de la hiérarchie de la présentation, il faut envisager d'autres solutions pour préserver un raisonnement local aux objets.

2.4.2. L'exploitation des connaissances

L'interface fonctionnelle entre le module d'aide et le progiciel transmet l'activité de l'utilisateur. Chaque lancement de commande est ainsi répercuté dans le module d'aide. Il est alors nécessaire de penser une structure pour interpréter ces informations brutes, qui ne se suffisent pas à elles mêmes. Nous avons déjà introduit le concept de noyau sémantique dans le module d'aide, qui constitue une abstraction du fonctionnement du progiciel.

Pour attacher une sémantique aux données transmises par l'interface fonctionnelle, il faut une connaissance sur cette connaissance. Appelée aussi métaconnaissance, elle est un composant essentiel du noyau sémantique. Ainsi, pour exploiter le lancement d'une commande, il faut pouvoir déterminer si cette dernière est autorisée, détecter si elle est sans conséquence sur le travail en cours, vérifier qu'elle n'est pas redondante. Le module d'aide a donc recours à des connaissances qui décrivent le fonctionnement du progiciel, pour réaliser le suivi d'utilisateur en exploitant son activité. Ces connaissances rendent le module « intelligent » en ce sens qu'il détermine quels sont les conseils pertinents dans le contexte donné.

La notion de métaconnaissance est exploitée dans notre représentation par objets, puis pour la première maquette du système d'aide, discutée ci-après. Cette approche a distingué les niveaux de représentation, en regroupant au niveau méta la définition des connaissances parta-

gées par plusieurs classes d'objets, et au niveau des classes la définition des attributs des instances de cette classe. Nous renvoyons le lecteur à l'ouvrage de J. Pitrat [Pitrat90] pour avoir une vue complète sur les métaconnaissances.

Le problème de savoir où placer ces métaconnaissances dans le système reste entier. J.M. Fouet [Fouet94] discute ce problème pour un système à base de connaissances, qui comprend une base de faits et un moteur d'inférence. L'introduction des métaconnaissances au niveau de la base pose le problème de ne pas tenir compte du problème couramment traité : l'ordonnement des prémisses par exemple ne sera pas optimal pour tous les problèmes. Leur introduction dans le moteur rend plus ardue sa mise au point, et il ne pourra fonctionner qu'avec des bases utilisant le même vocabulaire. J.M. Fouet propose alors un stratège extérieur, auquel le moteur a recours lorsqu'il est confronté à une alternative. Le moteur doit formuler cette alternative sous la forme d'un métaproblème, et la soumettre au stratège. Dans notre module d'aide, le choix est de privilégier le raisonnement local permis par la représentation par objets. L'introduction des métaconnaissances est discuté au chapitre 4.

L'emploi de métaconnaissances restreint le nombre de plates-formes candidates pour supporter le module d'aide, et en particulier la réalisation rapide d'une maquette. Après un premier examen, les systèmes Clips et Shood ont été retenus. L'article de W Mettre [Mettrey91] présente les avantages de Clips comme système expert, qui néanmoins n'offre pas l'efficacité des règles de Shood décrites par la thèse de F. Bounaas [Bounaas95]. Le choix s'est porté sur la plate-forme Shood, développé à l'INRIA, qui permet d'introduire aisément des explications attachées à chaque objet. Le choix de cet environnement a été motivé par l'usage des métaconnaissances, qui complètent la représentation par objets. De plus, le système Shood manipule une hiérarchie de règles de production du type Événement, Conditions, Actions, qui est couplée à la hiérarchie des objets représentés. Cette structure a par la suite inspiré le fonctionnement du module d'aide en insistant sur le raisonnement local aux objets.

Concernant l'indépendance des connaissances entre elles, T.M Mitchell, S. Mahadevan, L.I Steinberg [Mitchell et al.85] proposent de considérer deux ensembles de règles : d'une part les solutions indépendantes pour un même problème et d'autre part les connaissances pour choisir une solution adaptée formalisée par des règles liées. Ils appliquent cette distinction dans le système LEAP, qui s'appuie sur le système VEXED offrant une aide et des conseils pour la conception VLSI. Leur but est de formaliser une démarche inconnue d'un utilisateur expert, à qui il est demandé d'étudier un processus dont la fonctionnalité est préalablement connue. Ils comptent ainsi incorporer un module d'apprentissage pour acquérir une nouvelle connaissance basée sur l'expérience du concepteur. Pour notre travail, nous retenons la notion d'indépendance des règles, qui nous permet de déléguer à l'objet un raisonnement local, sans recourir à un système superviseur. Nous reprenons dans le paragraphe suivant la discussion sur l'ajout de connaissances, sans toutefois parler d'apprentissage automatique.

2.5. L'enrichissement des connaissances du système d'aide

Les utilisateurs n'expriment pas toujours clairement leurs besoins, ou les améliorations qu'ils voudraient voir. Ainsi, la maquette était indispensable pour obtenir une première réaction devant un système proposé. Cette réaction est une base de discussion sur les besoins et les réponses à envisager. Néanmoins, il ne faut pas négliger les contributions constructives de quelques utilisateurs, qui font part de leurs remarques, et parfois de leurs suggestions, pour améliorer le progiciel. On comprend que ces personnes qui s'impliquent sont déçues lorsque

l'équipe de développement du progiciel ne retient pas leurs propositions, comme le signale N. Landesman [Landesman97]. Dans le cadre particulier du module d'aide, la prise en compte des demandes des utilisateurs reste le meilleur moyen pour encourager son utilisation.

Si dans un premier temps, nous avons envisagés de fonder le système d'aide sur le Raisonnement à Partir de Cas, RàPC, il est rapidement apparu qu'il existait une connaissance déjà formalisée sous forme de règles pour l'utilisation du progiciel, d'une part dans le manuel, et d'autre part dans la manière de procéder. En effet, dès le début de ce travail, les principaux facteurs qui déterminaient les conseils étaient cernés : le processus étudié, et le contexte du déroulement de la session. Cependant, les règles s'appliquent sur des caractéristiques générales du processus, et elles sont avantageusement complétées par le savoir-faire des experts pour des situations plus spécifiques. Le RàPC est parfaitement adapté à l'introduction d'une connaissance spécifique, qui complète les règles obtenues avec les experts.

Nous décrivons tout d'abord comment le système d'aide peut être enrichi, et nous détaillons ensuite les possibilités offertes par le RàPC.

2.5.1. L'intégration de nouvelles connaissances

L'enrichissement des explications n'est pas une source de dysfonctionnement, dans la mesure où elles sont concrétisées par des textes extérieurs au module d'aide. Les explications sont des ressources de l'application, et elles sont accessibles à travers des identificateurs. Il est toutefois nécessaire de vérifier la cohérence des textes proposés pour qu'ils décrivent effectivement le fonctionnement du progiciel. Cette validation n'est pas vitale pour le fonctionnement du module d'aide, mais elle conditionne la qualité des conseils donnés à l'utilisateur. La construction interactive de ces explications en collaboration avec les utilisateurs n'est pas exclue, mais elle sort de notre domaine de recherche, et n'a pas été envisagée.

La situation est radicalement différente pour l'adjonction d'une nouvelle règle dans le système d'aide. En effet, cette connaissance a la capacité d'altérer le fonctionnement actuel, dans la mesure où elle positionne des données réputées protégées. Ces conséquences sont inévitables si la règle est déclenchée ; dans le cas contraire son influence est nulle du point de vue du module d'aide. Ainsi, il est possible d'ajouter une règle, mais cela demande un ensemble de précautions que seule l'équipe de développement peut gérer. En particulier, il ne s'agit pas de conserver des règles redondantes, ou pire, des règles conflictuelles, qui pour les mêmes conditions de déclenchement ont des conséquences contradictoires dans l'environnement. Dans la perspective de l'apprentissage, Y. Kodratoff [Kodratoff88] traite du problème du maintien de la cohérence lors de l'ajout de règles. A partir du moment où cette modification n'est pas simple, elle ne pourra pas être confiée à l'utilisateur. Une autre solution devra être envisagée si l'utilisateur a besoin d'enrichir lui-même, sur son site, le système d'aide.

Il est clair que l'équipe en charge du développement du module d'aide pourra ajouter les connaissances collectées ultérieurement auprès des concepteurs, et exprimées par des règles. A partir du moment où une nouvelle connaissance formalisée, le travail d'implémentation déjà mené se poursuit. La seule restriction concerne les apports jugés trop spécifiques au type de travail mené par un concepteur, pour être intégrés dans le module d'aide dédié à tous les concepteurs.

Dans la pratique, le problème est moins simple, car les concepteurs ne communiquent pas spontanément ces connaissances. En premier lieu, ils sont méfiants vis-à-vis de la diffusion d'une connaissance étroitement liée à leur conception. En second lieu, ils sont conscients que

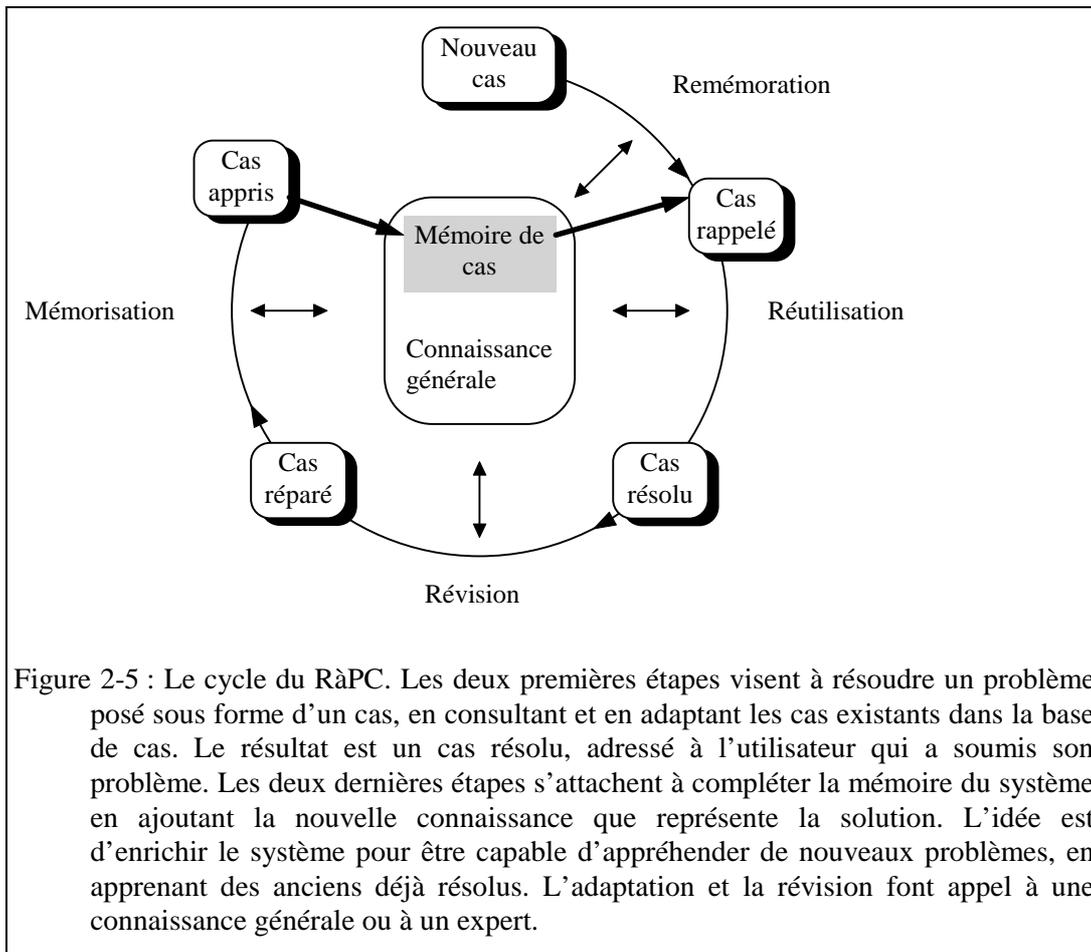
cette connaissance n'est pas suffisamment formalisée : elle est le résultat de quelques expériences, mais les conditions précises d'application ne sont pas cernées. Comme la formalisation n'entre pas dans leur champ de travail, ils ne se consacrent pas à cette activité. Il reste donc difficile de transmettre cette information, tant que la connaissance est étroitement liée au processus, mais dans des proportions mal définies. Si nous souhaitons néanmoins la rendre accessible, il faut transposer l'expérience aussi complètement que possible dans le module d'aide, et permettre au concepteur de la commenter. Pour réaliser cette apport de connaissance par l'utilisateur, nous avons précisé que les règles étaient inadaptées. La solution du RàPC est par contre tout à fait prometteuse.

2.5.2. La remémoration efficace par RàPC

Sachant que notre système d'aide ne représente pas la connaissance exhaustive sur l'utilisation du progiciel, il est attrayant de penser dès sa conception le problème de la complétion des connaissances générales par le savoir-faire des experts. Le premier point est de mémoriser cette expérience, et ensuite de pouvoir la remémorer. Le RàPC occupe une place privilégiée pour la remémoration [Silverman97]. Sa particularité est d'appréhender, à travers les expériences, des domaines où la connaissance n'est pas encore formalisée, en commentant ces expériences. La diversité des domaines et des utilisations du RàPC est illustrée par H. Mignot [Mignot92], qui commente les nombreux projets déjà existants.

Le cycle du RàPC comprend en général les quatre étapes suivantes [Aamodt et al.94], illustrées par la Figure 2-5 :

- la remémoration des cas les plus similaires au cas courant, qui représente un problème à résoudre,
- l'adaptation des cas remémorés, pour obtenir une solution au problème,
- la révision, qui évalue la solution générée pour la réparer si nécessaire,
- et enfin la mémorisation de la nouvelle connaissance que cette expérience apporte dans la base de cas.



Dans notre cas, nous nous concentrons sur les opérations touchant à la mémoire, c'est-à-dire aux étapes de remémoration et de mémorisation. Nous n'avons pas de connaissance générale, autre que la structure du module d'aide, à apporter au système pour lui permettre de réaliser les étapes d'adaptation et de révision. En fait, la base de cas regroupe l'ensemble des situations mal appréhendées par les connaissances générales du système d'aide écrites sous forme de règles. Notre proposition est donc une combinaison entre les règles et les cas. Les premières sont validées par les experts, les deuxièmes sont une trace d'expérience qu'un concepteur juge utile de garder. Le couplage entre les règles et les cas est déjà exploré par de nombreux travaux. Ces derniers juxtaposent les deux modes de raisonnement, comme nous l'envisageons dans le dernier chapitre du mémoire, ou ils réalisent une véritable combinaison comme l'expose [Golding91]. Dans ce dernier travail, les cas apportent deux contributions. D'une part ils sont utilisés pour les exceptions aux règles, et sont alors la mémorisation des exemples négatifs. D'autre part, les exemples positifs illustrent les règles afin de faire une prédiction. Notre problématique impose une indépendance entre les règles et les cas. En effet, le progiciel et le système d'aide évoluent indépendamment des bases de cas créées sur les sites clients.

Nous présentons maintenant les principaux éléments du RàPC, qui sont la définition du cas, l'étiquetage (ou indexation) d'un cas pour être extrait ou inclus dans la base, et le choix

de la structure de la mémoire. Tous ces thèmes sont discutés dans l'ouvrage de référence sur le RàPC de J. Kolodner [Kolodner93].

De manière générale, J. Kolodner décrit les composants d'un cas comme :

- La description du problème à résoudre, lors de la création du cas. Elle comprend les compromis sur la résolution à mener et les caractéristiques du problème par rapport à l'état du monde.
- La proposition pour résoudre le problème, c'est-à-dire les étapes successives et leurs justifications. Elle justifie par ailleurs l'ensemble des solutions acceptables qui n'ont pas été retenues, ainsi que les solutions qui n'étaient pas acceptables.
- Le résultat de l'application de la proposition, qui peut être un succès ou un échec.

La définition est en fait plus large, car elle comprend aussi la notion d'évaluation du résultat de la proposition, pour tenter d'expliquer le résultat réel par rapport au résultat es-compté, et pour réparer la proposition si nécessaire.

Les deux thèmes de l'indexation du cas et de l'organisation de la mémoire conditionnent la remémoration, lorsque la base de cas est explorée pour rechercher l'ensemble des cas utiles au problème courant. Ces thèmes sont complémentaires, car une organisation hiérarchique de la base de cas est équivalente à une indexation particulière.

La notion d'utilité est une ligne constante : il ne sert à rien de multiplier le nombre de cas, s'ils ne sont pas exploitables. De même, les étiquettes doivent posséder un pouvoir prédictif sur le problème et la solution d'un cas. Elles doivent être suffisamment générales pour permettre la remémoration du cas dans des situations différentes, et par ailleurs, suffisamment spécifiques pour effectivement remémorer le cas dans des situations futures. Le choix des index est lié au domaine d'application, et à l'usage des cas. Etant donné que dans notre application les cas complètent les propositions devant une situation d'échec, le vocabulaire adapté aux types d'échecs que propose Hammond [Hammond87] est retenu pour indexer nos propositions. Hammond l'utilise pour retrouver les explications des causes de cet échec, mémorisées par les cas. Ce travail sur le vocabulaire est poursuivi par Leake [Leake91], qui utilise les anomalies.

Les processus de mémorisation et de remémoration sont étroitement liés à l'organisation de la base de cas. Une première approche est d'utiliser une mémoire plate, où tous les cas sont mémorisés séquentiellement dans une liste. Cette structure simple présente une remémoration précise, puisque tous les cas sont testés, mais peu efficace dès que la base est importante. Pour remédier à cet inconvénient, M. Malek [Malek96] propose un modèle hybride qui associe une mémoire plate et un réseau de neurones à base de prototypes. Le système aboutit à une remémoration efficace, sans compromettre la simplicité de la mémorisation d'un nouveau cas. La seconde approche est une organisation hiérarchique de la base de cas, qui prend souvent la forme d'un arbre. Barletta et al. [Kriegsman et al.93] exploite la hiérarchie des concepts sur laquelle repose l'utilisation de symboles pour indexer les données. Chacun des noeuds de l'arbre regroupe les cas qui partagent des caractéristiques, et qui sont mémorisés comme les feuilles de l'arbre. La remémoration consiste à sélectionner le noeud approprié et se limite aux feuilles correspondantes (voir J. Kolodner). Cette organisation garantit une remémoration efficace mais influence sa précision, qui sera d'autant meilleure que les cas utiles pour une situation sont effectivement regroupés sous un même noeud. Dans le cas général elle requiert la construction de l'arbre qui se fait à partir de l'ensemble des cas collectés, et elle nécessite une mise à jour de l'arbre lors de l'ajout d'un nouveau cas.

Dans le cadre de notre application, où les cas viennent compléter un système existant, il est plus attrayant d'exploiter la structure du système d'aide, dans la mesure où l'organisation hiérarchique a aussi été pensée pour accueillir des cas. En effet, cette hiérarchie permet à l'utilisateur de consulter soit les cas correspondant au noeud sélectionné par le système d'aide, soit de consulter les autres noeuds s'il le juge nécessaire.

Ce chapitre a exposé les éléments de solution qui ont influencé notre réflexion pour proposer un système d'explications et de conseils pour une bonne utilisation d'un progiciel. Le chapitre suivant est une synthèse de nos choix pour la mise en œuvre du système d'aide.

Chapitre 3

3. L'interaction avec l'utilisateur

Ce troisième chapitre est une synthèse des choix retenus pour mettre en œuvre le système d'aide, qui répondent aux besoins des utilisateurs formulés dans le premier chapitre.

Dans le premier chapitre, nous avons décrit l'interaction souhaitée par les utilisateurs qui ont évalué la maquette, puis le prototype. En résumé, le module d'aide doit renseigner l'ensemble des possibilités du progiciel, mais en étant capable de distinguer, au moment où il est interrogé, quelles sont les possibilités qui s'appliquent effectivement au contexte de travail en cours. La vocation du système d'aide n'est pas d'améliorer l'interface Progiciel/Utilisateur existante : pour cet aspect, il serait plus judicieux de faire appel à un ergonomiste spécialiste en la matière.

Les utilisateurs donnent priorité à l'avancement de leur travail et ils ne sont pas en permanence disposés à approfondir leur connaissance du progiciel. Il faut donc veiller à ne pas imposer le système d'aide, mais au contraire, encourager les utilisateurs à y faire appel. De cette manière, ils recherchent seulement l'information nécessaire, pour à terme se dispenser de cette assistance. L'approche retenue est d'impliquer l'utilisateur pour valoriser ses connaissances et pour les compléter.

Nous justifions maintenant notre approche, qui n'a pas recours à un profil d'utilisateur.

3.1. Pourquoi se dispenser du profil utilisateur ?

Le profil caractérise les besoins de l'utilisateur pour lui offrir une interface adaptée, de manière à faciliter sa compréhension. Quand plusieurs types d'utilisateurs sont susceptibles d'exploiter le progiciel, il convient de définir leurs profils respectifs pour mieux répondre à leurs besoins spécifiques. L'étude du profil utilisateur se focalise donc sur la différence entre les types d'utilisateurs, comme par exemple la distinction entre un utilisateur néophyte et un utilisateur averti, qui en général ne rencontrent pas les mêmes problèmes.

Or notre étude du progiciel de simulation montre que sa maîtrise ne dépend pas seulement du concepteur, mais avant tout du circuit qu'il simule, car l'expertise est attachée aux fonctionnalités. Les notions d'utilisateur néophyte ou expert ne s'appliquent qu'aux cas extrêmes : l'étudiant ayant peu ou pas manipulé de simulateurs, et les concepteurs en électronique qui font partie de l'équipe de développement du simulateur.

En règle générale, un utilisateur novice dans la manipulation du simulateur devient rapidement capable de l'exploiter. En effet, grâce à son expérience sur d'autres progiciels similaires, il connaît les principales étapes pour mener son étude, et il sait rapidement identifier les commandes associées dans le simulateur. Par contre cette exploitation reste partielle, et les options particulières qui font la spécificité du progiciel, ne sont pas aussi rapidement acquises. Ainsi, paradoxalement, le novice peut manipuler avec facilité des notions réputées délicates, sans pour autant maîtriser pleinement le progiciel.

Par ailleurs, la multiplicité des progiciels manipulés contrarie les concepteurs dans leur expertise. Comment, en effet, maîtriser un progiciel manipulé deux à trois mois par an, quand ce dernier évolue au rythme minimum d'une version par an et qu'il présente en outre de fortes similarités avec d'autres progiciels ? L'expert oublie inéluctablement des facilités spécifiques à ce progiciel et il peut faire des erreurs quand il le reprend en main après une longue inactivité.

La notion de profil général de l'utilisateur n'est donc pas adaptée. Une autre possibilité consiste à déterminer l'expertise de l'utilisateur sur chaque fonctionnalité, comme le propose [Berthomé et al.95]. Cependant, l'approche plus positive qui consiste à impliquer l'utilisateur en l'aidant dans ses choix permet de mettre l'accent sur les explications.

L'exercice délicat de la compréhension des explications par tous les utilisateurs est ramené à présenter les explications en des termes et notions partagés grâce aux connaissances du domaine. Nous avons effectivement eu des requêtes lors de l'évaluation de la maquette. Par exemple, le terme « analyse de convergence » était jugé trop flou : les concepteurs préféraient l'expression « recherche du point de fonctionnement ». En conclusion, il est plus approprié d'inclure les termes précis du domaine de connaissances partagées par les utilisateurs potentiels, que de vouloir construire le contenu des explications adapté à chaque utilisateur.

Le domaine de la conception nous amène donc à considérer des utilisateurs qui partagent davantage de connaissances qu'ils ne présentent de différences. Notre proposition est d'exploiter les aspects communs, pour présenter à tous les utilisateurs l'expertise des utilisateurs avisés, sans chercher à construire un profil utilisateur, mais en prenant en compte le contexte d'utilisation qui inclut par exemple les caractéristiques du circuit.

3.2. La valorisation de l'utilisateur

Considérer comme expert tout utilisateur du progiciel ne devrait pas surprendre. En effet, l'utilisateur connaît le domaine d'application qu'est l'électronique, les spécifications du produit à concevoir, les limites du modèle choisi pour travailler avec le progiciel : il doit « seulement » acquérir les connaissances sur les capacités du progiciel. En dépit de la complexité des connaissances impliquées dans la manipulation efficace d'un progiciel, les connaissances du domaine restent autrement plus difficiles à formaliser et à comprendre.

Cette approche permet en outre de valoriser son esprit critique devant l'examen des conseils du système d'aide. Rappelons que dans le domaine de la conception, et *a fortiori* de la simulation, le concepteur doit toujours remettre en cause le travail réalisé pour satisfaire au mieux des compromis ou améliorer sa conception. Le système d'aide avive cet état d'esprit s'il responsabilise l'utilisateur dans le choix des conseils.

L'évaluation de la maquette par des concepteurs a montré que ces derniers souhaitent préserver leur contrôle sur le progiciel et qu'ils acceptent difficilement une automatisation, même

si la solution envisagée est efficace. Par exemple, un expert critiquait l'ajout d'un élément capacitif sur tous les noeuds analogiques d'un circuit, préférant exécuter manuellement la modification là où elle était indispensable. Ce contrôle global lui permet de « savoir ce qui est réellement modifié » dans sa conception. La proposition sera néanmoins appliquée par un néophyte de l'outil, ou par un concepteur qui accepte cette modification pour obtenir un premier résultat.

Pour valoriser l'esprit critique, le module d'aide doit expliciter les causes d'un conflit détecté et justifier les conseils pour le résoudre. Ainsi l'utilisateur peut vérifier la pertinence des messages et prendre en main le problème s'il en est capable. Par ailleurs, il peut consulter les suggestions du module pour choisir une solution adaptée à la résolution du problème. Toujours dans le souci d'informer, les suggestions sont accompagnées de leurs avantages et de leurs inconvénients, ainsi que des conditions dans lesquelles elles s'appliquent. Ces informations permettent à l'utilisateur d'avoir une image du fonctionnement du module. En outre elles peuvent être consultées pour découvrir des alternatives envisageables.

3.3. Le mode d'interaction

Le contenu des explications fournies par le système d'aide doit répondre d'une part aux questions sur le fonctionnement posées lors d'une prise en main, et d'autre part à une situation d'échec. Les connaissances sont donc générales, pour exposer les possibilités du progiciel, et contextuelles quand il s'agit d'exploiter ces options pour le travail de conception en cours. Néanmoins l'interaction n'est pas seulement définie par le contenu des informations, mais aussi par la façon d'établir cet échange, surtout si la contrainte de ne pas interrompre le travail de l'utilisateur est respectée.

Le mode d'interaction définit les instants propices pour retenir l'attention de l'utilisateur. Les moments privilégiés sont le réglage des paramètres avant de lancer une analyse, et la fin d'une simulation, en particulier si elle se solde par une situation d'échec. Les propositions exposées ci-après sont le fruit d'une réflexion conduite avec des utilisateurs.

3.3.1. Une interaction souple

L'utilisateur n'a pas toujours besoin des interventions du module d'aide, mais par ailleurs il peut en avoir l'utilité à tout moment. En conséquence il doit pouvoir inhiber ces interventions momentanément, sans que cela pénalise une demande ultérieure, surtout quand l'hypothèse de travail est que le concepteur peut étudier un circuit, pour lequel les connaissances du système ne sont pas réellement adaptées. Le fonctionnement du module d'aide doit permettre cette facilité d'utilisation. Dans le cas contraire, si le module d'aide est soit actif, soit désactivé, dans un fonctionnement tout ou rien, dès que l'utilisateur n'accepte pas une suggestion il risque fort de tout désactiver.

L'option de désactivation, accessible par le menu Advisor présenté par la Figure 3-1, pour masquer les interventions du module d'aide momentanément, était initialement pensée pour les experts qui savent exploiter le progiciel, même sur des circuits atypiques. Néanmoins, les néophytes sont aussi tentés de désactiver le système d'aide. Lors de l'expérimentation du prototype auprès des étudiants en électronique, leur souci était avant tout de réaliser leurs travaux. Faisant d'avantage d'erreurs de manipulation, ils supportaient mal de devoir acquitter la prise de connaissance de chaque message. Cela devenait une charge qui pesait sur leur travail, et il y avait trop d'erreurs signalées pour les analyser avec bénéfice. C'est à leur demande que

les mises en garde du module d'aide n'ont plus été bloquantes. Lorsque le module a détecté un conflit, il le signale dans l'interface d'une manière aussi précise que possible, comme le montre la Figure 1-15 dans le premier chapitre.

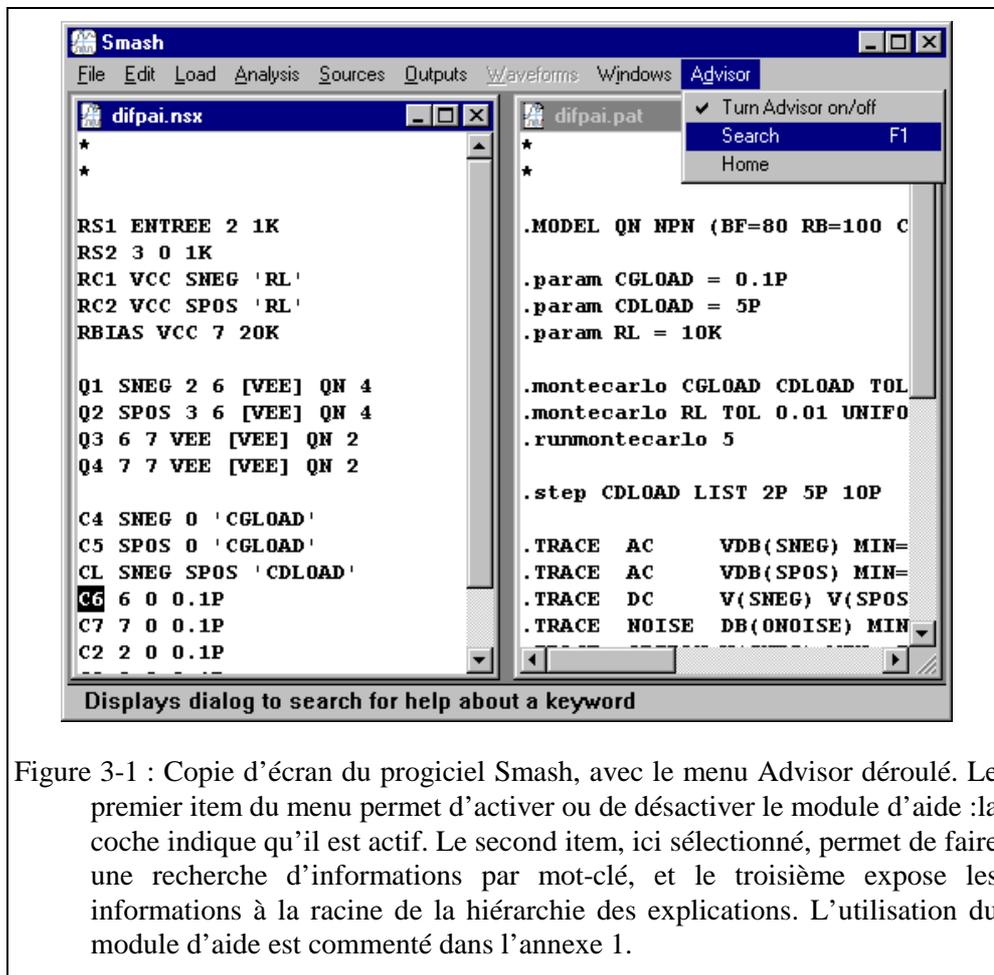


Figure 3-1 : Copie d'écran du progiciel Smash, avec le menu Advisor déroulé. Le premier item du menu permet d'activer ou de désactiver le module d'aide : la coche indique qu'il est actif. Le second item, ici sélectionné, permet de faire une recherche d'informations par mot-clé, et le troisième expose les informations à la racine de la hiérarchie des explications. L'utilisation du module d'aide est commenté dans l'annexe 1.

Aujourd'hui l'utilisateur demande explicitement une aide sur le conflit signalé, en utilisant le bouton « Help ». Ce n'est qu'à ce moment que la boîte de dialogue présentant les informations produites par le module d'aide est affichée à l'écran. En sus des connaissances dédiées à la notion concernée par le conflit, l'utilisateur peut exploiter l'ensemble de pointeurs vers des sujets corrélés, dans la liste « related topics ». Ainsi, l'attention et la curiosité sont sollicitées par une incitation à la découverte des capacités du progiciel.

La discrétion du signalement des conflits présente aussi l'avantage de gagner du temps quand la cause du conflit est une erreur d'étourderie. Une fois mise en évidence, cette dernière pourra être immédiatement corrigée par un connaisseur.

3.3.2. Inciter l'activité de l'utilisateur

La charge de l'utilisateur est soulagée par la suppression d'une consultation systématique des explications relatives au conflit détecté. En contre-partie, elle est accrue par la nécessaire détermination des informations par l'utilisateur. La consultation des explications devient une volonté marquée par l'initiative de l'utilisateur. Cette implication favorise la mémorisation

des nouvelles informations, surtout si ces dernières sont aussitôt mises en application : l'action facilite l'apprentissage.

Un utilisateur moins aguerri aura besoin, d'une part de découvrir la raison du conflit, et d'autre part d'apprendre à corriger l'erreur. C'est en fonction de ses connaissances qu'il fera appel au module d'aide pour combler ses lacunes. Impliqué dans la résolution du problème, son esprit critique est sollicité par les explications du module d'aide.

L'initiative de l'utilisateur est une hypothèse de base pour le bon fonctionnement du module d'aide. Toutes les opportunités pour renforcer cette activité doivent être saisies. Cependant, force est de constater que, dans la pratique, seuls les problèmes qui arrêtent le travail sont systématiquement analysés pour être résolus. La curiosité de l'utilisateur est réfrénée par la nécessaire réalisation du travail.

3.3.3. Profiter des situations d'échec

La situation d'échec est très favorable à l'intervention du module d'aide. En effet, arrêté dans son travail, le concepteur accepte de se renseigner pour résoudre son problème. Dans la pratique, c'est à ce moment que les utilisateurs consultent le manuel ou contactent le support technique téléphonique.

En fait, le progiciel présentait déjà des indications générales dans la situation d'échec d'une analyse temporelle, comme le montre la Figure 3-2. Ces conseils ne mettaient pas en évidence leurs conditions d'application, ni même l'indication de leur efficacité dans le contexte de travail. Le seul énoncé de solutions ne permet donc pas à l'utilisateur de s'appropriier le progiciel. Il faut au moins préciser à l'utilisateur les conditions d'application, et au mieux lui indiquer si elles sont ou non validées dans le contexte de travail.

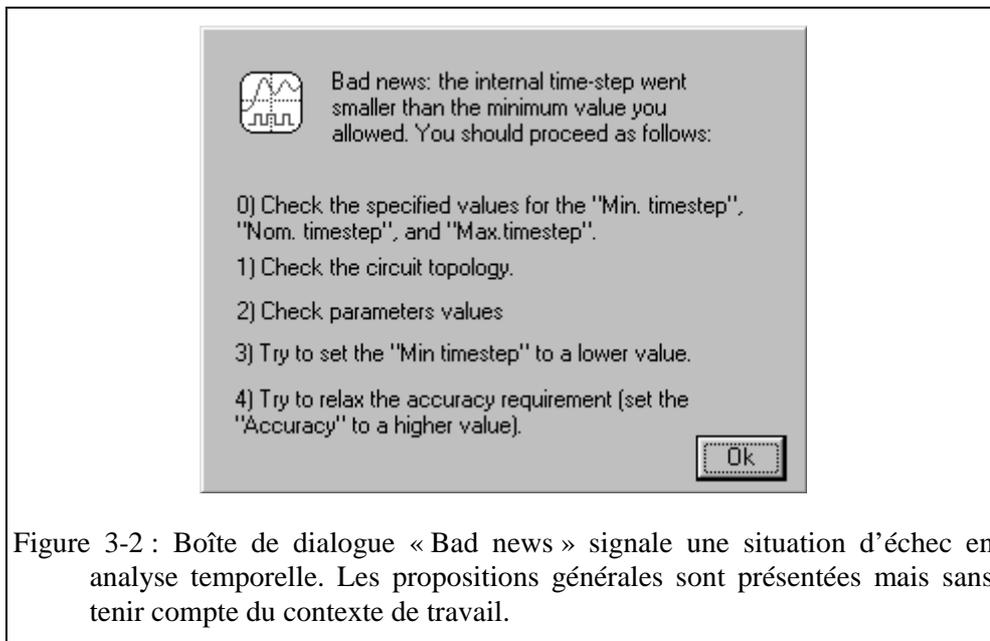


Figure 3-2 : Boîte de dialogue «Bad news» signale une situation d'échec en analyse temporelle. Les propositions générales sont présentées mais sans tenir compte du contexte de travail.

Dans une situation d'échec, non seulement l'utilisateur est plus disponible, mais il est surtout très attentif aux suggestions faites. Il regarde de près les avantages et les inconvénients

d'une proposition, pour faire judicieusement son choix par rapport à ses critères. La proposition retenue sera d'autant mieux mémorisée que les conseils étaient pertinents et qu'ils permettent de résoudre l'échec.

Cette attitude de conseil laisse l'utilisateur juger de la pertinence de l'intervention du système d'aide. Dans une situation d'échec du progiciel, c'est-à-dire quand l'exécution d'une action ne peut pas être achevée avec les contraintes fixées, le système d'aide laisse aussi le soin à l'utilisateur de choisir sa solution parmi les alternatives connues du système, ou de construire sa propre solution éventuellement comme une adaptation des suggestions précédentes.

3.4. L'interaction retenue

Nous avons cerné les besoins de l'utilisateur sur les modes d'interaction avec le module d'aide. Avant de décrire le fonctionnement du module d'aide dans les chapitres suivants, nous présentons l'interaction retenue.

Nous avons illustré précédemment comment le module d'aide signale une situation de conflit, sans bloquer le travail (voir la Figure 1-15). Cette interaction repose sur l'implication de l'utilisateur, qui juge s'il sait corriger une étourderie, ou s'il vérifie la cause du conflit qui ne lui semble pas évidente.

En dehors des conflits, il existe des situations non satisfaisantes, constatées à la fin d'une analyse :

- la situation d'échec, qui résulte d'une analyse numérique qui ne converge pas. C'est la situation dans laquelle l'aide est attendue en priorité ;
- l'insatisfaction d'une analyse jugée trop lente. Il faut savoir que des circuits comprenant des milliers de transistors demandent des heures de calcul ;
- l'insatisfaction d'une précision insuffisante pour observer des phénomènes physiques attendus par l'utilisateur.

Les deux derniers points impliquent un échange avec l'utilisateur, car le progiciel ne peut pas conclure sur ces insatisfactions subjectives. Aujourd'hui, l'interaction se concentre sur la situation d'échec, qui est détectée par le progiciel et qui peut donc être prise en compte par le système d'aide sans intervention de l'utilisateur.

Pour permettre à l'utilisateur de contrôler la pertinence des conseils du système d'aide, il faut qu'il puisse consulter :

- la liste des conseils connus du système d'aide pour résoudre cette situation d'échec. L'utilisateur peut ainsi évaluer les connaissances du système d'aide, et éventuellement constater qu'une possibilité n'est pas connue ;
 - la liste des conseils candidats, c'est-à-dire retenus pertinents dans le contexte de travail, avec la description de leurs conditions d'application. Ceci permet de cerner les possibilités du progiciel, pour les appliquer par la suite sans le support de l'aide ;
 - la liste des conseils influents, c'est-à-dire candidats et reconnus pour résoudre le problème dans ce contexte. Ce sont en particulier les options dédiées à des contextes spécifiques et qui de ce fait sont souvent méconnues de l'utilisateur ;
 - les raisons pour ne pas avoir considéré qu'un conseil connu était candidat,
-

- enfin, les avantages et les inconvénients des conseils candidats pour que l'utilisateur puisse en choisir un en connaissance de cause ;

Nous insistons sur le fait que l'implication de l'utilisateur est toujours à l'origine de l'interaction avec le système d'aide, pour demander ou non un conseil.

Prenons l'exemple d'une situation d'échec obtenue par une analyse pour calculer le point de fonctionnement du circuit. La liste des possibilités est en grande partie exposée dans le manuel de l'utilisateur, qui indique d'abord les vérifications à faire, puis cinq méthodes proposées successivement. Ceci sous entend qu'il n'est pas garanti que l'une d'elles résolve le problème. Le manuel mentionne par ailleurs les principaux paramètres qui influencent cette analyse.

Le système d'aide devra indiquer de manière équivalente cette liste de méthodes connues.

What to do when it does not work

Some circuits can exhibit convergence problems. Smash™ has some powerful features to help you . Here are a number of hints.

First check that the vmin and vmax values you have set are correct. [...] The best thing to do is to have vmin and vmax set to the actual minimum and maximum values which can appear (or 10% more).

[...]

Methods

- 1) Verify that with no .OP directives, that is with everything « by default » in the pattern file, it does not work.*
- 2) The first thing to try is to reduce the deltav parameter and retry. [...]*
- 3) If it still does not work, try the « deltav=0 » method : in the deltav box, type 0, select « Reset everything », then relaunch. This will activate an alternate algorithm[...]*
- 4) If nothing works yet, consider using this method, the « deltav=-1 » method [...].*
- 5) Consider using a powerup analysis to obtain a bias point.[...]*

Some other features are provided to deal with problematic circuits. See the .GMINJUNC, .GDSMOS, .GBDSMOS, .OPHELP, .PIVMIN and .VSTART directives.

Figure 3-3 : Extrait du manuel de l'utilisateur [Desclèves95], qui commente l'analyse de recherche du point de fonctionnement. La section donne des directives générales et explique comment procéder pour les appliquer. Dans un premier temps il s'agit de vérifier que la plage de recherche donnée par les paramètres vmin et vmax est correcte. Ensuite, faute d'avoir des informations sur le circuit, le manuel propose l'application successive des différents algorithmes en favorisant les plus rapides.

Les avantages du système d'aide contextuelle sont :

- de faire les vérifications, et signaler à l'utilisateur un conflit sur la plage de recherche s'il y a lieu ;

- de sélectionner, parmi la liste des méthodes possibles, celles qui s'appliquent effectivement, en fonction du circuit et du contexte en général. De cette manière, l'attention de l'utilisateur se focalise sur les méthodes pertinentes.

Nous illustrons maintenant les choix de l'interaction retenue avec les deux boîtes de dialogue extraites d'une exécution de la maquette. La Figure 3-4 proposée à l'utilisateur dans une situation d'échec précise la nature du problème, ainsi que la liste des propositions adaptées au contexte d'utilisation. L'utilisateur peut consulter le descriptif de chaque proposition, grâce au bouton « Descriptif » qui ouvre la boîte de dialogue de la Figure 3-5.

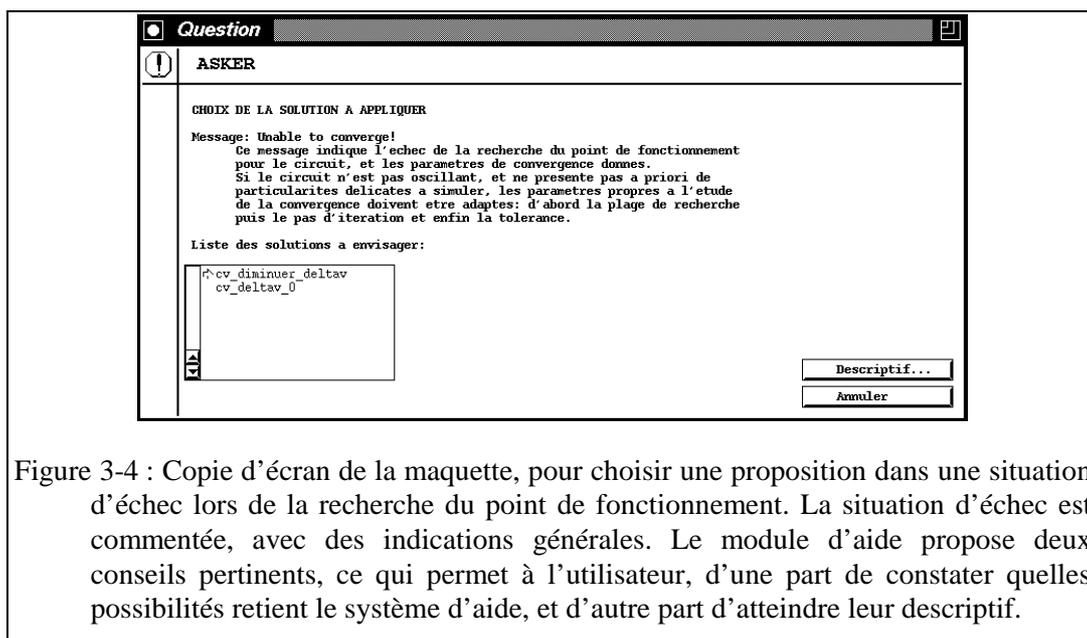


Figure 3-4 : Copie d'écran de la maquette, pour choisir une proposition dans une situation d'échec lors de la recherche du point de fonctionnement. La situation d'échec est commentée, avec des indications générales. Le module d'aide propose deux conseils pertinents, ce qui permet à l'utilisateur, d'une part de constater quelles possibilités retient le système d'aide, et d'autre part d'atteindre leur descriptif.

Ce descriptif de proposition, comprend une information sur les conditions d'utilisation, les avantages et les inconvénients, et enfin les modifications pour la mettre en œuvre. Dans l'exemple de la Figure 3-5, les actions portent sur la modification de trois paramètres, en précisant les valeurs suggérées.

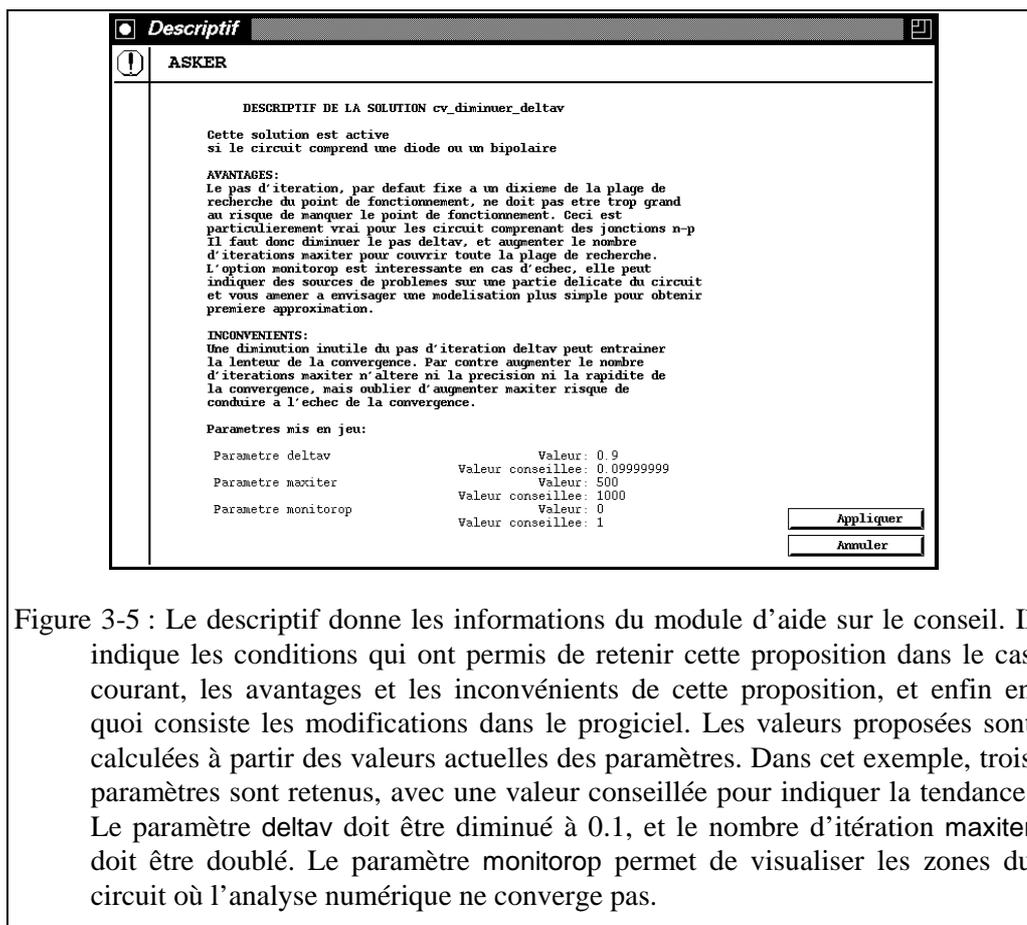


Figure 3-5 : Le descriptif donne les informations du module d'aide sur le conseil. Il indique les conditions qui ont permis de retenir cette proposition dans le cas courant, les avantages et les inconvénients de cette proposition, et enfin en quoi consiste les modifications dans le progiciel. Les valeurs proposées sont calculées à partir des valeurs actuelles des paramètres. Dans cet exemple, trois paramètres sont retenus, avec une valeur conseillée pour indiquer la tendance. Le paramètre deltav doit être diminué à 0.1, et le nombre d'itération maxiter doit être doublé. Le paramètre monitorop permet de visualiser les zones du circuit où l'analyse numérique ne converge pas.

Ce chapitre est une synthèse de la collaboration avec les utilisateurs et présente l'interaction qu'ils ont souhaitée et les besoins qu'ils ont exprimés. Les propositions introduites dans le module d'aide ont été ensuite validées avec les utilisateurs, soit avec la maquette, soit avec le prototype d'Advisor. C'est grâce à un développement itératif du projet que le système d'aide a pu s'ajuster aux demandes, exprimées aussi bien par des concepteurs avisés que par des néophytes dans la manipulation du progiciel.

Chapitre 4

4. Les fondements du module d'aide

Ce chapitre expose les fondements du module d'aide, qui ont été d'abord validés par la maquette, puis retenus pour le prototype et la version commerciale. Ces fondements sont de différentes natures :

- les outils construits pour étudier le progiciel : ce sont le contexte, le schéma de phases et la distinction entre les connaissances structurelles et contextuelles,
- la représentation de connaissance par objets,
- l'apport des métaconnaissances.

L'implication de l'utilisateur dans l'utilisation du module d'aide ne doit pas pénaliser son travail. Pour ne déléguer à l'utilisateur que la décision dans la recherche d'une solution à un problème, il faut qu'il puisse consulter l'état des données qui interviennent dans ce problème. L'équilibre consiste à laisser l'utilisateur choisir, mais en lui donnant accès à l'ensemble des informations utiles. Or l'intérêt des informations du point de vue de l'utilisateur dépend étroitement du contexte d'utilisation.

Dans le premier chapitre, l'exemple de l'influence du circuit sur le contrôle d'une simulation a introduit la notion de contexte. Avant d'aborder l'organisation des connaissances du module d'aide, qui permettent de donner à l'utilisateur des informations contextuelles, nous définissons ce que le contexte recouvre.

4.1. La définition du contexte

Le contexte englobe toutes les informations sur l'état du progiciel à un instant donné :

- l'objet de la conception en cours,
 - la situation de l'utilisateur dans le déroulement d'une session,
 - l'historique de la session avec l'ensemble des actions réalisées avec leurs résultats (satisfaction ou échec de la commande).
 - la connaissance qui détermine la pertinence des actions en tenant compte des trois points précédents.
-

De manière plus concrète, le simulateur permet d'illustrer cette définition. Ainsi, l'objet de la conception est le circuit électronique à tester. La situation de l'utilisateur correspond au type d'analyse menée sur le circuit, au cours d'une session de simulation. L'historique est utile si le résultat des actions précédentes donnent une information pour les actions futures.

Le dernier point retenu dans la définition du contexte est essentiel. Il représente les connaissances du noyau sémantique sur le fonctionnement du progiciel, qui permettent un suivi de l'activité. En effet, une succession d'actions de l'utilisateur est inexploitable sans la connaissance des contraintes qui régissent l'enchaînement. Le mécanisme complet de suivi du déroulement de la session est la combinaison du schéma de phases et du noyau sémantique qui modélisent le fonctionnement du progiciel.

4.2. Le schéma de phases pour le suivi de l'utilisateur

Le schéma de phases modélise les enchaînements possibles entre les principales étapes prévues dans le fonctionnement du progiciel et présentées par l'interface avec l'utilisateur. Il inclut les connaissances relatives à l'enchaînement des phases, telles que les pré-conditions pour chaque analyse.

Parce qu'il est organisé autour des analyses, le schéma de phases prend aussi en compte les commandes qui lancent ces analyses, et qui sont justement celles qui conduisent parfois à une situation d'échec. Cette organisation des connaissances autour des phases de travail de l'utilisateur permet de donner une sémantique à l'ensemble de ses actions.

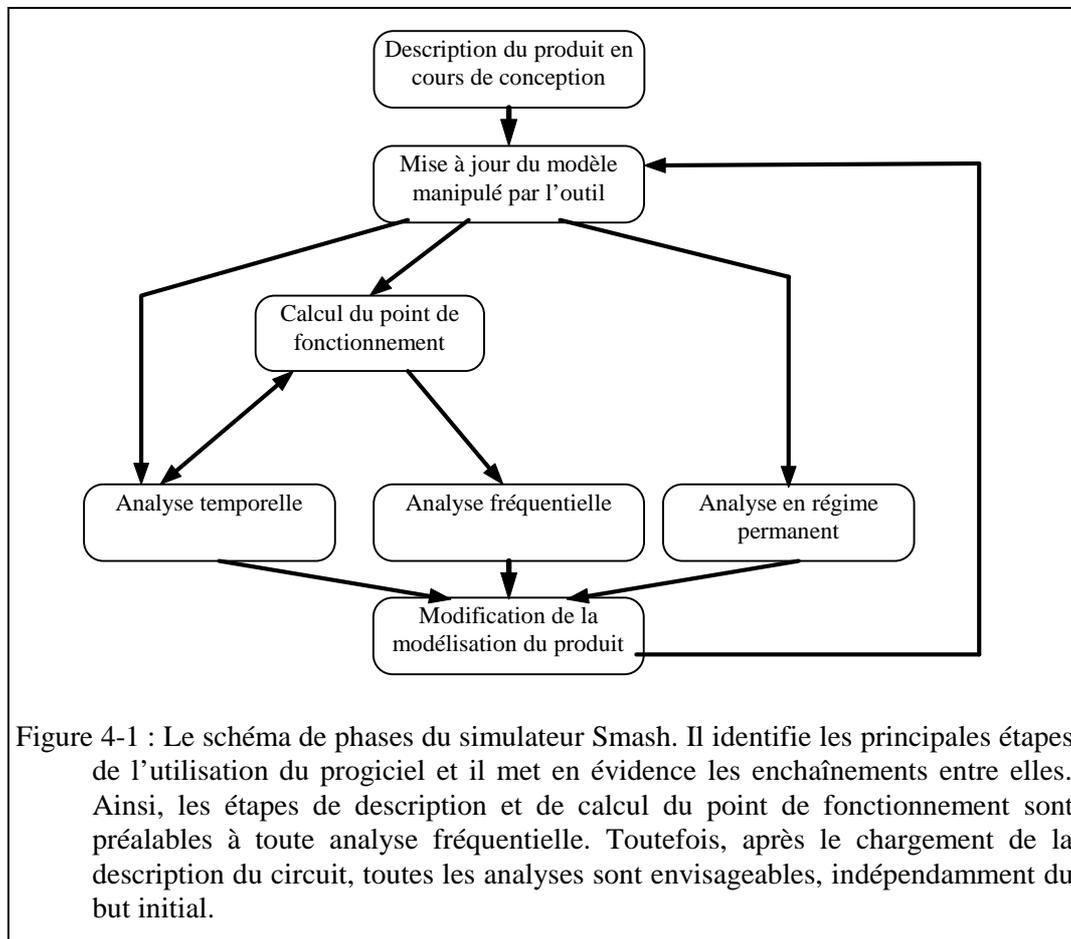
En effet, les actions de l'utilisateur transmises au module d'aide sont interprétées à l'aide du schéma de phases. Chaque phase modélise les connaissances pour la mise au point d'une analyse et présente un ensemble de caractéristiques :

- les conditions d'application d'une analyse, liées à l'objet de conception,
- la commande d'entrée dans la phase, qui correspond par exemple à la commande qui lance l'exécution de l'analyse,
- les analyses préalables, qui doivent être appliquées avec succès avant de considérer cette phase,
- les analyses suivantes envisageables,
- la liste des commandes qui en général peuvent être appliquées,
- la liste des paramètres qui de manière générale ont un effet sur l'analyse correspondante.

Du point de vue du système d'aide, une session est donc une succession de phases qui représentent les enchaînements exécutés par l'utilisateur. Au cours d'une session, une même analyse peut être appliquée plusieurs fois pour mettre au point le circuit, ou pour affiner la simulation. Dans le schéma de phases, cela correspond à des passages multiples dans une même phase, mais avec un contexte différent. Le contexte peut aussi évoluer au cours d'une même phase, pour refléter les modifications appliquées par l'utilisateur avec les commandes et les paramètres accessibles.

Le schéma de phases suivant (Figure 4-1) correspond au simulateur Smash. La première phase correspond au chargement du circuit pendant laquelle le simulateur construit les structures de données à partir de la description textuelle du circuit (voir netlist présentée au pre-

mier chapitre). Les phases suivantes sont les différentes études accessibles avec ce simulateur : analyse fréquentielle, temporelle ou en régime permanent. Quand la description du circuit est modifiée, une étape de chargement s'impose pour en tenir compte, avant de relancer l'analyse et vérifier si son résultat est devenu satisfaisant en temps et en précision.



Le schéma de phases représente les grandes étapes de travail au cours d'une session, mais il faut atteindre le niveau plus fin des actions appliquées dans chacune des phases. En conséquence, il faut modéliser dans le noyau sémantique, non seulement les phases, mais aussi les commandes et les paramètres mis à disposition de l'utilisateur.

Par contre, c'est grâce à la position de l'utilisateur dans le schéma de phases, qu'il sera possible de définir si les commandes et les paramètres sont ou non pertinents dans le contexte. L'aide contextuelle repose donc sur l'actualisation de la position de l'utilisateur, pour ainsi mettre à jour les informations contextuelles du module d'aide.

Cette actualisation doit être permanente, afin que l'aide donnée au cours de la session soit cohérente avec les actions précédentes de l'utilisateur. Dans la situation où un concepteur désactive le système d'aide, seules les interventions du système d'aide dans l'interface avec l'utilisateur sont inhibées. Ainsi, même s'il ne se manifeste pas, le système d'aide poursuit la construction de l'historique et la vérification de conflit, pour répondre à tout moment aux demandes de conseils.

4.3. La distinction entre les connaissances structurelles et les connaissances contextuelles

Le système d'aide d'une part renseigne l'utilisateur sur le fonctionnement du progiciel (ses options de manière générale), et d'autre part exerce un contrôle sur la bonne utilisation de ces options, en justifiant les suggestions dans le contexte. Ces deux aspects représentent deux types de connaissances.

Le fonctionnement du système d'aide repose sur un ensemble de connaissances que nous nommons structurelles, même si le terme plus précis serait « structurantes ». Ce sont les connaissances qui définissent le fonctionnement permanent du progiciel, sans tenir compte des restrictions du contexte. Elles correspondent à l'utilisation potentielle de chacune des options du progiciel, et forment l'ossature du noyau sémantique. Ces connaissances sont collectées auprès des développeurs du progiciel et des utilisateurs experts. Elles ne sont pas toutes représentées dans le schéma de phases, où en particulier les explications n'apparaissent pas. Néanmoins, le schéma de phases met en évidence le rôle de ces connaissances structurelles : pour chaque phase, elles représentent les actions *a priori* possibles. L'étape suivante est de sélectionner les actions effectivement pertinentes dans le contexte.

Le rôle des connaissances contextuelles est justement de définir si une action est ou non pertinente dans un contexte donné, et surtout pourquoi. En effet, quand le système d'aide signale un conflit dans le contexte, il faut qu'il indique quelle proposition n'est pas vérifiée. De cette manière, l'utilisateur peut comprendre la cause précise du conflit et ainsi apporter les corrections nécessaires. Dans le cas où une action est effectivement pertinente, il n'est pas prévu que le système d'aide félicite l'utilisateur, pour l'encourager à exploiter des options méconnues. Il est regrettable de toujours critiquer, mais la contrainte de limiter les interventions dans le cours du travail pousse à les réserver pour la prévention ou la correction.

Les connaissances contextuelles sont donc des règles de bonne utilisation qui explicitent pour chaque commande ou paramètre, son état par rapport au contexte. Elles sont l'expression de la restriction des connaissances structurelles au cas couramment traité, et constituent le noyau sémantique. En effet, selon le contexte courant, certaines des actions *a priori* possibles ne sont pas pertinentes, et d'autres actions deviennent influentes pour corriger un conflit. Pour fournir des explications adaptées au contexte, le système d'aide s'appuie sur l'état courant du progiciel et sur l'historique de la session.

4.4. La représentation par objets

La représentation par objets organise les connaissances et décompose le progiciel étudié, en privilégiant certaines de ses caractéristiques qui seront représentées par des classes d'objets. Le lecteur peut consulter l'ouvrage [Masini90], pour un exposé complet sur la représentation des connaissances par des objets.

Une classe rassemble les instances d'objets qui partagent une même définition et un même comportement, décrits respectivement par les attributs et par les méthodes. Les méthodes permettent de traiter des données et apportent ainsi un raisonnement local à l'objet, lui accordant une certaine autonomie. Par la suite, nous employons le terme « d'objet réactif » pour marquer l'importance de ce raisonnement local. Les attributs définis dans la classe sont instanciés pour tout nouvel objet de la classe, ce qui permet de prendre en compte la spécificité de chacun dans la limite de la définition de la classe. Les données de l'objet sont encapsulées,

c'est-à-dire qu'elles ne sont accessibles que par une demande explicite à l'objet de transmettre une de ses données propres. Cette encapsulation protège les données jugées sensibles et interdit toute modification abusive.

L'organisation des classes est hiérarchique, ce qui permet de représenter des connaissances partagées par plusieurs classes au niveau de leur classe mère commune. Chaque instance d'objet peut ainsi faire appel aux méthodes définies dans sa classe et dans les classes mères.

Le choix des classes et leur structure hiérarchique reflètent les caractéristiques qui sont considérées comme utiles pour étudier le progiciel. Plus concrètement, pour modéliser le progiciel et son fonctionnement, les caractéristiques retenues sont par exemple :

- les phases de travail décrites par le schéma de phases, avec l'analyse correspondante,
- les commandes et paramètres qui interviennent dans la mise au point de cette analyse.

L'organisation des commandes et des paramètres intègre la notion de phase de travail dans la hiérarchie des connaissances modélisées. Comme le montre le parallèle fait en Figure 4-2 entre la hiérarchie des phases et la hiérarchie des commandes, le haut de la hiérarchie est organisée par des classes abstraites, qui représentent les commandes dédiées à une analyse.

L'avantage de cette représentation est de prendre en compte les objets concrets, connus et manipulés par l'utilisateur du progiciel. Les conditions de fonctionnement s'expriment donc en fonction de paramètres et de commandes qui sont compréhensibles par l'utilisateur. La situation serait différente si, au lieu d'exploiter les éléments de l'interface avec l'utilisateur, une condition s'exprimait avec une donnée du noyau fonctionnel inconnue de l'utilisateur. Le choix de cette représentation n'est cependant pas une condition pour construire un système d'aide, tant il est possible de séparer les explications textuelles et les objets réellement manipulés. Néanmoins, il permet d'exploiter simplement la structure de classes au niveau de l'implémentation dans un langage de programmation par objets.

Une exploitation intéressante est la construction d'une trace de l'activité de l'utilisateur. La représentation par objets des contrôles, c'est-à-dire des paramètres et des commandes présents dans l'interface, permet d'obtenir facilement une telle trace. Dans le système d'aide, une commande est représentée par une classe d'objets. Son appel par l'utilisateur génère une nouvelle instance de cet objet commande. Par la génération dynamique des instances il est donc possible de tracer l'activité de l'utilisateur, si à chaque contrôle correspond une classe d'objets. Cette trace est nécessaire pour construire l'historique et en déduire ensuite le contexte pour une aide adaptée.

La modélisation par objets a permis l'expression des connaissances générales et contextuelles pour chaque composant du noyau sémantique. Elle représente avec le même formalisme :

- des objets concrets, comme le circuit,
 - des objets présents dans l'interface avec l'utilisateur, comme les paramètres et les commandes,
 - et des objets abstraits, comme les phases du schéma de phases qui sont des concepts introduits par la modélisation.
-

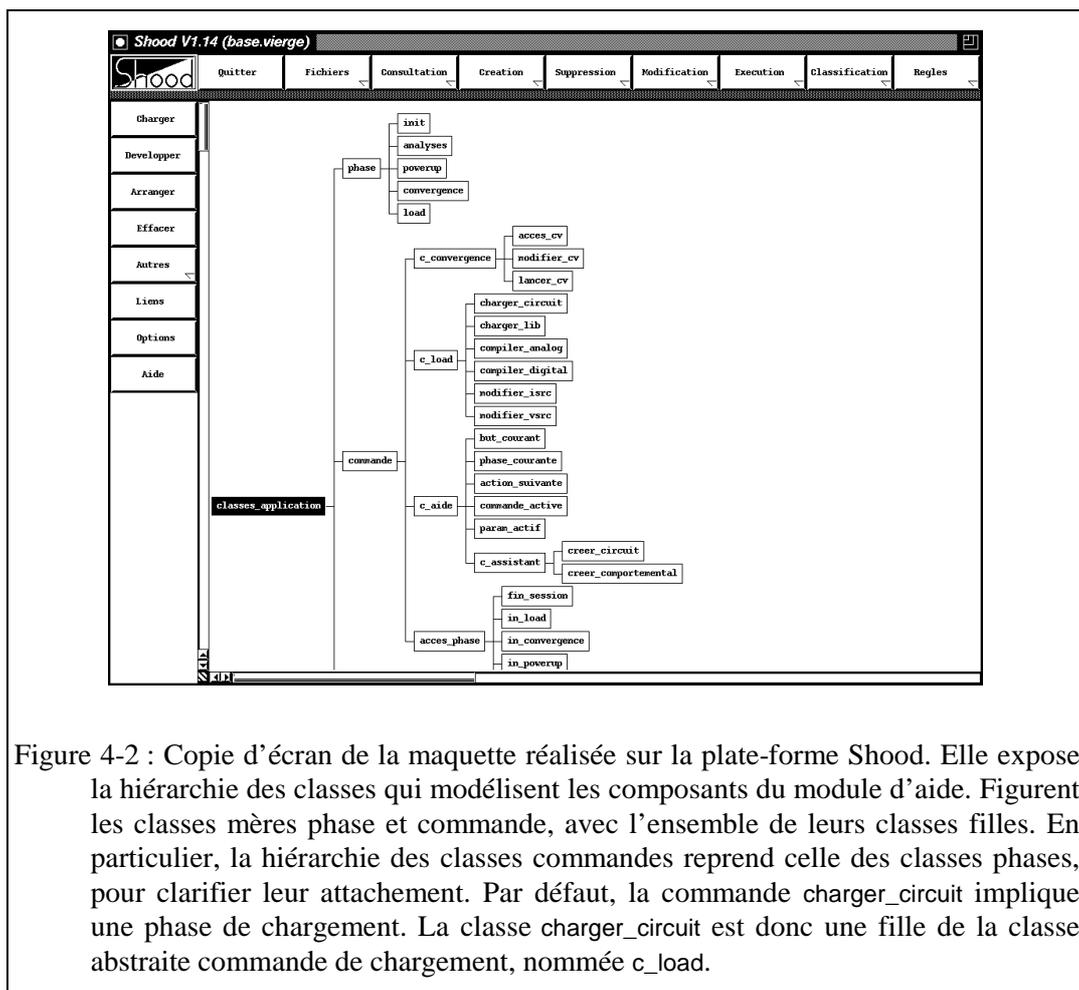


Figure 4-2 : Copie d'écran de la maquette réalisée sur la plate-forme Shood. Elle expose la hiérarchie des classes qui modélisent les composants du module d'aide. Figurent les classes mères `phase` et `commande`, avec l'ensemble de leurs classes filles. En particulier, la hiérarchie des classes commandes reprend celle des classes phases, pour clarifier leur attachement. Par défaut, la commande `charger_circuit` implique une phase de chargement. La classe `charger_circuit` est donc une fille de la classe abstraite `commande` de chargement, nommée `c_load`.

La représentation par objets a aussi facilité l'identification des connaissances nécessaires pour chaque objet : d'une part pour déterminer son état par rapport au contexte, c'est-à-dire pour savoir si sa modification était pertinente ou non, d'autre part, pour identifier quelles caractéristiques de l'objet renseignaient d'autres objets. La représentation par objets a ainsi mis en évidence les contraintes entre les composants du système d'aide qui modélisent le logiciel.

Du point de vue de l'utilisation, les objets sont donc une opportunité pour renseigner de manière concise chacun des éléments ainsi modélisés. En prenant l'exemple d'un paramètre, la représentation par objets permet d'explicitier ses contraintes d'utilisation qui jusqu'alors étaient disséminées dans le manuel. Les contraintes, les influences, les interdictions sont mises en évidence par les liens entre les objets correspondants. Ces liens constituent une connaissance du noyau sémantique, exploitée pour déterminer le contexte de travail. En effet, un conflit résulte du non respect d'une contrainte existant sur les données propres de l'objet, ou sur les relations avec les autres objets. Par exemple, l'intérêt de modifier un paramètre dépend des caractéristiques du circuit. L'encapsulation des données conduit à expliciter les liens par des échanges d'information entre les objets : pour déterminer si ce paramètre est utile, il faudra vérifier les caractéristiques du circuit en cours d'étude.

4.5. Les métaconnaissances

La maquette a contribué à une distinction simple entre les connaissances structurelles et les connaissances contextuelles, au niveau de leur représentation. En effet, la plate-forme Shood qui a servi au développement de la maquette, présente un mécanisme puissant d'expression des métaconnaissances [Bounaas95], couplé à une représentation par objets. La maquette a en conséquence établi une distinction naturelle entre :

- les connaissances structurelles, représentées par des métaconnaissances qui constituent la structure de la maquette ;
- les connaissances qui expriment les restrictions par rapport au contexte et qui sont écrites sous forme de règles de production de la forme « si A alors B ».

En fait, les connaissances sont réparties en trois niveaux, du général au spécifique, avec les méta-classes, les classes et les instances d'objet classes. La classe est un niveau d'abstraction, qui définit les connaissances que toutes ses instances partagent. Certaines classes partagent entre elles des connaissances générales qui peuvent être énoncées dans une méta-classe. Il existe donc une instanciation multi-niveaux pour distinguer l'information permanente et spécifique.

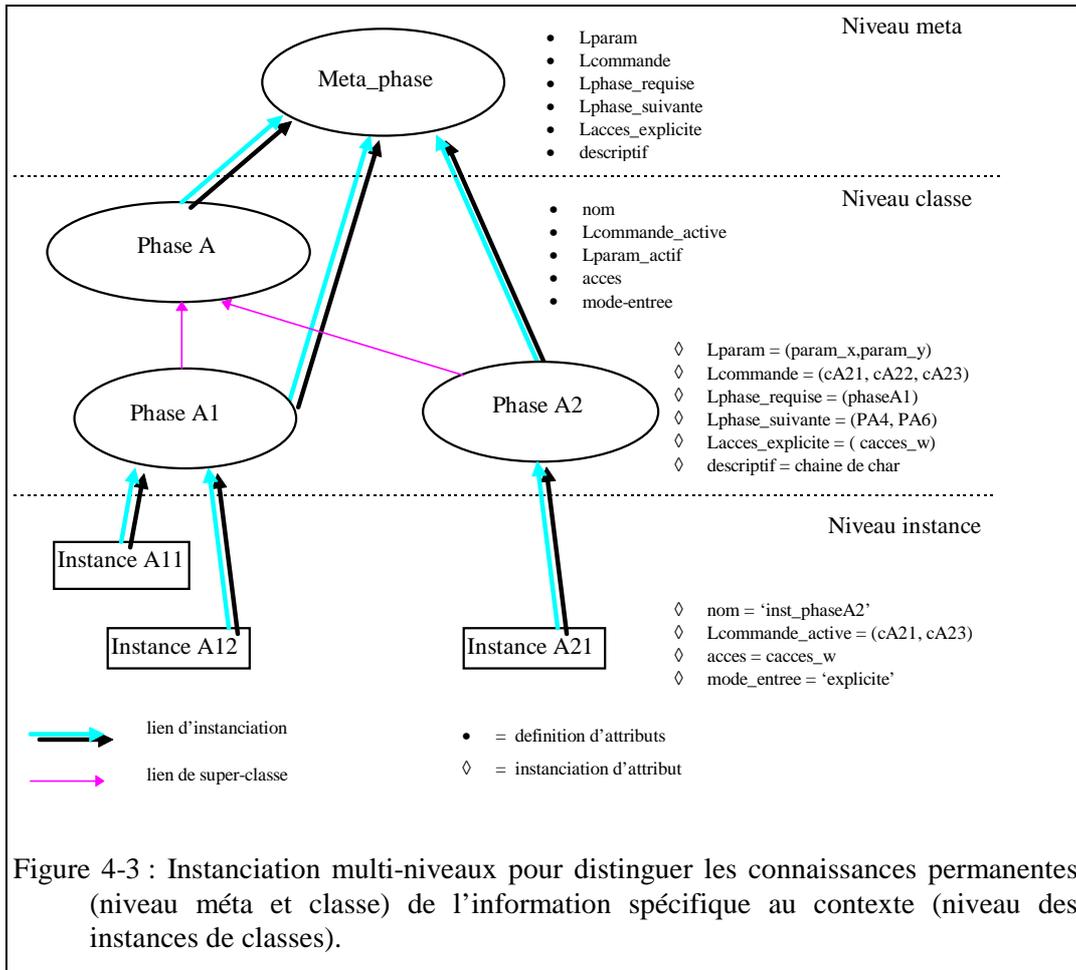
Nous illustrons cette organisation du noyau sémantique avec l'exemple de la modélisation des connaissances du schéma de phases. Ce dernier introduit deux classes d'objets :

- la classe *Phase* qui modélise les caractéristiques des phases mises en évidence par le schéma présenté par la Figure 4-3 ;
- la classe *Session* qui modélise le déroulement de l'étude d'un circuit comme une succession de phases.

Considérons la notion de phase durant laquelle une analyse du circuit est mise au point. Une phase est décrite par trois niveaux de représentation : la méta-phase, la classe phase et les instances de cette classe :

- Le niveau méta-phase est la description d'une phase abstraite, qui indique qu'au cours d'une phase concrète un ensemble de commandes et un ensemble de paramètres peuvent *a priori* être manipulés. En conséquence, la méta-phase définit les structures de données qui seront des listes de commandes et de paramètres. Par ailleurs, le niveau méta précise qu'une phase doit connaître ses conditions d'application, et en particulier il définit les structures de données pour modéliser les phases précédentes requises et les phases suivantes possibles, avec la possibilité de spécifier une préférence exprimée par l'attribut *phase_suivante*. A ce niveau méta est aussi défini un accès aux explications décrivant notamment le rôle de la phase.
 - Le niveau de la classe modélise une phase, dans laquelle une analyse particulière est appliquée. Les attributs définis au niveau méta-phase sont instanciés dans cette classe phase. Par exemple, la classe phase de chargement du circuit n'a aucune phase préalable et présente toutes les autres phases comme étant ensuite envisageables. Son attribut *phase_suivante* sera instancié par une classe « phase de recherche du point de convergence ». Les attributs instanciés définissent ainsi les caractéristiques propres à ce type de phase. L'ensemble des classes de phases constituent les connaissances structurelles sur le schéma de phases. C'est aussi au niveau de la classe de phase que les attributs des instances sont déclarés.
-

- Une instance de la classe phase modélise l'état actuel ou un état déjà passé. Pour tout nouvel état, une instance de phase est créée pour représenter ses caractéristiques, telles que les commandes et les paramètres qui sont effectivement pertinents dans le contexte. Ces caractéristiques sont un sous-ensemble des commandes et des paramètres définis dans la classe phase comme *a priori* envisageables, et nous verrons par la suite comment elles sont construites. Au cours de la session, les instances de phases constituent un historique des états parcourus par l'utilisateur.



La métaconnaissance est une connaissance sur la connaissance manipulée. Typiquement, « on sait » qu'une phase admet un descriptif général et qu'elle a des caractéristiques d'enchaînement dans le schéma de phases. Néanmoins, la métaconnaissance n'est pas exploitée pour la construction des explications, ni même pour « méta-expliquer », c'est-à-dire pour améliorer les performances du système d'aide en faisant en sorte qu'il s'explique les intérêts d'une stratégie par rapport à d'autres, ou par rapport au contexte. Cette dernière approche est discutée par J.Pitrat [Pitrat96], où le problème est d'apprendre à la machine à faire mieux. Notre exploitation de la métaconnaissance est plus modeste. Elle permet surtout de structurer l'ensemble des connaissances, y compris pour les explications destinées à l'utilisateur.

4.6. La maquette valide le système réactif

La maquette est le résultat de la mise en œuvre de ces fondements du module d'aide. En dépit des erreurs de modélisation qu'elle visait justement à identifier, son fonctionnement satisfaisant a validé ces points essentiels. Pour le développement suivant, ces points fondamentaux sont donc à nouveaux exploités, en apportant les corrections nécessaires. Nous faisons ici un résumé de l'intérêt des différents moyens mis à contribution.

L'encapsulation des données répond au besoin d'une information centrée sur la définition et les données d'un objet. Cette information doit être transmise à l'utilisateur quand l'objet interrogé correspond à un élément concret de l'interface, comme un paramètre par exemple. Cependant, le système d'aide ne repose pas seulement sur la description du progiciel, mais aussi sur son fonctionnement.

A l'encapsulation des données, la modélisation par objets ajoute le raisonnement local à l'objet, qui va permettre de satisfaire le besoin de contrôle du progiciel. En effet, pour que l'utilisateur s'approprie le progiciel, il faut qu'il puisse découvrir les contraintes sur chaque composant de l'interface, qui sont justement exprimées par les connaissances du noyau sémantique. Par contre, l'évaluation de la maquette a montré que l'utilisateur ne tirait aucune information de l'exposition du fonctionnement du système d'aide lui-même. Ainsi, il faut définir une structure qui offre une visibilité utile à l'utilisateur sur le fonctionnement du progiciel, mais sans tomber dans l'excès de rendre visible le fonctionnement du système d'aide.

Le besoin de contrôle porte sur chacun des objets manipulés par l'utilisateur. L'idée est alors de construire le système d'aide comme un « ensemble d'agents » avec leurs connaissances et leur raisonnements, construits sur chacun des objets modélisés. Ces objets sont en particulier capables de répondre à une demande d'information faite par l'utilisateur. En appliquant un raisonnement local, un objet est capable de se déterminer par rapport au contexte. L'ensemble des objets composent alors un système réactif, dans le sens où ce dernier réagit à l'évolution du contexte.

Les objets liés au suivi de l'utilisateur constituent un premier exemple d'objets réactifs, qui a pu être validé au stade de la maquette. Voyons d'abord comment le suivi des actions de l'utilisateur, grâce au schéma de phases, permet de construire le contexte et en particulier, comment le raisonnement local permet de mettre à jour la « position » de l'utilisateur dans ce schéma, à chacune de ses actions.

Le suivi de l'utilisateur se fait dans un cycle de traitement de chaque nouvelle commande. Au début du cycle, les données sont : une session ouverte *S*, une phase courante *Pc* et une nouvelle commande *C*. Le traitement de l'application de la commande consiste de manière générale à :

- Vérifier si la nouvelle commande *C* est définie comme une des commandes envisageables dans la phase *Pc*. Dans l'affirmative, l'actualisation du schéma de phases se limite à la mémorisation de cette commande dans l'historique de la phase.
- Dans la négative, la commande *C* doit créer une nouvelle phase. Le choix de la phase créée est défini par la métaconnaissance de la commande : un attribut fixe la phase à laquelle par défaut la commande est attachée. Par exemple, l'ajout de bibliothèques de composants est une commande qui par défaut est attachée à la phase de chargement.

- La nouvelle phase créée renseigne alors la session **S**, qui contrôle l'enchaînement des phases. La session mémorise le nouvel état et ajoute à l'historique des phases l'état précédent, qui est une instance d'une classe phase.

Ce traitement est nuancé par le résultat de l'application de la commande. Dans le cas d'une situation d'échec par exemple, il est nécessaire de travailler dans la phase prévue pour résoudre le problème obtenu. C'est justement la phase à laquelle la commande est attachée par défaut. Elle connaît, par son niveau méta, l'ensemble des conseils successibles de résoudre les problèmes des commandes attachées.

Le traitement de l'application d'une commande est donc d'abord effectué au niveau de l'objet commande, en utilisant le raisonnement propre à cet objet. Ce raisonnement est décrit par les méthodes de l'objet commande, qui peuvent tenir compte du résultat de la commande. Au cours du traitement, l'objet commande propage le raisonnement à d'autres objets, comme une phase ou la session.

Ainsi, chaque commande est modélisée par une instance d'objet qui peut, de manière autonome, décider de créer un nouvel état ou de simplement actualiser la session en construisant l'historique des actions. En conséquence, la position dans le schéma de phases est actualisé à la fin du cycle. La prise en compte de l'évolution du contexte est donc assurée par l'explicitation des conséquences de chaque action, qui dépend du contexte initial.

La maquette a montré que le suivi de l'utilisateur construit par l'ensemble des objets était efficace. Les états déduits des commandes successives étaient effectivement compatibles avec la situation réelle de l'utilisateur dans le progiciel.

L'idée est ensuite d'étendre cette approche d'objets autonomes, capables de prendre en compte le contexte initial pour en définir un nouveau. En effet pour le suivi de l'utilisateur, seules les commandes et les états ont été discutés.

Chapitre 5

5. Les connaissances sur le fonctionnement du progiciel

Les objets manipulés dans l'interface avec l'utilisateur sont un moyen privilégié pour solliciter le module d'aide, car ils donnent un support concret pour demander des informations. Avant d'étendre la notion d'objets réactifs à tous les composants du système d'aide, nous insistons sur l'intérêt de cette distinction entre les connaissances structurelles et contextuelles pour la description du progiciel.

Initialement, l'importance des connaissances structurelles est apparue pour les conseils que le système devait proposer à l'utilisateur. Notre souci est de laisser l'utilisateur vérifier quelles connaissances ont été considérées par le système d'aide pour proposer *in fine* un conseil particulier. Pour évaluer les conseils, il est intéressant de distinguer parmi les connaissances :

1. l'ensemble des connaissances manipulées par le système d'aide ;
2. l'ensemble des propositions retenues dans le contexte en cours ;
3. les explications de ce choix, en indiquant en particulier pourquoi une possibilité n'était pas retenue.

Pour répondre au point 1, il est nécessaire de modéliser les connaissances générales sur chaque problème, afin que le système d'aide puisse clairement indiquer ce qu'il connaît : c'est le rôle des connaissances structurelles. Par ailleurs, les points 2 et 3 appellent la description du raisonnement appliqué pour sélectionner les propositions en fonction du contexte. Les contraintes du contexte sur les connaissances structurelles sont exprimées par les connaissances contextuelles.

5.1. L'organisation des connaissances

Nous présentons ici l'organisation des connaissances sur le fonctionnement du progiciel, qui sont requises par le système d'aide pour effectivement apporter des conseils pour mieux manipuler le progiciel.

Le système d'aide exploite pour son propre fonctionnement (**Erreur! Source du renvoi introuvable.**) :

- le schéma des phases de travail au cours d'une session, avec les métaconnaissances sur les paramètres et les commandes disponibles,
-

- le noyau sémantique concerne les contrôles sur le progiciel, accessibles par l'utilisateur, c'est-à-dire les paramètres et les commandes présentées dans l'interface. Il a pour rôle de décrire leurs caractéristiques générales et les connaissances contextuelles qui leur sont attachées, pour commenter les données modifiées par l'utilisateur. Le module d'aide duplique donc les données du noyau fonctionnel dans le noyau sémantique.

Le système d'aide communique à l'utilisateur les possibilités et les contraintes modélisées avec les explications, dont l'organisation est discutée ultérieurement.

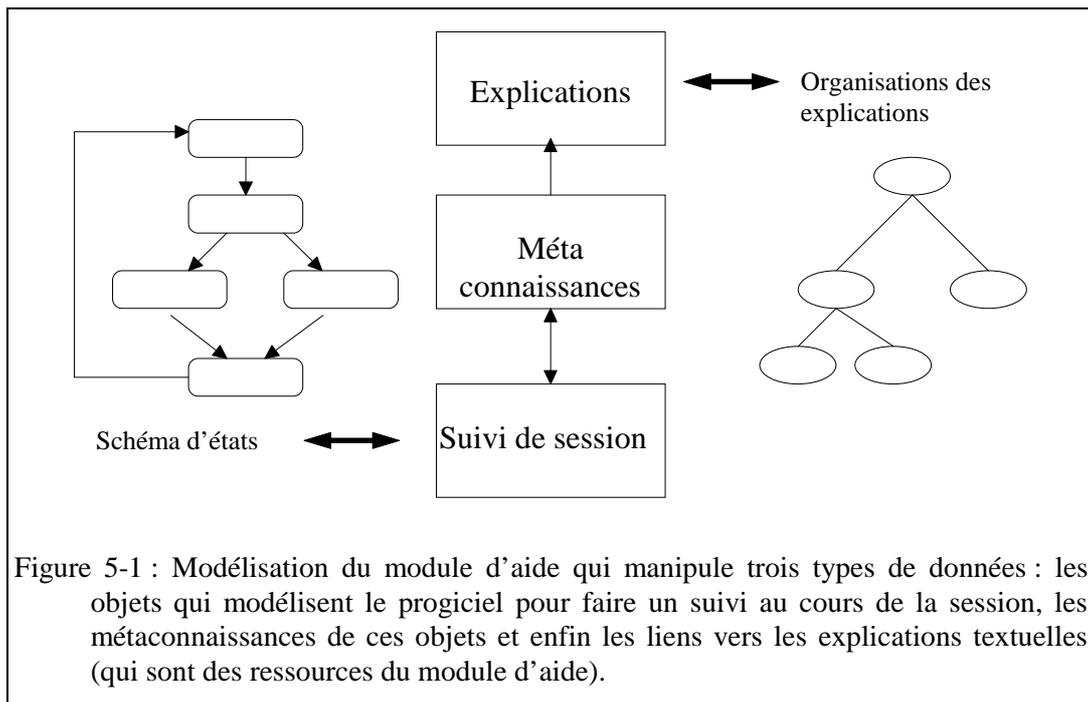


Figure 5-1 : Modélisation du module d'aide qui manipule trois types de données : les objets qui modélisent le progiciel pour faire un suivi au cours de la session, les métaconnaissances de ces objets et enfin les liens vers les explications textuelles (qui sont des ressources du module d'aide).

L'« intelligence » du système d'aide est tributaire de son noyau sémantique et de sa capacité à déterminer la position occupée dans le schéma des phases au cours d'une session. En effet, les explications ne font que refléter la modélisation du fonctionnement du progiciel.

5.1.1. L'organisation autour du schéma de phases

Chaque phase de travail correspond à la mise au point et à l'application d'une analyse dans le progiciel. L'idée de cette modélisation est de rassembler les connaissances susceptibles de renseigner cette analyse, sans devoir considérer toutes les options du progiciel. La description d'une phase comprend donc :

- la commande dite principale qui lance l'analyse,
- l'ensemble des paramètres qui sont susceptibles d'intervenir dans l'analyse,
- les commandes secondaires qui interviennent dans cette phase (voir Figure 5-2). Ce sera soit une spécialisation de la commande principale suffisamment courante pour être prévue comme une option à part entière, soit un traitement sur les résultats obtenus de la commande principale.

Cette modélisation du fonctionnement en phases de travail n'a pas pour but de limiter la maîtrise de l'utilisateur sur le progiciel. Son intérêt est au contraire de communiquer à l'utilisateur l'ensemble des possibilités connues du système d'aide. De cette manière il est possible de mettre en évidence l'influence d'un contrôle, que ce soit un paramètre ou une commande, sur une phase et ce indépendamment de sa présentation. Ici, le contrôle n'est pas nécessairement attaché visuellement à cette analyse dans l'interface, en étant par exemple disponible dans la même boîte de dialogue que l'analyse. La décision d'inclure un paramètre dans la liste des contrôles d'une analyse dépend ainsi de son influence, et non pas de l'interface avec l'utilisateur.

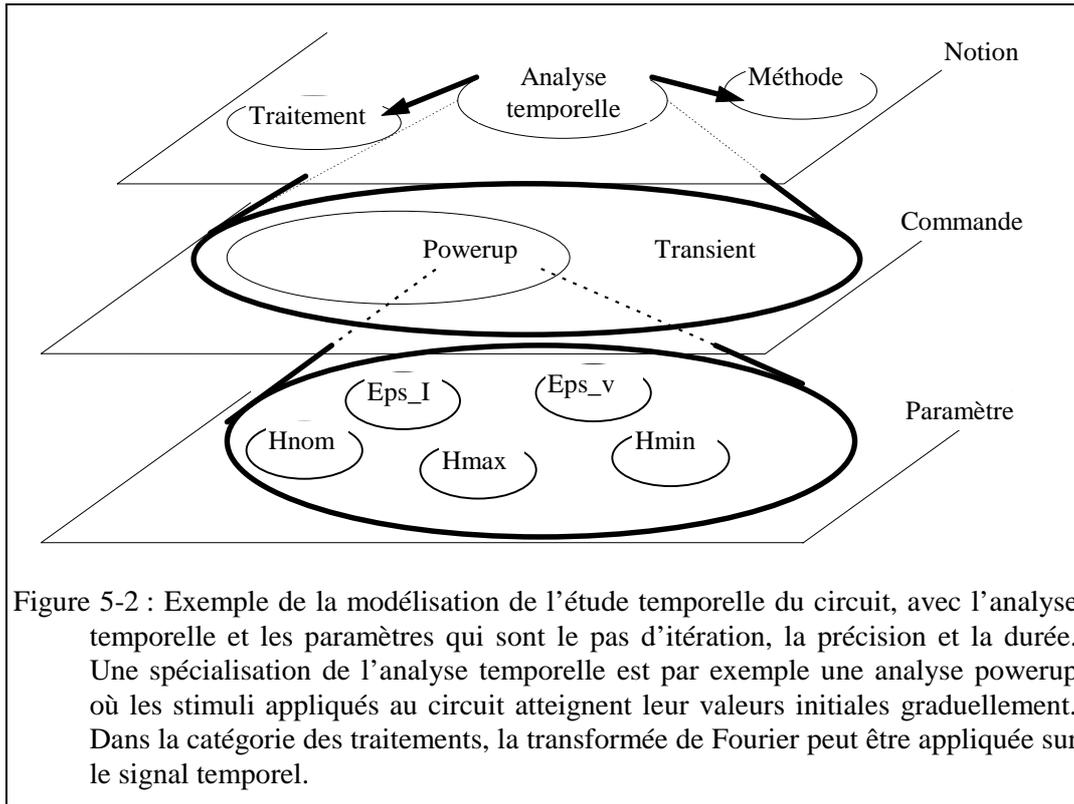


Figure 5-2 : Exemple de la modélisation de l'étude temporelle du circuit, avec l'analyse temporelle et les paramètres qui sont le pas d'itération, la précision et la durée. Une spécialisation de l'analyse temporelle est par exemple une analyse powerup où les stimuli appliqués au circuit atteignent leur valeurs initiales graduellement. Dans la catégorie des traitements, la transformée de Fourier peut être appliquée sur le signal temporel.

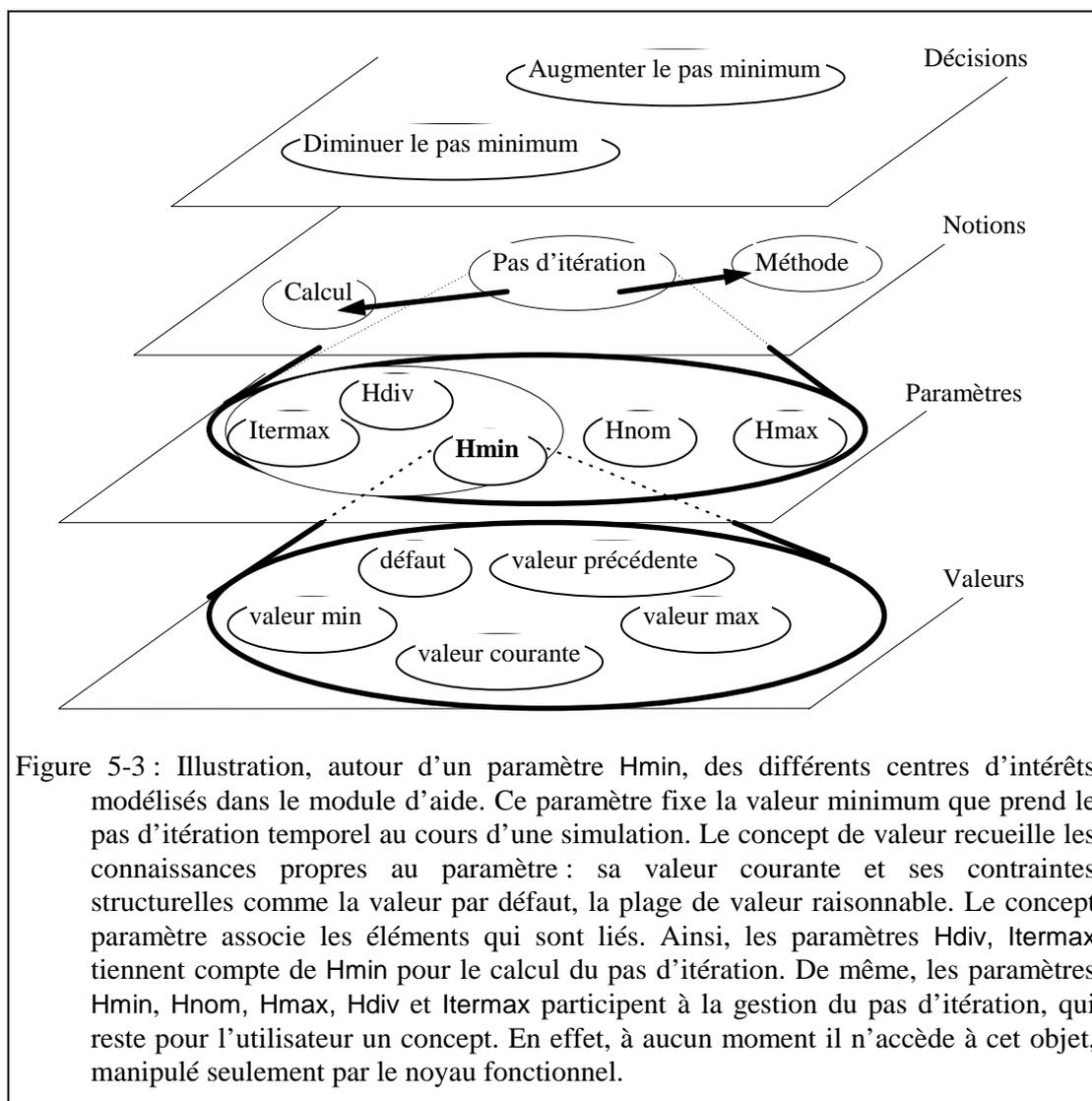
Cette modélisation concentre l'information sur l'utilisation du progiciel pour les phases de travail modélisées. Cependant son intérêt dépend de la capacité du système à correctement déterminer la position occupée dans le schéma des phases. C'est pourquoi il est nécessaire de définir quels sont les critères pour décider du début et de la fin d'une phase de travail. Ce problème est discuté avec le suivi de l'activité de l'utilisateur, dans le chapitre 6.

5.1.2. Les connaissances du noyau sémantique

Nous avons introduit le noyau sémantique dans le premier chapitre, comme le moyen d'exploiter certaines données du noyau fonctionnel du progiciel, par le système d'aide. En effet, toutes les données du noyau fonctionnel ne sont pas connues de l'utilisateur. Le noyau sémantique renseigne les seules données accessibles, telles que les contrôles sur le progiciel que sont les paramètres et les commandes, et la description du processus étudié qui est le circuit dans notre application.

La sélection des données en fonction de leur accessibilité est simplement liée à l'objectif de l'aide : les explications et les conseils doivent permettre à l'utilisateur de mieux manipuler le progiciel. Nous verrons que cette réalité palpable est un support efficace pour insérer les interventions du système d'aide sans perturber le travail en cours.

La figure suivante (Figure 5-3) illustre les connaissances sur l'utilisation d'un paramètre, qui doivent être manipulées par le noyau sémantique. Elle met en particulier l'accent sur les influences entre les paramètres.



Une connaissance du noyau sémantique est destinée d'une part à l'utilisateur, sous forme d'explication textuelle, et d'autre part au système d'aide sous forme de caractéristiques des objets modélisés. Dans un premier temps, nous discutons de la nature des connaissances du noyau sémantique, et nous illustrons leurs expressions exploitées dans le fonctionnement du module d'aide. Les explications textuelles destinées à l'utilisateur sont discutées ultérieurement.

5.1.2.1. Les conditions d'utilisation

De manière générale, l'application ou la modification d'un contrôle, selon que ce soit une commande ou un paramètre, est subordonnée à la vérifications de pré-conditions. Cette situation est souvent mise en évidence par des menus « grisés » dans l'interface entre le progiciel et l'utilisateur. Par contre, il n'est pas toujours évident de comprendre quelle est, ou quelles sont les pré-conditions qui ne sont pas satisfaites, dans la mesure où un contrôle « grisé » ne répond plus aux actions de l'utilisateur. A moins que dans cet état particulier, le contrôle réponde à une sollicitation en indiquant sa pré-condition principale. Ce comportement est d'ores et déjà proposé dans certains progiciels. Par exemple, le traitement de texte utilisé pour ce mémoire, affiche dans la barre de message le signalement « Commande non disponible : aucun texte n'a été sélectionné » si la commande de copie de texte est appelée sans sélectionner de texte.

Le traitement des pré-conditions est envisagé dans le noyau sémantique, dans la mesure où il peut être affiné en fonction du contexte. L'apport du module d'aide est alors d'évaluer la pertinence d'un contrôle, c'est-à-dire de s'assurer que non seulement ses pré-conditions fonctionnelles sont vérifiées, mais aussi que les conditions d'utilisation qui peuvent être évaluées dans un contexte connu sont elles aussi vérifiées. Le module d'aide permet de ne pas seulement considérer les contrôles potentiels, mais de se concentrer sur ceux, dits candidats, qui ont une influence sur le travail en cours.

5.1.2.2. Les règles de bonne utilisation

L'expression des conditions d'utilisation d'un contrôle permet d'informer l'utilisateur d'un problème potentiel, avant qu'il ne confirme son application. Néanmoins, son emploi se définit d'abord en fonction de ce qu'il permet de faire. En particulier pour s'appropriier le progiciel, l'utilisateur a besoin de connaître les analyses disponibles dans le progiciel et les moyens de les maîtriser.

Il attend donc des informations sur les différents contrôles :

- pour quelle étude sont-ils pertinents ;
- leur description, c'est-à-dire quelles analyses déclenchent-ils ;
- les contrôles qui les influencent, comme c'est le cas pour les paramètres d'une commande ;
- les contrôles connexes, spécialisés ou au contraire plus général, pour découvrir des options adaptées ;

Dans le module d'aide, ces connaissances sont formalisées par :

- Le schéma de phases, où chaque objet phase connaît les paramètres et les commandes qui sont potentiellement utiles pour mener l'étude correspondant à la phase.
 - Les objets commandes et paramètres, qui référencent les phases dans lesquels ils sont susceptibles d'être appliqués. De cette manière, il est possible de savoir dans quelles études il peuvent être impliqués.
 - Les objets commandes référencent leurs paramètres potentiels et chacun d'eux évaluera son influence pour le contexte en cours.
-

Commande	
nom :	chaîne de caractères
explication_obj :	instance de la hiérarchie des explications
meta_obj :	instance de la hiérarchie du niveau méta
historique_liste :	liste des instances de la classe
satisfaction :	identificateur {Id_satisfy, Id_failure, Id_cancel}
msg_erreur :	identificateur de l'erreur
est_achevé :	booléen pour indiquer si la commande est achevée
phase_courante :	phase d'appel de la commande
param_liste :	liste des instances de paramètres utiles à la commande
Id_relevant :	id de l'explication, mis à jour par la règle réflexe Is_relevant
Id_redondant :	id de l'explication, mis à jour par la règle réflexe Is_redondant
Id_influent :	id de l'explication, mis à jour par la règle réflexe Is_influent
Is_relevant :	méthode pour déterminer si la commande est déclenchable
Is_redondant :	méthode pour déterminer si la commande est redondante
Is_influent :	méthode pour déterminer si la commande est influente

Figure 5-4 : La classe commande présente un ensemble d'attributs privés qui ne peuvent être modifiés qu'à travers les méthodes publiques, c'est-à-dire accessibles par tous les objets du système. Seules les méthodes correspondant aux règles réflexes sont listées. Elles actualisent les attributs qui référencent les justifications de leurs résultats. La notation OMT est ici employée ; elle distingue les attributs des méthodes.

La représentation de ces connaissances dans le noyau sémantique repose sur le principe de la distinction entre les connaissances structurelles et les connaissances contextuelles (Figure 5-4), c'est-à-dire entre les possibilités déclarées et les règles qui définissent si une possibilité est ou non concrétisée dans un contexte donné.

Pour notre application, nous avons retenu trois types de règles :

- la règle `Is_relevant`, qui vérifie les pré-conditions d'utilisation d'un contrôle,
- la règle `Is_redondant`, qui vérifie que l'application du contrôle n'est pas redondante avec les actions précédentes de l'utilisateur,
- la règle `Is_influent`, qui permet de distinguer parmi les contrôles candidats ceux qui sont connus pour intervenir dans la résolution d'une situation d'échec donnée.

Ces trois règles ne sont pas indépendantes, mais elles apportent chacune une information pour l'utilisation adéquate du contrôle. En particulier, la notion d'action redondante est traitée à part entière, bien qu'elle soit aussi une condition pour déclarer un contrôle candidat.

La règle `Is_influent` intervient aussi pour signaler un problème potentiel, en dehors d'une situation d'échec. En particulier, quand la valeur numérique d'un paramètre est en dehors d'une plage raisonnable, sa règle est validée pour signaler à l'utilisateur cette anomalie qui peut expliquer un problème futur ou déjà avéré.

Le noyau sémantique associe par ailleurs chaque contrôle modélisé, à un objet dans la hiérarchie distincte des explications. Ces explications sont l'expression :

- des connaissances structurelles et contextuelles, jusqu'ici formalisées pour être exploitées par le module d'aide,
- des connaissances sur l'utilisation du progiciel, qui ne sont pas modélisées dans le module d'aide. Par exemple, la notion de description du circuit permet d'expliquer pourquoi le simulateur exploite deux fichiers.

5.1.2.3. Les vérifications avant une confirmation

En particulier pour un paramètre numérique, il existe des contraintes sur la valeur qu'il peut ou ne peut pas prendre. Ces restrictions assurent une cohérence avec la sémantique du paramètre. Il existe plusieurs catégories de contraintes :

- Une restriction absolue qui reste toujours valide. Par exemple, une durée ne peut pas être un nombre négatif.
- Une restriction relative, qui reflète un lien logique et simple entre plusieurs paramètres (Figure 5-5). Par exemple, le pas nominal d'itération est compris entre la borne minimale fixée par la valeur du paramètre « pas inférieur » et celle maximale fixée par la valeur du « pas supérieur ». Cette longue phrase s'exprime plus simplement par la relation $h_{min} < h_{nom} < h_{max}$, si le pas d'itération est noté h .

```

BOOL Is_Relevant_param_hnom( CNAVIGSESSION* pSession )
{
    BOOL Is_valid = FALSE;
    //Get influent parameters : pParam stands for current parameter, pDuration for the duration
    //of the analyse set by user.
    CNAVIGPARAMETER* pParam = (CNAVIGPARAMETER*)pSession->GetLParam()-
    >FindParam( ID_TIME_HNOM )->GetHead();
    CNAVIGPARAMETER* pDuration = (CNAVIGPARAMETER*)pSession->GetLParam()-
    >FindParam( ID_TRANSIENT_DURATION )->GetHead();
    if( pSession->IsAnyLoadCircuit() )
    { Is_valid = TRUE;          //will be TRUE if circuit is loaded
      double val = pDuration->GetValueTemporal();
      // According to Duration value, sets appropriate expected values for the current parameter :
      pParam->GetExpli()->SetMinExpected( val*1e-6 );
      pParam->GetExpli()->SetMaxExpected( val*1e-2 );
    }
    return Is_valid;
}

```

Figure 5-5 : Exemple de la règle `Is_relevant` pour le paramètre `hnom`. La modification de ce paramètre est acceptée dès qu'un circuit est chargé, indépendamment de la phase de travail. La partie grisée concerne la définition de la plage de valeur raisonnable : elle dépend de la durée de l'analyse fixée par l'utilisateur, avec le paramètre `Duration`.

Ces vérifications avant une confirmation peuvent soit être déléguées au noyau fonctionnel, soit être traitées par le noyau sémantique pour expliciter les restrictions qui ne sont pas respectées. Dans ces situations, il s'agit d'un conflit bloquant étant donné que la vérification

assure la cohérence des données modifiées dans le noyau fonctionnel. Si elle est confiée au le noyau sémantique, elle sera assurée par la règle `Is_relevant`.

La dernière catégorie de contraintes porte sur la définition d'une plage de valeur raisonnable, qui est incluse dans le domaine des valeurs possibles, défini par les restrictions assurées par le noyau fonctionnel. La définition d'une plage raisonnable est destinée à identifier les causes d'un problème potentiel ou avéré. Le non respect de cette plage ne doit pas pour autant empêcher une étude. La règle `Is_influent` devra donc tenir compte de cette source de problèmes.

5.1.3. Les règles réflexes pour tenir compte du contexte

Les règles réflexes sont des méthodes attachées à chacun des objets du module d'aide : elles permettent de tenir compte du contexte. Elles représentent leurs connaissances contextuelles, évaluées chaque fois que l'utilisateur utilise le contrôle modélisé par l'objet. Les principales règles, `Is_relevant`, `Is_influent`, `Is_redundant`, définissent respectivement si l'objet peut être modifié, si sa modification a effectivement une influence dans le contexte, et si son appel est redondant vis-à-vis des actions précédentes.

Dans le module d'aide, l'application d'un contrôle est représenté par une instance de cet objet. Les attributs de l'instance sont destinés à mémoriser les informations contextuelles, obtenues par l'application des règles qui sont des connaissances contextuelles.

5.1.3.1. Les pré-conditions d'un contrôle : un exemple de règle

Les conditions d'utilisation de chaque contrôle sont exprimées par une méthode de l'objet qui le modélise. Cette méthode, nommée `Is_relevant`, vérifie les pré-conditions de fonctionnement et les pré-conditions d'influence, pour indiquer si le contrôle est pertinent dans le contexte en cours.

Nous prenons l'exemple d'un paramètre présent dans le progiciel, qui permet de modifier certains composants électriques du circuit. Il existe en fait deux pré-conditions :

- le circuit doit être correctement chargé. Dans le cas contraire, ce contrôle ne peut pas être appliqué ;
- le circuit doit inclure au moins un composant sensible à ce paramètre. Dans le cas contraire, ce contrôle n'aura aucun effet immédiat sur le travail ;

Ainsi les contrôles de l'interface modélisés dans le module d'aide expriment leurs conditions d'utilisation en fonction de l'influence qu'ils ont effectivement sur l'étude en cours. Par ailleurs, chacun des objets mémorise la pré-condition qui n'est pas vérifiée de manière à donner à l'utilisateur une des deux explications suivantes :

- « le contrôle n'est pas disponible car le circuit n'est pas correctement chargé »
- « le contrôle n'a pas d'influence sur le circuit en cours qui ne comprend pas de composant de type transistor bipolaire ».

Les pré-conditions sont organisées en fonction de leur priorité. Dans l'exemple précédent, le fonctionnement du progiciel impose d'abord de charger un circuit pour ensuite évaluer ses caractéristiques.

Dans notre application, le module d'aide complète la gestion des pré-conditions faite par le progiciel, dans la mesure où il intervient quand un contrôle est autorisé par le progiciel : c'est

une première condition. La seconde condition détermine parmi les contrôles potentiels, ceux qui ne sont pas candidats. L'utilisateur distingue donc ces deux niveaux de pré-conditions, traitées respectivement par le progiciel et par le système d'aide. Il mesure de cette manière en quoi le contexte intervient dans la bonne utilisation du progiciel. Il peut surtout choisir de ne pas tenir compte d'une pré-condition d'influence liée à la description du circuit, alors qu'une pré-condition de fonctionnement est incontournable. En effet, il arrive que l'utilisateur modifie un contrôle en sachant qu'il va compléter la description du circuit. Le module d'aide signalera donc un conflit potentiel, qui ne sera pas retenu par l'utilisateur qui est conscient de son anticipation. Dans le cas contraire, le signalement permettra à l'utilisateur de repérer son erreur de manipulation, ou son étourderie.

Cette dynamique délègue un raisonnement au niveau de chacun des objets. Ainsi, une phase interroge l'ensemble des commandes potentielles pour identifier celles qui s'appliquent dans le contexte courant comme l'illustre la Figure 5-6. Les commandes interrogées retournent le résultat de leur méthode `Is_relevant`, qui indique si les conditions pour déclencher la commande sont, ou non, réunies. La deuxième règle réflexe, `Is_influent`, indique si la commande est pertinente, c'est-à-dire si elle est susceptible de répondre à un problème courant.

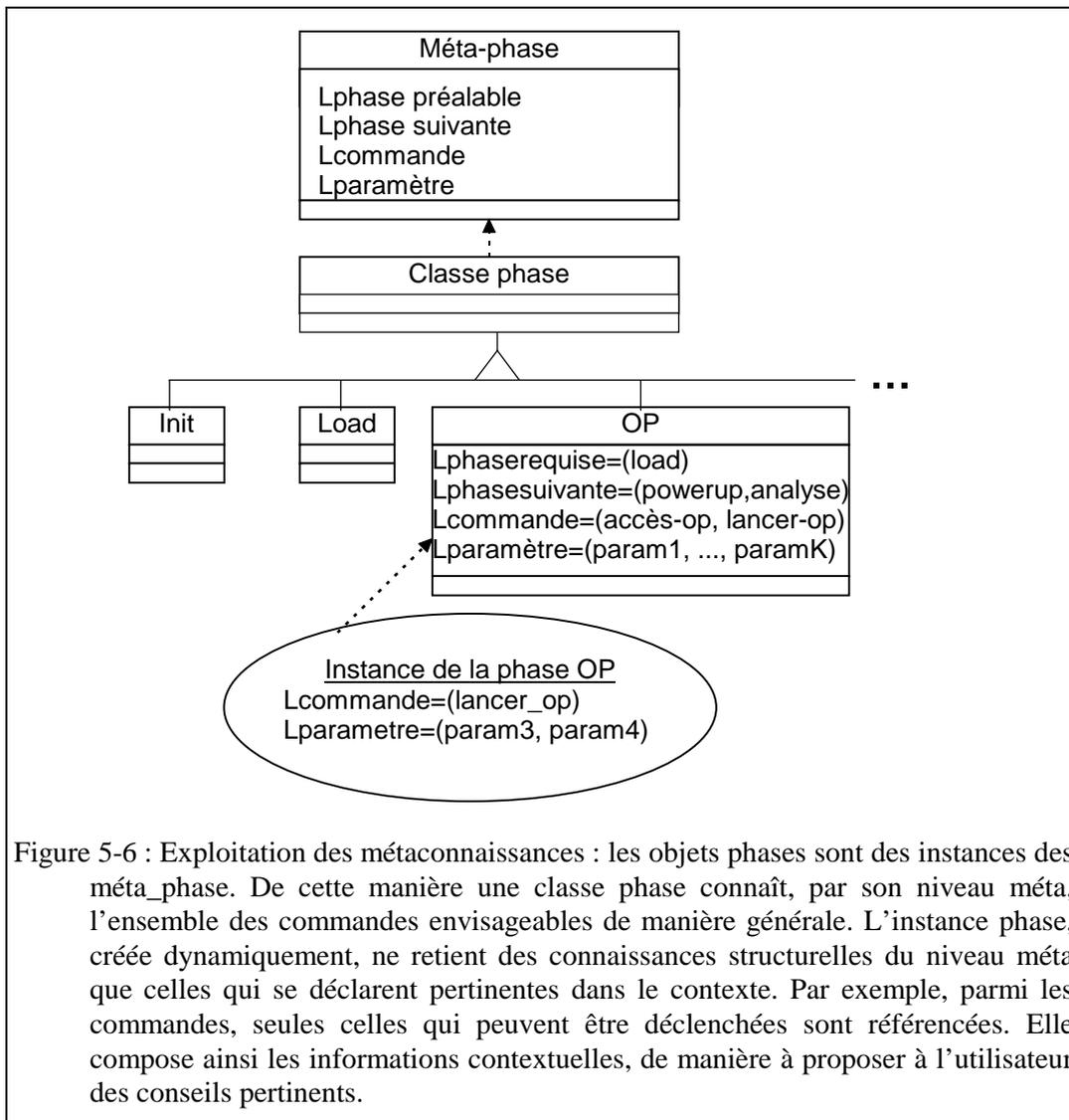


Figure 5-6 : Exploitation des métaconnaissances : les objets phases sont des instances des méta_phase. De cette manière une classe phase connaît, par son niveau méta, l'ensemble des commandes envisageables de manière générale. L'instance phase, créée dynamiquement, ne retient des connaissances structurelles du niveau méta que celles qui se déclarent pertinentes dans le contexte. Par exemple, parmi les commandes, seules celles qui peuvent être déclenchées sont référencées. Elle compose ainsi les informations contextuelles, de manière à proposer à l'utilisateur des conseils pertinents.

5.1.3.2. La formalisation des règles réflexes

Les règles sont une transcription des connaissances sur l'utilité et l'activité des éléments renseignés. La démarche retenue considère individuellement les composants du module d'aide et détermine pour chacun l'ensemble des sources d'influence. La règle d'un objet A représente la connaissance qui permet de sélectionner les objets qui exercent une influence sur cet objet, c'est-à-dire qui le contraignent ou qui le modifient.

Les règles s'expriment sous une forme procédurale « si A alors B ». Cependant, au niveau d'un objet, la connaissance structurelle n'est pas un ensemble de règles simples mises en concurrence pour obtenir une solution. Les règles expriment un compromis entre les diverses contraintes recensées sur l'objet. C'est donc le développeur du module d'aide qui garantit la cohérence d'une règle. Ce traitement des contraintes détermine exhaustivement les connaissances mises en jeu dans l'écriture d'une règle. Ceci permet de signaler quelle contrainte, exprimée par la règle, est mise en échec lors d'une évaluation. Ainsi, quand un paramètre signale qu'il ne devrait pas être utilisé, le résultat de l'évaluation est aussi une explication qui reflète une cause du conflit. Ce système n'expose à l'utilisateur qu'un seul motif de conflit, même si plusieurs contraintes sont violées (Figure 5-7). Néanmoins, les explications présentées lors d'un conflit permettent de prendre connaissance de l'ensemble des contraintes.

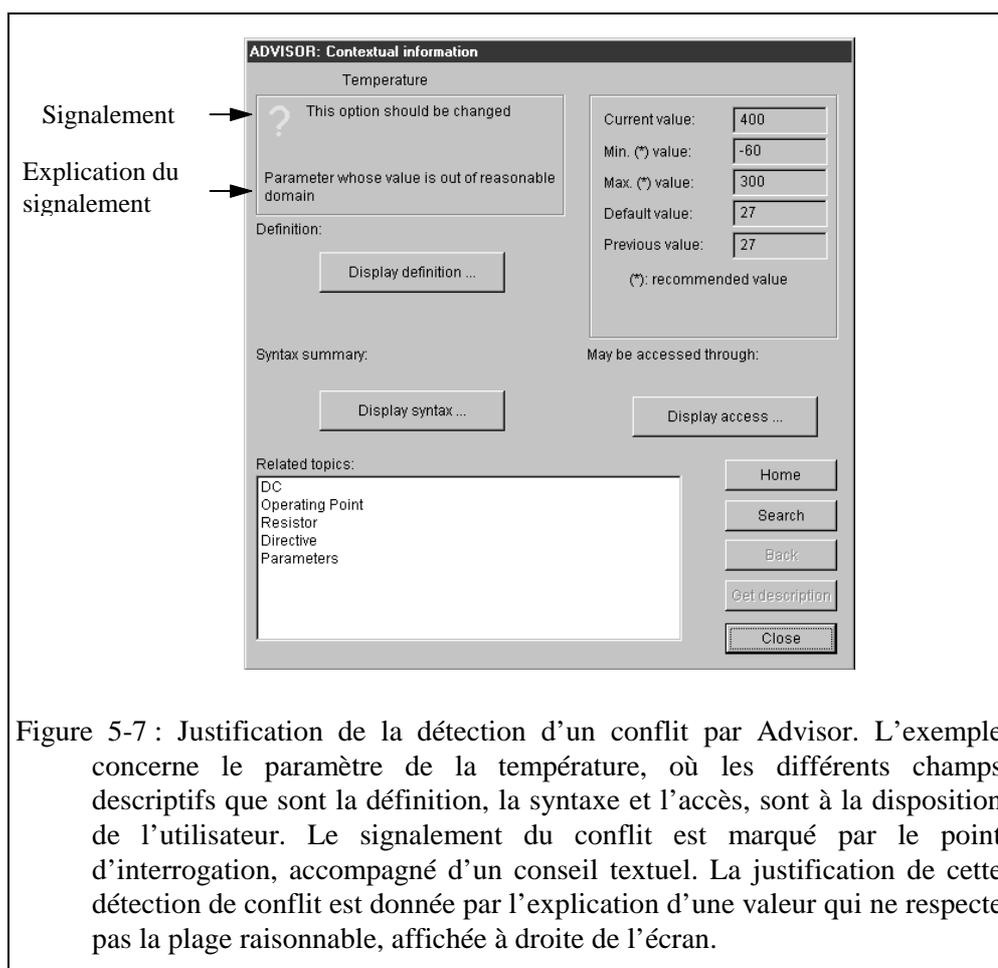


Figure 5-7 : Justification de la détection d'un conflit par Advisor. L'exemple concerne le paramètre de la température, où les différents champs descriptifs que sont la définition, la syntaxe et l'accès, sont à la disposition de l'utilisateur. Le signalement du conflit est marqué par le point d'interrogation, accompagné d'un conseil textuel. La justification de cette détection de conflit est donnée par l'explication d'une valeur qui ne respecte pas la plage raisonnable, affichée à droite de l'écran.

Aujourd'hui, l'utilisateur ne peut pas ajouter ou modifier une règle. L'ensemble des connaissances contextuelles sont introduites par le développeur du système d'aide, en

l'occurrence moi-même. Par la suite, il sera intéressant de pouvoir modifier les contraintes arbitraires aujourd'hui appliquées, comme les plages de valeurs raisonnables ou les valeurs par défaut. En dépit de son écriture non modifiable, la règle tient compte du contexte et le résultat de son évaluation est une information contextuelle et commentée.

5.1.3.3. L'intervention des règles réflexes

Les règles d'un objet sont systématiquement déclenchées lorsque ce dernier est sollicité, d'où le nom de règles réflexes. Dans le souci de renseigner l'utilisateur de manière compréhensible, seules ses actions directes sont soumises au contrôle du système d'aide. L'interactivité est favorisée pour signaler immédiatement un conflit sur l'élément modifié et bénéficier ainsi du contexte connu par l'utilisateur. Ce dernier se rappelle bien quel élément a été modifié et comment il a pu le modifier. Ceci est particulièrement important si le module d'aide suggère une correction de cette modification.

Les règles sont déclenchées aussi bien pour prévenir des erreurs, que pour signaler celles déjà commises. Quand le module d'aide est actif, les règles agissent en prévention. Quand l'utilisateur a inhibé les interventions du module d'aide, ce dernier poursuit l'observation pour mettre à jour le noyau sémantique et ainsi pouvoir être réactivé à tout moment. La prévention consiste à signaler un conflit sur une modification, avant qu'elle ne soit effective dans le noyau fonctionnel. De cette manière, l'utilisateur peut la corriger ou la confirmer.

Toutefois, les règles ne se déclenchent pas seulement lors de la modification de l'objet qu'elles renseignent. Hormis la règle `ls_redondant` qui n'est pertinente que sur une nouvelle action, un mauvais emploi n'est pas seulement dû à une modification. Un objet peut aussi devenir conflictuel lorsque le contexte change. Bien souvent, l'utilisateur calque, sur un nouveau processus de conception, les paramètres utilisés précédemment. Cette méthode de travail donne au mieux une première approximation. Quand les deux processus ne sont pas « assez » similaires, les anciens paramètres ne sont pas appropriés. Il n'est pas rare qu'une situation d'échec résulte de cette copie inadaptée et qu'il suffise d'adopter les valeurs par défaut des paramètres pour la résoudre.

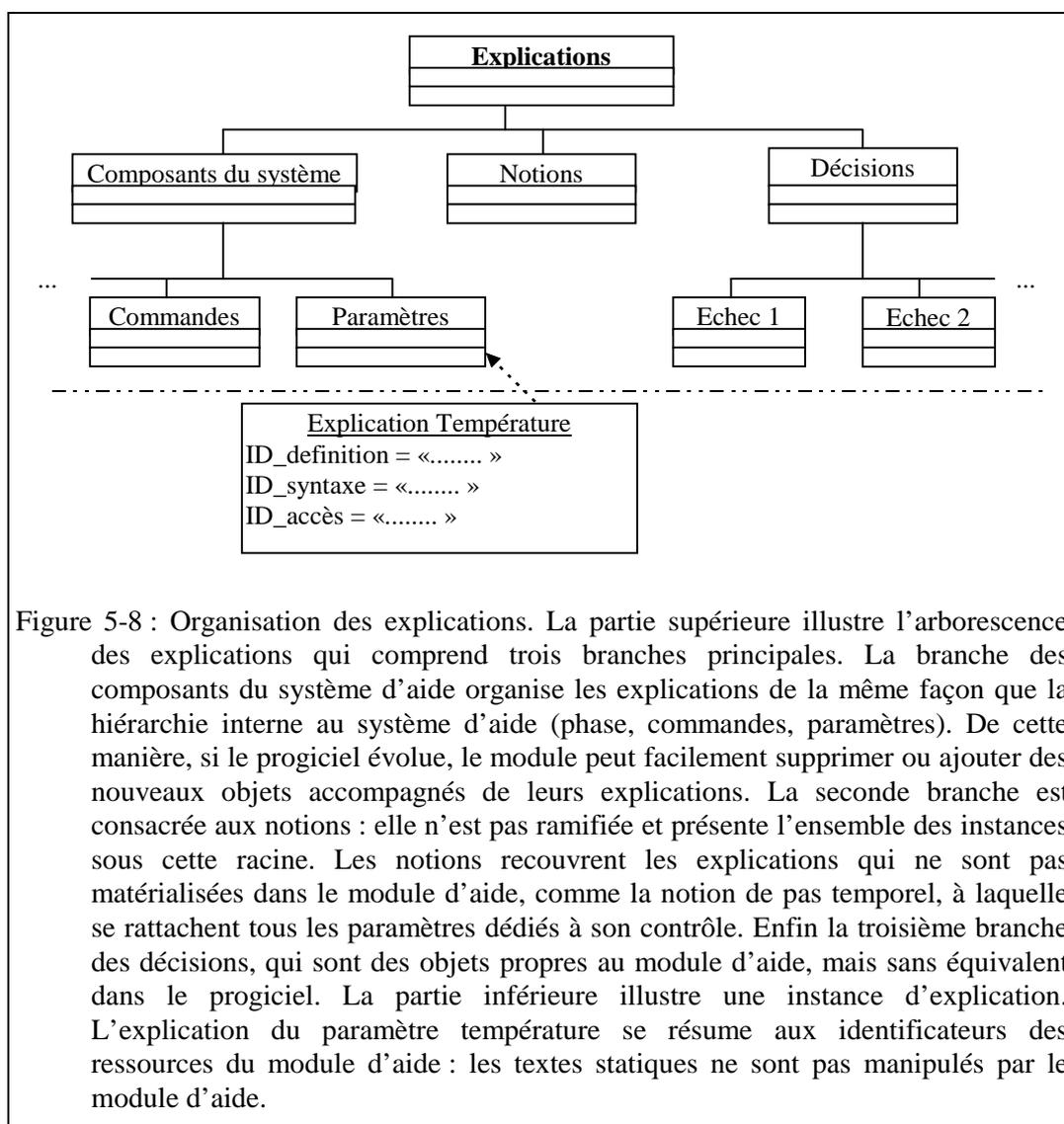
5.2. Les explications

Les explications ne sont pas dynamiquement construites dans le système d'aide : les textes explicatifs sont définis en dehors des objets du noyau sémantique. De cette façon le contenu des explications est facilement adapté aux besoins des utilisateurs. Néanmoins, le fait de ne pas construire les explications n'interdit pas de présenter les explications adaptées au contexte : ce sont les connaissances contextuelles des objets du noyau sémantique qui sélectionnent les explications en fonction du contexte pour chaque instance d'objet.

Au cours du développement itératif de notre application, la modélisation des connaissances structurelles s'est affinée. Tout d'abord, elle a distingué les explications des connaissances exploitées par le module d'aide pour le raisonnement. Les explications ne sont pas une nouvelle connaissance pour le module d'aide, mais une information destinée à l'utilisateur. En conséquence, la gestion des explications et celles des objets du noyau sémantique sont indépendantes : un objet ne vérifie pas le contenu de ses explications, ni ne pose d'hypothèse sur leur construction.

Les explications sont organisées autour de trois axes, comme le montre la Figure 5-8 :

- La description des objets du noyau sémantique, elle-même organisée dans une arborescence calquée sur la hiérarchie du noyau sémantique : à chaque objet modélisé correspond un objet descriptif qui est une explication. Cette simplicité facilite ultérieurement une modification des explications, car elles sont traitées en dehors des connaissances structurelles et contextuelles du noyau sémantique. Par ailleurs, les relations entre les explications reflètent les liens entre les composants du module d'aide.
- Les notions, qui décrivent des concepts qui ne sont pas matérialisés dans l'interface.
- Les conseils sont les explications des propositions du système d'aide pour résoudre une situation d'échec. Les objets associés dans le module d'aide sont discutés dans le chapitre 7 de ce mémoire.



5.2.1. Le descriptif des objets du noyau sémantique

L'utilisateur attend un descriptif contextuel pour un élément qu'il manipule dans l'interface. En conséquence, chaque instance correspondant à une action est construite de manière à disposer de l'objet explication correspondant à la classe d'action (Figure 5-9). C'est ensuite au niveau de l'instance que les explications sont sélectionnées en fonction du contexte.

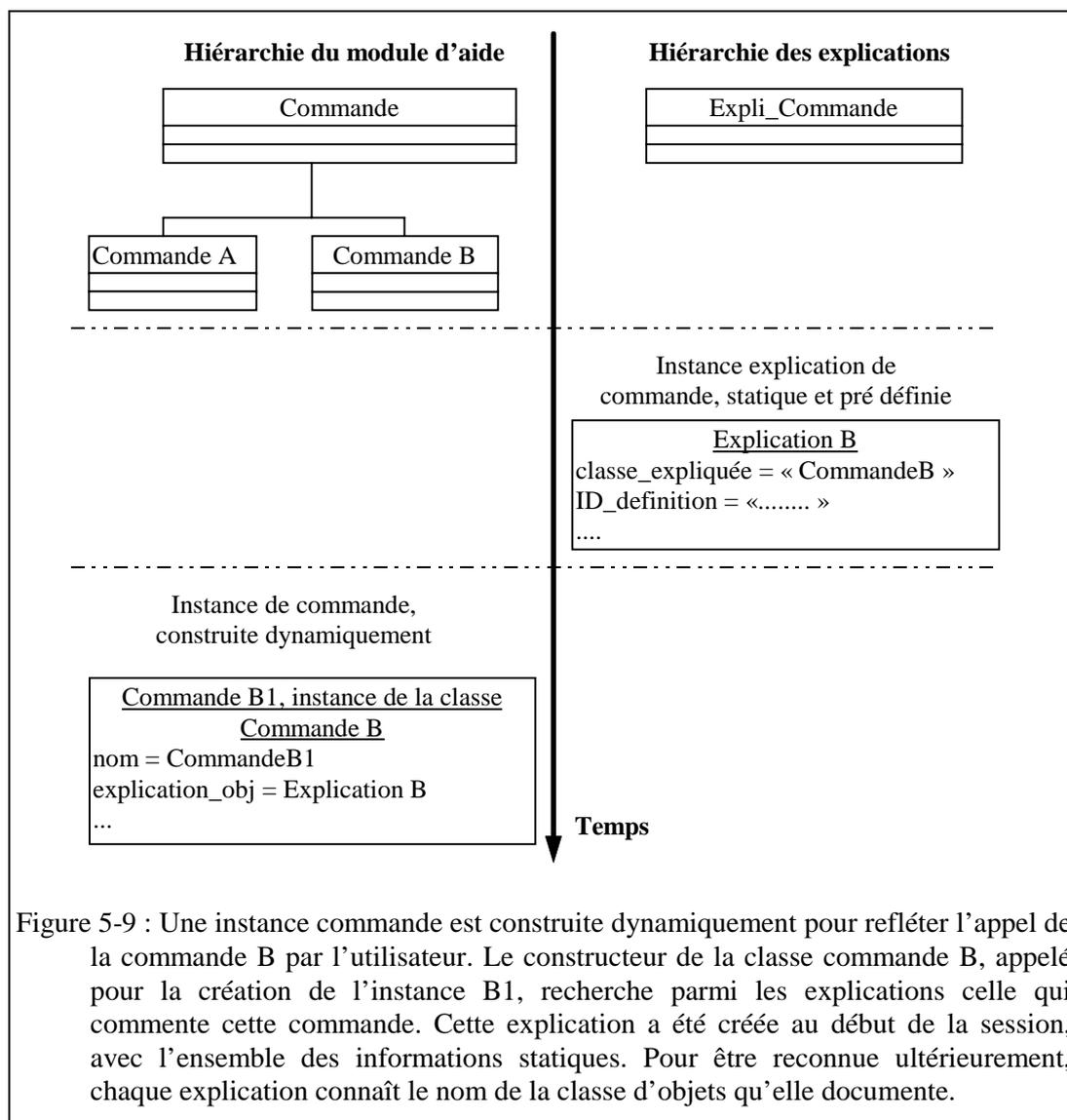


Figure 5-9 : Une instance commande est construite dynamiquement pour refléter l'appel de la commande B par l'utilisateur. Le constructeur de la classe commande B, appelé pour la création de l'instance B1, recherche parmi les explications celle qui commente cette commande. Cette explication a été créée au début de la session, avec l'ensemble des informations statiques. Pour être reconnue ultérieurement, chaque explication connaît le nom de la classe d'objets qu'elle documente.

Les explications sur un objet du noyau sémantique portent sur :

- sa définition ;
- sa syntaxe, qui renseigne aussi l'unité de mesure physique d'un paramètre (volt, seconde...) ;
- la description de son accès dans l'interface, c'est-à-dire la boîte de dialogue ou le menu où est défini le contrôle ;

- les indications de valeurs numériques pour un paramètre, sur sa valeur par défaut, sa plage de valeur raisonnable, ou sa valeur précédente ;
- une liste de contrôles connexes, c'est-à-dire de paramètres et de commandes liés ;
- une liste de notions, qui permet de situer le contrôle par rapport aux concepts mis en jeu dans le progiciel.

L'exemple suivant (Figure 5-10) montre les explications proposées sur le paramètre température. En exploitant le statut d'expert de l'utilisateur, une explication concise lui est dispensée, en lui laissant la charge de se renseigner sur les sujets connexes. La température est un paramètre global du simulateur et elle influence les primitives comme les résistances ou les capacités, dont la valeur peut être dépendante de la température. Les études du circuit en fonction d'une variation de la température sont un autre sujet connexe, que l'utilisateur peut explorer. C'est donc l'utilisateur qui décide de vérifier en quoi un composant comme une résistance est influencé par la température. Il pourra ainsi apprendre comment il faut définir une résistance, pour qu'elle tienne compte des variations de température.

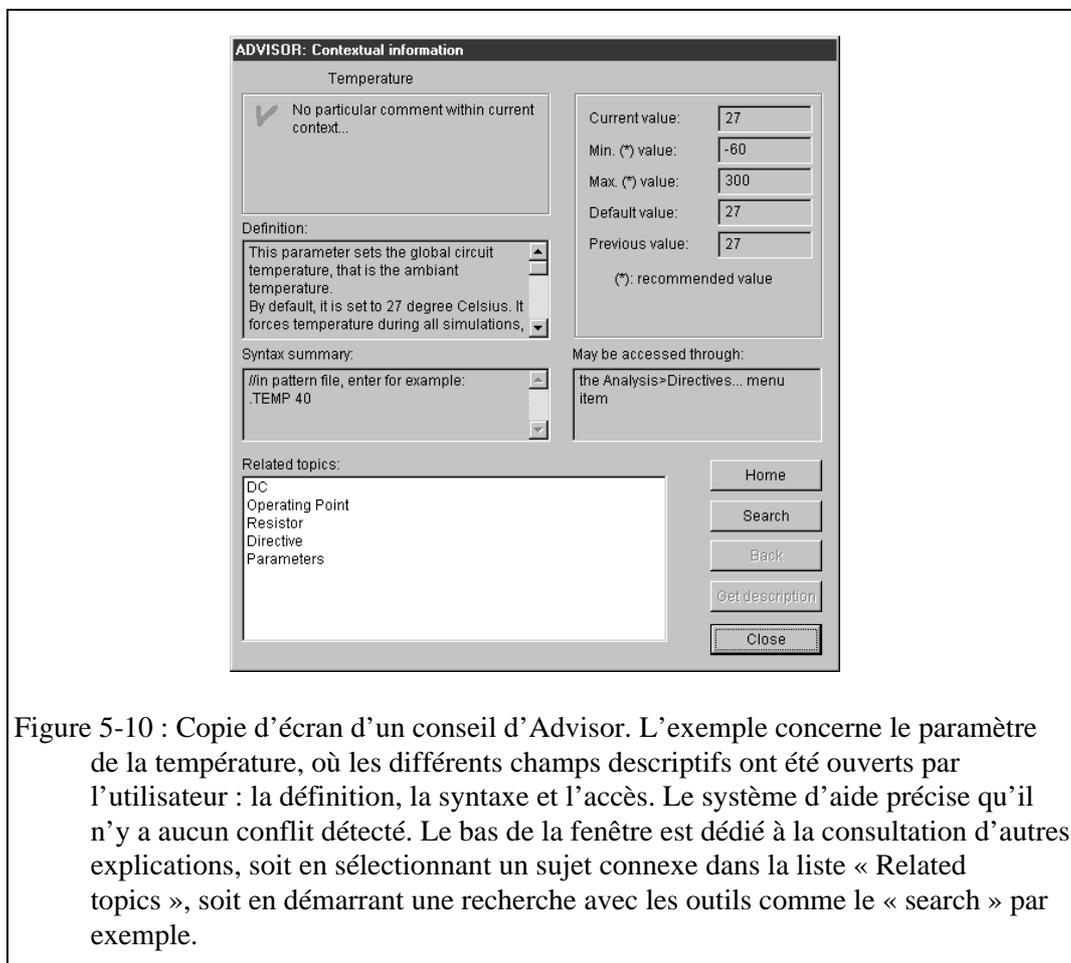


Figure 5-10 : Copie d'écran d'un conseil d'Advisor. L'exemple concerne le paramètre de la température, où les différents champs descriptifs ont été ouverts par l'utilisateur : la définition, la syntaxe et l'accès. Le système d'aide précise qu'il n'y a aucun conflit détecté. Le bas de la fenêtre est dédié à la consultation d'autres explications, soit en sélectionnant un sujet connexe dans la liste « Related topics », soit en démarrant une recherche avec les outils comme le « search » par exemple.

Les notions et les contrôles connexes sont déduits de l'étude des liens exprimés par le niveau méta des objets : les explications sont leur expression textuelle destinées à l'utilisateur. De manière générale, toutes les connaissances sur le fonctionnement du progiciel formalisées pour le module d'aide sont aussi utiles à l'utilisateur et doivent être transcrites textuellement à partir de la modélisation. Les liens apportés par ces objets explications connexes sont essen-

tiels à l'appropriation du progiciel, car à partir d'un contrôle de l'interface, l'utilisateur peut découvrir de nouvelles possibilités du progiciel en explorant l'arborescence des explications. C'est ainsi que pour une phase, ses explications renvoient aux explications sur les commandes qu'elle manipule, aux explications des paramètres qui l'influencent. La phase donne un point de vue plus général sur l'analyse, que chacune des commandes ne pouvaient pas apporter.

5.2.2. Les notions

Les notions sont des explications générales, ajoutées à celles du noyau sémantique pour renseigner l'utilisateur sur les concepts implicitement mis en œuvre dans le progiciel. Elles permettent d'explicitier à l'utilisateur des liens entre des objets du noyau sémantique. Par exemple, le pas d'itération d'une analyse temporelle est le dénominateur commun des paramètres qui le contrôlent, alors qu'il n'est pas matérialisé dans l'interface.

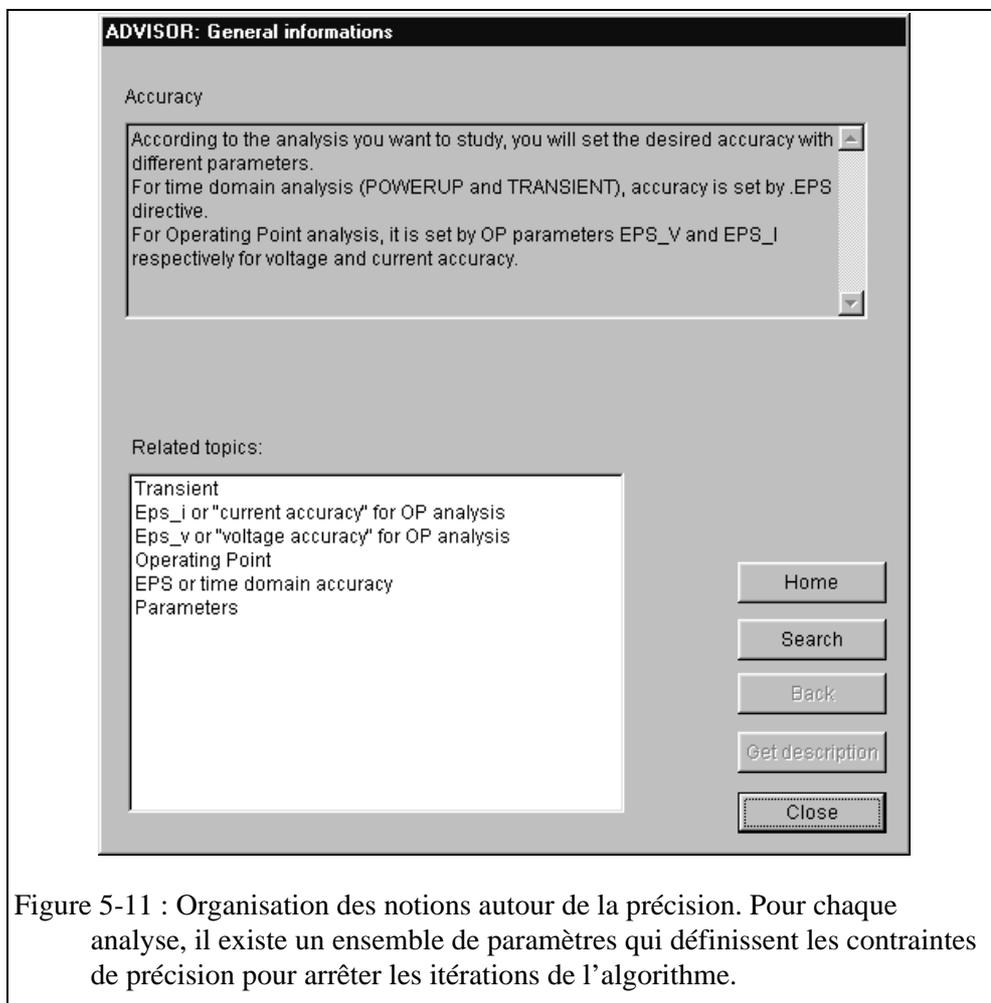


Figure 5-11 : Organisation des notions autour de la précision. Pour chaque analyse, il existe un ensemble de paramètres qui définissent les contraintes de précision pour arrêter les itérations de l'algorithme.

Lors de l'évaluation de la maquette, les utilisateurs ont exprimé le besoin de cerner ces concepts. La raison invoquée était de pouvoir vérifier qu'ils comprenaient correctement la structure implicite des explications. L'introduction des notions a permis d'organiser les explications par rapport aux phases de travail et par rapport à leur maîtrise, qui de manière géné-

rale est un compromis entre la vitesse et la précision de l'analyse. Ces explications sont toujours destinées à l'utilisation du progiciel.

Les notions ont aussi permis d'élargir le vocabulaire choisi pour le noyau sémantique en donnant explicitement les termes équivalents. Par exemple un composant inductif peut être indifféremment nommé « inductor » ou « self » en anglais. Exclure un terme est totalement arbitraire et risque de laisser perplexe l'utilisateur néophyte qui ne pensera pas toujours à rechercher un synonyme. Les notions permettent aussi une ouverture vers des concepteurs d'un domaine spécifique en corrélant leur vocabulaire avec celui du progiciel, en attendant de pouvoir totalement personnaliser l'interface avec l'utilisateur. Ce dernier pas soulèvera le problème de la construction des explications dédiées à un utilisateur, car le vocabulaire propre au progiciel ne sera plus un discours partagé.

Les notions essentielles sont déjà présentes dans le prototype. Elles expliquent en particulier le concept d'analyse vis-à-vis des diverses commandes. Ainsi, l'analyse en transitoire et la mise sous tension d'un circuit sont toutes deux des analyses temporelles, et l'utilisateur peut observer qu'il existe trois types d'analyses fondamentales et plusieurs fonctionnalités de traitement du signal (calcul du signal sur bruit, taux de distorsion, analyse de Fourier...). Cependant, les évaluations du prototype auprès des étudiants révèlent qu'ils endossent lentement le statut d'expert. Les notions sont aussi le support de la compréhension d'un vocabulaire de base utilisé dans le progiciel, et elles doivent s'attacher à tous les termes employés.

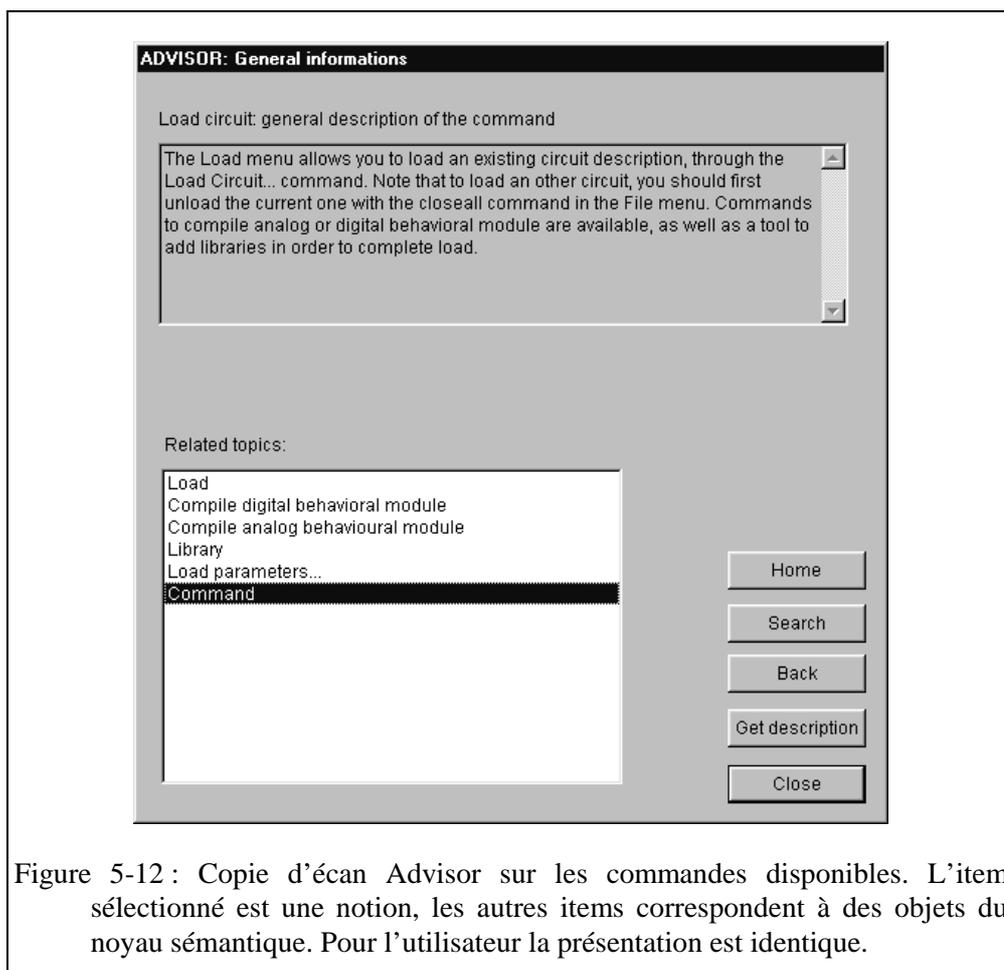


Figure 5-12 : Copie d'écran Advisor sur les commandes disponibles. L'item sélectionné est une notion, les autres items correspondent à des objets du noyau sémantique. Pour l'utilisateur la présentation est identique.

5.2.3. Les conseils

Contrairement aux contrôles sur le progiciel, les conseils n'apparaissent pas dans l'interface existante entre l'utilisateur et le progiciel. Les conseils sont des objets propres au module d'aide, décrits dans le chapitre 7, qui modélisent la connaissance pour résoudre une situation d'échec. Nous nous limitons dans cette section aux explications associées à ces objets, indépendamment de leur structure propre : cette connaissance textuelle devra être exploitée mais nous ne nous préoccupons pas ici du comment. Les explications utiles pour un conseil sont :

- la situation d'échec à laquelle elle répond,
- ses conditions d'application, autre que la situation d'échec,
- sa description générale pour indiquer en quoi il consiste,
- ses avantages, c'est-à-dire quelles difficultés il permet de traiter,
- ses inconvénients,
- comment la mettre en œuvre en indiquant quels contrôles doivent être modifiés et comment ils doivent l'être.

En conséquence, l'objet associé au conseil comprend des attributs pour mémoriser les identificateurs des ressources textuelles et une liste d'explications connexes qui renvoient aux contrôles utiles pour cette solution potentielle.

En résumé, ce chapitre est consacré à l'organisation des connaissances sur le progiciel, qui ont pu être formalisées avec les experts. Nous avons distingué deux types de connaissances :

- celles qui participent à la définition du contexte dans le module d'aide,
- celles qui concernent les explications textuelles qui apportent essentiellement une description générale.

Le chapitre suivant est consacré au fonctionnement du module d'aide, qui doit maintenant exploiter ces connaissances pour dispenser une information pertinente à l'utilisateur.

Chapitre 6

6. Le fonctionnement du système d'aide

Ce chapitre est dédié au fonctionnement du système d'aide, c'est-à-dire à la façon d'exploiter la connaissance sur l'utilisation du progiciel formalisée dans le chapitre précédent, en tenant compte du contexte défini dans le chapitre 4. Le problème est donc de répercuter dans le système d'aide l'évolution du contexte consécutive aux actions de l'utilisateur, pour ensuite définir à partir des connaissances l'information contextuelle utile.

Le fonctionnement du système d'aide repose sur le traitement cyclique des actions de l'utilisateur qui modifient le contexte de travail. Pour chaque action le cycle se décompose en :

- l'observation de l'action, qui consiste à l'intercepter lorsque l'utilisateur l'applique dans le progiciel;
- l'interprétation de l'action dans le schéma des phases de travail ;
- la détermination des conseils utiles dans le nouveau contexte.

Ce cycle de traitement permet de répercuter les actions de l'utilisateur dans le module d'aide. Ce fonctionnement est permanent pour garantir une cohérence entre les actions et leur suivi dans le module d'aide.

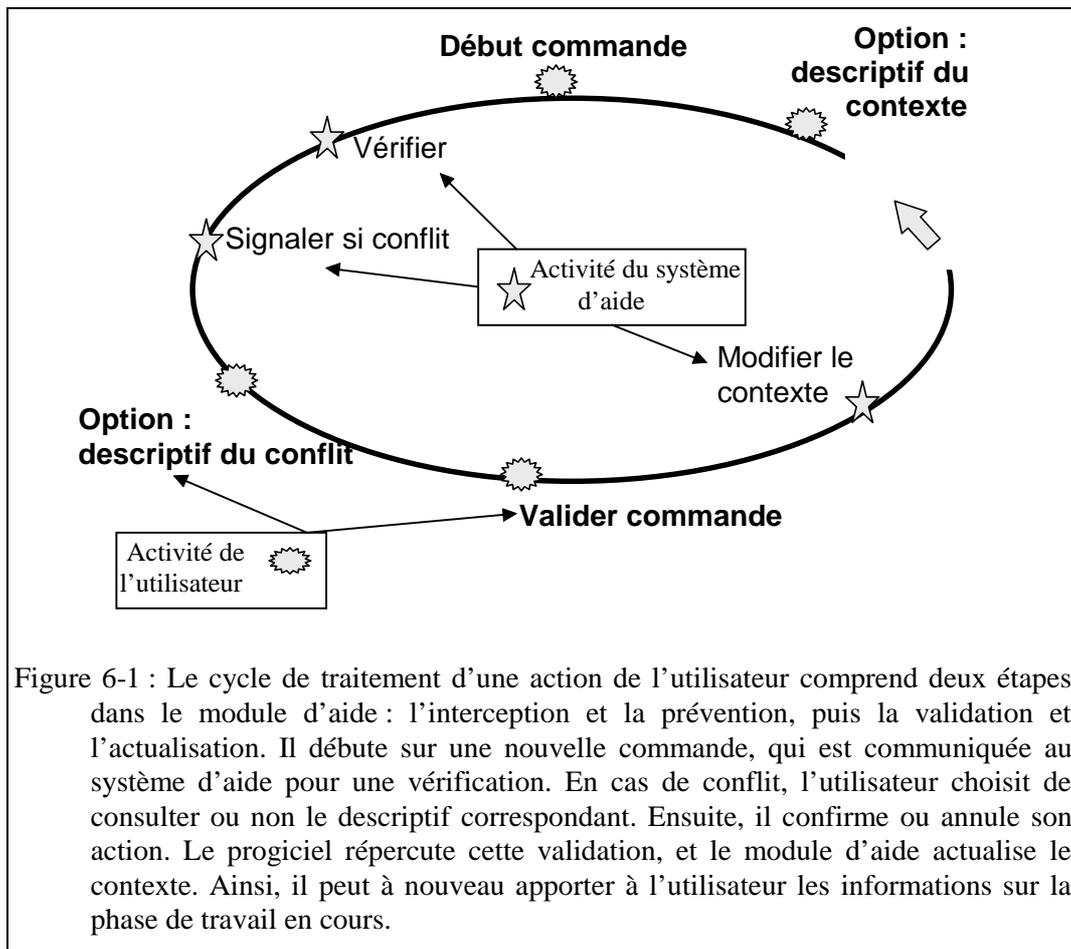
6.1. Répercuter les actions dans le module d'aide

L'observation des actions de l'utilisateur est nécessaire pour construire leur suivi. Elle est réalisée dans l'interface entre le progiciel et l'utilisateur, en ajoutant des messages à destination du module d'aide dans la chaîne des événements qui traite une action. Ce traitement spécifique est ajouté soit dans l'API du progiciel si elle existe, soit dans l'interface fonctionnelle qui gère le « call-back » de chaque contrôle, c'est-à-dire l'appel au traitement nécessaire pour répercuter son activation dans le noyau fonctionnel. Dans notre application, le module d'aide est informé par la fonction `Set_new_command(ld_command)`, qui est ajoutée dans la gestion du « call-back » de chaque commande avec l'identificateur adéquat.

Dans la mesure où le module d'aide est aussi prévu pour dispenser une aide préventive, l'interception des actions de l'utilisateur doit se faire au moins à deux moments :

- au début de l'action, c'est-à-dire quand l'utilisateur accède à un paramètre, ou lance une analyse : le module d'aide reçoit le message d'un début de cycle avec l'identificateur de l'action ;
- à la fin du traitement par le progiciel, pour que le noyau sémantique s'actualise avec les données du noyau fonctionnel. C'est en particulier à la fin du cycle que le résultat d'une commande ou la valeur donnée à un paramètre sont connus. L'action peut par ailleurs être annulée, ce qui devra aussi être répercuté dans le noyau sémantique à la fin de son traitement.

Le cycle de traitement d'une action présente donc plusieurs étapes aussi bien dans le progiciel que dans le module d'aide : elles sont visualisées sur la Figure 6-1.



La nature des échanges entre le progiciel et le module d'aide est limitée à des identificateurs clés, qui suffisent à déterminer l'évolution du contexte. L'interface avec le progiciel est le vecteur pour actualiser le contexte dans le module d'aide et seuls les résultats objectifs sont transmis par le progiciel, avec la variable `ID_RESULT`. Ainsi, parmi les situations d'insatisfaction, seule une situation d'échec qui peut être reconnue par le progiciel, sera transmise comme un échec, avec l'identificateur qui spécifie sa nature. Par contre les jugements subjectifs d'une analyse trop lente ou pas assez précise ne sont pas gérés, car ils nécessitent l'intervention de l'utilisateur.

Le résultat d'une commande dans notre application peut être :

- un échec et sa nature dans la mesure où le progiciel peut identifier différents types d'échec ;
- une annulation ;
- une interruption pour arrêter volontairement une analyse, trop lente par exemple ;
- et la situation où aucun problème n'est détecté par le progiciel.

Le cycle de traitement d'une commande indique les instants favorables pour actualiser le module d'aide, pour intervenir en prévention, pour effectuer le suivi de l'utilisateur ou encore pour répondre à un problème rencontré.

6.1.1. L'observation des actions

La communication des identificateurs des actions et de leurs résultats ne constitue pas une observation des actions en soi. L'observation consiste à donner une signification à ces informations en construisant les objets correspondants dans le noyau sémantique, et en les modifiant en fonction des nouvelles informations transmises par le progiciel. En conséquence, dans le module d'aide, chaque action interceptée est concrétisée par une instance de la classe d'action correspondante. Cette association entre une action et une instance donne la possibilité d'accéder aux connaissances structurelles et contextuelles de la classe correspondante, grâce au noyau sémantique.

De cette manière, à condition de connaître les informations sur le contexte au moment où l'action est appliquée, l'objet remplit toutes les caractéristiques nécessaires pour être réactif. Il exploite en effet :

- ses possibilités et ses contraintes dans le cas général, définies dans le noyau sémantique, c'est-à-dire dans la définition de la classe correspondante,
- le contexte, qui est mémorisé par l'objet session,
- ses connaissances contextuelles, explicitées par les méthodes de sa classe, pour déduire, à partir des connaissances générales, les informations pertinentes dans le contexte courant.

Pour illustrer ce fonctionnement, nous prenons l'exemple d'une commande, notée C. Comme indiqué précédemment, le module d'aide reçoit le message de début de cycle, avec l'identificateur de la commande correspondante notée ID_C. C'est en fonction de cet identificateur que la classe d'objets du noyau sémantique est retrouvée pour ensuite invoquer son constructeur : une instance est créée, qui représente la commande en cours dès le début du cycle de traitement. Elle sera en particulier affectée par le résultat de la commande, transmis par le progiciel au système d'aide.

A ce moment du cycle, la commande n'est pas encore validée dans le progiciel, mais elle est déjà concrétisée dans le module d'aide. Cet objet peut alors réagir par rapport au contexte qu'il n'a pas encore modifié. De cette manière, il est possible d'évaluer si les pré-conditions sont ou non respectées.

Le deuxième temps du traitement d'une action correspond à la prise en compte de son résultat, pour achever sa répercussion dans le module d'aide. L'instance correspondante est mise à jour pour mémoriser ce résultat, mais l'essentiel du traitement est de définir quelle est dans le module d'aide la phase adaptée pour donner à l'utilisateur les meilleurs conseils. Le suivi

de l'activité a en effet pour but d'actualiser le module d'aide de manière à répondre aux nouvelles données, et ainsi anticiper les problèmes. L'organisation du suivi repose donc sur le schéma des phases et son contrôle par l'objet `Session`, qui vont tout deux participer à la définition du nouveau contexte.

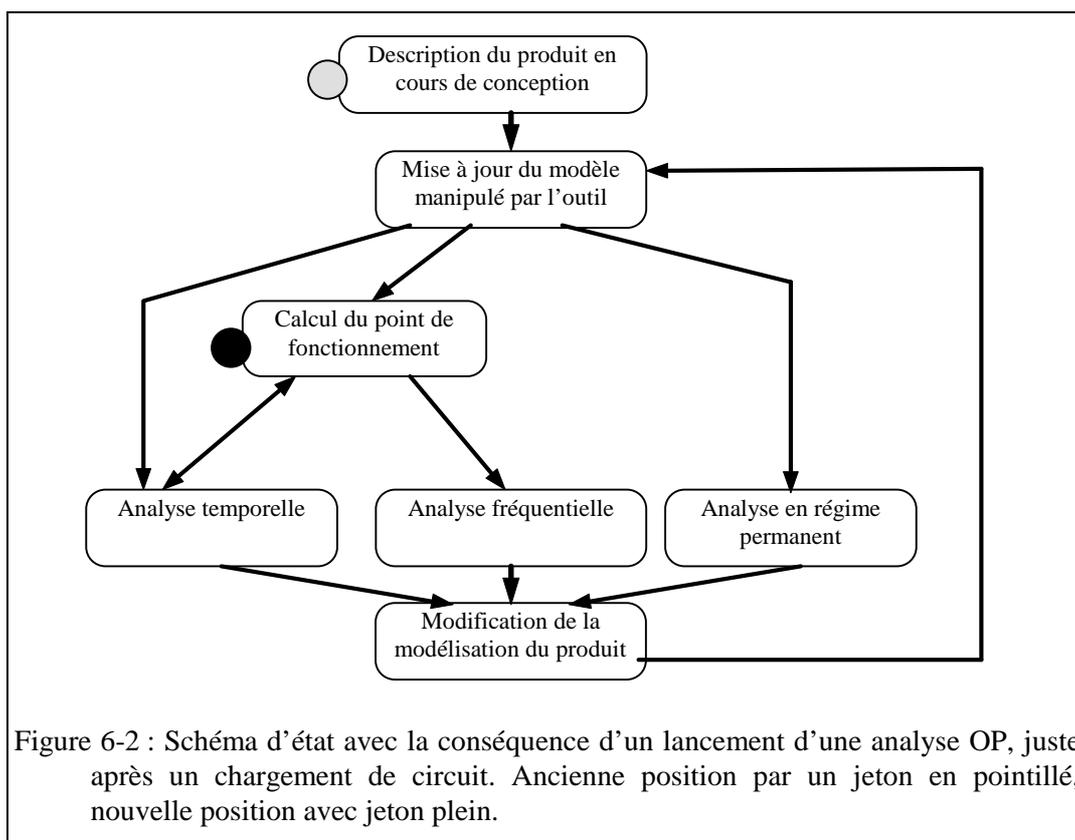
La dernière caractéristique pour que l'objet soit « réactif » est qu'il définisse quelle est justement la phase la plus adaptée, qui apporte le plus d'informations sur la nouvelle situation obtenue en fin de cycle de traitement. Il s'agit donc de définir dans le module d'aide le nouveau contexte et en particulier la nouvelle position dans le schéma de phases. Ainsi, quand une commande a échoué, décider que la phase de travail correspond maintenant à la phase qui renseigne précisément cette commande est un choix qui permet de donner à l'utilisateur des informations sur ce problème.

Cette connaissance est contextuelle, et elle définit de manière exhaustive les situations qui conduisent à modifier la position dans le schéma de phases pour s'adapter à la nouvelle situation. Elle est concrétisée par la méthode `MakeEvolution` de chaque classe du noyau sémantique. Ce raisonnement local permet de corriger ou d'affiner cette méthode au fur et mesure que les connaissances sur le fonctionnement du progiciel sont enrichies.

Nous avons vu que l'observation conduit aux données utiles au suivi de l'activité, en exploitant le raisonnement local aux objets, d'une part pour évaluer leurs possibilités dans le contexte courant, d'autre part pour actualiser le contexte du module d'aide.

6.1.2. Organisation du suivi pour une mise à jour du contexte

Le traitement d'une action par le module d'aide consiste à définir quelle est la nouvelle phase de travail. C'est ensuite en exploitant les connaissances du schéma de phases que le module d'aide peut conseiller l'utilisateur. En effet, c'est la structure de données de l'objet `phase` qui explicite, pour la phase de travail, les commandes potentielles, candidates et nécessaires suivant le contexte. De manière symbolique, le rôle du module d'aide dans le suivi est de déplacer dans le schéma de phases un jeton, qui représente l'analyse courante appliquée au circuit étudié (Figure 6-2).



Le suivi de l'activité est construit sur trois types d'objets : les commandes, regroupées dans des phases de travail, elles-mêmes organisées comme un enchaînement d'analyses au cours d'une session. Nous rappelons ici que la session correspond à une étude d'un circuit particulier. Elle ne coïncide pas obligatoirement avec l'ouverture et la fermeture du progiciel, mais avec le chargement et le déchargement de la description d'un circuit.

L'organisation du suivi est construite sur :

- une description globale de l'activité dans le schéma de phases : à une phase de travail correspond un objet phase,
- une description détaillée de l'activité dans cette phase, avec l'ensemble des actions effectuées.

Le rôle de l'objet Session est de centraliser les informations sur l'enchaînement des phases, sachant qu'à partir de ces dernières les commandes sont accessibles. Par ailleurs, comme le montre la Figure 6-3, l'objet Session recueille les informations du progiciel ainsi que celles sur le contexte de travail : la description du circuit, l'historique des phases, la phase et la commande en cours composent le contexte et sont donc pris en compte dans la structure de données de cet objet. A tout instant, il existe au plus une instance de la classe Session, car il est impossible de charger simultanément plus d'un circuit.

Session	
nom :	chaîne de caractères
historique_liste :	liste des instances de la classe
circuit :	instance de l'objet circuit chargé par le progiciel
param_liste :	liste des paramètres instanciés
commande_courante :	commande en cours
phase_courante :	phase d'appel de la commande
phase_en_cours :	phase précédente qui n'a pas été conclue
phase_historique :	liste des instances des phases précédentes.
explication_liste :	liste des instances explications prédéfinies dans le module d'aide

Figure 6-3 : L'objet Session centralise les données sur l'état courant. Il référence des objets concrets de l'outil, comme le circuit ou les paramètres qui doivent être initialisés dès le début d'une session. Il référence aussi la commande en cours pendant son cycle de traitement. Les objets plus abstraits, comme les phases d'étude du schéma de phases et les conseils formalisés dans le module d'aide, sont aussi contrôlés par l'objet Session.

L'objet Session est le point d'échange entre le progiciel et le module d'aide. Quand une nouvelle commande est appliquée, c'est cet objet pivot qui déclenche la création de l'instance associée et il la mémorise comme la commande courante. En fin de cycle, la session reçoit le résultat de cette commande, transmis sous la forme d'un identificateur et actualise l'objet associé dans le module d'aide. L'objet Session assure donc un rôle de coordination entre le fonctionnement du progiciel et celui du module d'aide, de manière à ne pas perturber le premier.

La difficulté du suivi est donc d'identifier, à partir des commandes communiquées au module d'aide, les phases correspondantes et en particulier les transitions entre elles. En effet, toutes les actions ne conduisent pas à une nouvelle phase, et une même action ne conduit pas systématiquement à une nouvelle phase. Cette constatation corrobore l'intérêt d'un raisonnement local aux objets associés aux actions, pour définir au cas par cas l'actualisation du module d'aide, qui dépend aussi du contexte. C'est donc la méthode intitulée *MakeEvolution* qui pour chaque objet actualise le module d'aide, en créant si nécessaire une nouvelle phase.

En tout premier lieu, il faut définir quels sont les critères pour décider du début et de la fin d'une phase de travail. Dans notre application, la fin de phase n'est pas explicitée par l'utilisateur : après une analyse, aucun indice ne permet de déduire qu'elle va être relancée avec certaines modifications, ou que l'utilisateur souhaite en mener une autre. C'est donc en fonction de l'action suivante qu'il est possible de déterminer si la phase actuelle est toujours capable de donner des informations à l'utilisateur : ce sera le critère pour créer une nouvelle phase dans le module d'aide. Néanmoins, quand la phase précédente n'est pas satisfaite, c'est-à-dire que l'analyse correspondante a conduit à une situation d'échec, elle reste une tâche de fond mémorisée par l'objet Session.

Cette notion de phase en attente permet de gérer la situation d'une phase mise en échec par une étape précédente. En illustrant la situation sur le schéma de phases, considérons la phase « analyse transitoire ». Après une analyse satisfaisante, l'utilisateur peut modifier son circuit

pour poursuivre ensuite cette même analyse. Pour cela, le progiciel charge d'abord le circuit modifié. Deux situations peuvent se présenter :

- Soit le chargement est un succès, et l'analyse transitoire est à nouveau lancée. Dans ce cas si des problèmes interviennent, ils concernent la phase « analyse transitoire » et il est inutile de compliquer le suivi des actions en ajoutant la phase implicite de chargement. En effet, le chargement est une phase préalable explicite dans le schéma de phases. Si le chargement est satisfait, il est donc simplement mémorisé comme une action de la phase en cours.
- Soit le chargement aboutit à un échec. Dans ce cas, la phase « analyse transitoire » n'est pas terminée, mais elle n'est pas non plus adaptée pour résoudre ce problème. C'est le rôle de la phase « chargement » de centraliser les connaissances pour aider à la résolution de ce problème. Tant que le chargement n'est pas réussi, c'est donc la phase de chargement qui doit être en cours, même si la phase « analyse transitoire » reste la phase à poursuivre. Ceci permet en particulier de réorienter l'utilisateur vers cette phase en attente, dès que le chargement est réussi.

En résumé, le suivi consiste d'abord à déduire la phase en cours, pour ensuite évaluer les actions potentielles, les actions candidates et les actions nécessaires. Le module d'aide détermine ainsi les actions possibles, mais sans en privilégier une particulière en fonction de la précédente : il ne s'appuie pas sur des enchaînements de commandes, mais sur une logique de succession de phases de travail. Le suivi privilégie les phases inachevées, mais il ne contraint pas l'utilisateur à poursuivre une analyse jusqu'à sa réussite.

6.1.3. L'historique mémorise les actions

La création dynamique d'instances permet de garder une trace des actions, et mémorise aussi les informations sur le contexte du moment. En effet, au cours d'une session, une même action peut être appliquée plusieurs fois, sans que les problèmes de son utilisation soient identiques. Dans le module d'aide, cette réalité est concrétisée par autant d'instances de la même classe, qui ont chacune leurs propres informations sur le contexte. Ces informations sont par exemple une pré-condition non respectée, ou une situation d'échec qui résulte d'une analyse lancée par la commande correspondante.

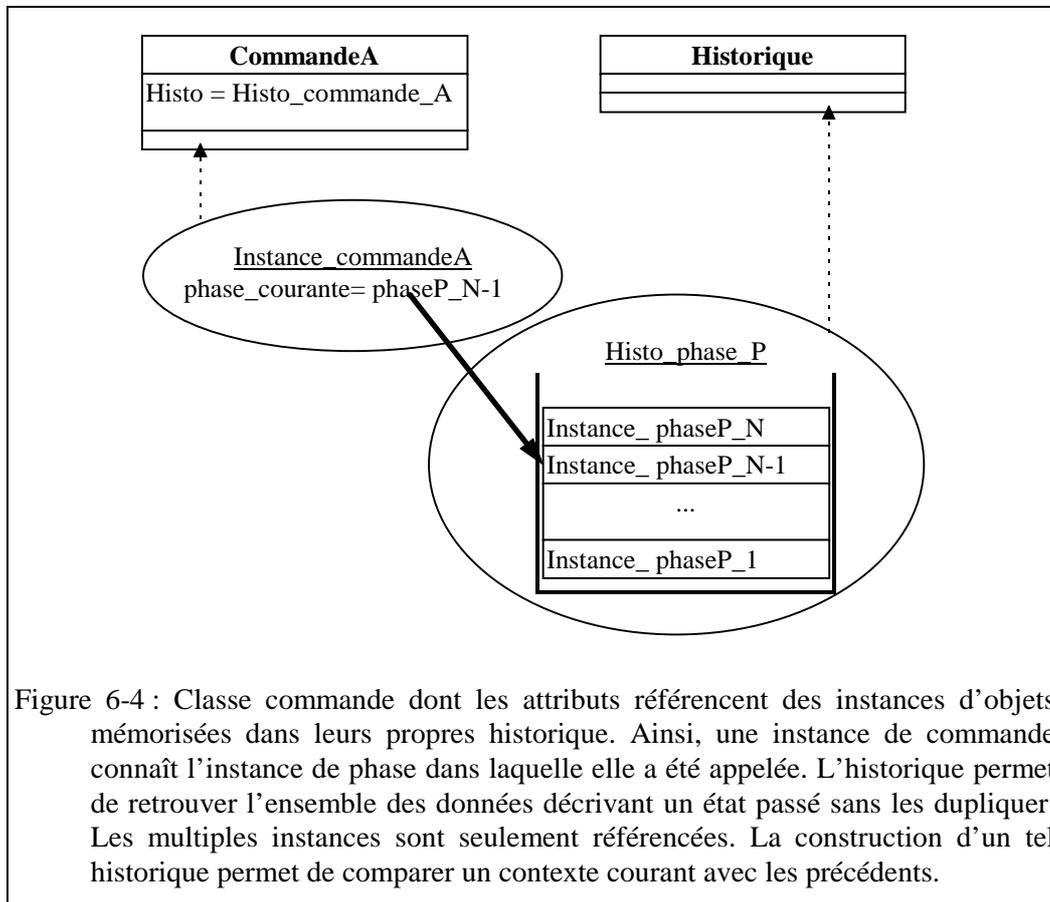
L'historique est aujourd'hui exploité dans deux situations typiques :

- Besoin d'informations sur une action appliquée précédemment. Par exemple il peut être intéressant de connaître la valeur précédente d'un paramètre pour évaluer en quoi son évolution pose problème : l'historique est une information sur un contexte passé.
- Besoin de connaître quelles sont les actions précédemment appliquées dans une phase donnée. Par exemple la comparaison de la précédente action avec la nouvelle donne des informations pour identifier les actions redondantes. Mais surtout, connaître les précédentes actions et leurs résultats permet, dans une situation d'échec, de ne pas proposer un conseil qui correspond à une action déjà tentée par l'utilisateur.

L'historique est donc un outil pour trouver les informations contextuelles sur l'utilisation précédente du progiciel. Pour répondre aux deux besoins exprimés ci-dessus, il existe deux organisations de l'historique :

- la première est attachée à chaque classe d'objet, qui liste ses instances au fur et à mesure de leur création, en préservant l'ordre de leur application,
-

- la seconde correspond aux liens entre les actions et leur position dans le schéma de phases lors de leurs applications. C'est-à-dire que chaque objet phase a son historique des actions effectuées par l'utilisateur, et que l'objet Session a son historique de phases.



Dans la mesure où l'historique est construit sur les instances correspondant aux actions, toutes les informations contextuelles prévues dans la classe de ces objets sont accessibles à tout moment : ces instances ne sont détruites qu'à la fermeture d'une session de travail. Ces informations sont par exemple le résultat de l'action, codé par un identificateur. L'historique est utile dans la mesure où chaque nouvel objet y a accès. Dans notre application, il est choisi de toujours pouvoir « remonter » l'historique vers des actions plus anciennes, en partant de l'objet Session qui centralise les informations contextuelles.

La construction de l'historique se fait en deux temps au cours du cycle de traitement d'une action. D'abord lors de la création de l'instance associée à l'action, c'est-à-dire en début de cycle, la classe de cet objet actualise son historique : aussitôt créée, elle est systématiquement ajoutée à la liste représentant l'historique de la classe. Cette liste d'instances peut être implémentée comme un attribut statique de la classe. En conséquence la première organisation de l'historique est calquée sur l'organisation hiérarchique des classes d'objets du module d'aide : à chaque classe correspond un historique.

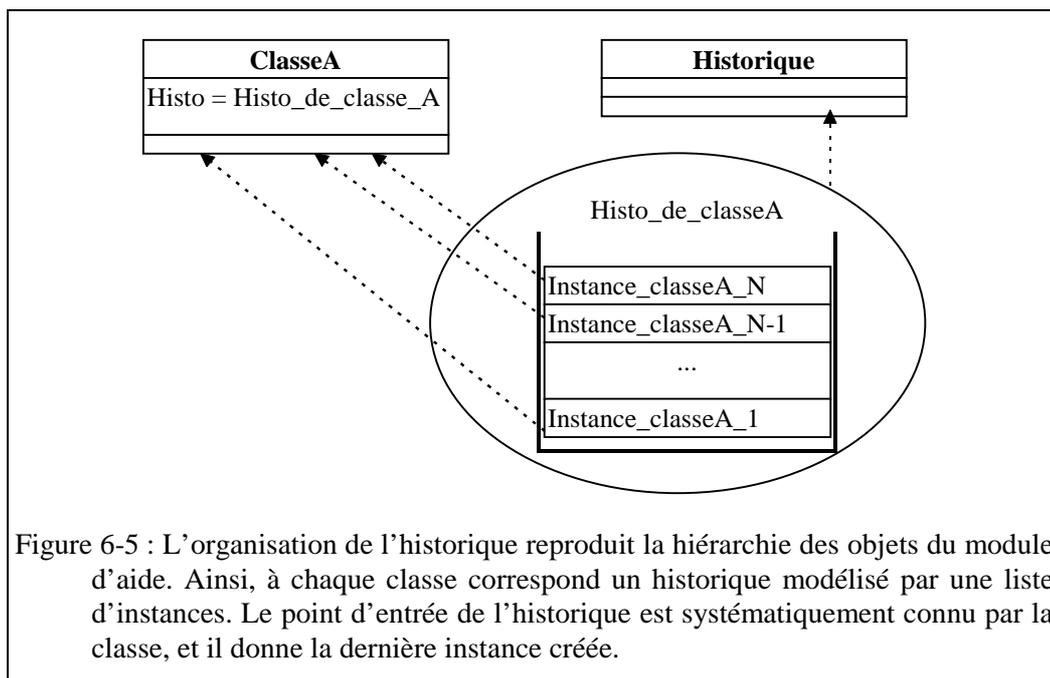
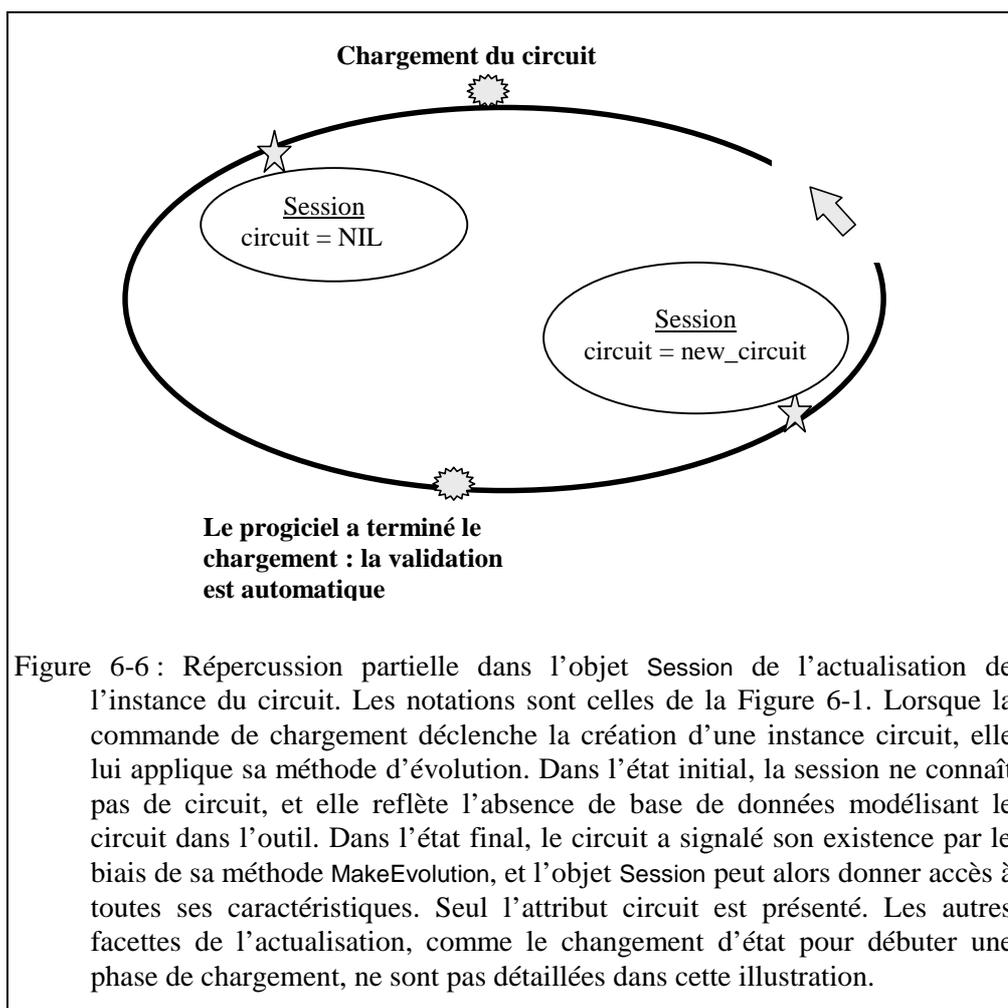


Figure 6-5 : L'organisation de l'historique reproduit la hiérarchie des objets du module d'aide. Ainsi, à chaque classe correspond un historique modélisé par une liste d'instances. Le point d'entrée de l'historique est systématiquement connu par la classe, et il donne la dernière instance créée.

Le second temps correspond à la fin du traitement, où l'action actualise le contexte du module d'aide en précisant dans quelle phase elle s'inscrit. Pour se positionner en fonction du contexte, un objet ne fait pas uniquement appel à son propre historique. L'objet Session, qui est un objet global, donne accès à tous les objets manipulés dans le module d'aide. Prenons l'exemple de la commande de chargement d'un circuit. Dans le module d'aide ses conséquences sont :

- la commande courante de l'objet Session correspond à une instance de chargement,
- la création d'une nouvelle instance de circuit,
- une actualisation de l'objet Session, pour qu'elle référence l'instance de circuit,
- si nécessaire la création d'une phase de chargement, qui devient la phase courante,
- l'ajout de la commande chargement dans l'historique de la phase courante.



Le précédent exemple du chargement du circuit illustre la construction locale de l'historique et la mise à jour de l'instance de l'objet Session qui centralise l'ensemble des objets du module d'aide. Aujourd'hui la trace des actions est exploitée par le module d'aide, mais elle peut aussi être utile pour une étude a posteriori du déroulement d'une session. De manière schématique, il s'agit de reporter dans un fichier les éléments de l'historique au fur et à mesure de l'évolution du contexte. Cette étude pourrait apporter des indices pour remonter à la source des problèmes, en évaluant de manière statistique les « points noirs » de l'utilisation du progiciel, ou bien des lacunes dans les conseils du système d'aide. Cette mise en œuvre n'est pas immédiate, car il faut au préalable définir si des informations supplémentaires sont nécessaires pour tirer des enseignements de la trace, et par ailleurs remplir deux conditions :

- convaincre les utilisateurs d'adresser les traces à l'équipe du support technique du progiciel,
- construire un fichier lisible pour que les utilisateurs évaluent les informations ainsi transmises.

En résumé, le suivi de l'utilisateur doit déterminer ce qui devient possible. Pour cela, l'action s'appuie sur sa description sémantique dans le module d'aide et sur le contexte défini. Cette information est apportée par l'actualisation de la position dans le schéma de phases,

opérée par l'objet représentant l'action. En réponse à son application, l'objet réagit en appelant ses mécanismes internes, représentés par ses méthodes.

6.2. Une aide préventive

L'aide préventive concerne les actions qui, dans leur cycle de traitement, ont une étape explicite de confirmation. L'exemple typique est la modification d'un champs édité dans une boîte de dialogue : cette modification est effectivement répercutée dans le noyau fonctionnel du progiciel quand l'utilisateur valide et quitte la boîte de dialogue. Néanmoins, il est possible d'observer cette modification avant sa confirmation, et ainsi la commenter. De cette manière il est possible de signaler des problèmes potentiels de mauvaise utilisation, ou tout au moins détectés comme tels par le module d'aide, avant qu'ils ne soient avérés.

6.2.1. Intervention avant la confirmation d'une action

L'idée est donc d'observer les actions de l'utilisateur avant qu'elles ne soient validées dans le noyau fonctionnel du progiciel. Les informations sur les actions de l'utilisateur ne sont donc pas communiquées par le progiciel à travers son API, car il n'en a pas encore connaissance. Elles sont directement interceptées dans l'interface avec l'utilisateur et adressées au module d'aide, ce qui représente un traitement spécifique ajouté à celui de chaque élément de l'interface.

Chacune de ces actions, encore inconnue du noyau fonctionnel, est répercutée dans le module d'aide pour déterminer si la modification est effectivement une action candidate dans le contexte de travail. Dans la négative, une des règles « réflexes » préventive, discutée au chapitre 5, est mise en défaut et le résultat de son application donne un indice sur le problème d'utilisation. Ce résultat de l'évaluation de la règle est mémorisé par l'instance associée à l'action. Cette mémorisation doit en particulier fournir l'identificateur de l'explication destinée à l'utilisateur.

Les signalements de l'aide préventive ne sont pas coercitifs : l'utilisateur peut décider de passer outre. Même si les conseils du module d'aide ne sont pas suivis, parce que l'utilisateur estime agir en connaissance de cause, le signalement de problèmes potentiels est une bonne manière d'attirer l'attention sur des utilisations atypiques. Si l'utilisateur n'est pas suffisamment confiant dans ses choix, il consultera le système d'aide pour vérifier pourquoi il existe un conflit.

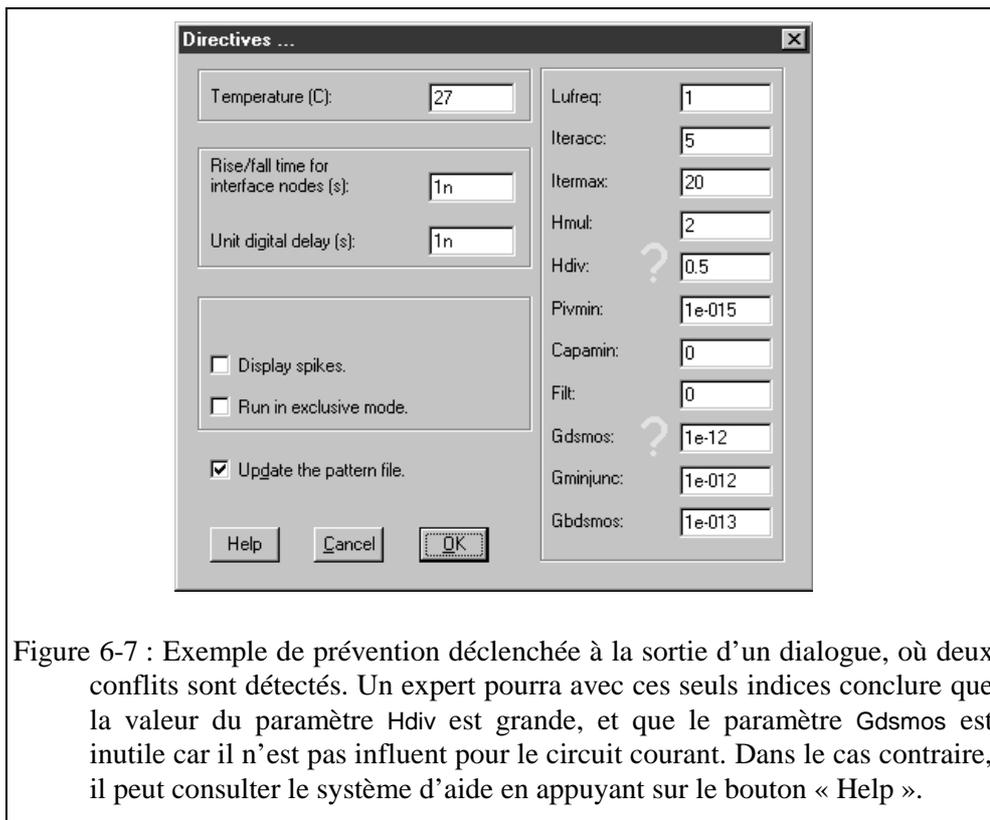


Figure 6-7 : Exemple de prévention déclenchée à la sortie d'un dialogue, où deux conflits sont détectés. Un expert pourra avec ces seuls indices conclure que la valeur du paramètre Hdiv est grande, et que le paramètre Gdsmos est inutile car il n'est pas influent pour le circuit courant. Dans le cas contraire, il peut consulter le système d'aide en appuyant sur le bouton « Help ».

6.2.2. Exemple de prévention

Nous considérons maintenant les vérifications faites par le module d'aide à l'étape préventive du cycle d'une action, avec l'exemple du paramètre *deltav* qui contrôle le pas d'itération lors de l'analyse du point de fonctionnement. Ce paramètre est accessible dans la boîte de dialogue de l'analyse du point de fonctionnement, qui regroupe les paramètres nécessaires à cette analyse.

Une boîte de dialogue est un élément de l'interface privilégié pour le signalement des conflits qui portent sur les paramètres. En effet, la prévention intervient :

- à l'ouverture de la boîte de dialogue, où la valeur des différents paramètres présents dans l'interface est vérifiée ;
- à la confirmation des modifications faites par l'utilisateur, qui sont préalablement évaluées dans le module d'aide.

La prévention doit être demandée par le progiciel, et dans ce cas précis par la boîte de dialogue : ce n'est pas le module d'aide qui interrompt le fonctionnement du progiciel. Ainsi, la gestion de la boîte de dialogue transmet au module d'aide la liste des paramètres qu'elle présente avec les résultats de la vérification faite par le module d'aide. L'interaction entre le progiciel et le module d'aide se déroule comme suit :

1. Le progiciel transmet les valeurs de tous les paramètres au module d'aide, et ne se limite pas seulement à ceux de la boîte de dialogue. En effet, un paramètre qui n'est pas présenté dans cette boîte de dialogue peut néanmoins influencer les paramètres

qu'elle rassemble. L'actualisation des valeurs du module d'aide est faite à partir de celles du noyau fonctionnel du progiciel, qui reste la référence.

2. Le module d'aide évalue chacun des paramètres en appliquant leur règle préventive `ls_relevant`. A cette étape, les instances des paramètres sont donc mises à jour non seulement avec les valeurs courantes, mais aussi avec l'évaluation de leur pertinence et de leur influence : le contexte du module d'aide est actualisé.
3. Le module d'aide transmet au progiciel le résultat de l'évaluation pour chacun des paramètres de la liste initialement adressée. Ce résultat est soit un conflit potentiel, soit une confirmation d'un paramètre pour lequel aucune mise en garde n'est identifiée.
4. Le progiciel présente ce résultat à l'utilisateur qui souhaite bénéficier d'une aide préventive. Dans le cas contraire, l'inhibition des interventions du module d'aide a pour conséquence de ne pas présenter dans l'interface d'informations supplémentaires. Toutefois le cycle décrit actualise toujours le module d'aide, même si l'utilisateur n'observe aucune modification dans l'interface.

Ce cycle est donc automatiquement déclenché dès l'affichage de la boîte de dialogue, donnant ainsi à l'utilisateur les conseils du module d'aide sur l'état actuel des valeurs. Considérons le traitement du paramètre `deltav`, qui correspond à une valeur numérique dans la boîte de dialogue. Quand le progiciel transmet cette valeur, il communique au module d'aide un couple de données : (identificateur du paramètre, valeur transmise).

Dans le module d'aide, l'identificateur permet de connaître la classe de l'objet, qui elle-même accède à l'historique de ses instances et ainsi retrouve l'instance courante. En fonction de la valeur transmise par rapport à celle connue dans le module d'aide, l'objet associé au paramètre est ou non actualisé. Quand le contexte du progiciel est reporté dans le module d'aide, l'intérêt du noyau sémantique est de vérifier si la valeur transmise est cohérente avec l'ensemble des informations contextuelles et des connaissances.

Si nous considérons le paramètre `deltav`, sa représentation dans le module d'aide comprend d'une part les informations contextuelles :

- la valeur courante du paramètre, dans ce cas c'est une valeur numérique (`value_current`) ;
- la valeur temporaire du paramètre, qui correspond à la valeur modifiée mais pas encore validée par l'utilisateur (`value_temp`) ;
- la valeur précédente du paramètre (`value_previous`);
- l'existence d'un conflit sur le paramètre, qui sera un booléen (`Is_conflictual`) ;
- la nature de ce conflit, donnée par l'identificateur du conflit (`Id_conflict`);

et d'autre part les connaissances contextuelles et structurelles :

- la valeur par défaut, calculée en fonction de la plage de valeurs des tensions atteintes dans le circuit (`value_default`) ;
 - la valeur conseillée par le module d'aide (`value_suggested`) ;
 - la borne minimale de la plage de valeur raisonnable (`value_min_suggested`) ;
 - la borne maximale de la plage de valeur raisonnable (`value_max_suggested`) ;
 - l'identificateur de l'explication textuelle pour la syntaxe du paramètre (`Id_syntax`) ;
-

- l'identificateur de l'explication textuelle pour la définition générale du paramètre (Id_definition) ;
- l'identificateur de l'explication textuelle pour la localisation dans l'interface du progiciel du paramètre (Id_access) ;
- et les règles de bonnes utilisations qui sont les méthodes de la classe ;

Le calcul de la valeur par défaut du pas d'itération suit une règle appliquée par le progiciel, qui la fixe à un dixième de la plage des tensions d'alimentation du circuit. Cette heuristique permet de faire fonctionner une majorité de circuit électroniques. Néanmoins, elle n'est pas adaptée quand le circuit comprend des composants électroniques du type transistors bipolaires. Dans ce cas, une autre règle doit être prise en compte :

RI : « La valeur du pas d'itération ne doit pas être « trop élevée » si le circuit comprend des transistors bipolaires. Une valeur usuellement satisfaisante est de 100mV. Par contre, une faible valeur d'itération doit être compensée par un nombre d'itérations plus grand pour couvrir toute la zone de convergence. »

Cette règle de bonne utilisation était connue des utilisateurs experts seulement, car le progiciel ne prévoyait pas de définir les valeurs des paramètres en fonction du circuit. Le module d'aide permet de transmettre cette connaissance à des utilisateurs néophytes. Qui plus est, les explications sur ce paramètre mettent en évidence le lien avec le paramètre maxiter, qui contrôle le nombre d'itération de l'analyse numérique.

Prenons l'exemple de l'évolution de l'instance dans le module d'aide, en considérant que le circuit modifié comprend maintenant des transistors bipolaires. A l'ouverture de la boîte de dialogue, la valeur du paramètre deltav est transmise au module d'aide qui applique la règle ls-relevant. Dans cette règle, l'objet deltav interroge les caractéristiques de l'objet Circuit à travers l'objet Session, qui centralise les informations contextuelles. Dans la mesure où l'objet Circuit confirme qu'il comporte des composants du type transistor bipolaire, la valeur conseillée est fixée à 100mV :

```

Si Session->Circuit->Comprend_des_bipolaires()
  et Session->analyse_point_fonctionnement=Id_Failure
Alors
  Value_suggested = 100mV,
  Value_min_suggested = 0.9*Value_suggested,
  Value_max_suggested = 1.1*Value_suggested,
  Id_conflict=Valeur_faible_pour_composant_bipolaire
  ...

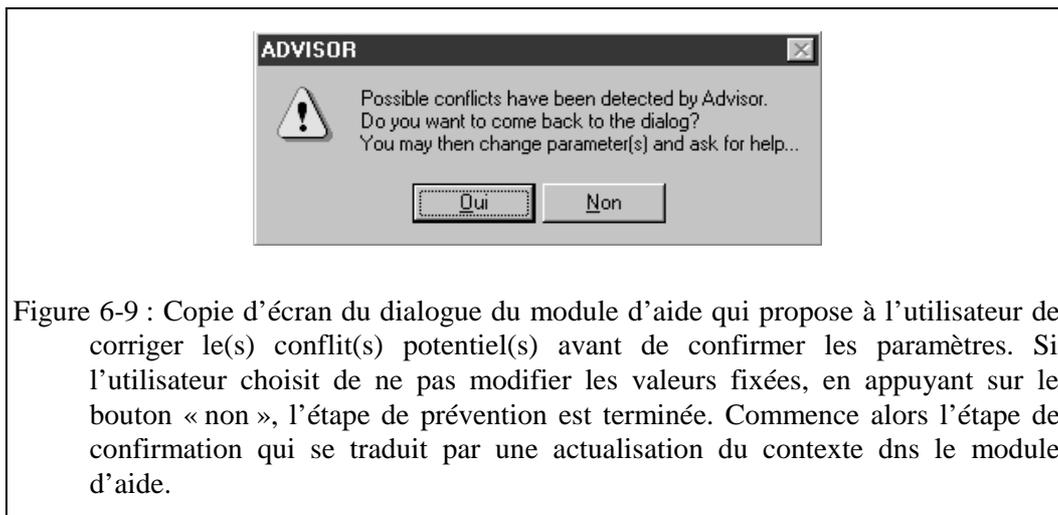
```

Figure 6-8 : Extrait de la règle pour définir les conseils concernant l'objet deltav, en fonction du contexte représenté par l'objet Session.

Le module d'aide est alors capable de transmettre au progiciel l'évaluation de ce paramètre dans le contexte d'utilisation. Si la valeur temporaire du paramètre est en dehors de la plage conseillée, l'objet mémorise un conflit par son attribut ls_conflict. A noter que tant que l'utilisateur n'a pas modifié le paramètre correspondant, la valeur temporaire est égale à la valeur courante. Pour la présentation de l'information, nous avons choisi d'indiquer dans

l'interface du progiciel seulement le signalement du conflit, de manière à obtenir une intervention discrète du module d'aide, comme cela est illustré précédemment par la Figure 6-7.

L'actualisation des objets associés aux paramètres dans le module d'aide est limitée à ces seuls objets en modifiant leurs valeurs et leurs informations contextuelles. En effet, à la fermeture d'une boîte de dialogue, le module d'aide vérifie les paramètres et signale les conflits détectés donnant ensuite le choix entre la confirmation des modifications faites, ou leur correction. Une confirmation se traduit alors par la fermeture définitive du dialogue, et dans le module d'aide par une mise à jour de la valeur courante : l'attribut `value_current` prend la valeur de `value_temp`.



Cette mise en garde systématique peut agacer l'utilisateur qui souhaite délibérément enfreindre les recommandations du système d'aide. Il pourra alors désactiver les interventions du module d'aide, qui poursuivra néanmoins le cycle de traitement. Dans tous les cas, le suivi des actions est maintenu, ce qui permet à l'utilisateur de conserver une aide contextuelle à jour. En effet, bien que désactivé, le module d'aide est toujours accessible pour un utilisateur qui souhaite le questionner : ce suivi permanent laisse la possibilité à l'utilisateur d'activer et de désactiver le module d'aide selon ses besoins.

En résumé, le signalement d'un conflit n'est pas une contrainte que l'utilisateur doit obligatoirement lever : le module d'aide donne à l'utilisateur une idée de ce qu'il peut faire, et non pas de ce qu'il doit faire. Par contre c'est une opportunité pour le module d'aide de communiquer quelles sont les raisons qui conduisent à ce conflit, et ainsi inciter l'utilisateur à consulter les influences entre les différents paramètres et commandes du progiciel.

6.3. La modification du contexte dans le module d'aide

De manière plus générale, l'étape de confirmation dans le cycle de traitement d'une action implique des modifications dans l'ensemble du module d'aide : sur les phases et sur les informations contextuelles de l'objet `Session`, en plus de l'objet associé à cette action. Nous prenons l'exemple de l'actualisation du module d'aide qui intervient en fin de cycle de traitement, comme une conséquence de la confirmation d'une commande.

6.3.1. Répercuter la confirmation d'une commande

La nature de l'actualisation est particulière à chaque commande. Elle est par ailleurs influencée par le contexte et en particulier par le résultat de la commande dans le progiciel. Pour tenir compte de cette disparité, les conséquences d'une commande sont traitées par la méthode nommée *MakeEvolution*, de la classe d'objets associée à chaque commande. Cette méthode explicite le raisonnement local qui permet de définir les possibilités dans le nouveau contexte.

Concrètement, lorsque l'utilisateur confirme ou annule sa commande, le progiciel transmet au module d'aide ce résultat en affectant la variable *ID_RESULT*, présentée au début de ce chapitre. Cette information contextuelle est mémorisée par l'instance de commande créée en début de cycle. L'actualisation du contexte dans le module d'aide ne commence que lorsque le progiciel indique la fin de son propre traitement, qui correspond aussi à un contexte stable : l'analyse numérique est terminée, le noyau fonctionnel est mis à jour... Cette indication de fin de commande dans le progiciel assure la cohérence du contexte du module d'aide, qui ne sera actualisé qu'à partir de données stables du noyau fonctionnel. Par ailleurs, le fonctionnement du module d'aide ne risque pas de pénaliser celui du progiciel : il ne peut pas ralentir une analyse numérique par exemple. Dans la pratique, le temps de fonctionnement du module d'aide n'est pas perceptible par l'utilisateur. Toutefois cela ne peut pas être généralisé, ni pris comme hypothèse pour de futurs systèmes car la modélisation des connaissances n'est pas étrangère à cette performance : le système ne doit pas choisir entre plusieurs règles. Pour un objet il existe une seule règle pour traiter chaque « réflexe », écrite en tenant compte de l'ensemble des comportements possibles. En conséquence, il n'existe pas d'étape dans le fonctionnement du module d'aide consacrée au choix d'une règle dans un ensemble.

Il est d'autant plus compréhensible d'attendre la fin du traitement d'une commande dans le progiciel, que l'utilisateur peut arrêter une analyse lancée s'il estime la durée trop longue, ou qu'il identifie une erreur sur la partie des graphes déjà affichés. Il faut noter qu'une simulation de plusieurs heures n'est pas exceptionnelle pour un circuit complexe. Même si cet abandon n'est pas pleinement exploité par le module d'aide qui n'en connaît pas la raison, il doit être distingué d'une analyse qui aboutit, ou qui échoue. Dans l'état actuel des connaissances sur l'utilisation du progiciel, les motivations de l'abandon ne sont pas connues du module d'aide. En conséquence, le traitement d'une analyse abandonnée consiste à mémoriser ce résultat dans l'historique de la classe associée à la commande.

De surcroît, l'actualisation du module d'aide après le déroulement de l'action facilite le traitement des annulations. Ces dernières interviennent lorsque l'utilisateur réalise à temps une erreur de manipulation, avec ou sans l'assistance du module d'aide. Par exemple, dans la boîte de dialogue des paramètres « Directives » de la Figure 6-7, les modifications faites sur ces paramètres depuis l'ouverture du dialogue peuvent être annulées. Dans la mesure où il n'existe pas de connaissance pour déduire la signification de l'annulation, les conséquences dans le module d'aide se limitent à la mémorisation de la commande annulée dans l'historique de la classe correspondante.

En résumé, dans le cas d'une annulation ou d'un abandon, les possibilités définies au cycle précédent restent valables : l'actualisation du module d'aide se borne à une observation des actions. La situation est différente quand la commande est satisfaite ou au contraire si elle aboutit à une situation d'échec : il faut alors évaluer les nouvelles possibilités en fonction du contexte observé. Dans la sous-section « Organisation du suivi pour une mise à jour du contexte », nous avons déjà illustré l'actualisation de la position dans le schéma de phases suite à

une analyse OP réussie, voir Figure 6-2. Nous complétons maintenant la description du suivi avec la méthode MakeEvolution de l'objet associé à la commande de chargement d'un circuit.

6.3.2. Exemple de suivi lors de la confirmation d'une commande

L'exemple suivant illustre l'actualisation du module d'aide, suite à la commande de chargement d'un circuit. Dans le progiciel, cette commande correspond à la construction de la base de données à partir d'une description textuelle du circuit. Pour le système d'aide, la conséquence de cette commande, est

- de se placer dans le schéma de phases, soit au niveau du chargement, soit dans le contexte de travail précédent s'il s'agit d'une modification dans le processus d'essai - erreur. Par exemple si le résultat de la commande est un échec, la méthode impose de se placer en phase de chargement, de manière à disposer des informations et des connaissances pour résoudre ce problème.
- si cette commande ouvre une nouvelle session, la méthode demande au noyau fonctionnel l'ensemble des informations sur les caractéristiques du circuit et sur les valeurs courantes des paramètres. Elle actualise les objets associés à ces éléments dans le module d'aide, avec leurs caractéristiques utiles pour fournir une aide. Cette actualisation nécessite éventuellement la création de nouveaux objets.

Pour exemple, nous donnons la méthode MakeEvolution de la commande de chargement de circuit, qui par défaut est traitée par la phase LoadPhase. Trois situations mènent à la création de la phase référencée par la commande : soit le circuit n'existe pas encore, soit la phase courante correspond à l'initialisation de la session, soit enfin le chargement n'a pas pu être mené à bien. Les autres situations se résument à une commande satisfaite, avec un circuit déjà défini : une nouvelle phase n'est pas nécessaire dans le suivi. La commande de chargement est alors intégrée dans l'historique de la phase en cours.

```

Méthode MakeEvolution pour la commande de chargement de circuit

SI LoadPhase n'est pas la phase en cours
  ALORS
  Créer une nouvelle phase LoadPhase QUAND
    - le circuit n'est pas défini
  ou bien    - la phase en cours correspond à la phase d'initialisation InitPhase
  ou bien    - le chargement du circuit est en échec
SINON, c'est-à-dire quand le circuit existe, et la commande est satisfaite ou annulée
  la phase en cours n'est pas modifiée, mais le circuit doit être actualisé
//...

```

Figure 6-10 : Cet exemple illustre comment la méthode MakeEvolution participe au suivi des actions. Il décrit l'actualisation du schéma de phases suite à l'application d'une commande de chargement de circuit, Load circuit. La phase attachée est LoadPhase, qui sera satisfaite si le circuit est correctement chargé. A l'application de la commande, trois situations imposent la phase LoadPhase. Les deux premières sont liées au début d'une session : si le circuit n'est pas encore défini dans le module d'aide, ou si la phase courante correspond à InitPhase qui modélise le démarrage. La troisième situation correspond à un échec du chargement, où l'objectif est d'aboutir au chargement : la nouvelle phase en cours doit être celle de chargement.

Le contexte sur lequel repose l'évaluation de la situation de l'utilisateur, doit lui aussi être mis à jour : c'est le deuxième volet de la méthode MakeEvolution. Outre la mise à jour du schéma de phases, la méthode MakeEvolution d'une commande est aussi chargée de déclencher la mise à jour des objets altérés par l'application de la commande. Elle répercute ainsi dans le module d'aide le nouvel état de ces éléments à partir de la référence que constitue le noyau fonctionnel.

```

Méthode MakeEvolution pour la commande de chargement de circuit (2ème partie)
Mise à jour des caractéristiques du circuit et des directives de simulation
...
session->SetCircuit(); //déclenche l'actualisation du circuit, via l'objet session
session->SetLParam(); //déclenche l'actualisation de l'ensemble des paramètres
//la phase courante détermine sa satisfaction en fonction de celle de la commande :
session->phase_current->SetSatisfy();
//l'instance de la commande marque la fin de son traitement
this->SetComplete( true );

```

Figure 6-11 : La méthode MakeEvolution d'une commande déclenche les mises à jour nécessaires pour que le module d'aide soit en phase avec le contexte réel de l'outil. Elle adresse l'ensemble des objets susceptibles d'être altérés par l'application de la commande. L'objet session est le pivot des échanges entre les objets du module d'aide, car en tant que contrôle de la session il les centralise. L'initialisation est déléguée à chacun de ces objets, qui interroge le noyau fonctionnel via l'API.

Si le circuit n'existait pas encore, la méthode `MakeEvolution` de la commande déclenche la création d'une instance de circuit. Dans le cas contraire, l'objet associé au circuit serait simplement mis à jour. Cette actualisation est du ressort de chaque objet, qui centralise les informations et les connaissances nécessaires.

La méthode `MakeEvolution` concentre la connaissance et la procédure pour réaliser l'actualisation d'un objet. Elle est attachée à chacun des composants du module d'aide qui modélise l'outil. Ainsi, chaque paramètre, chaque commande modélisée connaît les conséquences de son application dans le progiciel. La méthode construit dans le module d'aide l'image des modifications du progiciel.

6.4. Les capacités du module d'aide dans une situation d'échec

Considérons maintenant les informations et les connaissances disponibles dans le module d'aide pour apporter des conseils à l'utilisateur dans une situation d'échec.

Prenons l'exemple d'une phase « recherche du point de fonctionnement », qui s'inscrit dans le schéma de phases après la phase préalable de chargement du circuit. Dans la mesure où le progiciel est capable de nommer les types d'échecs, le module d'aide doit établir une correspondance avec les phases qui détiennent les connaissances utiles pour résoudre ces échecs. Le tableau suivant est un extrait des correspondances établies dans le module d'aide.

Type d'échec	Phase associée dans le module d'aide
<code>Id_too_many_nodes</code>	Phase de chargement
<code>Id_unknown_device</code>	Phase de chargement
<code>Id_unable_to_converge</code>	Phase de recherche point de fonctionnement
<code>Id_bad_news</code>	Phase d'analyse temporelle

Figure 6-12 : Extrait du tableau des correspondances entre la nature des échecs détectés dans le progiciel, et la phase adaptée dans le module d'aide pour résoudre ce problème. Le descriptif correspond à la signification de cet échec.

Le type des échecs n'identifie pas la cause précise, mais il permet à l'utilisateur expert de sélectionner un certain nombre d'actions correctrices potentielles. Dans le module d'aide, cette sélection se fait en deux étapes :

- déterminer la phase qui doit traiter l'échec, en exploitant le tableau de correspondance ;
- identifier les commandes susceptibles de corriger l'échec, en adressant à chaque commande potentielle de cette phase la demande de déterminer leur intérêt pour résoudre ce problème.

Plaçons nous dans la situation de la phase « recherche du point de fonctionnement », quand un échec du type « `id_unable_to converge` » est transmis au module d'aide. Dans cette situation, si la phase en cours ne correspond pas à celle indiquée par le tableau des correspondances, le suivi impliquera d'abord une actualisation du module d'aide pour l'établir comme étant la phase en cours. Voyons maintenant comment la liste des commandes candidates et influentes est construite.

Dans le module d'aide, la phase « recherche du point de fonctionnement » connaît la liste des commandes potentielles. Pour connaître les commandes candidates et influentes, il suffit donc de déterminer parmi ces commandes celles qui sont utiles pour traiter la situation courante. En exploitant le raisonnement local aux objets du module d'aide, le principe est de demander à chacun de ces objets s'ils sont ou non influents. Il est alors possible de recueillir dans une liste les actions susceptibles de résoudre le problème.

Le fonctionnement du module d'aide est donc similaire pour la prévention ou pour la confirmation, mais en exploitant les méthodes adaptées des objets qui sont respectivement `ls_relevant` et `ls_influent`. Basé sur l'exploitation des connaissances de chaque élément du logiciel, le module d'aide a la capacité de commenter leurs intérêts et leurs influences en fonction du contexte. Aujourd'hui, seules les informations sur les paramètres sont transmises à l'utilisateur, car il peut immédiatement les exploiter.

Il est plus délicat de communiquer de manière satisfaisante une information sur une commande, qui soit concise, discrète et néanmoins utile. Par exemple, une commande influente dans une situation d'échec sera utile si préalablement certains paramètres sont modifiés : le conseil ne se limite pas à lancer la commande. Aujourd'hui, une commande identifiée comme influente par le module d'aide n'est pas signalée dans l'interface avec l'utilisateur. L'utilisateur doit de lui-même consulter les explications contextuelles pour vérifier s'il existe ou non un conflit pour une commande donnée.

Pour pleinement exploiter l'ensemble des capacités du module d'aide, le chapitre suivant introduit une notion de conseil, pour indiquer la procédure qui associe des paramètres et des commandes ou qui détermine un enchaînement de commandes.

Chapitre 7

7. L'aide à la prise de décision

L'implémentation actuelle du module d'aide dans le progiciel présente l'inconvénient de ne pas signaler à l'utilisateur toutes les informations disponibles sur les commandes, faute d'une interface adaptée. Le besoin est de mettre en évidence les liens entre les commandes et entre les commandes et les paramètres. Ce signalement ne peut pas se faire dans le cadre de l'interface actuelle avec l'utilisateur, car les commandes sont accessibles séparément dans des menus déroulants. Ainsi, contrairement aux paramètres qui sont eux regroupés dans une boîte de dialogue, toutes les commandes ne sont pas portées à la connaissance de l'utilisateur : il n'est pas possible de donner les informations souhaitées à l'aide des simples signalements, qui seront donc réservés aux paramètres.

L'introduction complémentaire d'une aide à la décision est donc motivée par le besoin de transmettre à l'utilisateur, sous une forme directement exploitable, des informations sur le fonctionnement des commandes. Bien que certaines de ces connaissances soient déjà manipulées par le module d'aide, il faut constater que les utilisateurs ne les consultent pas faute d'être mises en évidence au moment où elles seraient utiles, et en particulier dans une situation d'échec. Comme nous l'avons précisé précédemment, si une situation d'échec est regrettable, elle donne paradoxalement l'opportunité à l'utilisateur de s'appropriier les options du progiciel, dans la mesure où le système d'aide les lui suggère.

Le module d'aide présenté jusqu'ici est capable de suivre les actions de l'utilisateur. En s'appuyant sur les connaissances contextuelles, il identifie les actions influentes et les conflits, en expliquant les motifs de ses choix. L'aide à la prise de décision complète le module d'aide pour répondre aux besoins de l'utilisateur dans une situation d'échec, en indiquant l'ensemble des alternatives à envisager pour la résoudre. Les moyens retenus pour compléter l'aide actuelle sont :

- des conseils connus des utilisateurs experts, qui apportent un niveau de description pour centraliser les objets concernés par la résolution d'une situation d'échec, de manière à exploiter les connaissances et les informations déjà disponibles ;
- des expériences passées, mémorisées dans le module d'aide et exposées à l'utilisateur quand elles sont susceptibles de fournir des indices pour résoudre le problème dans le contexte courant.

Le terme « aide à la décision » est abusif, et il nous faut clairement indiquer que nous n'avons pas recours à la théorie de la décision. Le terme est de fait employé comme une abré-

viation de *l'aide pour prendre une décision*. Dans notre application l'utilisateur est réellement confronté à un problème de décision, c'est-à-dire à un choix parmi une liste d'alternatives possibles étant donné un « état du monde », mais il n'existe pas aujourd'hui de connaissances pour estimer l'intérêt d'une alternative par rapport à une autre : le système d'aide n'apporte que des conseils pour permettre à l'utilisateur de décider en connaissance de cause.

En effet dans une situation d'échec liée à l'application d'analyses numériques, il n'est pas possible d'évaluer les chances, ou la probabilité, qu'une suggestion aboutisse à la convergence numérique de l'analyse. La connaissance se limite aux conditions nécessaires, mais en rien suffisantes, pour qu'une suggestion puisse influencer positivement l'analyse. Dans notre application, le système d'aide s'applique à évaluer l'état du monde, mais il n'est pas capable d'accomplir la phase de projection, autrement qu'en apportant des explications générales sur les avantages et les inconvénients des alternatives. De fait, dans le domaine d'application étudié qu'est la simulation, les utilisateurs experts procèdent par essai et erreur pour d'abord aboutir à une solution, et ensuite l'optimiser.

7.1. Les conseils

Nous décrivons maintenant comment un conseil est modélisé dans le module d'aide, par un objet Advice, et comment ces conseils interviennent dans le fonctionnement du module d'aide et du progiciel. La maquette illustre nos propos, sachant que l'implémentation de l'aide à la prise de décision dans le module Advisor n'est pas achevée à ce jour.

Deux préoccupations guident l'introduction des conseils, formulés pour une situation d'échec donnée :

- Une information sur l'utilisation des commandes qui associe non seulement une commande, mais aussi des paramètres et d'autres commandes.
- Une aide qui ne gêne pas l'utilisation du progiciel. Pour éviter toute intervention du système d'aide sur une situation d'échec, c'est l'utilisateur qui doit solliciter les conseils en appliquant la commande particulière « Demande de conseils » dans le progiciel.

7.1.1. La définition des conseils formalisée autour des échecs

Pour comprendre les choix retenus pour représenter ces conseils, il faut cerner les caractéristiques des connaissances attachées à la résolution d'une situation d'échec. Nous insistons sur l'importance du message d'échec, qui est le point de départ pour sélectionner les tentatives de résolution. Toutefois un message ne définit pas à lui seul la situation d'échec, car il n'est pas toujours possible d'associer dans le progiciel une impossibilité de calcul à la cause physique de ce problème de calcul. L'exemple exposé dans la Figure 7-1 suivante montre que la non convergence de l'analyse de la recherche du point de fonctionnement aboutit à un échec du type « Unable to converge ». Mais la cause physique, qui peut être liée à des contraintes trop fortes, ou à une mauvaise topologie du circuit ne peut pas être déduite. Ni par le progiciel, ni de manière certaine par un expert. Or la résolution de la situation d'échec consiste justement à corriger la cause physique.

Aujourd'hui le module d'aide exploite les messages définis dans le noyau fonctionnel du progiciel. Une étape supplémentaire consiste à affiner les messages d'échec dans le progiciel, quand cela est possible et susceptible de conduire à de meilleurs conseils.

Prenons l'exemple concret d'un échec lors de l'analyse du point de fonctionnement d'un circuit. Les conseils formalisés avec les experts du progiciel sont répertoriés dans le document [Duprez95], qui éclaire les principales difficultés d'utilisation du progiciel Smash. Ce document est une synthèse des connaissances obtenues par le dialogue avec les experts et par une pratique personnelle de la simulation avec le progiciel. L'extrait suivant concerne le message « unable to converge », qui indique précisément le problème, c'est-à-dire que l'analyse n'a pas convergé, mais n'explique pas ses causes. Les différentes causes à envisager sont donc répertoriées.

Message « unable to converge »

Observer l'évolution du résidu. S'il oscille autour d'une valeur, la conserver et relancer une analyse de convergence (UseOP automatique). Après la deuxième tentative, si l'oscillation persiste, proposer une analyse de convergence car le circuit est peut-être physiquement oscillant [...]

Si le résidu n'oscille pas, mais reste constant à partir d'une certaine itération, cela peut indiquer que la plage de recherche n'est pas assez large. Il faut d'une part vérifier la présence de sources de tension commandée, et d'autre part vérifier que les noeuds ont atteint des valeurs extrêmes de la plage en vérifiant le compte rendu. Si l'une ou l'autre des recherches aboutit, il faut proposer de modifier la plage (et éventuellement proposer d'utiliser une analyse $\text{deltav} = -2$ mais gare au risque de divergence).

Faire l'analyse par défaut (en mettant la commande en commentaire) et en repartant de zéro (reset everything) : c'est la première solution à proposer si aucun indice ne permet de préférer les deux autres solutions. Eventuellement proposer de diminuer deltav (surtout si grand par défaut devant les valeurs à utiliser : typiquement en bipolaire il doit être autour de 0.1V) et d'augmenter la valeur de maxiter .

Appliquer la méthode $\text{deltav} = -1$ (surtout si le circuit peut vérifier un flot de signal).

Appliquer la méthode $\text{deltav} = 0$ avec $\text{Maxiter} = 1000$.

Si le circuit présente des jonctions, envisager une approximation de la solution avec la directive OPHELP sur Gdsmos, Gbdsmos dans le cas de Mos. Si le circuit contient des bipolaires et/ou des diodes, faire OPHELP sur Gminjunc. Si le circuit contient des JFET, faire un gminjunc et gdsmos. Terminer l'analyse par un OP avec USEOP.

...

Figure 7-1: Extrait du document « Informations sur le progiciel Smash », qui liste les conseils pour résoudre un échec de la recherche du point de fonctionnement. L'ordre ici proposé est arbitraire, mais ne reflète pas un ordonnancement judicieux. La démarche est d'une part de rechercher des indices supplémentaires dans le compte-rendu de l'analyse, en particulier sur les valeurs atteintes et la variation du résidu. D'autre part, il faut vérifier les caractéristiques du circuit avant d'envisager certaines possibilités.

Par exemple, s'il existe des transistors Mos, il est intéressant de modifier les conductances des jonctions pour obtenir une première approximation, utilisée ensuite comme point de départ pour appliquer l'analyse.

En conséquence, pour un même message d'erreur, il n'existe pas une solution, mais bien un ensemble de possibilités qui doivent être envisagées dans un processus d'essais et erreurs. L'intérêt des conseils est justement de cerner au mieux les causes possibles, en fonction des caractéristiques du circuit et du contexte en général. Ensuite, pour chacune des causes possibles, le conseil propose des modifications et l'utilisation des options spécifiques du progiciel.

De cette manière, le système d'aide valorise les capacités du progiciel, qui restent trop souvent méconnues.

Un conseil se définit par rapport à la situation d'échec qu'il commente, c'est-à-dire par rapport à une analyse, complétée par le type de l'échec à traiter. Comme le montre la figure suivante, Figure 7-2, le conseil est un ensemble d'informations. Il répond à une situation d'échec précise, et il a pour but d'inciter l'utilisateur à modifier les contraintes fixées dans le progiciel. Dans le module d'aide, ces informations sont obtenues par l'objet Advice associé au conseil, qui a les attributs suivants :

- l'analyse qui est mise en échec ;
- le type d'échec ;
- une description générale du conseil ;
- les conditions dans lesquelles il est candidat, qui seront exprimées par des règles réflexes ;
- les avantages escomptés de son application ;
- les inconvénients possibles ;
- les paramètres à modifier, et la tendance de leur modification ;
- l'analyse qui teste le nouvel état ;

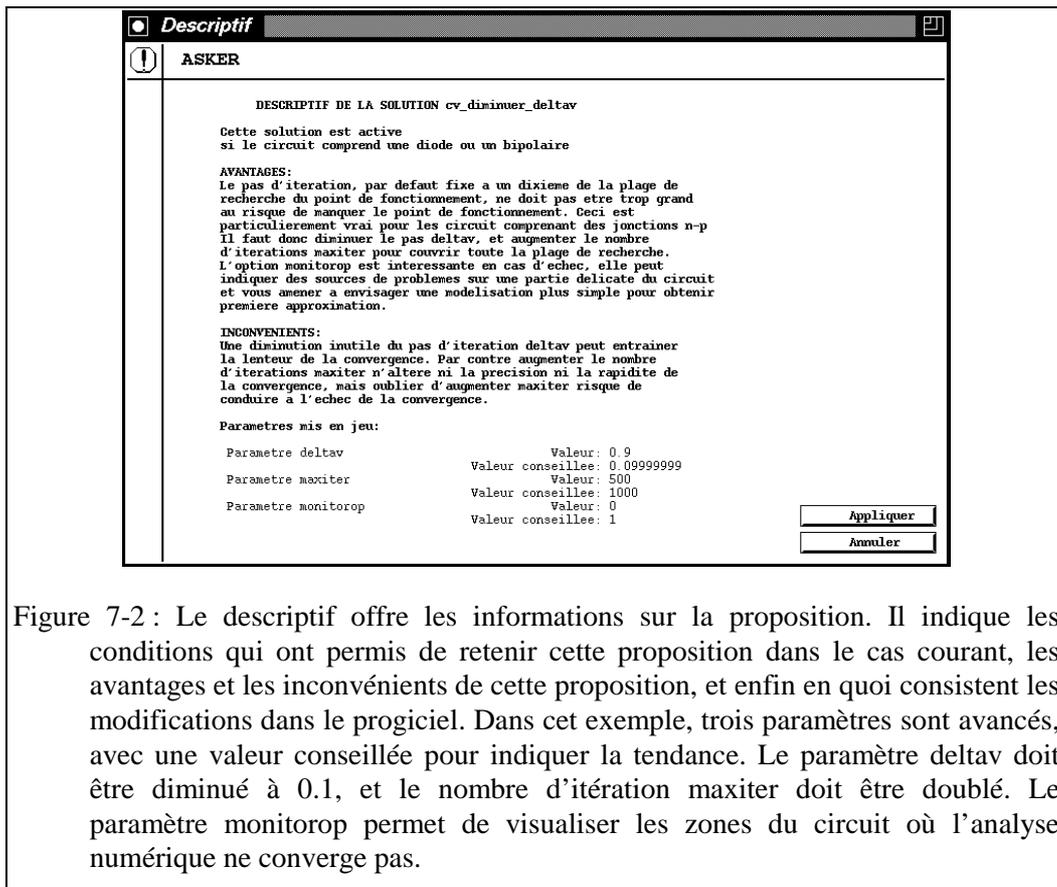


Figure 7-2 : Le descriptif offre les informations sur la proposition. Il indique les conditions qui ont permis de retenir cette proposition dans le cas courant, les avantages et les inconvénients de cette proposition, et enfin en quoi consistent les modifications dans le progiciel. Dans cet exemple, trois paramètres sont avancés, avec une valeur conseillée pour indiquer la tendance. Le paramètre deltav doit être diminué à 0.1, et le nombre d'itération maxiter doit être doublé. Le paramètre monitorop permet de visualiser les zones du circuit où l'analyse numérique ne converge pas.

Ces connaissances générales sont destinées à l'utilisateur, qui peut les consulter pour résoudre une situation d'échec. Ces informations ont été validées avec la maquette, dans

laquelle l'utilisateur devait choisir, soit d'appliquer une proposition dans sa totalité, soit de consulter simplement les informations.

7.1.2. Le raisonnement local pour les conseils

Les conseils sont donc étroitement liés aux situations d'échec qu'ils commentent. Leur pertinence est de surcroît dépendante du contexte général. Si l'importance des caractéristiques du circuit est évidente dans l'extrait précédent (Figure 7-2) où les conseils s'appuient sur la présence de transistors bipolaires, les informations contextuelles des actions précédentes jouent aussi leur rôle pour identifier celles qui ont déjà été appliquées. Nous listons maintenant les traits principaux d'un conseil :

- Il répond à une situation d'échec particulière, définie par le message d'échec et l'analyse insatisfaisante. Par exemple, dans l'analyse de recherche du point de fonctionnement, quand l'algorithme ne converge pas, le message correspondant est Unable to converge. Pour une analyse temporelle qui ne converge pas, le message est Bad news.
- Le contexte discrimine l'intérêt d'un conseil. Ainsi, pour résoudre une situation d'échec, seules les propositions candidates sont envisageables. En reprenant les conseils généraux extraits du manuel, le système d'aide ne propose pas de modifier le paramètre .GDSMOS, si le circuit considéré contient uniquement des transistors bipolaires. L'aide à la décision s'appuie sur le noyau sémantique du module d'aide décrit précédemment : un conseil n'est pertinent que si les modifications qu'il suggère sont pertinentes.
- Le résultat de l'application d'un conseil n'est pas prévisible. Lorsque le contexte est connu, il contribue à la détermination des propositions candidates, c'est-à-dire susceptibles de résoudre le problème. Cependant, si les conditions pour qu'un algorithme soit envisageable sont connues, il n'existe pas de garantie quant à sa convergence. En l'absence de la détermination précise des limites d'application d'un algorithme, il est utile de connaître les algorithmes candidats pour le problème courant.
- A chaque conseil correspond une tendance, décrite par ses avantages et ses inconvénients. Ceci explique pourquoi le simulateur présente de nombreux algorithmes différents pour mener une même analyse. La richesse du progiciel est de proposer des algorithmes complémentaires : certains privilégient la précision de l'analyse, et d'autres sa rapidité.
- Les conseils candidats ne sont pas proposés dans un ordre de priorité. C'est un choix de l'utilisateur de sélectionner une proposition, parmi celles qui peuvent être appliquées.

En fait, ces traits caractéristiques ont guidé toute la modélisation du système d'aide, pour qu'il soit capable de répondre aux besoins de l'utilisateur. Pour ce faire, nous avons préalablement défini les connaissances sur le fonctionnement du progiciel (chapitre 5), et nous avons exposé le fonctionnement du module d'aide pour qu'il soit capable de justifier les choix proposés. Dans le noyau sémantique, tous les objets sont capables de déterminer si leur application est pertinente.

Parce que nous avons traité l'ensemble des objets du module d'aide de manière à ce qu'ils puissent indiquer quand et pourquoi leur application est une bonne suggestion, nous ne devons pas être surpris de retrouver pour les objets Advice un fonctionnement similaire. Les conseils sont aussi modélisés par des objets réactifs, c'est-à-dire qu'ils identifient l'intérêt d'être

ou non appliqués en fonction du contexte, et en fonction des objets du module d'aide qu'ils impliquent. En conséquence, le noyau sémantique est préalable à la mise en place des conseils. Par exemple, pour conseiller l'ajout de conductance dans le circuit, il faut évaluer son intérêt en fonction de la présence ou non de jonctions dans le circuit, et plus simplement en vérifiant si le paramètre qui contrôle la conductance est effectivement candidat pour une modification.

Les règles réflexes des objets Advice sont les trois règles de base présentées au chapitre 6 pour l'ensemble des objets du module d'aide : Is-relevant, Is-influent et Is-redondant. Les deux premières règles permettent d'identifier, parmi les conseils potentiels, ceux qui sont candidats et ceux qui sont influents. Par ailleurs, la règle Is-redondant permet de vérifier que la proposition du conseil n'a pas déjà été faite, ni appliquée spontanément par l'utilisateur. Dans ces deux cas, le conseil ne devra pas être à nouveau suggéré, et en particulier quand son application n'a pas résolu la situation d'échec. Dans la mesure où les résultats ne peuvent pas être anticipés, le module d'aide doit suivre l'utilisateur dans sa démarche d'essais et erreurs, pour éviter les mêmes suggestions. La règle exploite donc l'historique des commandes et des conseils de la phase en cours.

Règle Is_relevant :

« La directive Relax est pertinente pour une analyse temporelle quand le circuit est constitué de transistors Mos, et qu'il vérifie le critère d'ordonnancement de ces derniers. De plus le circuit doit exclusivement contenir des composants numériques, ou les composants analogiques suivants : Mos, capacités, source de tension indépendante et reliée à la masse, source de courant, modules comportementaux dont l'impédance de sortie est spécifiée »

Règle Is_influent :

« Si le circuit vérifie les conditions d'application de l'algorithme de relaxation, il est probable qu'il permette une simulation temporelle avec un excellent ratio précision par rapport au temps »

Règle Is_redondant :

« L'algorithme de relaxation a été appliqué sans succès »

Figure 7-3 : Exemple de règles réflexes pour la commande d'une analyse temporelle avec un algorithme de relaxation qui est une option peu utilisée, bien qu'elle soit très efficace dans certaines conditions.

Le paragraphe suivant montre comment les conseils s'insèrent dans le module d'aide.

7.1.3. L'exploitation des conseils dans le fonctionnement

Les conseils s'appuient sur le noyau sémantique du module d'aide et exploitent le contexte. Ils s'organisent autour de :

- l'échec signalé, qui est le résultat d'une analyse ;
- la phase mise ainsi en échec, qui regroupe l'ensemble de l'historique des conseils déjà tentés, des commandes, des modifications de paramètres, et des phases précédentes pour vérifier si cet échec a déjà été résolu au cours de la session ;
- la spécificité du processus de conception, qui détermine en partie les conditions dans lesquelles le conseil peut être envisagé.

Cette organisation autour des phases permet de connaître pour chacune d'elle les paramètres et les commandes, qui sont définis pour la contrôler dans le système d'aide. Ainsi dans une situation d'échec, le système peut indiquer ses connaissances générales si l'utilisateur

désire vérifier qu'une commande est ou non envisagée pour résoudre le problème. L'organisation et l'exploitation des objets Advice sont très proches de celles des objets Commande : comme ces derniers, ils sont connus de l'objet Phase susceptibles de les utiliser. De la même façon, l'objet Phase exploite une liste de conseils potentiels, une liste de conseils candidats et un historique des conseils déjà appliqués. De cette manière, l'utilisateur peut connaître :

- l'ensemble des conseils que le système peut consulter sur une situation d'échec ;
- l'ensemble des conseils qui se sont déclarés candidats, c'est-à-dire qui ne sont pas redondants, et qui peuvent être déclenchés ;
- l'ensemble des conseils qui se déclarent déjà appliqués, sans qu'ils aient résolu la situation d'échec.

Pour donner à l'utilisateur les informations sur les conseils connus et leur positionnement par rapport au contexte, les objets Phase doivent les manipuler. Les phases connaissent, au niveau méta, l'ensemble des objets Advice susceptibles de conseiller une situation d'échec, obtenue par une commande de cette phase. Quand une situation d'échec est avérée, la phase sélectionne l'ensemble des conseils qui répondent au type d'échec courant, puisque des erreurs différentes peuvent survenir dans une même phase. La phase adresse ensuite un message pour que chaque objet se positionne par rapport au contexte. L'utilisateur peut ensuite consulter ces différents objets comme il le faisait dans la maquette illustrée par la Figure 7-4 suivante.

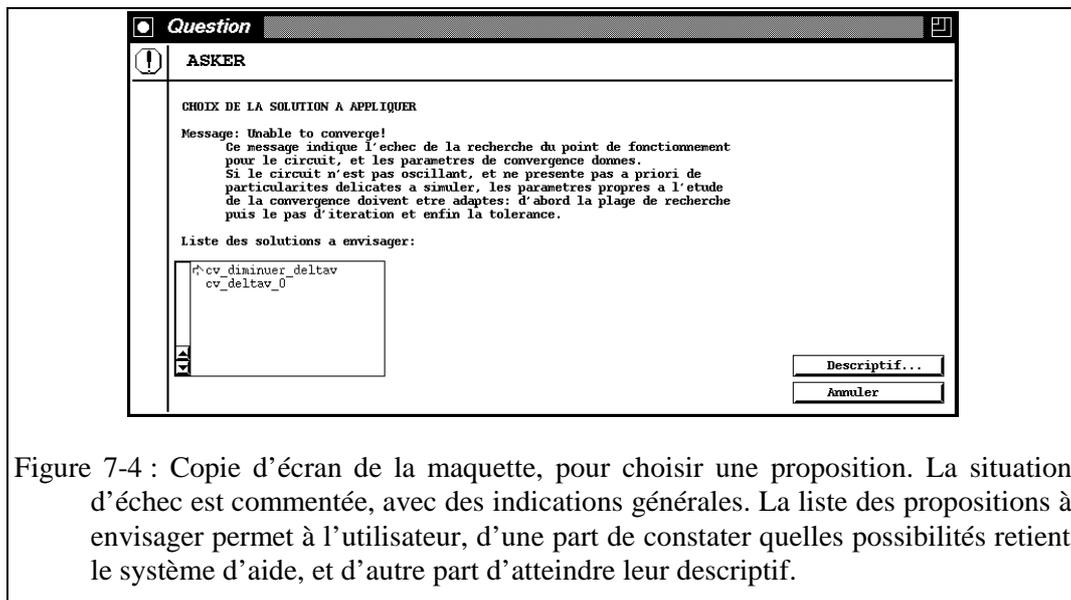
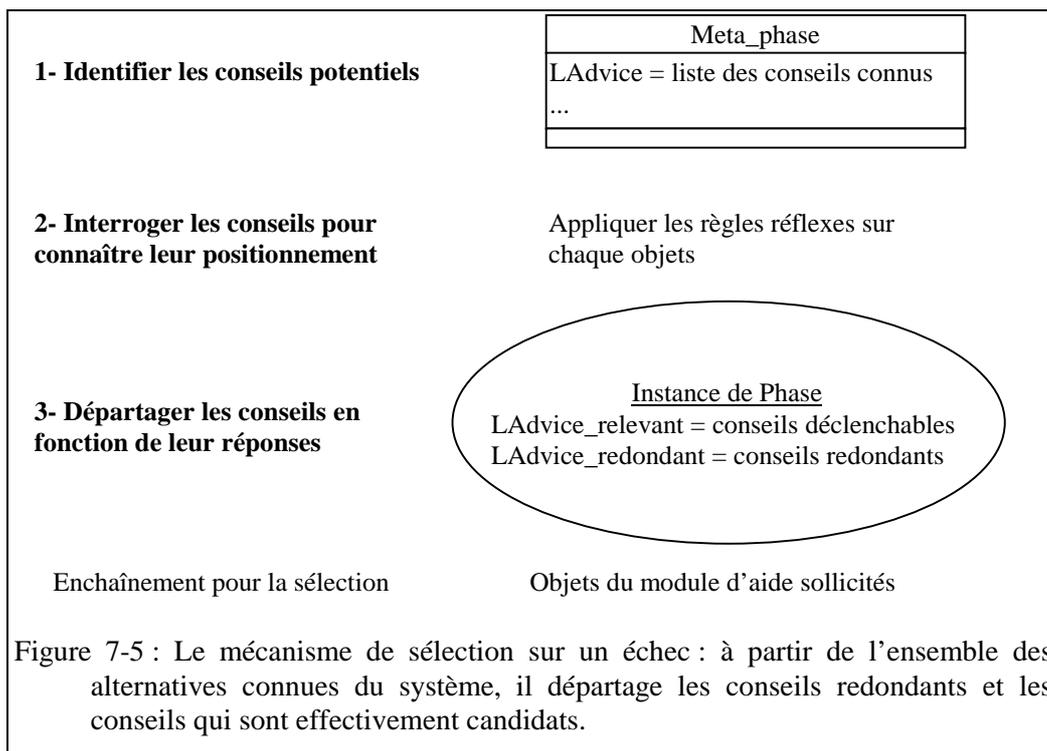


Figure 7-4 : Copie d'écran de la maquette, pour choisir une proposition. La situation d'échec est commentée, avec des indications générales. La liste des propositions à envisager permet à l'utilisateur, d'une part de constater quelles possibilités retient le système d'aide, et d'autre part d'atteindre leur descriptif.

Le module d'aide a la charge de pré-sélectionner les propositions candidates dans le contexte courant. La Figure 7-4 illustre les étapes du module d'aide pour identifier les conseils :

- Déterminer l'ensemble des conseils susceptibles de permettre de résoudre l'échec courant.
- Marquer les conseils qui ont déjà été essayés sans succès.
- Retenir les conseils qui sont candidates dans le contexte donné.



Pour l'utilisateur, les conseils sont obtenus par une commande explicite, qui donne dans un premier temps la liste des suggestions. L'utilisateur peut alors consulter chacune de ces suggestions avant de décider d'en appliquer une. Il pourra aussi choisir de mener seul la résolution de la situation d'échec, en utilisant lui-même les informations générales sur les conseils.

7.2. La mémorisation des expériences

La connaissance aujourd'hui formalisée en collaboration avec les différents experts, n'est pas exhaustive. Sans renoncer à formaliser et intégrer de nouvelles connaissances dans le module d'aide, il est par ailleurs nécessaire de tenir compte de l'expérience des concepteurs. Malheureusement ces derniers ne jugent pas opportun d'exporter leur savoir-faire, pour des raisons aussi variées que le temps nécessaire pour le formaliser, la confidentialité de son travail, les liens étroits avec la spécificité de leur conception, ou tout simplement faute de retombée positive pour eux-mêmes. Si faire part de son expérience à l'équipe de développement de l'outil, distante et inconnue, n'est pas spontané, le partage des connaissances avec ses seuls collègues de travail est plus attrayant. Il y a plus de chance pour que les concepteurs travaillent ensemble, ou sur des sujets de conception proches. De plus, la communication de cette expérience se fait en pratique par un tutorat, sans garantir qu'il couvre l'ensemble des problèmes déjà résolus par les plus anciens du métier. Sans prétendre devenir une mémoire d'entreprise, le module d'aide a l'ambition d'apporter une mémorisation des procédures de solution « au cas par cas ».

Un système hybride, où les expériences mémorisées par des cas complètent les règles, permet de préserver la cohérence d'une connaissance formalisée avec les experts sous forme de règles, en présentant une ouverture pour la mémorisation du savoir-faire. Les utilisateurs sont alors amenés à consulter et à ajouter des cas quand les heuristiques du système sont in-

suffisantes. Cette démarche renforce l'initiative de l'utilisateur qui juge seul de la pertinence des justifications de ses collègues vis-à-vis de son problème de conception. L'ajout de cas dans la base locale demande une importante initiative à l'utilisateur. Cet effort est le seul pseudo contrôle envisagé pour l'ajout de cas. Le système d'aide n'a en effet aucun moyen de valider les cas de la base. De plus, comme la base de cas doit pouvoir être enrichie sur les sites de conception, nous n'avons pas la possibilité de superviser l'évolution de la base, en tant que fournisseur du système d'aide.

La mémorisation des expériences présente deux intérêts majeurs pour répondre à l'introduction d'un savoir-faire dans le système d'aide :

- elle permet d'envisager un enrichissement du système sur un site client. Ainsi, le système d'aide peut mémoriser des propositions spécifiques au type de travail ;
- elle travaille sur une base de cas, qui se compose justement d'expériences. Ce point ouvre le système aux concepteurs, qui souhaitent mémoriser une situation, sans se soucier de l'identification formelle des informations qu'elle représente. C'est l'ensemble de la situation qui est représentative.

Par rapport au raisonnement à partir de cas (RàPC), nous exploitons l'organisation de la base de cas, l'ajout ou la mémorisation d'une nouvelle expérience et l'extraction des cas susceptibles de correspondre au problème en cours. Le but est d'extraire un ensemble de cas susceptibles de répondre aux caractéristiques du cas à résoudre : les deux étapes sont donc la mémorisation et la remémoration. Nous n'exploitons qu'une partie du cycle du RàPC, dont la particularité n'est pas tant la remémoration que l'adaptation des cas connus pour ajouter dans la base un nouveau cas adapté à la situation courante.

7.2.1. La définition d'une expérience

La base de cas envisagée doit compléter les conseils. La structure aujourd'hui retenue pour un cas est équivalente à une instance d'objet Advice. Cependant, étant dépourvu des mécanismes d'un objet réactif, le cas mémorise l'ensemble des informations relatives à son activation au niveau de ses étiquettes. Le cas comprend :

- la classe de phase pour laquelle il est mémorisé,
- la commande qui a mené à la situation d'échec,
- le type d'échec,
- les autres conditions d'activation du cas, qui sont des spécificités par rapport aux conseils. Ces spécificités portent par exemple sur le type de méthode (méthode directe ou relax), sur des caractéristiques du processus (circuit purement analogique, numérique ou mixte),
- les conditions d'activation (textuelle).
- les particularités du processus, quand ces dernières sont déterminantes (textuelles). Un exemple sera la présence de composants spécifiques, comme un quartz avec une valeur élevée du facteur de qualité.
- la liste des paramètres modifiés, avec leurs valeurs initiales et finales,
- le résultat de l'application de la commande de validation de la proposition.

Le corps du cas est donc divisé entre la description de l'état du monde pour lequel il a été appliqué, des explications textuelles sur les conditions d'activation et enfin la proposition,

avec le résultat de son évaluation. Aujourd'hui les cas sont principalement envisagés pour apporter de nouvelles propositions positives, c'est-à-dire des conseils qui permettent de résoudre un problème.

7.2.2. Comment les expériences complètent les heuristiques

Pour chaque type d'échec correspond un ensemble de conseils, et un ensemble d'expériences. Ces expériences sont aujourd'hui prévues comme un recours quand les connaissances générales ne permettent pas de résoudre une situation d'échec. Dans la mesure où la connaissance sur le « comment adapter une expérience existante au problème courant » n'est pas formalisée, nous prévoyons dans un premier temps de proposer pour chaque problème une base de cas « à plat », c'est-à-dire non structurée, afin d'observer comment l'utilisateur l'exploite.

L'organisation des expériences est donc calquée sur celles des conseils. Par la suite, si l'usage des cas est intensif et conduit à une base importante pour certains types de problèmes, il sera toujours temps d'identifier des caractéristiques pour structurer pour chaque problème la base plus finement.

La structure de la base des cas s'inspire de l'expérience de la formalisation de l'expertise sous forme d'heuristiques. Dans le système d'aide, les cas sont destinés à combler les lacunes des conseils formalisés sous forme d'heuristiques. L'organisation de la base est prépondérante pour obtenir une mémorisation efficace. C'est elle qui détermine les différents niveaux à partir desquels la sélection des cas peut intervenir. En fonction du niveau choisi par l'utilisateur, les cas correspondants de la hiérarchie seront mis à sa disposition. Ainsi, suivant la confiance de l'utilisateur dans l'organisation de sa base, il pourra être très précis et consulter les cas en bas de sa hiérarchie, ou au contraire interroger un niveau plus général pour englober plus de cas.

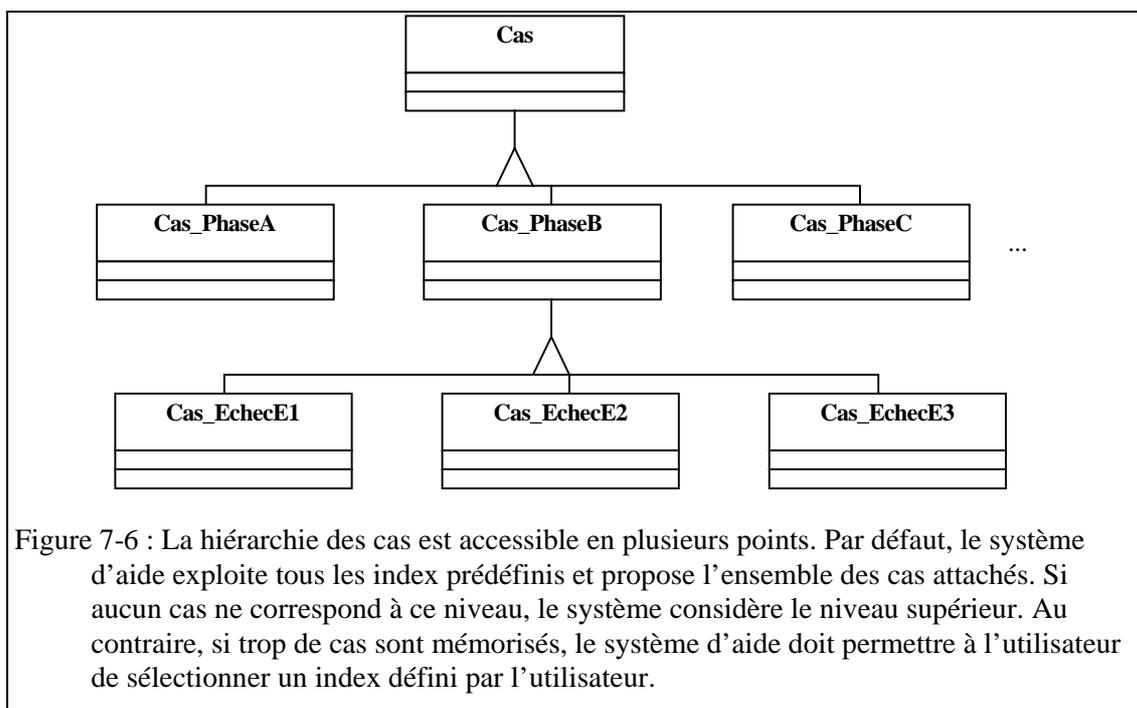


Figure 7-6 : La hiérarchie des cas est accessible en plusieurs points. Par défaut, le système d'aide exploite tous les index prédéfinis et propose l'ensemble des cas attachés. Si aucun cas ne correspond à ce niveau, le système considère le niveau supérieur. Au contraire, si trop de cas sont mémorisés, le système d'aide doit permettre à l'utilisateur de sélectionner un index défini par l'utilisateur.

Le détour par les connaissances heuristiques a été nécessaire pour proposer une organisation des cas exploitables dans le fonctionnement de l'outil. Cette organisation s'inspire de celle des connaissances structurelles, qui privilégie l'état du fonctionnement (décrit par l'analyse en cours) par rapport aux spécificités du processus. Cette priorité conduit à une indexation des cas où l'analyse est un index plus discriminant que la spécificité du processus.

Afin de poursuivre l'implication de l'utilisateur dans la résolution d'une situation d'échec, ce chapitre expose comment peuvent être mémorisées les expériences des concepteurs. Le premier moyen qui sera mis en œuvre à court terme est pour le concepteur de transmettre à l'équipe de développement du module d'aide une connaissance formalisée. Par la suite, si l'utilisateur peut apporter sa connaissance pour compléter la connaissance du système d'aide, l'implication de l'utilisateur s'oriente vers un début de collaboration.

Chapitre 8

8. Conclusion

L'introduction dans le progiciel Smash™ d'un système d'aide a été initialement suggérée par des clients, concepteurs en micro-électronique. Leur proposition était de tenir compte du « contexte », pour donner seulement les informations pertinentes. L'intérêt de l'aide contextuelle était donc établie par les futurs bénéficiaires, rester à mettre en œuvre cette proposition...

La première étape consista à cerner toute la problématique. La demande des utilisateurs est de s'approprier un progiciel de conception déjà existant, sans toutefois être contraint dans leur emploi du progiciel. Ceci conduit à identifier d'une part ce qu'est une « bonne utilisation » du progiciel qu'il faut encourager, et d'autre part les options spécifiques au progiciel, qui ne sont pas pleinement exploitées. Le but est double :

- faciliter l'exploitation du progiciel,
- valoriser les nombreuses options qui démarquent le progiciel de ses concurrents.

Les étapes suivantes sont menées en étroite collaboration avec les utilisateurs. Elles ont permis de :

- définir l'interaction qu'ils souhaitaient avoir avec le système d'aide ;
- formaliser les connaissances sur l'utilisation du progiciel, et en particulier l'importance du schéma de phases, autour duquel elles s'expriment simplement ;
- valider et d'enrichir notre proposition de système d'aide, en le soumettant à la critique constructive des concepteurs.

Riches de ces connaissances, nous les avons organisées en distinguant les connaissances structurelles, qui forment la structure du module d'aide, des connaissances contextuelles qui permettent de déduire en fonction du contexte les informations contextuelles destinées à l'utilisateur. Cette organisation est appliquée à tous les objets qui constituent le système, de manière à leur donner une autonomie, pour « se définir » et justifier leur intérêt dans le contexte courant. De cette manière, l'utilisateur peut venir interroger chacun de ces objets pour obtenir une information contextuelle. Cette structure permet d'impliquer l'utilisateur dans sa recherche d'informations, en l'incitant à être critique vis-à-vis des justifications des conseils.

Le système d'aide n'a pas pour mission de se substituer à l'utilisateur à l'heure du choix. Dans la limite de sa connaissance, il a pour but de prévenir les conflits, et quand une situation d'échec est rencontrée, de proposer l'ensemble des alternatives à envisager pour la résoudre. De cette façon, l'utilisateur s'approprie petit à petit les capacités de l'outil, et il peut ensuite ne plus avoir recours au module d'aide.

La perspective immédiate est la poursuite de l'implémentation des conseils, déjà en cours de réalisation. La maquette a validé auprès des utilisateurs le fonctionnement de l'aide à la prise de décision.

L'étape suivante sera d'enrichir les connaissances du module d'aide, pour pallier les lacunes ou les désaccords. La solution envisagée est l'ajout d'une base de cas pour représenter une connaissance locale, sur laquelle il est impossible d'exercer une vérification. Bien que construite sur l'organisation du module d'aide, pour faciliter la coopération entre les deux types de connaissances, la base de cas ne perturbe pas les règles établies et validées avec les experts.

Les perspectives plus lointaines concernent la coopération et la négociation entre l'utilisateur et le système d'aide. En effet un outil de vérification, comme le simulateur, présente la particularité d'un arbitrage. Typiquement, dans une situation d'échec, seul le progiciel valide une proposition qui peut être proposée indifféremment par l'utilisateur ou par le module d'aide. Dans cette perspective, il peut être satisfaisant que l'utilisateur et le module d'aide négocient les suggestions, pour ensuite les soumettre au progiciel, qui arbitrera. La négociation apporte un plus par rapport à l'aide à la décision, si elle permet à l'utilisateur de modifier des contraintes qui guident le module d'aide. Un exemple sera de modifier les intervalles de valeurs raisonnables pour les paramètres, en accord avec le type de processus étudié. Aujourd'hui ces valeurs sont empiriquement fixées, et validées pour une utilisation en micro-électronique.

Une toute autre perspective concerne l'historique de la session. Abordée dans le chapitre 6, l'exploitation de la trace de l'ensemble de la session ouvre des champs d'analyses intéressants, sous réserve de pouvoir extraire des informations effectivement pertinentes. Cette traçabilité est d'autant plus attrayante qu'elle permettrait d'évaluer l'emploi des conseils et par la suite, la fréquence des recours faits à la base de cas. Elle apporterait une photographie précise des sites clients, et de leurs difficultés dans l'utilisation du progiciel.

9. Bibliographie

L'ensemble des références citées et commentées dans ce mémoire sont présentées dans l'ordre alphabétique.

- [Aamodt et al.94] A. Aamodt et E. Plaza, *Case-based reasoning : Fundamental issues, methodological variations, and system approaches*, AICOM, vol 7 (1), mars 1994.
- [Amergé et al.94] C. Amergé et O. Corby, *Acquisition de connaissances pour des explications en contexte d'évaluation: application à une tâche de conception*, ERGO-IA94, Eds. I.D.L.S, Biarritz, pp. 681-692, 26-28 octobre 1994.
- [Bastien et al.93] J. M C. Bastien et D. Scapin, *Ergonomic criteria for the evaluation of human computer interfaces*, Rapport technique INRIA n°156, juin 1993.
- [Bastien et al.94] J. M. C. Bastien et D. Scapin, *Evaluating a user interface with ergonomic criteria*, Rapport de recherche INRIA n°2326, août 1994.
- [Berthomé et al.95] A. Berthomé-Montoy et J-M Fouet, *Utilisation de méta-connaissances pour adapter l'aide à l'utilisateur*, INFORSID XIII Congrès, Grenoble, pp. 329-342, 30 mai-2 juin 1995.
- [Bounaas95] F. Bounaas, *Modèle dynamique et représentation de connaissance*, Thèse en informatique de l'Institut National Polytechnique de Grenoble, octobre 1995.
- [Cawsey92] A. Cawsey, *Explanation and interaction, the computer generation of explanatory dialogues*, ACL-MIT Press Series in Natural Language Processing, 1992.
-

-
- [Coutaz90] J. Coutaz, *Interface homme-ordinateur : conception et réalisation*, Dunod Informatique, 1990.
- [Coutaz et al.91] J. Coutaz, L. Bass, *Developing software for the user interface*, SEI Series in software engineering, 1991.
- [Descleves95] C. Descleves, *Smash3 User manuel*, manuel du simulateur Smash, diffusé par Dolphin Integration, Meylan, 1995.
- [Duprez95] K. Duprez, *Informations sur Smash*, document interne à Dolphin Integration, août 1995.
- [Duprez96] K. Duprez, *Aide interactive pour un outil de conception*, Explication'96, Sophia-Antipolis, Eds. Inria France, pp. 107-120, 19-21 juin 1996.
- [Fischer et al.91] G. Fischer, A. Lemke, T. Mataglio & A. Morch, *The role of critiquing in cooperative problem solving*, ACM Transaction on Information Systems, Vol 9, n°3, pp. 123-151, avril 1991.
- [Forslund95] G. Forslund, *Toward cooperative advice giving system*, IEEE Expert, vol10, n°4, pp. 56-62, août 1995.
- [Fouet94] J-M. Fouet, *Utilisation de méta-connaissances pour l'acquisition, la transformation et la découverte de connaissances*, INFORSID94, Aix-en-Provence, pp. 33-46, 17-24 mai 1994.
- [GENE97] Groupe GENE, *Transposer les principales fonctions d'un dialogue explicatif dans une interface graphique*, Actes des 6^{ème} journées nationales du PRC-GDR Intelligence Artificielle 1997, Grenoble, Eds. Hermès, pp. 339-355, 1997.
- [Golding91] A.R. Golding et P.S. Rosenbloom, *Improving rule-based system through case-based reasoning*, AAAI91, Anaheim (Californie USA), vol 1, pp. 22-27, 14-19 juillet 1991.
- [Hammond87] K.J. Hammond, *Learning and reusing explanations*, International Workshop on Machine Learning, Irvine (Californie USA), pp. 141-147, 22-25 juin 1987.
- [Kodratoff88] Y. Kodratoff, *Introduction to Machine Learning*, Pitman publishing, 1988.
- [Kolski93] C. Kolski, *Ingénierie des interfaces homme-machine, conception et évaluation*, Traité des nouvelles technologies, série automatique, Eds Hermes, 1993.
- [Kolodner93] J. Kolodner, *Case-Base Reasoning*, Eds. Morgan Kaufmann Publishers Inc 1993.
-

-
- [Kriegsman et al.93] M. Kriegsman et R.Barletta, *Building a case-based help desk application*, IEEE Expert, Vol 8 n°6, pp. 18-26, Décembre 1993.
- [Kundert95] K. S. Kundert, *The design's guide to Spice and Spectre*, Kluwer Academic Publishers, 1995.
- [Landesman97] N. Landesman, *Les utilisateurs ont-ils voix au chapitre ?*, Science et vie micro, n°145, pp. 195-200, janvier 1997.
- [Leake91] D. Leake, *An indexing vocabulary for case-based explanation*, AAAI91, Anaheim (Californie USA), vol 1, pp. 10-15, 14-19 juillet 1991.
- [Maïs88] C. Maïs, *Pour des systèmes d'aide à la réalisation de procédures sous-optimales*, ERGO-IA88, pp. 174-187, 1988.
- [Malek96] M. Malek, *Un modèle hybride de mémoire pour le raisonnement à partir de cas*, Thèse de doctorat en informatique de l'Université Joseph Fourier de Grenoble, octobre 1996.
- [Masini90] G. Masini, A. Napoli, D. Colnet, D. Léonard, K. Tombre, *Les langages à objets, langages de classes, langages de frames, langages d'acteurs*, InterEditions, 1990.
- [Mc Calla et al.92] G. Mc Calla, J E. Greer, *Two and one-half approaches to helping novices learn recursion*, Proc. of the Workshop on cognitive models and intelligent environment for learning programming, Gênes Italie, 1992.
- [Mettrey91] W. Mettrey, *A comparative evaluation of expert system tools*, IEEE Computer , Vol 24, n°2, pp.19-31, février 1991.
- [Mignot92] H. Mignot, *Etude d'une mesure de similarité pour le raisonnement par cas*, DEA université de Paris-Sud, 1992.
- [Mitchell et al.85] T. M Mitchell, S. Mahadevan, L. I. Steinberg, *LEAP: a learning apprentice for VLSI design*, IJCAI 85, pp. 573-580, 1985.
- [Muller97] P-A. Muller, *Modélisation objet avec UML*, Eds Eyrolles, 1997.
- [Nguyen-Xuan et al.95] A.Nguyen-Xuan & J-F. 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 n°2, pp. 145-171, 1995.
- [Norman86] D.A. Norman, *Cognitive engineering, User centered system design: new perspectives in computer interaction*, Eds : D.A. Norman, S.W. Draper, Hillsdale, New Jersey, Lawrence Erlbaum Associates, pp. 31-61, 1986.
-

-
- [Pitrat90] J. Pitrat, *Métaconnaissance, futur de l'Intelligence Artificielle*, Hermès, 1990.
- [Pitrat96] J. Pitrat, *Méta-expliquer pour apprendre une stratégie*, Explication'96, Sophia Antipolis, Eds. Inria, pp. 123-131, 19-21 juin 1996.
- [Rasmussen86] J. Rasmussen, *Information processing and human-machine interaction : an approach to cognitive engineering*, Eds : North-Holland series in system science and engineering, A.P. Sage, 1986.
- [Rewari93] Rewari, *AI for customer service and support*, IEEE Expert, Vol 8, n°6, pp. 5-8, décembre 1993.
- [Richard83] Richard, *Logique de fonctionnement et logique d'utilisation*, Rapport de recherche INRIA n°202, avril 1983.
- [Safar90] B. Safar, *Répondre à des questions de type Pourquoi-pas ?*, Revue d'intelligence artificielle, Vol 4 n°2, 1990.
- [Saleh et al.94] R. Saleh, S-J. Jou, A. R. Newton, *Mixed-mode simulation and analog multilevel simulation*, Kluwer Academic Publishers, 1994.
- [Scapin et al.89] D. L. Scapin, C. Pierret-Goldbreich, *Towards a method for task description : MAD*, Deuxième conférence scientifique internationale sur le travail à l'écran de visualisation, Montréal, septembre 1989.
- [Senach87] B. Senach, *Méta-communication, gestion de contexte et modélisation cognitive de l'utilisateur: trois perspectives pour améliorer l'intelligence des systèmes d'aide à l'utilisation*, Bulletin de liaison de la recherche en informatique et en automatique, n°115, pp. 22-31, 1987.
- [Shimazu et al.94] H. Shimazu, A. Shibata and K. Nihei, *Case-based retrieval interface adapted to customer-initiated dialogues in help desk operations*, AAI94, Seattle (Washington USA), vol 1, pp.513-518, 31 juillet-04 août 1994.
- [Silverman97] B. G. Silverman, *Computer reminders and alerts*, IEEE Computer Vol 30 n°1, pp. 42-49, janvier 1997.
- [Tarby94] J-C. Tarby, *Application de la logique d'utilisation à l'aide contextuelle*, ERGO-IA94, Eds. I.D.L.S, Biarritz, pp. 181-190, 26-28 octobre 1994.
- [Tourniaire et al.97] F. Tourniaire, R. Farrell, *The art of software support*, Eds. Prentice Hall PTR, 1997.
-

[Willamowski94]

J. Willamowski, *Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur*, Thèse en informatique de l'Université Joseph Fourier, Grenoble1, 1994.

A n n e x e A

10. Annexe A : Manuel Advisor

Le lecteur intéressé peut obtenir une version d'évaluation du progiciel Smash™, dans lequel est introduit le module d'aide, sous le nom d'Advisor.

Contact : Dolphin Integration, Chemin des clos BP 65, ZIRST, F-38242 Meylan

L'URL du site Dolphin est : <http://www.dolphin.fr>

Les pages suivantes sont extraites du manuel du progiciel. Elles sont consacrées à l'utilisation du module d'aide. Le manuel est actuellement uniquement disponible en anglais.

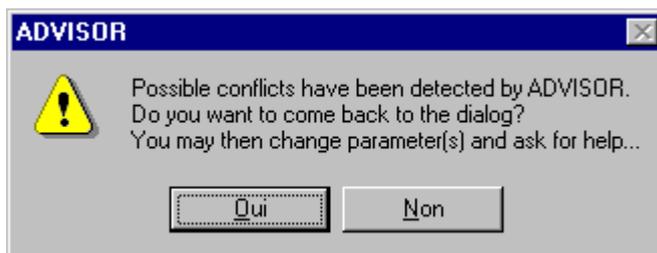
Advisor menu

This menu contains commands regarding Advisor, the contextual help system of Smash. Be aware that Advisor takes into account the specificities of the circuit you are working on to produce contextual advice. If no circuit is loaded, you still can access help, but do not be surprised if some parameters warn you that their usage requires a successful load.

At the time this manual is printed, Advisor is offered on PC platform only, and requires a 32 bits system (Window95 or NT) .

Turn Advisor on/off

This command allows you to turn Advisor « on » or « off ». By default, Advisor is « on » each time you run Smash, and thus the menu item is checked. Whenever Advisor is « on », it will pop up a small warning box (see figure below) when a conflict is detected among the parameters of a dialog. This behavior will help you to enter a correct set of parameters, by pointing out any relevant problems or inconsistencies. If you want to avoid these messages, you can turn Advisor « off »: the menu is then unchecked, and Advisor becomes mute. Please note that at any time, you can turn Advisor « on » or « off ».



Advisor (if turned « on ») suggests you to solve a detected conflict.

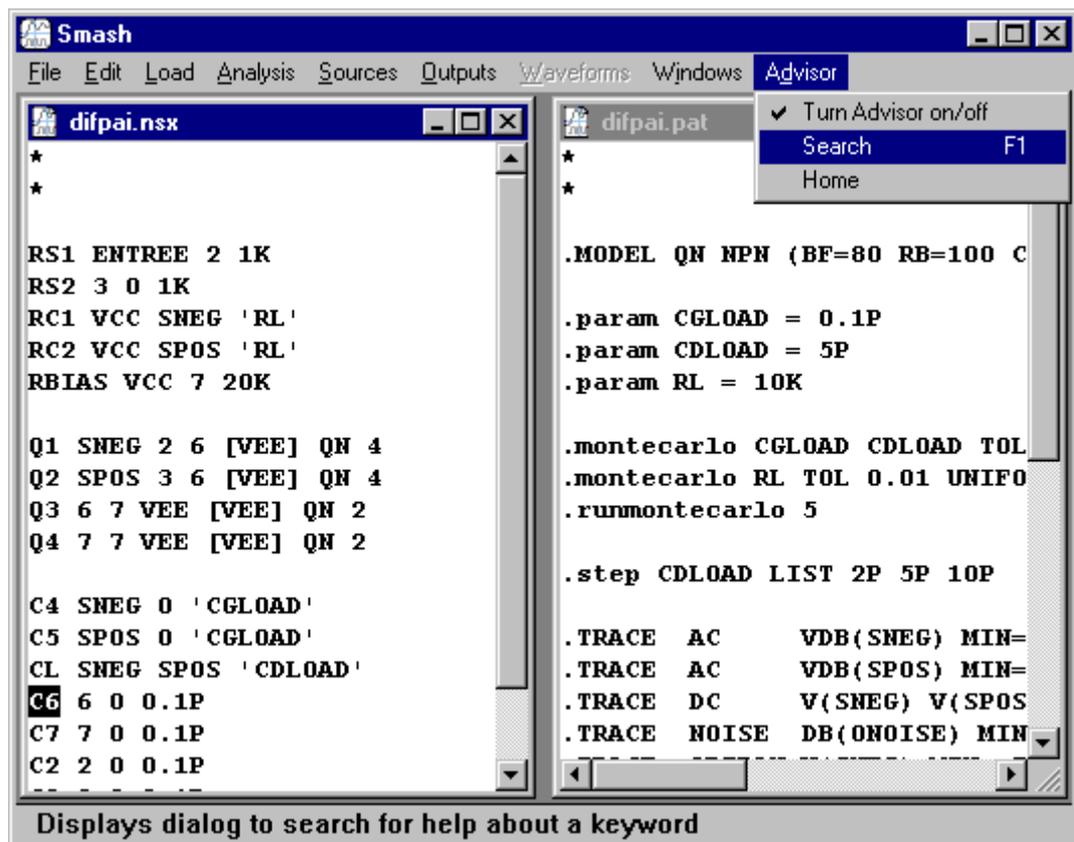
When this warning occurs, you can either choose the default Yes selection, to come back in the parameter dialog and check the reason of the conflict. Then if needed, you can solve the conflict before proceeding. If you wish to carry on despite the warning, just click on the No button.

Search

This command activates a basic search tool to help you find Advisor comments about a selected topic. This command will activate a dialog box with a string editor, and the list of keywords handled by Advisor. If the string is not among known keywords, you will have to scroll through the list in order to find an entry as close as possible.

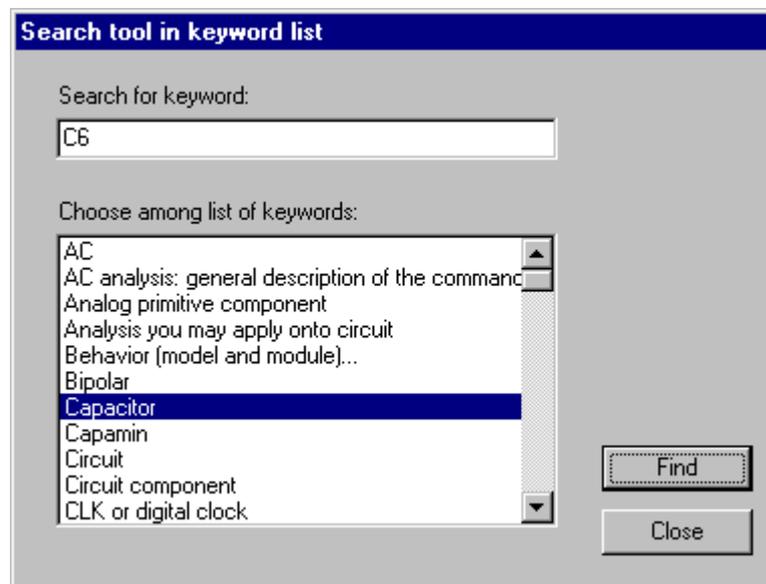
Please note that the string editor is empty unless you previously selected a string in a text file, such as the netlist (.nsx) or pattern (.pat) file. Then this selected string is matched with the keyword list, and if it is a known keyword, you immediately get Advisor comments. General information about the keyword, syntax and values (if any) are then displayed.

The search menu can be called via the menu Advisor Search, or with the F1 keyboard accelerator.



Displays dialog to search for help about a keyword

Search tool activation with C6 (in netlist) previously selected



As C6 is not a keyword, Advisor chose the closest known keyword.

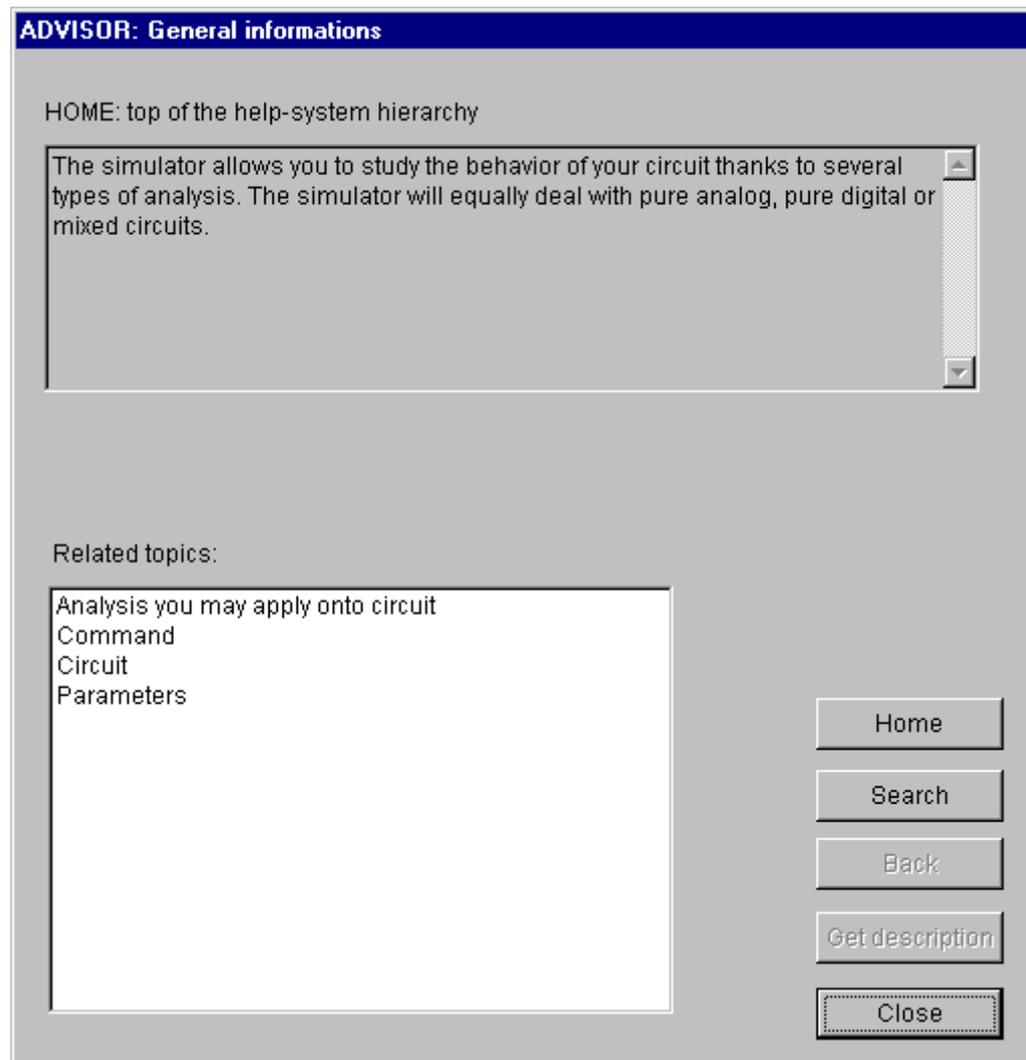
Home

This command activates a dialog box which opens the top-level of the explanation hierarchy. From this point, you can easily move to a related topic by double-clicking on a list

element, or use the Get Description button once you selected an item in the list box. This menu command is accessible even if Advisor is « off » (see Turn Advisor on/off).

Advisor is a contextual interactive system : it keeps track of the modifications you make in the dialogs, but it cannot keep track of the manual modifications in the pattern file.

From the Home dialog, the search tool can be called thanks to the « Search » button.

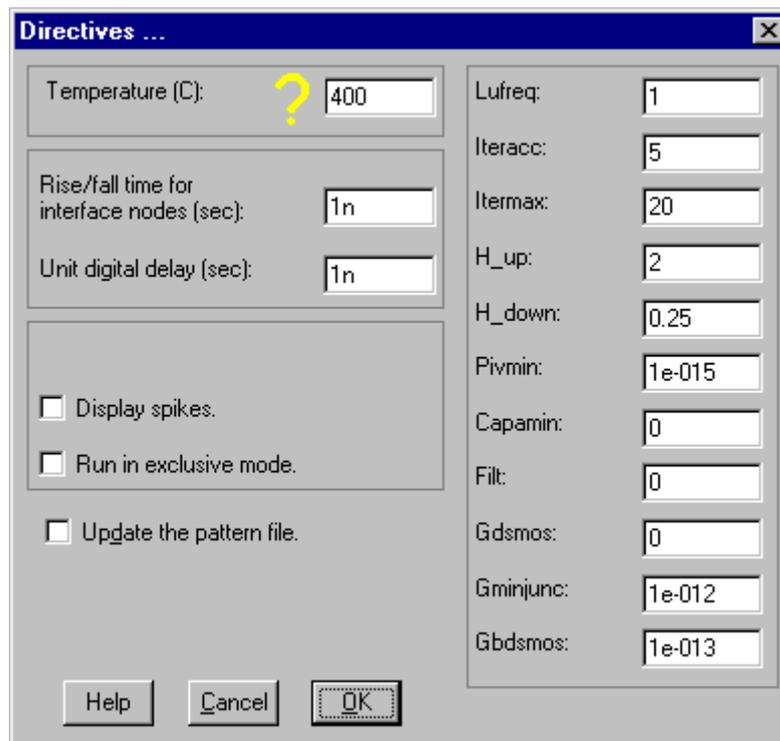


Home is the top of the hierarchy of explanation

Advisor access

The contextual help system, Advisor, can be reached through several ways. One way is to enter via the top of explanation hierarchy (Home), thanks to the Advisor Home menu (see Home section). However, if you are looking for a special item, it is more convenient to use the search tool (see Search section).

Last but not least, access is provided in a Smash dialog through an « Help » button. This access is very useful when a conflict is detected regarding a dialog parameter. If you click « Help » at this time, the faulty parameter is automatically selected in Advisor.



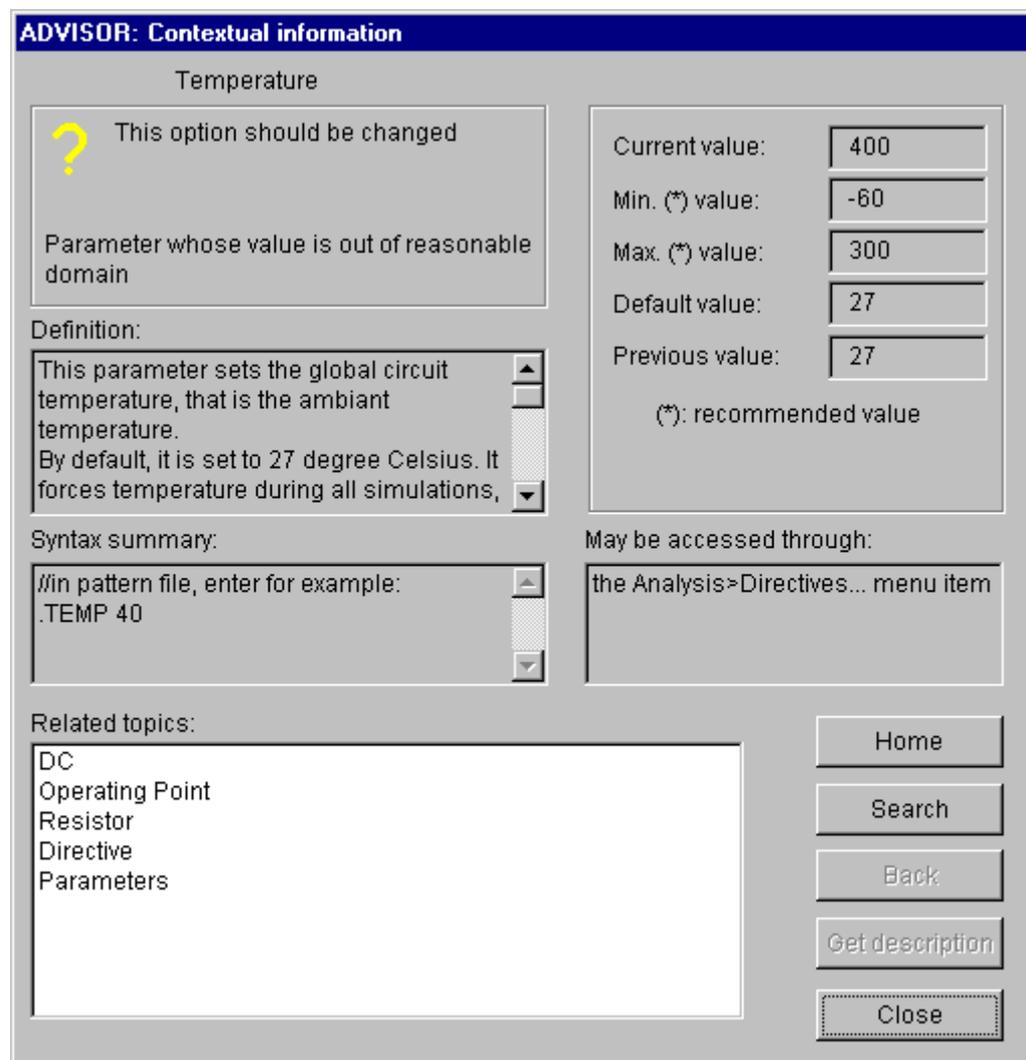
Access via dialog with the « Help » button. Useful when conflict is detected.

When Advisor is called from a Smash parameter dialog including an « Help » button, this dialog is moved, so that you can see both the parameter dialog, and Advisor comment dialogs. Notice that Advisor dialog is modal, thus it is the only one which can be moved once displayed. Any other action outside the dialog will be ignored.

Navigating in Advisor

Once you enter Advisor, you may wish to look for more information (indeed you should !) about related topics. To do so, all Advisor dialogs have a direct access to the top of explanation hierarchy with the button « Home ». A « Back » button displays the previous comments. The « Get description » button allow you to look for the item selected in the « Related topics » list. A double-click in the list has the same effect, and it will automatically display the corresponding comments.

Also, an access to the search tool with the button « Search » is provided. This button is useful to check the definition of a concept. Please note that a search call will erase the historic of Advisor comments.



Information on temperature when a conflict is detected.

When a conflict is detected on a parameter, a question mark is displayed in the Advisor dialog with a laconic message like «...parameter is out of reasonable domain ». If you want to get more information, have a look at the definition, by clicking on the «Display definition...» button. For a numeric parameter, some informations are displayed regarding its possible values : the current value can easily be compared with the minimum and maximum

values recommended by Advisor : it can give you some clue. Default and previous values of this parameter are also displayed.

When looking for parameter help, you will also find the syntax description and how to access to this parameter, thanks to the «Display access...» button.

The Close button will let you leave the Advisor help system, and return to your simulation.

Other hints

You can also have a look at the Appendixes about error and warning messages, to have a short explanation about an error or a warning, and also the chapter to consult for more details.

An other useful appendix is the CookBook (in Appendix too), which provides many hints to solve simulation problems.

Annexe B

11. Annexe B : Formulaire pour l'évaluation du prototype

Formulaire distribué pour l'évaluation de la version beta de Advisor dans la version 3.3 du progiciel Smash.

Evaluation de ADVISOR

En vous remerciant de remplir ce questionnaire après votre évaluation de l'option Advisor, système d'aide intégré au simulateur Smash™.

11.1. Votre profil

Nom :

Prénom :

Société ou Institut :

Période d'évaluation : du .../.../... au .../.../...

Date de réponse au questionnaire : le .../.../...

11.1.1. L'évaluation de votre expérience en simulation de circuits électriques

Quelle est votre expérience en simulation avec Smash™, avant l'évaluation de cette version 3.3 :

- Aucune expérience
- Moins de 10h consacrées à la mise au point(*) des paramètres.....

- Plus de 10h consacrées à la mise au point (*)des paramètres.....

(*) temps consacré à la détermination des paramètres, en dehors du temps de simulation

Avez-vous déjà utilisé d'autre(s) simulateur(s): OUI NON

Si oui, lesquels?

- SPECTRE..... Verilog XL.....
- ANALOGY..... VSC.....
- PSPICE..... VeriBest.....
- HSPICE.....
- Autre :

11.1.2. Vos types de circuits simulés avec SMASH™ :

- purement analogiques.....
- purement logiques
- mixtes
-

Particularités des circuits (HF, électronique de puissance ...) :

.....

11.2. Vos commentaires :

Merci d'argumenter vos critiques et requêtes. Vous pouvez de plus illustrer vos commentaires en détaillant ce que vous souhaiteriez voir dans ADVISOR, dans la partie C de ce questionnaire.

11.2.1. La facilité d'utilisation d'ADVISOR

L'option ADVISOR vous semble-t-elle bien mise en évidence? OUI NON

L'accès à l'aide est-il aisé via le menu? OUI NON

L'accès à l'aide est-il aisé via les dialogues des paramètres? OUI NON

Quelles autres facilités d'accès souhaiteriez vous? :

11.2.2. Les caractéristiques d'ADVISOR

ADVISOR offre-t-il une réponse à votre besoin d'aide en ligne? OUI NON

Les descriptions textuelles sont-elles suffisantes? OUI NON

Les liens entre les paramètres vous aident-ils? OUI NON

Les descriptions contextuelles sont-elles pertinentes? OUI NON

Avez-vous désactivé ADVISOR (turn off)? OUI NON

Si oui : rarement souvent toujours

est-ce par curiosité?.....

est-ce pour désactiver l'aide en raison de votre maîtrise de l'outil ?.....

Cette option de désactivation vous semble-t-elle indispensable? OUI NON

Pour quelle raison ?.....

.....

11.2.3. L'utilisation d'ADVISOR

Dans quel cas avez-vous fait appel à ADVISOR:

pour répondre à un problème précis?.....

pour découvrir les possibilités du simulateur SMASH?.....

Dans quel cas ADVISOR vous a été utile:

pour répondre à un problème précis(*)?.....

pour découvrir les possibilités du simulateur SMASH?.....

* Merci de fournir l'exemple détaillé pour les cas d'insatisfaction.

Avez-vous été surpris par les interventions d'ADVISOR? OUI NON

Si oui,

la première fois seulement.....

lors du lancement d'analyse (RUN dans un dialogue).....

à chaque intervention d'ADVISOR.....

Ces interventions vous ont-elles sensibilisé à un problème pertinent?

toujours pertinent.....

souvent pertinent.....

rarement pertinent.....

jamais pertinent.....

11.2.4. Ce qui vous a manqué dans la recherche d'une explication

- Les solutions possibles en situation d'échec de simulation.....
- Les explications sur les paramètres des modèle de composant (MOS, bipolaires...).....
- L'absence des paramètres pour les analyses AC, DC.....
- L'absence des explications pour les noeuds d'interface en simulation mixte.....

-
- Autre:
-
-

11.2.5. Ce que vous avez appris par l'utilisation d'ADVISOR

- Les liens entre les paramètres (ex: MAXITER et DELTAV) OUI NON
- Les options pour la convergence (ex: OPHELP pour l'analyse OP) OUI NON
- Les options de simulation (PUREANALOG, SMALLWINDOW) OUI NON
-
- Autre:
-
-

11.3. Vos suggestions vis-à-vis du système d'aide ADVISOR :

(Présentation de ADVISOR, exemple où l'aide est insuffisante, exemple d'aide souhaitée mais inexistante...)

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....