POLYPUBLIE
Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Titre: Une Approche pour le développement et la maintenance de systèmes informatiques complexes
Title:

Auteurs: Richard St-Denis, & Pierre N. Robillard
Authors:

Date: 1986

Type: Rapport / Report

Référence: St-Denis, R., & Robillard, P. N. (1986). Une Approche pour le développement et la maintenance de systèmes informatiques complexes. (Rapport technique n° EPM-RT-86-31). https://publications.polymtl.ca/10159/
Citation:

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: https://publications.polymtl.ca/10159/
PolyPublie URL:

Version: Version officielle de l'éditeur / Published version

Conditions d'utilisation: Tous droits réservés / All rights reserved
Terms of Use:

Document publié chez l'éditeur officiel
Document issued by the official publisher

Institution: École Polytechnique de Montréal

Numéro de rapport: EPM-RT-86-31
Report number:

URL officiel:
Official URL:

Mention légale:
Legal notice:

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal

https://publications.polymtl.ca

0 8 JAN. 1987

EPM/RT-86/31

UNE APPROCHE POUR LE DEVELOPPEMENT ET LA MAINTENANCE DE SYSTEMES INFORMATIQUES COMPLEXES

Richard St-Denis, étudiant, Ph.D.
Pierre -N. Robillard, Directeur de thèse

Département de Génie Électrique

École Polytechnique de Montréal
Août 1986

gratuit

Tous droits réservés. On ne peut reproduire ni diffuser aucune partie du présent ouvrage, sous quelque forme que ce soit, sans avoir obtenu au préalable l'autorisation écrite de l'auteur.

Dépôt légal, 2e trimestre 1984
Bibliothèque nationale du Québec
Bibliothèque nationale du Canada

Pour se procurer une copie de ce document, s'adresser au:

Éditions de l'École Polytechnique de Montréal
École Polytechnique de Montréal
Case postale 6079, Succursale A
Montréal (Québec) H3C 3A7
(514) 340-4000

Compter 0,10$ par page (arrondir au dollar le plus près) et ajouter 3,00$ (Canada) pour la couverture, les frais de poste et la manutention. Régler en dollars canadiens par chèque ou mandat-poste au nom de l'École Polytechnique de Montréal. Nous n'honorerons que les commandes accompagnées d'un paiement, sauf s'il y a eu entente préalable dans le cas d'établissements d'enseignement, de sociétés ou d'organismes canadiens.

PREMIERE PARTIE

UNE APPROCHE POUR LE DEVELOPPEMENT ET LA MAINTENANCE DE SYSTEMES INFORMATIQUES COMPLEXES

SOMMAIRE

Le génie logiciel s'intéresse à la production et à la maintenance de systèmes informatiques de haute qualité, livrés dans les délais prescrits et aux coûts estimés. Pour atteindre cet objectif fondamental, les outils logiciels offrent des éléments de solution intéressants, car ils complètent les méthodes et les langages en automatisant différentes tâches du processus de développement des systèmes informatiques. Dans ce rapport, nous prêtons notre attention aux outils logiciels de conception de systèmes en présentant de façon systématique les différents choix qui s'offrent aux spécialistes de l'informatique pour l'acquisition ou la création de tels outils. Après avoir identifié deux approches possibles de construction d'outils logiciels, nous explorons les concepts et nous dégageons les problèmes qui s'y rattachent. Certains éléments introduits dans ce rapport ne sont pas spécifiques à l'étape de conception et peuvent être généralisés aux autres étapes du cycle de vie, en particulier l'étape de spécification formelle du système.

TABLE DES MATIERES

1. Introduction

La production d'outils logiciels capables de fournir une assistance aux concepteurs de systèmes informatiques connaît depuis ces dernières années une croissance rapide. Il existe aujourd'hui plusieurs centaines d'outils logiciels. Certains d'entre eux sont disponibles commercialement, d'autres plus expérimentaux n'ont pas encore été transférés des institutions de développement et de recherche vers les sociétés commerciales et industrielles. Fonctionnant seuls ou intégrés à un environnement de génie logiciel, ces outils visent surtout une amélioration de la qualité du logiciel et un accroissement de la productivité des équipes de développement.

A partir du moment où l'on s'est aperçu que les concepteurs ne disposaient pas d'outils adéquats pour effectuer convenablement leur tâche, les premières études du génie logiciel ont conduit à l'élaboration et à la proposition de méthodes et de langages de conception de systèmes informatiques. Ce premier ensemble d'outils techniques, largement accepté et répandu, connaît encore aujourd'hui de nombreuses améliorations et additions. Toutefois, il faut bien reconnaître que l'application manuelle des méthodes et des langages de conception constitue un obstacle sérieux à un accroissement significatif de la productivité des concepteurs dans un environnement réel de développement. Rapidement, les informaticiens ont décidé d'utiliser l'ordinateur à leurs propres fins. Ils ont greffé autour de ces outils techniques toute une gamme d'outils logiciels capables d'éditer, de conserver, d'analyser et de documenter un ensemble volumineux d'éléments de données produits pendant l'étape de conception. Enfin,

depuis une quinzaine d'années, des groupes de chercheurs multidisciplinaires tentent de construire des outils capables d'effectuer automatiquement une partie des tâches créatives des concepteurs.

Il existe deux façons d'aborder le problème de conception, de réalisation et de mise en oeuvre d'outils logiciels de conception de systèmes informatiques:

-La première consiste à distinguer clairement le processus de conception et les produits qui résultent de l'étape de conception. Des outils logiciels, dits conventionnels, traitent les spécifications architecturales et détaillées du système produites par les concepteurs. Seuls ces derniers prennent des décisions, car ces outils ne disposent pas de données sémantiques quant à la façon dont les spécifications ont été générées. Ces outils enrichissent progressivement les méthodes et les langages de conception. Cette approche plus traditionnelle a donné lieu à de nombreux travaux de recherches théoriques et appliquées.

-La deuxième considère le processus de conception comme un produit au même titre que les spécifications [Balzer 85]. Il s'agit de comprendre et de formaliser le processus de conception pour en trouver une représentation et pour élaborer des méthodes de raisonnement appropriées. Des outils logiciels, dits à base de connaissances, guident, inspirent et parfois supplantent l'expertise des spécialistes du domaine. La logique, l'intelligence artificielle et la psychologie cognitive jouent un rôle prépondérant dans la réalisation de tels outils. Dans la mesure où à long terme elle pourra apporter des résultats intéressants, cette

nouvelle approche semble naturelle, car elle libère les concepteurs des détails techniques que leur imposent les méthodes, les langages et les outils conventionnels.

Nous ne tenterons pas de classifier, de répertorier ou de présenter les outils logiciels qui existent aujourd'hui. D'excellentes études ont fait le point sur l'état d'art dans ce domaine, qu'il s'agisse des outils logiciels conventionnels [Schindler 81; Howden 82; De Drouas 82; Vosbury 84; Hoffnagle 85] ou des outils logiciels à base de connaissances [Barr 82; Frenkel 85]. Avec un regard nouveau et à travers des recherches et des applications récentes, nous examinons la problématique liée à l'élaboration d'outils logiciels. Nous établissons des critères pour l'acquisition et la construction de tels outils. Enfin, nous dégageons quelques problèmes non encore résolus dans ce domaine.

Dans la deuxième section, nous présentons un modèle de conception de systèmes informatiques qui nous permet de mieux cerner les deux approches normalement présentées dans les publications scientifiques. Dans la troisième section, nous passons en revue les concepts de base qui interviennent dans un environnement de conception orienté vers l'utilisation d'outils logiciels conventionnels. Dans la quatrième section, nous présentons brièvement quelques outils logiciels à base de connaissances et nous examinons le rôle des techniques de l'intelligence artificielle dans le domaine de la conception des systèmes informatiques. Finalement, nous concluons notre étude en évoquant son utilité pratique et en avançant quelques suggestions de recherche.

2. La conception des systèmes informatiques

Avant d'aborder les concepts et les problèmes liés aux outils logiciels, nous rappelons le rôle de l'étape de conception dans le développement de systèmes informatiques. Nous présentons aussi un modèle de conception de systèmes informatiques qui nous permet de préciser les deux approches introduites dans la section précédente.

2.1 Le rôle de l'étape de conception

D'une façon générale, on représente le cycle de vie traditionnel comme la production contrôlée de descriptions successives du système qui convergent vers un produit livrable à l'utilisateur. Chaque description du système est appelée une spécification. La conception d'un système informatique constitue certainement l'étape critique de ce cycle, car elle influence fortement l'étape de maintenance qui représente une part importante du coût du système [Boehm 76]. Caractérisée par un processus de conception créatif, cette étape vise principalement à transformer la spécification formelle du système, elle-même obtenue de l'expression des besoins de l'utilisateur, en une spécification facilement traduisible dans un langage de programmation. L'étape de conception comporte deux activités principales:

- la conception architecturale pendant laquelle le concepteur décrit la structure du système informatique, aussi bien en termes des traitements qu'en termes des données, en précisant ses parties (modules et entités) et les jonctions entre les parties (interfaces et associations);

-la conception détaillée pendant laquelle le concepteur détaille chaque partie du système informatique en déterminant ses intrants, ses extrants, sa logique, ses algorithmes et ses structures de données.

Bien entendu, ces deux activités ne sont pas, dans la réalité, nécessairement séquentielles. Généralement, elles s'imbriquent et forment ensemble un réseau de sous-activités (noeuds) reliées par des mécanismes d'échange d'information (arêtes).

2.2 Un modèle de conception

Concevoir un système informatique représente une tâche complexe, qui pour être menée à bien, requiert des capacités intellectuelles, des mécanismes d'échange d'information et de bons outils techniques. Evidemment, exposé ainsi, le problème apparaît très général. Pourtant, le modèle que nous élaborons à partir de cette formulation, nous permettra d'analyser les artifices et la richesse des diverses orientations pour le développement d'outils logiciels.

Le modèle de conception illustré à la Figure 1 peut être schématisé par trois éléments de bases:

-le processus de conception;

-les entrées et les sorties effectives de l'étape de conception;

-les outils techniques de conception.

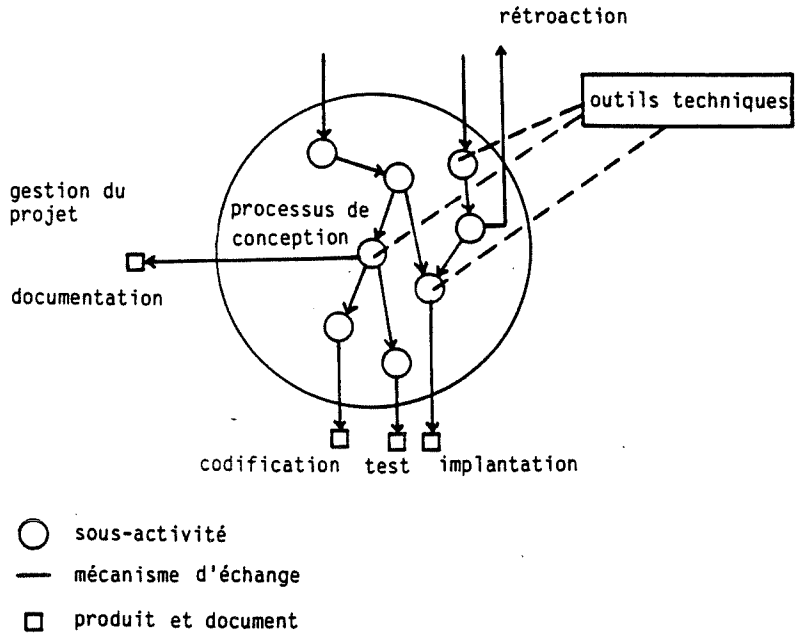

Figure 1 - Modèle de conception des systèmes informatiques

Le processus de conception est une activité humaine qui exige divers degrés de créativité. Dans chacune des sous-activités du réseau, les concepteurs doivent effectuer des choix, prendre des décisions, communiquer avec les autres équipes impliquées dans le projet, interpréter les spécifications des étapes précédentes, adapter leur travail en fonction de contraintes conflictuelles qui leur sont imposées, raisonner et appliquer un ensemble de règles pour obtenir finalement les résultats escomptés. Dans ce processus, la communication, qui s'établit grâce à des mécanismes d'échange d'information entre les sous-activités, tient une place importante. En effet, les

concepteurs consomment, produisent et échangent une très grande quantité d'information. Malheureusement, peu d'efforts ont été déployés pour proposer des techniques capables de capter toute la sémantique rattachée à l'information véhiculée durant l'étape de conception. Bien qu'elle pourrait être d'une très grande utilité pendant l'étape de maintenance, une grande partie de cette information est perdue ou devient obsolescente faute de moyens pour la conserver et la mettre à jour. Trois problèmes principaux peuvent être identifiés:

-décider du type d'information pertinente à conserver;

-trouver une représentation adéquate de cette information;

-rendre cette information disponible.

Les sous-activités du processus de conception, via les mécanismes d'échange, acceptent et fournissent un ensemble d'informations à caractère vertical et horizontal. Les informations à caractère vertical assurent un développement efficace et harmonieux du logiciel. Elles sont, par exemple, acheminées vers les étapes en aval (la spécification détaillée pour la codification, les spécifications pour les essais et l'implantation) et vers les étapes en amont (les rétroactions pour demander des ajustements ou des précisions). Les informations à caractère horizontal fournissent une aide constante à la gestion du projet ou permettent de documenter le système pour son utilisation, son opération et son entretien. Ainsi, les entrées et les sorties de l'étape de conception sont divisées en produits et en documents, et leur quantité varie en fonction du nombre de sous-activités et du guide de développement des systèmes dans lequel elles

s'inscrivent. De plus, des normes, généralement adoptées par l'entreprise, fixent leurs formats. La communication écrite joue un rôle primordial, car la communication orale, quoique largement utilisée, reste informelle et volatile.

Finalement, pendant la conception, les spécialistes ne sont pas laissés à eux-mêmes. Ils disposent d'outils techniques pour effectuer leur travail. Les outils techniques se répartissent en trois grandes classes: les outils conceptuels, les outils linguistiques et les outils logiciels.

2.3 Les environnements de conception

Dans l'optique d'un environnement orienté vers les outils logiciels conventionnels, le processus de conception est régi par des principes et des méthodes de travail empiriques qui ont été proposés à partir d'expériences vécues dans le développement du logiciel. Les arguments évoqués, justifiant l'efficacité et l'utilité de ces outils conceptuels, reposent principalement sur des intuitions acquises par les spécialistes du domaine durant de nombreuses années. Des outils linguistiques, tels que des diagrammes et des langages formels, servent de support pour l'expression des entrées et des sorties de l'étape de conception. Dans le cas où l'on ne peut formaliser l'expression de la communication, on a recours au langage naturel. Toutefois, contrairement aux diagrammes et aux langages formels, les langages naturels ne sont pas interprétables par les outils logiciels de cet environnement. Finalement, pour accélérer une partie du travail des spécialistes, cet environnement suggère l'utilisation

d'outils logiciels (conventionnels), tels que des éditeurs spécialisés, des générateurs de diagrammes et des analyseurs de spécifications, pour traiter les entrées et les sorties effectives de l'étape de conception.

Dans la perspective d'un environnement orienté vers les outils logiciels à base de connaissances, l'usage explicite d'un modèle sémantique du domaine d'application permet d'organiser le dialogue homme-machine, de comprendre les réponses de l'utilisateur, de traduire ces réponses en des actions appropriées et de fournir à l'utilisateur une explication sur les résultats générés. Le modèle sémantique décrit les classes d'objets, les événements, les relations entre les événements et les objets, les contraintes et les limites du domaine d'application. Les outils conceptuels regroupent des techniques de résolution de problèmes basées principalement sur les connaissances liées au domaine d'application et au processus de conception. Les langages formels occupent encore une place importante, mais on privilégie de plus en plus une interaction homme-machine en langage naturel. Les outils logiciels (à base de connaissances) empruntent des techniques de l'intelligence artificielle. Il s'agit le plus souvent de systèmes experts ou de systèmes de programmation automatique. Une étude théorique et une compréhension approfondie des sous-activités du processus de conception sont fondamentales pour la réalisation de cet environnement.

Ces deux environnements de conception de systèmes informatiques ne sont pas opposés. Ils peuvent être exploités conjointement pour bénéficier de tous leurs avantages. Ils ont été présentés ainsi pour

mettre en évidence deux approches fondamentalement différentes de développement d'outils logiciels. Les deux prochaines sections sont entièrement consacrées à l'étude détaillée de ces deux approches.

3. L'environnement de conception orienté vers les outils logiciels conventionnels

Dans cet environnement, les systèmes d'aide à la conception reposent sur trois concepts de base indissociables. La Figure 2, inspirée de Ludewig et al. [Ludewig 85], illustre une trilogie formée des classes d'outils conceptuels, linguistiques et logiciels. Ces classes sont interdépendantes et se complètent l'une l'autre. De plus, cette figure identifie quelques éléments appartenant à chacune de ces classes. Ce cadre simplifié permet de mieux distinguer les concepts à assimiler et de mieux cerner les problèmes inhérents à ce type d'environnement.

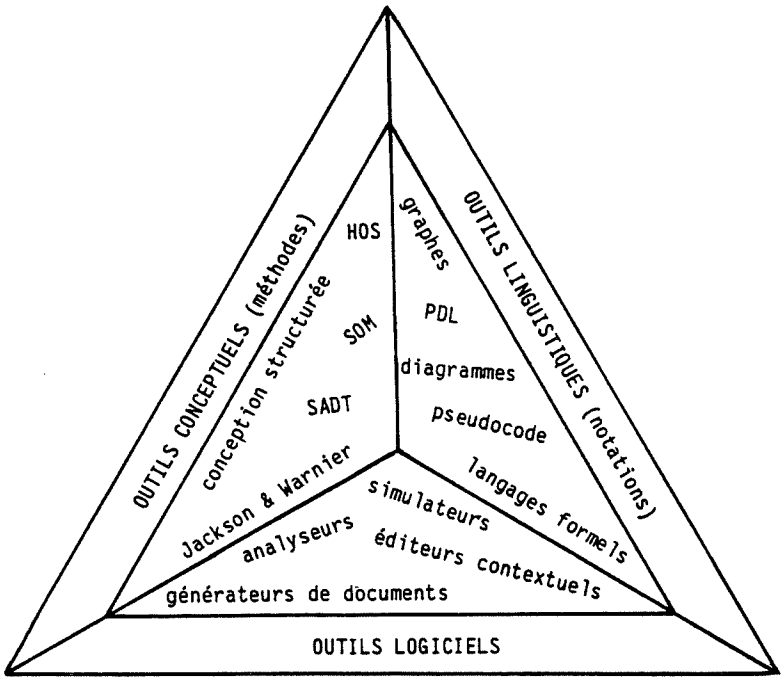

Figure 2 - Trilogie des concepts de base

Il est intéressant de noter que les outils logiciels constituent une partie de systèmes plus complexes appelés systèmes d'aide à la conception. Malheureusement, on confond souvent ces notions. Cette situation crée donc parfois des ambiguïtés au niveau de l'évaluation et de l'acquisition de systèmes d'aide à la conception. Par exemple, la rigidité des systèmes actuels force les utilisateurs à accepter d'emblée les méthodes, les notations et les outils logiciels qui leur sont associés et à renoncer à la possibilité d'intégrer de nouveaux outils pour une expérimentation ou pour des besoins spécifiques. En effet, pour un système d'aide à la conception donné, il existe généralement une seule instantiation de cette trilogie.

Une méthode est constituée d'un ensemble de règles et de principes sur lesquels reposent le développement des spécifications. Toute méthode est donc subordonnée à des principes élémentaires de conception de systèmes. La notation fournit un moyen de représenter les spécifications aussi bien à l'aide de textes qu'à l'aide de diagrammes. Finalement, les outils logiciels travaillent principalement sur les spécifications et guident les spécialistes dans un cheminement méthodologique. Ces derniers outils accélèrent le processus de conception, mais ils n'ont pas la possibilité de résoudre des problèmes. Le Tableau 1 donne quelques exemples de systèmes d'aide à la conception. Il précise pour chaque système les méthodes, les notations et les outils logiciels retenus par les auteurs.

Ainsi, pour l'environnement orienté vers les outils logiciels conventionnels, les chercheurs concentrent leurs efforts pour améliorer les outils existants ou pour trouver de meilleures méthodes, de

système d'aide la conception	méthodes	notations	outils
DASOM [Vefsnmo 85]	SOM (décomposition fonctionnelle, expression du comportement du système)	FS (diagramme de la structure fonctionnelle), ST (diagramme de transition d'états), SS (diagramme de la structure du logiciel), texte	éditeur (FS, ST,SS,texte), analyseur, générateur de documents, gestionnaire de projets
SARA/IDEAS [Krell 85]	SARA (décomposition et composition modulaire)	SM (diagramme de structure), GMB (graphe de comportement)	éditeur (SM, GMB), intégrateur, simulateur, "debugger"
PRISM [Rosenberg 85]	analyse stucturée, conception structurée	diagramme de flux de données, organigramme structuré, PDL	éditeur (PDL, DFD), générateur d'organi-grammes
SCHEMACODE [Robillard 81]	programmation structurée	PCS (pseudocode schématique)	éditeur (PCS), générateur de code
SPADES [Ludewing 85]	SPADE-M (modèle entité-association)	SPADE-L (texte)	SPADE-T (éditeur, analyseur)
TAGS [Sievert 85]	méthode TAGS (décomposition fonctionnelle, expression des flux de données et de contrôle, prototypage)	IORL (diagrammes et tables)	éditeur (IORL), analyseur, simulateur, gestionnaire de versions multiples
USE.IT [Hamilton 83]	HOS (décomposition fonctionnelle basée sur un modèle mathéma-tique formel)	AXES (diagrammes)	éditeur (AXES), analyseur, générateur de code

Tableau 1 - Systèmes d'aide à la conception

meilleures notations et de meilleurs outils logiciels non seulement pour faciliter la conception mais aussi l'entretien des systèmes informatiques. Dans cette section, nous préciserons davantage ces trois concepts et nous dégagerons des critères empiriques d'évaluation pour mesurer l'efficacité et la pertinence des outils existants et pour fixer des objectifs à atteindre lors de la réalisation de nouveaux outils.

3.1 Les méthodes de conception

Plus un système est complexe, plus il devient difficile d'organiser la structure du logiciel et de répondre aux attentes des utilisateurs et des différentes équipes impliquées dans le projet. Pour mener à bien l'étape de conception, il s'avère essentiel de disposer de méthodes de travail adéquates. A un niveau macroscopique, des méthodes partitionnent le travail en sous-activités et guident les concepteurs du début jusqu'à la fin de l'étape de conception afin d'assurer une progression sûre vers l'objectif final. A un niveau microscopique, des méthodes tracent la voie pour la décomposition et l'organisation du système informatique, pour la modélisation détaillée des traitements et des données, ainsi que pour la documentation, la vérification et la validation des spécifications. Ces méthodes consistent donc en un ensemble ordonné de procédures et de points de contrôle basés principalement sur des traditions, sur des expériences vécues et sur des principes de conception de systèmes.

Il existe plusieurs méthodes de conception de systèmes informatiques. Parmi les plus fondamentales et les plus répandues, citons

l'analyse structurée [DeMarco 78], la conception structurée [Yourdon 79], la méthode de Jackson et Warnier [Warnier 74; Jackson 75], la programmation structurée [Yourdon 75] et la méthode exposée par Gries [Gries 81]. Les deux premières méthodes ne concernent que la conception architecturale, alors que les trois dernières s'attaquent presque exclusivement à la conception détaillée. L'analyse structurée entraîne une décomposition fonctionnelle et une expression des flux de données du système. La conception structurée conduit à une structure hiérarchique et modulaire du système. La méthode de Jackson et Warnier propose de définir les algorithmes à partir des structures de données d'entrée et de sortie. La programmation structurée force le concepteur à exprimer les algorithmes à l'aide de trois structures de contrôle. Enfin, la méthode de Gries, basée sur le calcul des prédicats et sur la notion de précondition la plus faible, est radicale mais utile pour le développement de nouveaux algorithmes complexes. Le lecteur trouvera dans l'article de Yau et al. [Yau 86] une synthèse des méthodes de conception de systèmes et dans l'ouvrage de DeMarco [DeMarco 79] une description concise des principales méthodes.

Ces méthodes sont dites fondamentales parce qu'elles sont souvent employées pour élaborer des méthodes plus spécialisées. Par exemple, la méthode DARTS proposée pour la conception de systèmes en temps réel [Gomaa 84] utilise les principes de décomposition du système en tâches concurrentes et de masquage d'information [Parnas 72], ainsi que les méthodes d'analyse structurée et de conception structurée. Le guide de développement d'un système d'information, suggéré par la firme DMR [DMR 85], favorise les méthodes d'analyse structurée et de conception

structurée pour la modélisation des traitements et le modèle entité-association [Chen 76] pour la modélisation des données. Enfin, la méthode sous-jaçente à l'outil SCHEMACODE force les programmeurs à concevoir des programmes structurés en voilant l'aspect syntaxique des langages de programmation et en mettant en évidence la structure sémantique ou logique des programmes essentielle à une analyse détaillée d'un problème [Robillard 85].

L'évaluation d'une méthode reste encore aujourd'hui une tâche difficile à accomplir, car il n'existe pas une théorie générale des méthodes. A plus forte raison, la création des méthodes se fait plus par référence au bon sens humain qu'à des formalismes de mathématiques pures. A moyen terme, l'efficacité d'une méthode peut être mesurée à partir d'un grand nombre de systèmes réalisés à l'aide de celle-ci [Card 86]. A court terme, nous pouvons caractériser les méthodes à partir des cinq attributs suivants:

Spécificité. Les méthodes générales conviennent à un ensemble universel de systèmes. Par contre, elles ignorent leurs propriétés intrinsèques. Les méthodes spécifiques, souvent construites à partir des méthodes générales, permettent plus aisément la conception de systèmes spécialisés (systèmes d'information, systèmes en temps réel, systèmes répartis, systèmes d'intelligence artificielle). L'introduction de nouvelles règles facilite, par exemple, l'expression de la répartion, de la concurrence, de la communication ou de la synchronisation.

Profondeur. Une méthode s'appuie sur des principes éprouvés (le masquage d'information, l'abstraction des données [Liskov 75], la

séparation des mécanismes et des politiques [Brinch Hansen 70]) ou sur des structures mathématiques formelles (la méthode HOS [Hamilton 76]).

Complétude. Une méthode est complète si elle offre au concepteur un nombre suffisamment élevé de points d'ancrage. Par exemple, si la méthode utilise une technique de décomposition, elle doit lui fournir des critères de décomposition. Elle doit aussi lui indiquer une liste des vérifications et des validations à effectuer à chaque étape de la décomposition.

Praticabilité. Une méthode ne doit pas nuire à la productivité des concepteurs. Elle ne les encombre pas de détails inutiles ou d'un trop grand nombre de détails à la fois. Par contre, elle doit limiter les concepteurs dans leur propre démarche afin de garantir une uniformité et une qualité du logiciel. Idéalement, l'application des règles doit être transparente ou naturelle aux concepteurs.

Adaptabilité. Une méthode doit s'adapter facilement à de nouveaux environnements de développement de systèmes qui répondent mieux aux besoins et aux politiques d'une entreprise. A cause de l'inertie du milieu, il est de plus en plus difficile et même parfois coûteux de faire accepter de nouvelles méthodes.

3.2 Les notations

La notation ou la représentation des spécifications, qu'elle soit littérale ou graphique, fait l'objet d'une attention particulière. En effet, la notation étant un formalisme qui sert de support pour exprimer, communiquer, analyser et modifier les spécifications, elle

influence fortement la fonctionnalité des outils logiciels à mettre en oeuvre. De nombreuses notations ont été proposées et un choix parmi celles-ci ne constitue pas une tâche facile. C'est pourquoi nous soulignons quatre qualificatifs à prendre en considération pour ventiler les nombreuses notations qui existent aujourd'hui.

Complète. La notation comme moyen d'expression des spécifications doit posséder une syntaxe pertinente pour capter suffisamment d'éléments sémantiques et pour permettre une prise de décision éventuelle sur la modifiabilité du système informatique.

Compréhensible. La notation comme moyen de communication des spécifications doit être simple, facile d'accès et lisible aussi bien par les concepteurs que par les responsables de projets.

Analysable. La notation comme support pour analyser les spécifications produites par les concepteurs doit être formelle, pour vérifier et valider leur cohérence et leur complétude, et être exécutable pour simuler le comportement du système informatique.

Flexible. La notation comme support documentaire à la maintenance doit être assez souple pour faciliter les modifications des spécifications.

Les notations à l'aide de diagrammes apparaîssent les plus populaires pour plusieurs raisons. Premièrement, les diagrammes sont très attrayants pour l'oeil humain. Deuxièmement, les concepteurs sont souvent enclins à schématiser par des représentations sommaires certains traitements et données qui leur semblent difficile à verbaliser ou à énoncer autrement qu'en ayant recours à des figures.

Troisièmement, les diagrammes facilitent l'explication rapide des structures globale et détaillée du système informatique. Finalement, les diagrammes sont construits généralement à partir d'un nombre restreint de symboles, ce qui évite aux concepteurs de se préoccuper d'un trop grand nombre d'éléments syntaxiques et sémantiques. Ainsi, les diagrammes sont avant tout une représentation visuelle de l'ensemble des modules, des interfaces, des données et du comportement du système informatique. Toutefois, selon les notations proposées, les diagrammes comportent plusieurs inconvénients dont certains peuvent être contournés par l'introduction d'outils appropriés. Premièrement, ils sont souvent incomplets, peu analysables et/ou difficiles à modifier; car même si la partie graphique est formellement définie, la majeure partie de leur contenu est constitué de textes informels. Deuxièmement, des contraintes physiques du matériel, en particulier celles des écrans et des imprimantes, imposent des limites quant à la taille des diagrammes et au texte contenu dans une bulle. Finalement, si les diagrammes comportent un niveau de détails élevé, on atteint rapidement la limite visuelle de l'utilisateur.

Parmi les notations graphiques les plus répandues, mentionnons les diagrammes de flux de données, les organigrammes structurés, les diagrammes entités-associations, les graphes de transition d'états, les réseaux de Pétri, les organigrammes de Nassi-Scheiderman [Nassi 73] et le pseudocode schématique [Robillard 81]. Le lecteur trouvera dans l'article de Teplitzky [Teplitzky 79], un ensemble de critères différents de ceux présentés ici pour choisir parmi des notations graphiques couramment employées celle qui convient la mieux.

Les représentations des spécifications à l'aide de textes écrits dans un langage naturel ou dans un langage formel occupent aussi une place importante. Les langages formels possèdent l'avantage de bien capter la sémantique associée au système informatique et permettent plus aisément de déterminer si les spécifications sont cohérentes et complètes. Par contre, ils possèdent certains inconvénients. Souvent les spécifications écrites dans ces languages sont plus longues que les programmes qui en résultent. La codification est alors remplacée par une autre forme d'écriture plus abstraite dans laquelle le taux d'erreur est comparable à celui de la codification. Un formalisme excessif exige une bonne formation scientifique des personnels et implique une accumulation importante d'information dont l'exploitation n'est pas toujours immédiate et commode dans les étapes ultérieures du cycle de vie.

Il existe de nombreuses notations formelles; mentionnons celles basées sur des formalismes algébrique et axiomatique [Guttag 77] ou sur la logique mathématique (propositionnelle, du premier ordre, temporelle). Il existe aussi des langages de conception sans formalisme mathématique comme, par exemple, DREAM [Riddle 78] ou ADL [Vosbury 84]. Enfin, il y a les langages basés sur le pseudocode qui utilisent conjointement des langages naturels et des syntaxes de langages de programmation [Caine 75]. Ces notations ne possèdent aucun des avantages des langages formels. En plus, elles enferment les concepteurs dans un environnement de programmation.

Il faut éviter de séparer les deux types de notation, car ils se complètent l'un l'autre. En particulier, les langages formels peuvent

posséder une contrepartie graphique qui supprime une partie de leurs inconvénients. Ainsi, on aura tout avantage à offrir aux concepteurs un ensemble d'outils capables de traiter plus d'une représentation de spécifications.

Deux autres critères permettent de caractériser les notations. Premièrement, certaines d'entre elles sont plus appropriées pour la conception architecturale que pour la conception détaillée (bien qu'il existe plusieurs notations qui conviennent à la fois pour ces deux tâches) et inversement. Deuxièmement, elles permettent d'exprimer soit les flux de données, soit la logique du système. Par exemple, le pseudocode schématique s'emploie durant la conception détaillée et facilite la description de la logique des modules d'un système.

3.3 Les outils logiciels conventionnels

Dans la pratique courante, les tâches des concepteurs sont nombreuses. Même si la créativité constitue le dénominateur commun dans la réalisation de ces tâches, il n'en demeure pas moins qu'une bonne partie du travail des concepteurs est fastidieuse, exigeante et répétitive. Les outils logiciels conventionnels procurent donc une aide de plus en plus indispensable aux concepteurs, car ils automatisent les tâches routinières qui requièrent surtout des habiletés manuelles et peu d'habiletés intellectuelles. Ils facilitent aussi l'édification précise des dossiers techniques. Les outils logiciels conventionnels constituent un prolongement des outils classiques (compilateurs, éditeurs) qui opèrent normalement sur des programmes. En effet, ils comprennent des fonctions principalement axées vers

l'archivage, l'édition, l'analyse et la documentation des données générées pendant la conception. Si du point de vue fonctionnel, ces deux classes d'outils présentent une grande similitude, la différence fondamentale réside dans le fait que les outils logiciels de conception traitent des spécifications d'un niveau d'abstraction plus élevé.

L'archivage permet de conserver toutes les informations produites lors de la conception architecturale et détaillée du système informatique. Qu'il s'agisse d'un simple dictionnaire de données ou d'une base de données spécialisée, cet outil constitue le fondement essentiel au travail en équipe et à l'intégration d'outils en général. Nous discuterons ces deux aspects à la fin de cette section.

La manipulation des spécifications demande un travail considérable de la part des concepteurs. Leur formulation et leur mise à jour peuvent facilement devenir fastidieuses. Les éditeurs de diagrammes, les éditeurs syntaxiques contextuels et les éditeurs de tables constituent des outils essentiels, car en plus de résoudre le problème précédent, ils guident parfois les concepteurs dans leurs démarches en imposant des règles de construction de systèmes. Tout aussi importants sont les outils qui traduisent les spécifications d'une notation à une autre. Ils assurent aux concepteurs une vue multiple des données [Reiss 84]. D'autres outils permettent de passer d'un niveau de spécifications à un autre et ce dans les deux sens. Par exemple, un générateur de code produit automatiquement, à partir des spécifications détaillées, une partie (e.g. la définition des données en COBOL) ou la totalité du programme. Il est parfois utile d'effectuer l'opération inverse pour récupérer de vieux programmes et générer les

spécifications détaillées correspondantes dans une notation plus compréhensible pour les concepteurs. Par exemple, FLOWCHART génère des organigrammes de Nassi-Scheiderman à partir de programmes écrits en PASCAL ou en pseudocode [Roy 76].

L'analyse des spécifications produites par les concepteurs constitue une tâche obligatoire, car elle garantit leur travail. Une analyse statique permet de vérifier, par exemple, si les spécifications sont complètes, cohérentes, correctes et réalisables. Un autre aspect négligé est l'analyse de l'impact d'une modification dans un système complexe, car il est très difficile de déterminer quelles sont les composantes susceptibles d'être affectées par une modification et d'effectuer les changements sans causer des dommages irréparables. Enfin, une analyse dynamique permet de valider les spécifications par rapport aux intentions des utilisateurs. Elle simule le comportement du système. A ce stade, les simulateurs et les outils de mise au point jouent un rôle significatif. Le lecteur trouvera dans l'article de Boehm [Boehm 84] un ensemble de critères et de techniques pour la validation et la vérification des spécifications.

Finalement, la documentation alimente les étapes ultérieures du cycle de vie. Son volume peut être imposant si le système est d'une grande envergure. Plusieurs éléments de la documentation peuvent être produits à partir des données enregistrées dans la base de données. Des générateurs de diagrammes, des formatteurs de tables et des systèmes de traitement de texte constituent les outils les plus répandus. Aussi, pour mettre rapidement leurs produits sur le marché international, les Japonais construisent des outils capables de

traduire en anglais de la documentation originalement écrite en japonais [Mohri 84].

Nous concluons cette section en soulignant que les outils logiciels conventionnels doivent présenter, outre les fonctionnalités précédentes, deux qualités importantes. Ils doivent favoriser un véritable travail d'équipe et s'intégrer les uns aux autres.

Les systèmes complexes doivent être conçus par plusieurs personnes. Il se peut fort bien qu'il n'y ait personne qui comprenne tous les détails du système lors de sa conception. A un niveau supérieur toutefois, le système informatique doit être conçu de façon à être compris par au moins une personne. En deça de ce niveau, la conception des sous-systèmes est répartie entre plusieurs personnes. Pour faciliter le travail d'équipe, il faut exploiter les mécanismes offerts par les systèmes de gestion de base de données pour assurer l'intégrité, la protection et le partage des données. Il faut aussi mettre en oeuvre des mécanismes de communication qui captent l'information pertinente véhiculée lors de la conception, et des mécanismes qui facilitent l'intégration de toutes les parties du système et qui empêchent le travail redondant.

La majorité des outils sont monolithiques, car ils ont été développés et ils évoluent dans un environnement spécifique. Chaque outil gère ses propres fichiers ou sa propre base de données, et utilise un système d'exploitation et du matériel particuliers. Il présente à l'utilisateur un interface unique. Ainsi, la grande majorité des outils sont locaux, dans le sens qu'ils ne partagent pas leurs résultats. Pour faciliter l'intégration des outils logiciels,

il faut qu'ils soient portables, qu'ils soient conduits par un processus commun et qu'ils soient conçus en fonction de normes dictées pour l'interface des données, pour l'interaction homme-machine et pour les services offerts par les systèmes d'exploitation et les systèmes de gestion de bases de données. Les méta-outils semblent la solution la mieux adaptée pour offrir un cadre qui permet le développement harmonieux d'outils techniques. A titre d'exemple, citons le travail d'une équipe d'IBM [Hoffnagle 85] qui a proposé un modèle d'un méta-outil qui rencontre les objectifs énonçés ci-dessus.

4. L'environnement de conception orienté vers les outils logiciels à base de connaissances

Dans cet environnement, des outils logiciels fournissent une assistance intelligente aux concepteurs, et même parfois accomplissent automatiquement une partie de leurs tâches créatives. Ces outils diffèrent considérablement des outils conventionnels présentés dans la section précédente, d'une part par les fonctionnalités qu'ils possèdent et d'autre part par les moyens à prendre pour leur mise en oeuvre. Au début du développement de ces outils, les chercheurs aspiraient essentiellement à faire en sorte qu'ils génèrent automatiquement des spécifications utiles à partir de spécifications abstraites. Cette ambitieuse entreprise a donné lieu à un certain nombre de prototypes expérimentaux qui malheureusement n'ont été capables à ce jour que de résoudre de petits problèmes. CHI, PSI, DELADUS, NLPQ et SAFE sont des exemples de systèmes de programmation automatique qui acceptent en entrée une spécification d'un domaine de problèmes bien circonscrit et qui génèrent du code compilable ou interprétable

[Barr 82]. Aujourd'hui, des chercheurs concentrent davantage leurs efforts pour développer des outils logiciels capables:

-de guider le concepteur dans le choix et dans l'application des règles de décomposition pour raffiner de plus en plus les spécifications;

-d'assister le concepteur dans l'évaluation de l'impact d'une décision et dans l'exploration des différentes alternatives possibles de solutions;

-de conserver les justifications de certaines décisions envisagées et analysées, mais rejetées, dans le cas où ces dernières pourraient être reconsidérées un peu plus tard dans le processus de conception;

-de vérifier la cohérence et la complétude du système informatique et, dans le cas échéant, de remédier aux conflits et aux manques;

-de résoudre les contraintes conflictuelles et potentielles qui s'exercent sur les choix des concepteurs et qui proviennent des différentes sources de son environnement;

-d'évaluer les effets ou les répercussions néfastes d'une modification dans la spécification formelle du système sur les spécifications de conception déjà dérivées ou sur un système informatique opérationnel;

-de dialoguer avec le concepteur dans un langage naturel ou dans un sous-ensemble spécialisé et limité d'un langage naturel;

-de fournir à la demande du concepteur une explication à propos des résultats obtenus;

-de permettre l'ajout, la destruction ou la mise à jour des règles de conception.

Ces outils logiciels exécutent ces différentes actions soit en exploitant l'information contenue explicitement ou implicitement dans les spécifications, soit en référant à la base des connaissances, soit en interrogeant le concepteur. Ils diffèrent des systèmes de programmation automatique car ils ne réalisent pas, mais aident plutôt les concepteurs à accomplir des sous-tâches difficiles du processus de conception.

Ainsi, pour l'environnement orienté vers les outils logiciels à base de connaissances, les chercheurs doivent tendre au-delà des méthodes et des notations traditionnelles. Ils doivent franchir une autre étape en embrassant l'étude de la nature du logiciel et du processus de conception [Perlis 85; Soloway 84] et reconnaître le rôle primordial de la connaissance associée au domaine d'application [Barstow 85]. Ils doivent aussi exploiter les techniques de l'intelligence artificielle pour représenter les connaissances et pour générer et expliquer des solutions. Finalement, pour éviter aux concepteurs d'être familiers avec un langage de spécification trop compliqué, ils doivent renforcer ces outils par une interaction homme-machine conviviale permettant un dialogue en langage naturel.

Dans cette section, nous examinerons brièvement quelques outils logiciels à base de connaissances et nous donnerons leurs caractéris-

tiques. Nous évoquerons aussi les problèmes de la modélisation du domaine d'application et du processus de conception. Nous négligerons deux aspects importants, la construction des systèmes experts et le traitement des langues naturelles, qui sont communs à tous les domaines d'application et qui sont largement exposés dans les ouvrages d'intelligence artificielle.

4.1 Quelques exemples d'outils logiciels à base de connaissances

A titre d'indication et pour illustrer leur diversité, nous décrivons brièvement quelques outils logiciels automatiques. Bien entendu, cette présentation ne constitue pas une étude exhaustive. Nous voulons seulement attirer l'attention du lecteur sur quelques-unes de leurs caractéristiques internes et externes. Nous invitons donc le lecteur qui désirerait compléter ses connaissances à se référer aux articles cités dans le texte.

APE (Automatic Programming Expert) est un système expert pour la programmation automatique développé à l'Université de Bonn (Allemagne de l'ouest) [Bartels 81]. Ce système dispose de connaissances pour produire des programmes exécutables en INTERLISP à partir de spécifications algébriques de types abstraits de données et d'algorithmes abstraits formulés à l'aide de règles de production. Il comporte deux sous-systèmes indépendants qui utilisent des connaissances représentées sous forme de règles de production pour codifier respectivement les types de données et les algorithmes.

SECSI (Système expert pour la conception de systèmes d'information) est un système expert pour la conception de bases de données développé

à l'INRIA (France) [Bouzeghoub 83]. A partir d'une spécification d'une application décrite dans un sous-ensemble du français, ce système génère automatiquement un schéma relationnel normalisé ou optimisé ainsi qu'un ensemble de contraintes d'intégrité. La spécification est traduite en un réseau sémantique puis graduellement transformée, à l'aide d'un ensemble de règles de production, en un schéma relationnel.

Un prototype expérimental, nommé Designer/Verifier's Assistant a été développé par un étudiant de l'Université du Texas [Moriconi 79]. Ce système comporte deux sous-systèmes: un système expert qui facilite le développement incrémental de gros systèmes informatiques et un système traditionnel de preuve automatique. Le système expert interprète les effets possibles d'un changement dans une spécification et indique comment procéder méthodiquement pour la conception et la vérification. Il utilise un graphe pour représenter les connaissances déclaratives tels que les éléments qui interviennent dans la conception et la vérification ainsi que les relations entre ces éléments. De plus il gère, à l'aide de règles de production, un agenda qui suggère aux concepteurs les tâches à accomplir pour compléter correctement la conception du système informatique.

Un système expert a été développé à l'Université d'Illinois (Urbana-Champaign) pour supporter la conception de systèmes informatiques basée sur l'analyse structurée [Harandi 85]. Ce système expert utilise des patrons de segments de diagrammes de flux de données (DFD) pour représenter les composants nécessaires lors de la conception. Un ensemble de règles de transformation permet la combinaison et le

raffinement des segments afin d'obtenir un DFD détaillé du système désiré. Au fur et à mesure du raffinement, les spécifications de l'utilisateur et du DFD sont analysées pour vérifier si elles sont cohérentes ou complètes. Le système expert prend en considération la connaissance du domaine d'application, organisée sous forme de structures hiérarchiques, pour faciliter la sélection des segments et leur intégration dans le DFD.

Une petite base de connaissances écrite en PROLOG contient de l'information sur les attributs des modules, les relations entre les modules, la cohésion et le couplage. Des faits sont ajoutés dans la base de connaissances au fur et à mesure de la conception du système. Enfin, des règles permettent de déduire de nouveaux faits sur la structure du système lors de la conception ou de la maintenance [Leung 85].

La majorité des outils logiciels à base de connaissances sont en cours de développement et/ou dans leur enfance. Seulement quelques prototypes sont présentement employés dans des firmes spécialisées de logiciel sur une base expérimentale. De grands efforts doivent être déployés avant d'atteindre les objectifs initiaux, en particulier avant de démontrer qu'il s'agit d'une technique viable et praticable pour des problèmes réels.

4.2 Les caractéristiques des outils logiciels à base de connaissances

Outre les aspects utilitaires présentés au début de cette section, il convient de souligner huit signes distinctifs importants qui permettent de juger de leurs possibilités [Barr 82; Mostow 85].

La méthode de spécification. La méthode de spécification offre aux concepteurs un moyen de décrire en termes plus ou moins abstraits ses applications. Le choix d'une méthode influence la qualité et la complexité du module qui gère l'interaction entre l'outil et le concepteur. Ainsi le concepteur pourra selon l'outil exprimer son problème à partir d'exemples d'entrée et de sortie ou à partir de traces, ou à l'aide de langages formels ou de sous-ensembles d'un langage naturel. Bien que les langages naturels soient intéressants pour les humains, ils obligent l'outil à solutionner le problème avec des spécifications ambiguës.

La sortie. La sortie produite varie d'un outil à l'autre. Il peut s'agir d'un langage de programmation cible dans le cas des systèmes de programmation automatique ou d'une spécification plus raffinée dans le cas des outils plus spécialisés.

Le domaine d'application. Les outils ne peuvent résoudre que des problèmes appartenant à un domaine d'application bien précis. Plus le domaine est restreint, plus l'outil est puissant. Cette constatation est inhérente aux systèmes experts qui ont démontré l'utilité des techniques de l'intelligence artificielle lorsque celles-ci ne s'attaquent pas à une classe universelle de problèmes.

La méthode d'opération. La méthode d'opération réfère aux techniques d'intelligence artificielle mises en oeuvre pour résoudre le problème. Les outils s'appuient principalement sur des techniques de systèmes experts, mais des techniques de preuve automatique, de transformation de spécifications et de résolution de problèmes sont aussi employées.

Le degré d'automatisme. Le degré d'automatisme reflète en quelque sorte la puissance de l'outil. Il dépend surtout du domaine d'application et de la base des connaissances. Le but ultime des recherches dans ce domaine est d'atteindre un degré très élevé afin d'automatiser non seulement toute l'étape de conception mais l'ensemble du cycle de vie.

La base des connaissances. Les connaissances et leurs représentations permettent de modéliser le domaine d'application et le processus de conception. La connaissance descriptive décrit les éléments et les relations qui interviennent lors de la conception tandis que la connaissance normative indique comment inférer de nouvelles décisions. Pour représenter ces deux types de connaissances, on a recours par exemple aux réseaux sémantiques, aux règles de production, aux objets structurés, aux systèmes procéduraux et à la logique du premier ordre.

La portée de l'outil. La tâche à laquelle l'outil s'attaque, ainsi que l'écart entre le niveau des spécifications à l'entrée et le niveau des produits à la sortie, décident de la portée de l'outil dans l'étape de conception ou dans le cycle de vie.

4.3 La modélisation du domaine d'application

Les méthodes de conception élaborées jusqu'ici s'appuyaient essentiellement sur des techniques de résolution de problèmes. Or, un aspect négligé mais important est la connaissance du domaine d'application. En pratique, le domaine d'application n'est pas perçu exactement de la même manière par tous, car il n'existe pas de description explicite de ce domaine. De plus, chaque individu

acquiert cette connaissance indirectement par des intermédiaires qui risquent de ne pas être précis et constants dans leurs descriptions. Cette situation est donc source d'ambiguïtés et de conflits lors du développement des systèmes informatiques. Une solution à ce problème consiste à spécifier un modèle conceptuel du domaine d'application qui évolue indépendamment des systèmes informatiques à développer, et sur lequel repose l'analyse, la conception et la maintenance des systèmes informatiques. Les bénéfices à retirer d'une telle approche sont nombreux [Barstow 85]:

- -les concepteurs peuvent élaborer leurs spécifications en partie par consultation, étude et analyse du modèle conceptuel;
- -la simulation d'une partie du modèle conceptuel permet d'évaluer son comportement dynamique et d'éclaicir des points imprécis qui surgissent au moment de la conception;
- -tout système de programmation automatique requiert explicitement une connaissance formelle du domaine d'application;
- -l'importante quantité de connaissances contenues dans le modèle conceptuel constitue une partie intégrante de la base des connaissances de tout outil logiciel intelligent;
- -la documentation propre au domaine d'application est explicitement disponible via le modèle conceptuel.

Ainsi, le problème de spécifier un modèle conceptuel du domaine d'application est fondamental pour l'environnement de conception orienté vers les outils logiciels à base de connaissances.

L'analyse du domaine d'application implique [Arango 85]:

-la définition des frontières du domaine en fixant la classe des problèmes à étudier;

-la recherche et la classification des entités conceptuelles du domaine;

-le développement d'un modèle du domaine en établissant les relations et les dépendances fonctionnelles entre les entités identifiées;

-le développement de définitions réductibles en termes de modèles existants ou en termes de modèles intermédiaires spécialement conçus pour profiter d'ensembles d'abstractions réutilisables dans des développements ultérieurs.

La conception d'un domaine d'application est un processus qui transforme la structure sémantique du domaine dans une structure syntaxique appropriée. Plusieurs formalismes ont été proposés; parmi les plus importants, citons les langages Clear [Burstall 79] et RML [Borgida 85], les graphes conceptuels [Sowa 84], les réseaux de Pétri et l'usage de taxinomie [Borgida 84]. Ces formalismes visent principalement à capter aisément la connaissance descriptive du domaine et à faciliter la description structurée des problèmes.

4.4 La modélisation du processus de conception

Pour modéliser une sous-activité du processus de conception, on a recours aux techniques de la psychologie cognitive. Des observations

et des entrevues avec un ensemble de spécialistes qualifiés pour cette opération permettent d'exhiber les connaissances, les structures des connaissances, les mécanismes de raisonnement et les différentes stratégies qui interviennent pendant chacune des phases d'exécution du processus. Par exemple, pour la conception d'un algorithme, on recueille et on interprète des déroulements d'une série d'idées sur la compréhension du problème, l'ébauche d'une solution, le raffinement de la solution, l'exécution de la solution préliminaire, la formulation des difficultés et des opportunités, la vérification de la solution et l'évaluation de la solution [Kant 85]. Ensuite, on reconstitue le plus objectivement possible le comportement des spécialistes face à un ensemble de problèmes, puis on traduit leurs pensées en des plans et des règles appropriés. Cette démarche constitue un préalable à toute entreprise sérieuse de développement d'outils logiciels à base de connaissances, car elle conduit à l'élaboration de théories qui permettent de parvenir à une meilleure connaissance du processus de conception et à de nouvelles techniques qui cadrent mieux avec la nature humaine.

5. Conclusion

Dans ce rapport, nous avons fait une synthèse et une analyse des principaux concepts liés aux outils logiciels de conception de systèmes informatiques. Malgré leur grande diversité, nous avons montré comment ces outils pouvaient être classifiés en deux grandes catégories bien distinctes. La première catégorie regroupe les outils logiciels conventionnels et constitue une extension directe des outils qui agissent sur des programmes. Selon une étude récente commandée

par le Département de la défense américaine [Redwine 84], ces outils commencent à peine à être utilisés à une grande échelle. Toujours selon ce même rapport, les techniques requises pour leur mise en oeuvre semblent bien maîtrisées. Toutefois, un certains nombre de problèmes subsistent. En particulier, des efforts considérables doivent être faits pour proposer et développer des méta-outils qui faciliteront leur intégration et leur normalisation. De plus, aucune étude sérieuse n'a encore été faite pour évaluer leur performance relative dans un environnement réel de développement. Les critères d'évaluation dégagés dans ce rapport sont qualitatifs et permettent seulement de jauger rapidement leurs possibilités.

La deuxième catégorie regroupe les outils logiciels à base de connaissances. Même si cette approche n'a pas encore démontré qu'il s'agit d'une technique viable et efficace, elle semble être prometteuse. De nombreuses recherches sont en cours et les premiers prototypes sont représentatifs des possibilités offertes aux concepteurs, mais soulèvent de nombreuses questions et problèmes auxquels doivent faire face les chercheurs du domaine. La réussite de cette approche dépendra surtout des trois facteurs suivants: l'évolution des techniques de l'intelligence artificielle (en particulier les systèmes experts), la compréhension du processus de conception et la modélisation des domaines d'application. Le besoin persistant de réduire les coûts de développement et d'entretien du logiciel continuera à orienter les chercheurs dans cette direction.

BIBLIOGRAPHIE

[Arango 85]

G. Arango, P. Freeman: Modeling knowledge for software development; Proceedings of the 3rd Int. Workshop on Software Specification and Design, London, 1985, 63-66.

[Balzer 85]

R. Balzer: The role of logic and AI in the software entreprise; Proceedings of the 8th Int. Conf. on Software Engineering, London, 1985, 394.

[Barr 82]

A. Barr, E. A. Feigenbaum: The Handbook of Artificial Intelligence; William Kaufmann, Inc, Vol. 2, 1982, 295-379.

[Barstow 85]

B. Barstow, P. Barth, P. Dietz, R. Dinitz, S. Greenspan: Observations on specifications and automatic programming; Proceedings of the 3rd Int. Workshop on Software Specification and Design, London, 1985, 89-90.

[Bartels 81]

U. Bartels, W. Olthoff, P. Raulefs: APE: An expert system for automatic programming from abstract specifications of data types and algorithms; MEMO SEKI-BN-81-01, Universität Bonn, 1981.

[Boehm 76]

B. W. Boehm: Software engineering; IEEE Trans. on Computers, C 25 (12), 1976, 1226-1241.

[Boehm 84]

B. W. Boehm: Verifying and validating software requirements and design specifications; IEEE Software, 1 (1), 1984, 75-88.

[Borgida 84]

A. Borgida, J. Mylopoulos, H. K. T. Wong: Generalization/specification as a basis for software specification; In On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases and Programming languages. Ed. by M. L. Brodie, J. Mylopoulos, J. W. Schmidt, Springer-Verlag, New York, 1984, 87-114.

[Borgida 85]

A. Borgida, S Greenspan, J. Mylopoulos: Knowledge representation as the basis for requirements specifications; Computer, 18 (4), 1985, 82-91.

[Bouzeghoub 83]

M. Bouzeghoub, G. Gardarin: The design of an expert system for database design; Proceedings of the 1st Int. Workshop on New Application of Databases, Cambridge, UK, 1983, 203-223.

[Brinch Hansen 70]

P. Brinch Hansen: The nucleus of a multiprogramming system; Comm. ACM, 13 (4), 1970, 238-241.

[Burstall 79]

R. Burstall, J. Goguen: The semantics of Clear, a specification language; Abstract Software Specification, Lecture Notes in Computer Science, No. 86, D. Bjorner, Ed., Springer-Verlag, New York, 1979, 292-331.

[Caine 75]

S. H. Caine, K. Gordon: PDL - A tool for software design; Proceedings of the Nat. Comp. Conf., AFIPS, Arlington, 1975, 271-276.

[Card 86]

D. N. Card, V. E. Church, W. W. Agresti: An empirical study of software design practices; IEEE Trans. on Software Engineering SE 12 (2), 1986, 264-271.

[Chen 76]

P. P. Chen: The entity-relationship model: Toward a unified view of data; ACM Trans. on Database Systems, 1 (1), 1976, 9-36.

[De Drouas 82]

E. de Drouas, J.-M. Nerson: Les ateliers logiciels intégrés: développements français actuels; T.S.I., 1 (3), 1982, 211-232.

[DeMarco 78]

T. DeMarco: Structured Analysis and System Specification; Yourdon Press, New York, 1978.

[DeMarco 79]

T. DeMarco: Concise Notes on Software Engineering; Yourdon Press, New York, 1979.

[DMR 85]

DMR: Guide de développement d'un système d'information, partie 2 le développement du système; Ducros, Meilleur, Roy & Associés Ltée, deuxième édition, Montréal, 1985.

[Frenkel 85]

K. A. Frenkel: Toward automating the software development cycle; Comm. ACM, 28 (6), 1985, 578-589.

[Gomaa 84]

H. Gomaa: A software design method for real-time systems; Comm. ACM, 27 (9), 1984, 938-949.

[Gries 81]

D. Gries: The Science of Programming; Springer-Verlag, New York, 1981.

[Guttag 77]

J. V. Guttag: Abstract data types and the development of data structures; Comm. ACM, 20 (6), 1977, 397-404.

[Hamilton 76]

M. Hamilton, S. Zeldin: Higher order software - A methodology for defining software; IEEE Trans. on Software Engineering, SE 2 (1), 1976, 9-32.

[Hamilton 83]

M. Hamilton, S. Zeldin: The functional life cycle model and its automation: USE.IT; The Journal of Systems and Software, 3 (1), 1983, 25-62.

[Harandi 85]

M. T. Harandi, M. D. Lubars: A knowledge based design aid for software systems; Proceedings of Softfair Conference II, San Francisco, 1985, 67-74.

[Hoffnagle 85]

G. F. Hoffnagle, W. E. Beregi: Automating the software development process; IBM Systems Journal, 24 (2), 1985, 102-120.

[Howden 82]

W. E. Howden: Contempory software development environments; Comm. ACM, 25 (5), 1982, 318-329.

[Jackson 75]

M. A. Jackson: Principles of Program Design; Academic Press, New York, 1975.

[Kant 85]

E. Kant: Understanding and automating algorithm design; IEEE Trans. on Software Engineering, SE 11 (11), 1985, 1361-1374.

[Krell 85]

E. Krell, E. Lor: Current state of the SARA/IDEAS design environment; Proceedings of Softfair Conference II, San Francisco, 1985, 218-230.

[Leung 85]

C. H. C. Leung, Q. H. Choo: A knowledge-base for effective software specification and maintenance; Proceedings of the 3rd Int. Workshop on Software Specification and Design, London, 1985, 139-142.

[Liskov 75]

B. Liskov, S. Zilles: Specification techniques for data abstraction; IEEE Trans. on Software Engineering, SE 1 (1), 1975, 7-19.

[Ludewig 85]

J. Ludewig, M. Glinz, H. Huser, G. Matheis, H. Matheis, M.F. Schmidt: SPADES - A specification and design system and its graphical interface; Proceedings of the 8th Int. Conf. on Software Engineering, London, 1985, 83-89.

[Mohri 84]

T. Mohri, E. Ono, S. Uehara, T. Takao, H. Sato: PDAS: An assistant for detailed design and implementation of programs; Proceedings of the 7th Int. Conf. on Software Engineering, Orlando, 1984, 108-115.

[Moriconi 79]

M. S. Moriconi: A designer/verifier's assistant; IEEE Trans. on Software Engineering, SE 5 (4), 1979, 387-401.

[Mostow 85]

J. Mostow: Foreword: What is AI? And what does it have to do with software engineering?; IEEE Trans. on Software Engineering, SE 11 (11), 1985, 1253-1256.

[Nassi 73]

I. Nassi, B. Schneiderman: Flowchart techniques for structured programming; ACM SIGPLAN Notices, 8 (8), 1973, 12-26.

[Parnas 72]

D. Parnas: On the criteria to be used in decomposing systems into modules; Comm. ACM, 15 (12), 1972, 1053-1058.

[Perlis 85]

A. J. Perlis: Another view of software; Proceedings of the 8th Int. Conf. on Software Engineering, London, 1985, 395-396.

[Redwine 84]

S. T. Redwine Jr., L. G. Becker, A. B. Marmor-Squires, R. J. Martin, S. H. Nash, W. E. Riddle: DoD related software technology, requirements, practices, and prospects for the future; IDA paper P-1788, Institute for defense analysis, 1984.

[Reiss 84]

S. P. Reiss: Graphical program development with PECAN program development systems; Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Pittsburg, 1984, 30-41.

[Riddle 78]

W. E. Riddle, J. C. Wileden, J. H. Sayler, A. R. Segal, A. M. Stavely: Behavior modeling during software design; IEEE Trans. on Software Engineering, SE 4 (4), 1978, 283-292.

[Robillard 81]

P. N. Robillard, R. Plamondon: An interactive tool for descriptive, operational and structured documentation; Proceedings of the 23rd IEEE Comp. Int. Conf., Washington, D.C., 1981, 291-295.

[Robillard 85]

P. N. Robillard: A software tool and a schematic notation that improve the use of programming languages; Proceedings of Softfair Conference II, San Francisco, 1985, 149-158.

[Rosenberg 85]

D. Rosenberg: PRISM - Productivity improvement for software engineers and managers; Proceedings of the 8th Int. Conf. on Software Engineering, London, 1985, 2-7.

[Roy 76]

P. Roy, R. St-Denis: Linear flowchart generator for a structured language; ACM Sigplan Notices, 11 (11), 1976, 58-64.

[Schindler 81]

M. Schindler: Today's software tools point to tomorrow's tool systems; Electronic Design, 29 (7), 1981, 73-110.

[Sievert 85]

G. E. Sievert, T. A. Mizell: Specification-based software engineering with TAGS; Computer, 18 (4), 1985, 56-65.

[Soloway 84]

E. Soloway: A cognitively-based methodology for designing languages/environments/methodologies; Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Pittsburgh, 1984, 193-196.

[Sowa 84]

J. F. Sowa: Conceptual Structures - Information Processing in Mind and Machine; Addison Wesley, Reading, MA, 1984.

[Teplitzky 79]

P. Teplitzky: An approach for choosing a programming specification methodology; COMPSAC, 1979, 128-135.

[Vefsnmo 85]

E. A. M. Vefsnmo: DASOM - A software engineering tool for communication applications increasing productivity and software quality; Proceedings of the 8th Int. Conf. on Software Engineering, London, 1985, 26-33.

[Vosbury 84]

N. A. Vosbury: Process design; In Handbook of Software Engineering, Edited by C. R. Vick, C. V. Ramamoorthy, Van Nostrand Reinhold, New York, 1984, 544-564.

[Warnier 74]

J. D. Warnier: Logical Construction of Programs; Van Nostrand Reinhold Co., New York, 3rd Ed, 1974.

[Yau 86]

S. S. Yau, J. J.-P. Tsai: A survey of software design techniques; IEEE Trans. on Software Engineering, SE 12 (6), 1986, 713-721.

[Yourdon 75]

E. Yourdon: Techniques of Program Structure and Design; Prentice-Hall, Englewood Cliffs, 1975.

[Yourdon 79]

E. Yourdon, L. L. Constantine: Structured Design: Fundamentals of a Discipline of Computer and Systems Design; Prentice-Hall, Englewood Cliffs, 1979.

DEUXIEME PARTIE

DEFINITION DU PROBLEME

SOMMAIRE

Dans ce rapport, nous donnons une orientation à notre projet de recherche. Nous introduisons tout d'abord les systèmes informatiques guidés par un système expert qui associent la représentation procédurale à la représentation déclarative. Nous montrons l'utilité de tels systèmes, d'une part en précisant dans quel contexte cette solution est applicable, et d'autre part en dégageant leurs caractéristiques. Nous présentons ensuite une approche générale pour le développement et la maintenance de systèmes informatiques complexes. Nous identifions les problèmes inhérents à cette approche et nous indiquons quelques éléments de solutions basés sur des réalisations récentes. Enfin, nous concluons cette étude en énumérant une liste de problèmes encore ouverts. Chaque problème constitue un sujet de recherche potentiel à approfondir.

TABLE DES MATIERES

1. Introduction

Nous proposons comme objectif à notre projet de recherche d'apporter une contribution théorique et pratique au problème du développement et de la maintenance des systèmes informatiques complexes. Pour préciser ce que nous entendons ici par systèmes informatiques complexes, rappelons la classification des programmes présentée par Lehman [Lehman 80]. Cette classification partage les programmes en trois catégories distinctes: les S, P et E-programmes. Un S-programme constitue une solution informatique à un problème qui est précisément et complètement défini par une spécification statique. Les exemples suivants donnent un aperçu de quelques S-programmes: l'inversion d'une matrice carrée, la synchronisation de processus dans un système d'exploitation et le tri d'un vecteur d'éléments. L'entretien effectué sur les S-programmes est de nature perfectible; c'est-à-dire qu'il consiste à rendre les programmes plus efficaces, plus lisibles et plus élégants. Un P-programme contient des algorithmes heuristiques qui calculent des solutions approximatives suffisamment proches des solutions optimales d'un problème réel complètement spécifié pour lequel les spécialistes possèdent une connaissance incomplète, imprécise, contradictoire et évolutive. Les programmes joueurs d'échec, par exemple, entrent dans cette catégorie. Finalement, un E-programme est dérivé d'une spécification S, elle-même formulée à partir d'un modèle M. La construction du modèle M comporte un processus d'abstraction d'une partie de l'environnement. Les E-programmes diffèrent des P-programmes par le fait qu'ils font partie intégrante des applications qu'ils modélisent. A titre d'exemples, mentionnons les systèmes en temps réel, les systèmes d'exploitation, les systèmes répartis, les

systèmes de bureautique et les ateliers logiciels. Tout comme les P-programmes, les E-programmes sont sujets à de fréquentes modifications qui résultent principalement des changements de l'environnement ou des pressions des utilisateurs de plus en plus expérimentés vis-à-vis le système. L'entretien d'un E-programme reproduit les changements qui surviennent dans l'environnement, le modèle M, la spécification S et le programme. Dans ce rapport, nous associons l'expression "système informatique complexe" à un E-programme.

Selon Lehman, la décomposition d'un E-programme en modules permet de distinguer les éléments fonctionnels qui peuvent être complètement et précisément spécifiés, des éléments fonctionnels qui sont par nature heuristiques ou évolutifs. Dans ce dernier cas, ou bien ces éléments ne sont pas reconnus lors de la spécification du système et alors ils sont implicitement contenus dans le modèle, ou bien ils sont clairement identifiés et explicitement spécifiés. Au lieu d'utiliser l'approche procédurale pour implanter ces dernières spécifications, nous suggérons de traduire ces spécifications en une représentation exécutable par un système expert. Ainsi, un E-programme sera construit en deux parties distinctes: une partie stable construite à partir des techniques conventionnelles de conception de systèmes informatiques et une autre partie sujette à des changements continus supportée par des outils de systèmes experts. La première partie correspond à un ensemble de S-modules tandis que la deuxième partie correspond aux éléments heuristiques et évolutifs du E-programme, appelés E-éléments. Nous appelons un système informatique développé selon cette approche un système informatique guidé par un système expert (knowledge-driven system) [St-Denis 86].

Dans la prochaine section, nous justifions notre approche à ce problème en montrant comment les systèmes informatiques guidés par un système expert constituent une solution originale pour implanter des E-programmes. Dans la troisième section, nous décrivons une méthode générale à suivre pour développer et maintenir ce type de systèmes. Dans la quatrième section, nous présentons quelques exemples de systèmes informatiques guidés par un système expert. Dans la cinquième section, nous passons en revue un ensemble de problèmes partiellement résolus qui sont particuliers à cette approche. Enfin, dans la dernière section, nous fournissons une liste de problèmes à attaquer avec plus de profondeur.

2. Les systèmes informatiques guidés par un système expert

Un des moyens à prendre pour développer et maintenir plus efficacement des E-programmes consiste à transformer le cycle de vie conventionnel pour inclure l'utilisation intensive des techniques de l'intelligence artificielle. Une solution à long terme proposée par Balzer, Cheatham et Green exploite les outils logiciels à base de connaissances pour automatiser l'étape de développement [Balzer 83]. La solution à court terme présentée dans ce rapport est complètement différente. Nous suggérons de concevoir un E-programme comme un système informatique guidé par un système expert, c'est-à-dire d'intégrer un système expert dans chaque E-programme. Cette solution peut paraître à première vue inadéquate. Cependant les caractéristiques de l'environnement dans lequel s'exécutent les E-programmes s'apparentent aux critères qui définissent en quelque sorte la classe des problèmes auxquels s'attaquent les systèmes experts.

L'environnement dans lequel un E-programme s'exécute inclut un grand nombre d'objets. Chaque objet appartient à une classe qui est définie en terme d'une liste d'attributs. Les classes d'objets présentent entre elles des relations. Par exemple, une classe peut être définie comme une sous-classe d'une autre classe ou comme une combinaison de plusieurs classes. Des événements peuvent générer ou détruire des objets, ou altérer les valeurs courantes associées aux attributs d'un objet. De plus, chaque utilisateur du système applique ses propres règles pour manipuler les objets et pour réagir aux événements. Les règles sont exécutées sous certaines hypothèses et contraintes. Elles sont basées sur l'expérience de l'utilisateur et sur son appréhension actuel de l'environnement.

Au début, les utilisateurs ont des besoins qui ne sont pas toujours bien exprimés. Les besoins sont imprécis et incomplets. Les utilisateurs prennent de l'expérience au fur et à mesure qu'ils utilisent le système informatique. La spécification initiale est raffinée et améliorée pour inclure des heuristiques sophistiqués et de nouvelles facilités. Différentes perceptions de l'environnement et des besoins spécifiques propres à chaque utilisateur conduisent éventuellement à des connaissances contradictoires ou incompatibles, ce qui obligent les informaticiens à maintenir différentes versions du E-programme. L'altération des spécifications est aussi gouvernée par l'évolution de l'environnement. Des classes d'objets et des événements sont ajoutés, modifiés ou détruits. Les heuristiques deviennent de plus en plus complexes.

Les E-programmes doivent interagir avec les utilisateurs pour de multiples raisons. Deux d'entre elles sont importantes dans cette discussion. Premièrement, pour un E-programme, les aspects qualitatifs de la solution sont plus importants que les aspects quantitatifs. Ainsi, les utilisateurs sont plus confortables si le système informatique peut expliquer et justifier ses conclusions à propos de la solution générée. Deuxièmement, si le système informatique se trouve dans une impasse, il peut interroger l'utilisateur pour lui demander de l'information supplémentaire afin de trouver éventuellement une meilleure solution.

Les systèmes informatiques guidés par un système expert constitue certainement une solution appropriée au problème que nous voulons attaquer. En effet, ils offrent un ensemble de mécanismes qui permettent de capter sous une forme déclarative l'importante quantité de connaissances relatives à l'environnement, d'introduire ou de modifier rapidement et aisément les éléments heuristiques et évolutifs sans détériorer la structure du système informatique, d'exécuter une partie des spécifications sans les traduire dans un langage procédural, de raisonner avec des connaissances imprécises et incertaines, et enfin de dialoguer de façon conviviale avec les utilisateurs pour la recherche ou l'explication d'une solution.

Cette approche de concevoir des systèmes informatiques complexes possède trois avantages significatifs par rapport à l'approche conventionnelle. Premièrement, ils possèdent un plus haut degré de modifiabilité et donc d'adaptabilité vis-à-vis l'environnement et les utilisateurs, car la majorité des modifications ne sont plus faites dans le

code mais à un niveau d'abstraction plus élevé. Deuxièmement, la totalité ou des parties du système sont réutilisables, car elles sont indépendantes de l'environnement, des conditions d'opération et des stratégies heuristiques. Troisièmement, les systèmes qui en résultent apparaissent de plus en plus sophistiqués, car ils incorporent toute la puissance et les propriétés des systèmes experts.

Cette approche apparaît techniquement viable dû aux récents progrès et développements dans les domaines du prototypage, des systèmes experts et des langages de spécifications. Enfin, soulignons qu'une étude détaillée a démontré l'intérêt d'une telle approche pour une classe particulière de E-programmes, soit celle des systèmes répartis [St-Denis 86].

3. Un modèle général pour le développement et la maintenance de systèmes informatiques guidés par un système expert

La Figure 1 illustre de façon schématique un modèle pour le développement et la maintenance de systèmes informatiques guidés par un système expert. La première étape de ce processus consiste à décrire le domaine d'application en termes d'objets, d'événements, de contraintes et de stratégies heuristiques en utilisant, par exemple, le langage RML [Greenspan 84]. Cette description formalise le modèle M et permet de construire une première base de connaissances requise pour l'exécution du système informatique guidé par un système expert.

Dans la deuxième étape du processus, une analyse des besoins exprimés par l'utilisateur permet d'identifier les composantes fonctionelles et les éléments de données du système informatique. Les

données et les fonctions sont raffinées jusqu'à ce qu'elles ne requièrent plus aucune décomposition. A la fin de cette étape, on obtient la spécification formelle du système S.

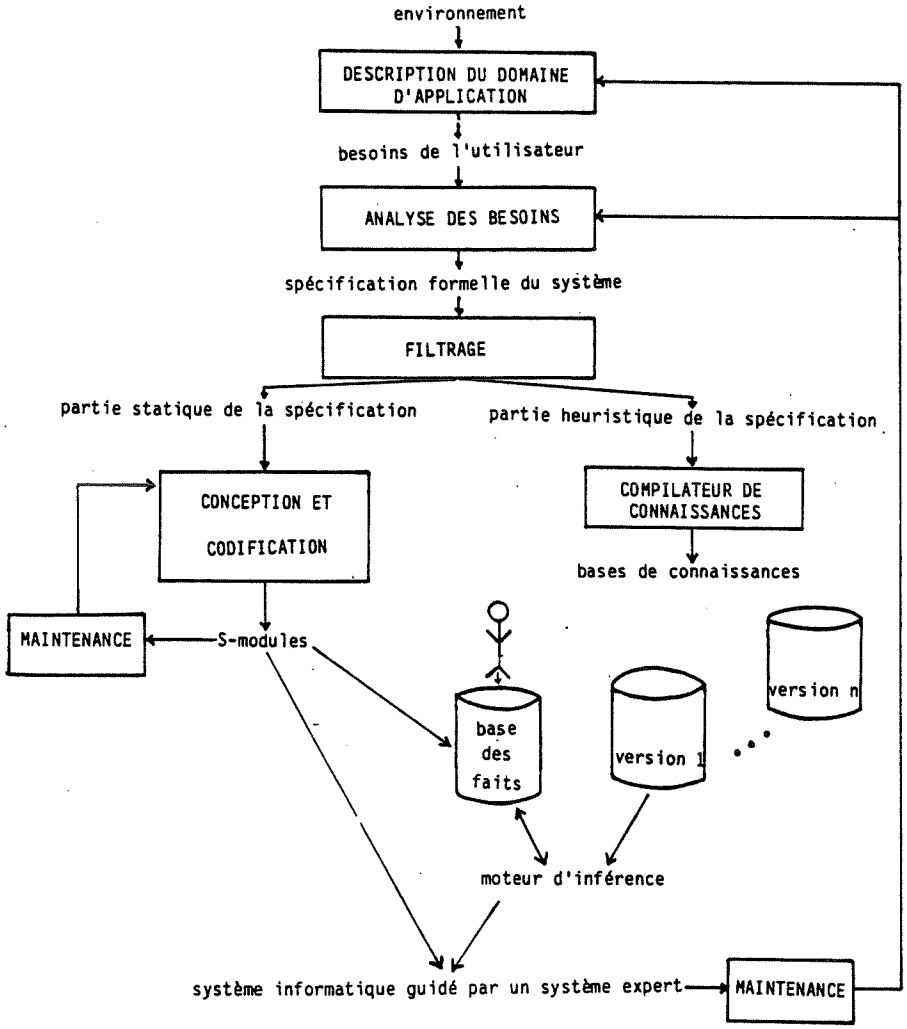

Figure 1 - Modèle de développement et de maintenance

La troisième étape comporte une opération de filtrage. La spécification S est divisée en deux parties. La première partie (S_1) identifie les composantes qui sont des S-modules tandis que la seconde partie (S_2) identifie les E-éléments. Un lien est établi entre un point de décision dans S_1 et une portion des connaissances déclarées dans la première étape. Il s'agit ici d'appliquer le principe de

séparation des mécanismes et des politiques, introduit par Brinch-Hansen [Brinch-Hansen 70], afin d'identifier les points de décision. L'applicabilité de ce principe est rendue nécessaire pour éviter aux concepteurs d'introduire dans la spécification détaillée de la partie procédurale des E-éléments formulés ultérieurement.

Une fois cette étape réalisée, le processus de développement se divise en deux branches parallèles. La branche de gauche représente essentiellement le cycle de vie traditionnel [Boehm 81]. La spécification S_1 est traduite successivement en une spécification architecturale, une spécification détaillée, puis en un ensemble de S-modules écrits dans un langage de programmation procédural. Les techniques de réutilisation de programmes sont possibles, car le système informatique est indépendant des stratégies heuristiques, des conditions d'opérations et des environnements.

La branche de droite représente le processus évolutif de développement des systèmes à base de connaissances [Hayes-Roth 84]. Les connaissances sont compilées sous une forme exécutable par un moteur d'inférence puis mémorisées. Enfin, elles sont vérifiées, et validées par rapport aux intentions des utilisateurs.

La dernière étape consiste à lier le moteur d'inférence et les S-modules via un mécanisme d'éditions de liens pour constituer le système informatique guidé par un système expert. Au moment de l'exécution, le moteur d'inférence prend des décisions à partir de la base des faits mise à jour par l'utilisateur et l'ensemble des S-modules.

Dans ce modèle, la maintenance apparaît à deux endroits différents. Un entretien de nature perfectible et corrective améliore la qualité et la performance, et corrige les défaillances des S-programmes. Cet entretien est fait sur le code. Un second entretien permet d'adapter le système à des changements de l'environnement ou à de nouvelles politiques des utilisateurs. Si ces nouveaux besoins n'entraînent pas de modification de l'architecture du système, les utilisateurs peuvent eux-mêmes modifier la base de connaissances pour améliorer la performance globale du système ou ajouter de nouvelles connaissances pour adapter le système à un nouvel environnement . Il s'agit dans ce cas d'un entretien correctif, perfectible et adaptatif sur les E-éléments seulement. Enfin, si les nouveaux besoins exprimés par les utilisateurs nécessitent l'ajout de nouvelles fonctionnalités au système, l'addition de nouveaux S-modules est inévitable et la structure du système doit être modifiée en utilisant un ensemble de techniques connues et appropriées.

Le problème de versions multiples est partiellement résolu en associant à chaque famille d'utilisateurs un ensemble de connaissances qui leur sont propres. L'activation du bon ensemble de connaissances au moment de l'exécution du E-programme peut s'opérer au niveau de la méta-connaissance d'une seule base de connaissances universelle ou à l'aide d'une sélection d'une base de connaissances spécifique. Cette dernière solution est présentée à la Figure 1.

En construisant les E-programmes de cette façon, nous bénéficions des avantages d'une approche classique et d'une approche moderne. Dans l'approche classique, les informaticiens disposent d'un grand

nombre d'outils qui sont techniquement viables et praticables. De plus, la programmation procédurale apparaît la plus appropriée pour le parc d'ordinateurs actuel et pour la réutilisation de programmes existants. Par contre, l'approche système expert permet une modification plus aisée du système, car les énoncés de la spécification initiale ne passent pas par toute une série de transformations successives qui ne sont pas aujourd'hui complètement systématisées. Les énoncés sont donc simplement déclarés en vrac puis compilés. Cette dernière approche s'apparente aux techniques de prototypage. Finalement, il est intéressant de noter que ce processus reconnaît trois types d'objets qui interviennent dans le développement d'un E-programme: les algorithmes (les S-modules), les données (les structures de données des S-modules et de la base des faits) et les connaissances liées à l'environnement et au domaine d'application (la base de connaissances).

4. Quelques exemples de systèmes informatiques guidés par un système expert

Voici une brève description de trois systèmes informatiques guidés par un système expert. Pour chaque système, nous indiquons le domaine d'application, nous décrivons l'architecture du système et nous présentons les particularités les plus intéressantes.

4.1 HEXSCON

HEXSCON (Hybrid EXpert System CONtroller) est un système expert destiné aux applications en temps réel aussi bien dans le domaine

militaire que dans le domaine industriel [Lattimer Wright 86]. Le système HEXSCON comporte quatre parties principales: le moteur d'inférence, la base de connaissances, le gestionnaire de la base des connaissances et un système d'exploitation qui gère les différentes tâches, les capteurs et les effecteurs. Le gestionnaire et la base des connaissances résident sur un ordinateur de grande taille. Les connaissances sont représentées à ce niveau dans un langage semblable à l'anglais. Cette connaissance est compilée, puis transférée dans un micro-ordinateur où s'exécutent le moteur d'inférence et le système d'exploitation en temps réel. Ce système opère dans un environnement dynamique. Les décisions doivent être prises pour permettre une réaction immédiate aux événements. Les signaux enregistrés par des capteurs sont interprétés puis ensuite analysés pour identifier les types d'objets et les événements connus de la base de connaissances. Les propriétés temporelles et spaciales ainsi que les relations qui existent entre les objets et les événements conduisent à des raisonnements progressifs dont la complexité augmente en fonction du temps disponible. De plus, ce système fonctionne raisonnablement et sûrement même à partir de données incertaines et imprécises. Enfin, il s'exécute sur des micro-ordinateurs avec des ressources matérielles limitées.

4.2 Un générateur de code pour compilateurs

Les compilateurs commerciaux développés récemment par la firme Intermetrics Inc. contiennent un système expert pour la génération de code de haute qualité [Haradhvala 84]. Ce type de système comporte un compilateur qui génère des arbres syntaxiques abstraits et un système

expert qui transforme ces structures en une séquence d'instructions exécutables par une machine cible. L'approche système expert est rendue nécessaire, car les machines cibles possèdent des jeux d'instructions sophistiqués et des architectures particulières. Les utilisateurs des compilateurs peuvent aisément adapter le générateur de code en fonction de besoins spécifiques. Ils n'ont pas à connaître la logique du compilateur et des outils de construction de compilateurs.

4.3 YES/MVS

YES/MVS est un système expert qui assiste les opérateurs dans la conduite d'ordinateurs [Ennis 86]. Il fournit des réponses rapides, cohérentes et précises à des situations aussi bien routinières que problématiques. Il réduit et réorganise le flot des messages entre l'opérateur et le système cible. L'architecture du système se compose principalement de quatre machines virtuelles. La machine virtuelle à opérer et trois autres machines qui s'exécutent sur un ordinateur distinct de la machine à opérer. Une première machine virtuelle constitue le système expert. Elle exécute les règles de la base des connaissances en fonction des messages qu'elle reçoit de la machine cible. En réponse aux décisions prises, le système expert retourne des commandes à la machine cible. Aussi, le système expert reçoit de et envoie à l'opérateur du texte. Une deuxième machine virtuelle constitue l'interface entre la machine cible et le système expert. Elle traduit tous les messages de la machine cible dans un format compatible pour le système expert et vice et versa. Finalement, une troisième machine virtuelle fournit une interface de communication entre le système expert et l'opérateur. Elle transmet aussi les

commandes suggérées par le système expert et autorisées par l'opérateur à la deuxième machine virtuelle. Le squelette de systèmes experts OPS5 a été utilisé pour construire YES/MSV. Certains changements ont été apportés à OPS5 pour prendre en considération les problèmes qui surviennent dans un environnement dynamique.

4.4 Autres systèmes en développement

Il existe plusieurs projets en développement qui nécessitent la construction de systèmes informatiques guidés par un système expert. Les systèmes de traitement de texte dans lequel l'expertise des typographes peut être incluse pour formatter du texte constitue un exemple intéressant. Bonham et al. [Bonham 85] entrevoit d'intégrer de l'expertise en écrivant un tel système en LISP. La robotique et l'opération d'avions commerciaux et militaires [Georgeff 86] constituent d'autres exemples d'applications en temps réel.

5. Problèmes particuliers au modèle

Dans cette section, nous dégageons les principaux problèmes inhérents à l'approche décrite dans la troisième section. Mais avant de présenter les différents problèmes et les solutions qui ont été retenues, il nous apparaît important de distinguer entre l'évolution de l'environnement et des événements qui surviennent dans l'environnement. L'évolution de l'environnement se traduit par des changements progressifs à des intervalles de temps macroscopiques. Ces changements sont pris en considération par des modifications apportées à la base des connaissances ou aux S-modules. Il existe aussi des

événements qui entraînent des changements rapides dans la base des faits. Ces changements surviennent à des intervalles de temps microscopiques. Les événements déclenchent des raisonnements au niveau du système expert. Dans ce dernier cas, on parle d'applications qui s'exécutent dans un environnement dynamique. Des problèmes de temps réponse et de cohérence dans le raisonnement doivent être résolus. Pour les applications qui s'exécutent dans un environnement statique, le système expert raisonne à partir de faits qui ne changent pas durant ses raisonnements. Le problème de temps réponse est moins aigu. Le problème de raisonnements incohérents n'existe pas.

5.1 Intégration du système expert

La façon dont le système expert s'intègre à la partie procédurale se traduit dans le contrôle ou la logique du système en général. Dans HEXSCON, le système d'exploitation considère le système expert comme une tâche qui est déclenchée par une interruption logique. Une interruption logique interrompt le raisonnement courant qui est abandonné et démarre un raisonnement sur un nouveau problème. Dans ce cas, on dit que le système expert est fortement intégré à la partie procédurale. Dans les deux autres systèmes, l'exécution du système expert est indépendante de la partie procédurale. Le système expert est faiblement intégré. Dans le cas des compilateurs, le système expert exécute une tâche bien circonscrite dans une séquence d'opérations (analyse lexicale, syntaxique, génération du code, optimisation du code). Dans le cas de YES/MVS, le système expert est indépendant de la machine à opérer et un mécanisme de communication inter-machines est mise en oeuvre pour répondre aux entrées/sorties.

5.2 Représentations incompatibles des données

Les représentations des données sont généralement différentes dans la partie procédurale et dans la partie déclarative. Dans l'approche procédurale, les attributs des objets sont contenus dans des structures de données. Le langage procédural permet de décrire ces structures et de manipuler leurs contenus. Des variables représentent des adresses où sont emmagasinées les données. Les systèmes experts n'utilisent pas la même représentation. Ils manipulent des données emmagasinées sous une forme symbolique. Cette incompatibilité dans les représentations internes des données conduit à définir une interface entre la partie écrite dans un langage procédural et le système expert. Quelques solutions ont été retenues. Par exemple, HEXSCON utilise un traducteur qui permet de passer d'une représentation à une autre. Le traducteur connaît la représentation des données dans chacune des parties et la façon de passer de l'une à l'autre. La solution retenue dans le système YES/MVS est différente. Une machine virtuelle traduit les messages générés par le système MVS en un format compatible pour le système expert. Ensuite les réponses du système expert sont transformées en des commandes de MVS. Contrairement au traducteur de HEXSCON qui travaille sur les données internes du programme, la machine virtuelle traite les entrées/sorties de MVS. Enfin, les compilateurs d'Intermetrics produisent des données interprétables par le système expert qui génère du code optimisé. Aucun échange n'est fait dans le sens inverse (échange unidirectionnel). Dans toutes ces solutions, on contourne le problème, car il n'existe pas une véritable intégration des données de la partie procédurale et des données de la base des faits. Le passage est effectué par une

traduction des données et par une édition de liens appropriée. Une solution plus élégante consiste à définir un langage de haut niveau qui permet la déclaration de structures de données, d'algorithmes et de connaissances. L'association données et faits est assurée par des mécanismes du langage, tout comme l'interaction entre le système expert et les algorithmes. Notons enfin qu'il existe une représentation des connaissances, appelée connaissance procédurale, qui intégre l'approche déclarative et l'approche procédurale [Georgeff 83].

5.3 Outils techniques

A notre connaissance, il n'existe pas véritablement d'outils techniques spécifiques au développement et à la maintenance des systèmes informatiques guidés par un système expert. La méthode présentée dans la section 3 doit être approfondie et des outils logiciels restent à développer.

5.4 Le temps réponse pour les applications en temps réel

Pour les applications en temps réel, le temps réponse est crucial. Le moteur d'inférence ne peut raisonner dans l'hypothèse qu'il dispose d'un grand temps. Plusieurs solutions ont été envisagées pour améliorer la performance des moteurs d'inférence. Par exemple, le moteur d'inférence de HEXSCON utilise de la connaissance compilée et introduit la notion de raisonnement progressif dont la complexité augmente avec le temps. Le système YES/MVS utilise une version modifiée du système OPS5 et s'exécute sur un ordinateur de très grande taille (IBM/370). Les parties droites des règles sont compilées et

les règles sont distribuées selon le sous-domaine d'activité entre plusieurs systèmes OPS5 qui s'exécutent concurremment sur des machines virtuelles distinctes. Pour les applications statiques le temps réponse n'est pas un facteur déterminant et ces mécanismes sont moins importants.

5.5 Raisonnements incohérents

Dans un environnement dynamique, une altération des faits initiaux durant un raisonnement peut conduire à des déductions incohérentes. La solution retenue pour résoudre ce problème dans le système YES/MVS consiste à reconnaître après chaque déduction les contradictions, et à considérer dans ce cas de nouvelles avenues de raisonnement. On appelle raisonnement non monotone ce type de raisonnement [Schor 86].

5. Conclusion

L'intérêt pour les systèmes informatiques guidés par un système expert est récent. Seulement quelques prototypes existent aujourd'hui. Outre la réalisation pratique d'un système informatique guidé par un système expert, cette approche soulève des problèmes techniques qui nécessitent un travail de recherche encore important. Plusieurs problèmes de recherche restent encore ouverts, sinon à découvrir, en particulier:

- l'approfondissement de la méthode présentée dans la troisième section;

-la construction d'outils logiciels pour la conception, le développement et la maintenance de tels systèmes;

-la définition d'un langage de programmation propre aux systèmes informatiques guidés par un système expert.

BIBLIOGRAPHIE

[Balzer 83]

R. Balzer, T.E. Cheatham, Jr., C. Green: Software technology in the 1990's: Using a new paradigm; Computer, 16 (11), 1983, 39-45.

[Boehm 81]

B. W. Boehm: Software Engineering Economics; Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Bonham 85]

M. Bonham, I. H. Witten: Towards distributed document preparation with interactive and non-interactive viewing; INFOR, 23 (4), 1985, 365-388.

[Brinch Hansen 70]

P. Brinch Hansen: The nucleus of a multiprogramming system; Commun. ACM, 13 (4), 1970, 238-241.

[Ennis 86]

R. L. Ennis, J. H. Griesmer, S. J. Hong, M. Karnaugh, J. K. Kastner, D. A. Klein, K. R. Milliken, M. I. Schor, H. M. Van Woerkom: A continuous real-time expert system for computer operations; IBM Journal Research and Development, 30 (1), 1986, 14-28.

[Georgeff 83]

M. P. Georgeff, U. Bonollo: Procedural expert systems; Proceedings of the 8th Int. Joint Conf. on Artificial Intelligence, Karlsruhe, Germany, 1983, 151-157.

[Goergeff 86]

M. P. Georgeff: Planning and reasonning in dynamic worlds; Centre de cours intensifs de l'Ecole Polytechnique de Montréal, 1986.

[Greenspan 84]

S. J. Greenspan: Requirements modeling: a knowledge representation approach to software requirements definition; PhD thesis, Technical report CSRG-155, University of Toronto, 1984.

[Haradhvala 84]

S. Haradhvala, B. Knobe, N. Rubin: Expert systems for high quality code generation; Proceedings of the 1st Conf. on Artificial Intelligence Applications, Denver, CO, IEEE Computer Society, Silver Spring, Md., 1984, 310-313.

[Hayes-Roth 84]

F. Hayes-Roth: The knowledge-based expert system: a tutorial; Computer, 17 (9), 1984, 11-28.

[Lattimer Wright 86]

M. Lattimer Wright, M. W. Green, G. Fiegl, P. F. Cross: An expert system for real-time control; IEEE Software, 3 (2), 1986, 16-24.

[Lehman 80]

M. M. Lehman: Programs, life cycles, and laws of software evolution; Proceedings of the IEEE, 68 (9), 1980, 1060-1076.

[Schor 86]

M. I. Schor: Declarative knowledge programming: better than procedural?; IEEE Expert, 1 (1), 1986, 36-43.

[St-Denis 86]

R. St-Denis: Expert systems for distributed systems; Interfaces in Computing, 3 (3-4), 1986, 217-225.

ÉCOLE POLYTECHNIQUE DE MONTRÉAL
3 9334 00289449 9