Les utilitaires et l'intelligence artificielle, pour un système d'aide à la conception en architecture. Étude d'un système expert orienté objet en CAAO

Dominique Caradant, Jean-Pierre Goulette, Patrick Pérez

323

Ecole d'Architecture de Toulouse . Chemin du Mirail
31057 TOULOUSE Cedex . Tel:(61)40.47.28 Poste 226

LES UTILITAIRES ET L'INTELLIGENCE ARTIFICIELLE POUR UN SYSTEME D'AIDE A LA CONCEPTION EN ARCHITECTURE
- 1985 -

Etude d'un systeme expert orienté objet en CAAO

Dominique Caradant
Jean-Pierre Goulette
Patrick Pérez

RAPPORT FINAL DE RECHERCHE

Rapport final d'une recherche financée par le Ministère de l'Urbanisme et du Logement, Direction de l'Architecture, Sous-Direction de l'enseignement et de la recherche, Secrétariat à la Recherche Architecturale (Décision de subvention du 7 Juillet 1985, n° 66322).

Décembre 1985.

Ecole d'Architecture de Toulouse.
Département de la Recherche.
Laboratoire d'Informatique Appliquée à L'Architecture.
Chemin Aristide Maillol 31100 Toulouse.
Tel : 61 40 47 28 poste 226.

LES UTILITAIRES ET L'INTELLIGENCE ARTIFICIELLE POUR UN SYSTEME D'AIDE A LA CONCEPTION EN ARCHITECTURE
- 1985 -

Etude d'un systeme expert orienté objet en CAAO

Dominique Caradant
Jean-Pierre Goulette
Patrick Pérez

RAPPORT FINAL DE RECHERCHE

Rapport final d'une recherche financée par le Ministere de l'Urbanisme et du Logement, Direction de l'Architecture, Sous-Direction de l'enseignement et de la recherche, Secrétariat à la Recherche Architecturale (Décision de subvention du 7 Juillet 1985, n° 66822).

Décembre 1985.

RESUME

Ce rapport est le compte rendu d'une recherche portant sur les utilitaires et l'intelligence artificielle en C.A.O. en Architecture.

Il expose plus particulièrement les notions de systèmes experts, de représentation de connaissances, de structures de "frames" et de langages orientés objets.

Il donne des exemples de prototypes de systèmes pour la phase initiale de conception en Architecture.
Ces prototypes sont :

- Menon, système expert de reconnaissance de scènes en architecture,
- Kosmigol, analyseur de langage naturel pour un dialogue avec l'architecte,
- Froc, langage alliant la puissance déclarative des frames à la modularité des structures de contrôle des langages orientés objet.

Des sessions d'utilisation de ces systèmes sont présentées.

La conclusion essaie de dégager les axes futurs de recherche en intelligence artificielle et CAO en Architecture.

SOMMAIRE

Introduction 0

1. Systèmes experts et bases de données sémantiques : Le système expert Menon. 4

1.1 Les SGBD intelligents : les BD sémantiques 5
1.2 Les systèmes experts 5
1.3 Les bases de données sémantiques et la CAO 6
1.4 Les systèmes experts et la CAO 6
1.5 Le logiciel "Menon" 9
1.5.1 Le point de vue architecture 9
1.5.2 Le point de vue informatique 10
1.5.3 Exemples d'utilisation 13
1.6 Conclusion 22

2. La représentation des connaissances : Le logiciel "Kosmigol" 23

2.1 Introduction 23
2.2 Les concepts abstraits et les objets concrets 24
2.3 La représentation des connaissances 24
2.4 Les réseaux sémantiques 25
2.4.1 Principe 25
2.4.2 Raisonnement avec la connaissance 26
2.5 Les "frames" ou schémas 26
2.5.1 Principe 26
2.5.2 Comment raisonner avec les frames 27
2.6 La représentation des objets architecturaux 27
2.7 Le logiciel "Kosmigol" 28
2.7.1 Analyse des différents constituants 30
2.7.1.1 Analyseur lexical 30
2.7.1.2 Analyseur syntaxique 30
2.7.1.3 Analyseur de règles 31
2.7.1.4 Analyseur d'affirmations 31
2.7.1.5 Analyseur de requêtes 31
2.7.1.6 Moteur d'inférences 32
2.7.2 Utilisation 32
2.7.3 Un exemple de session 35

sommaire

3. Un langage à structure de "frame" et orienté objets : le langage "Froc" 50

3.1 Rappels 50
3.1.1 Différentes possibilités de la structure déclarative 50
3.1.2 Les interfaces avec l'environnement 51
3.1.3 Mise en oeuvre de notre premier prototype 52
3.1.4 Les nouvelles formes d'héritage 52
3.2 Les langages objet 53
3.2.1 Les différents types de langages objet 53
3.2.2 La transmission de messages 55
3.2.3 La représentation des connaissances sous le formalisme objet 56
3.2.4 Limites de ces langages 58
3.3 Le langage Froc 59
3.3.1 La structure du langage 59
3.3.2 Syntaxe des instructions 60

4. Exemple d'implémentation d'un prototype en langage Froc 62

4.1 Introduction 62
4.2 Exemple de manipulation d'objets architecturaux 63
4.2.1 Les héritages dûs à l'utilisation d'envois de messages 63
4.2.2 Les héritages dûs à la spécialisation 65
4.3 Exemple de session

Conclusion 72

Bibliographie 74

Annexe 77

Annexe 1 : le source de "Menon" 77
Annexe 2 : le source de "Kosmigol" 94

INTRODUCTION

Ce texte est le rapport final d'une recherche financée par le Secrétariat de la Recherche Architecturale depuis 1983.

Le titre général de l'étude est "Les utilitaires et l'Intelligence Artificielle pour un Système d'Aide à la Conception en Architecture". Cette phase, qui est la dernière, concerne plus particulièrement l'étude d'un système expert orienté objets.

En se reportant aux rapports annuels précédents, on remarquera la continuité de notre démarche qui, partant des techniques élémentaires de l'Intelligence Artificielle (les systèmes experts), aboutit aujourd'hui à des modèles plus sophistiqués intégrant les "frames" et les langages orientés objet.

Cette sophistication s'accompagne, nous l'espérons, d'un rapprochement de plus en plus grand avec les méthodes et les savoir-faire invoqués en phase de conception initiale par les architectes.

Nos études théoriques dans le domaine sont exposées ici, mais un des principes de notre Laboratoire a toujours consisté à mettre au point un prototype illustrant nos considérations théoriques à chaque phase de l'avancement des travaux.

C'est pourquoi nous présentons ici chaque étape de notre démarche (en rappelant éventuellement les étapes précédentes), ainsi que le prototype qui en est issu : "Menon", pour un système expert de reconnaissance de scènes en architecture, "Kosmigol" pour un système expert orienté objets d'analyse de dialogues avec un architecte, et "Froc" pour un langage à structure de "frames" orienté objet.

Les possibilités de chacun de ces logiciels sont exposées, ainsi que des recherches futures dont ils ouvrent la voie.

Des exemples de sessions sont chaque fois donnés, permettant au lecteur de se faire une idée concrète du fonctionnement externe et interne.

Les listings en Lisp-Tangram et Lisp de deux de ces systèmes sont donnés en annexe.

1.SYSTEMES EXPERTS ET BASES DE DONNEES SEMANTIQUES : LE SYSTEME EXPERT "MENON"

Nous adopterons comme définition du terme CAO : "ensemble des techniques informatiques qui permettent l'exploitation, la mémorisation et la réexploitation du savoir-faire du concepteur". (F. Macias). C'est dans cette optique et avec la volonté de fournir des outils capables de répondre à ces objectifs que nous avons réalisé cette étude.

Lorsque l'objet à concevoir est complexe, et ce sera toujours le cas lorsque l'on utilisera la CAO, le concepteur ne pourra en assurer la synthèse globale. Il va décomposer ce produit en sous-ensembles qui seront assemblés par la suite. Cette approche conceptuelle fait appel à la notion de décomposition hiérarchique d'un produit, c'est à dire à la structuration arborescente des éléments constitutifs du projet final.

Cette approche engendre deux démarches de résolution : la démarche ascendante et la démarche descendante.

La démarche descendante ou analytique consiste à définir des spécifications, niveau par niveau, en partant de l'objet et en descendant vers les constituants élémentaires.
La démarche ascendante consiste en l'élaboration à partir des constituants physiques élémentaires des blocs plus importants pour aboutir "in fine" aux spécifications souhaitées.

Dans de nombreux domaines où la conception nécessite une part importante de créativité, (architecture, mécanique, génie civil), les deux démarches sont étroitement imbriquées au cours d'un processus de conception itératif, et fortement heuristique.

En effet, dans ce type d'activité, l'absence de règles formelles de conception nécessite, à tous les niveaux, une adéquation entre les spécifications propres à ce niveau, la réalisation, et les spécifications du niveau supérieur. Il s'en suit que les outils d'aide à la conception devront prendre en compte ce facteur dans leur constitution, pour permettre ce continuel va-et-vient entre les divers niveaux de réalisation et de spécifications.

Ce modèle général de conception a une certaine audience en architecture car il rejoint une démarche appelée "démarche struturaliste combinatoire" : "Dans une réflexion

par système, les échelles de conception s'imbriquent comme des poupées gigognes et l'ensemble possède la totalité de la complexité des éléments", faisait remarquer Michel Duplay (DUPLAY-82). Nous aurons l'occasion de revenir sur cette approche quand nous étudierons les caractéristiques d'un système intelligent d'aide à la conception en architecture.

L'intelligence artificielle actuellement se focalise sur deux fronts sensiblement distincts mais qui ont beaucoup de points en commun : les systèmes experts et les systèmes de gestion de bases de données.

1.1 Les SGBD intelligents : les bases de données sémantiques

Les SGBD sont des systèmes capables de gérer un grand nombre de données et de les retrouver en employant des techniques se basant sur des connaissances de bas niveau (dépendance fonctionnelle, relations simples, fonctions d'agrégats...).

On peut leur demander de prendre en charge quelques contraintes d'intégrité (telles que le type des valeurs d'une donnée), une dépendance fonctionnelle, et quelques automatismes.

Les bases de données sémantiques s'attachent surtout à la plausibilité des données en intégrant les problèmes des valeurs dépendantes, des cardinalités, et de la dynamique. Une autre orientation des SGBD sémantiques concerne les formes sémantiques de la base, et en particulier la transformation des données selon des règles de déduction (processus d'inférence).

1.2 Les systèmes experts

Les systèmes experts sont des systèmes hautement spécialisés dans des domaines particuliers : diagnostic médical, prospection minière, etc... Ils sont caractérisés par un faible volume de données, mais un grand nombre de règles du type :

CONDITION ---> ACTION.

La qualité d'un système expert au niveau décisionnel est liée intimement à l'importance du volume de règles employées. Il génère des décisions de haut niveau qui n'ont rien à voir avec celles employées par un SGBD sémantique.

Actuellement les systèmes experts tentent, par des méthodes de structuration des données et des connaissances complexes, de manipuler des grandes quantités d'informations. Comme l'a très bien montré Mitchell (MITCHELL-83) dans le domaine de la CAO ces deux approches se retouvent : "Des différents domaines où apparaissent les systèmes experts, la conception présente, sans doute, le plus grand problème au niveau de la gestion des données".

1.3 Les bases de données sémantiques et la CAO

Les SGBD de type relationnel sont, semble-t-il, les plus appropriés pour décrire un objet en cours de conception. En effet, les informations à manipuler sont de deux groupes :

- les entités qui représentent les objets du monde réel

- les relations qui décrivent les liens entre ces objets.

Actuellement apparaît un troisième groupe : les lois qui concernent la connaissance que l'on a des interactions entre les objets (contraintes face à une situation, contraintes des objets entre-eux). C'est au niveau de ces lois que se situe toute la sémantique de la base : le système ROSALIC (CHOL-82) en est un bon exemple, le lecteur intéressé pourra se référer au rapport LIZA-83 (CARADANT-84).

1.4 Les systèmes experts et la CAO

Les systèmes experts, en tant que manipulateurs de connaissances de haut niveau pour la résolution de problèmes, sont assez peu employés jusqu'a présent en CAO, à part dans quelques domaines particuliers :

-Implémentation de circuits électriques.
-Système prototype de résolution de problèmes en CAO tels que TROPIC (LAT-77)
-Systèmes experts d'aide au diagnostic de type EMYCIN (Van Melle-81)

Dans le domaine de l'implémentation de circuits électriques le problème est sensiblement simplifié : il est assez facile de représenter l'espace des états possibles du système, ceux-ci étant facilement formalisables. Les regles de production et les heuristiques employées sont connues depuis longtemps, elles sont très proches des connaissances de type technique.

Dans TROPIC, c'est un système général orienté vers la résolution de problèmes liés à la conception qui est mis en place. On exploite une méthode de raisonnement capable de générer des solutions à l'aide de règles de production évoluées.

La représentation des connaissances est, par contre, beaucoup plus pauvre et conduit à un système inexploitable dans une situation concrète (le système n'a pas assez d'à priori sur les objets qu'il manipule, il est difficile de les exprimer cet avec des règles).

La troisième approche est celle qui est la moins spécifique à la CAO, car elle se base sur les premiers systèmes experts orientés vers l'aide au diagnostic. Les systèmes experts qui ont été réalisés récemment avec cette méthode dans le domaine de l'architecture (Cullen-83) repose aussi sur cette démarche.

L'intérêt de ce genre de systèmes est actuellement surtout pédagogique. On peut voir alors l'influence de règles de production particulières dans des domaines où les faits (situation réelle) et les règles (connaissances générales qui permettent de manipuler ces faits) ne sont jamais nettement séparés. L'intérêt pédagogique devient alors évident : nous avons affaire à un outil de simulation intelligent dans sa démarche et dans ses conclusions.

Ce genre de système permettrait une étude plus approfondie des domaines dits "non scientifiques". Par exemple en architecture on peut penser à des systèmes d'aide à la décision dans le domaine de l'architecture solaire passive : choix de différentes techniques possibles en fonction de critères extérieurs de type climatique, architectural, sociologique...

Cette approche ne peut pas être réalisée à l'aide d'un système informatique classique car le domaine n'est pas complètement formalisable : il n'y a pas de modèles de résolution, même si des algorithmes de calcul sont nécessaires pour valider les choix techniques. On peut penser, dans ce cas, qu'il est intéressant d'avoir un système d'aide à la décision basé sur des connaissances expertes.

Nos précedents travaux de recherche nous ont amené à nous intéresser plus particulièrement aux possibilités d'assistance informatique dans les premiers instants d'une démarche de conception en architecture. Il ne s'agit donc pas, à proprement parler, d'étudier l'assistance informatique telle qu'elle est définie dans la plupart des logiciels de CAO en architecture existants (une assistance à

l'instrumentation du projet), mais bien plutôt de poser les bases d'un système souple et non contraignant pouvant être utilisé par l'architecte dès les premiers esquisses du projet architectural. Au stade de l'esquisse, le concepteur manipule en effet des éléments graphiques imprécisement dessinés : son but n'est pas alors de représenter exactement le bâtiment, mais seulement "d'esquisser" rapidement sa morphologie globale à partir de signifiants malléables. Nous ne nous étendrons pas sur la description de la nature dialectique du processus de conception ; nous retiendrons seulement un fait essentiel : il est nécessaire que les premiers dessins de l'acte de conception puissent être réalisés facilement et rapidement sans aucune contrainte quant à la précision des tracés et des signifiants manipulés.

Une remarque s'impose : les premières esquisses d'un projet architectural sont évidemment plus difficilement "déchiffrables" que ses représentations ultérieures. Les informations relatives au bâtiment y sont transcrites dans un code simpliste connu du concepteur et de ses proches collaborateurs (nous insistons sur le fait qu'il s'agit moins là de représenter sans ambiguité le bâtiment que d'imprimer les différentes traces d'une pensée conceptuelle). Ainsi un même élément d'architecture peut avoir, au fur et à mesure de l'avancement du projet, diverses représentations pour finalement être dessiné exactement selon les normes du rendu traditionnel d'architecture.

Un problème se pose alors pour les logiciels de CAO : comment permettre et gérer la multiplication des représentations d'un même élément ? Une solution possible (adoptée par la plupart des logiciels de CAO existants) est de ne permettre (et de ne gérer) qu'une seule représentation par élément (celle du rendu traditionnel). Ce type de réponse (qui est en fait une manière d'éluder le problème) nous laisse dans l'incapacité de fournir une assistance intéressante : c'est en effet au stade où les éléments d'architecture sont "incorrectement" dessinés (étape de gestation du projet) qu'une assistance à la conception serait souhaitable et pleinement efficace. Une autre solution, consistant à fournir au système la totalité des représentations possibles pour un élément (et des mécanismes de gestion de ces représentations, si toutefois ceci reste dans les limites du possible) serait sans aucun doute très complexe (et inintéressant) à réaliser et difficilement exploitable. Nous avons donc décidé d'orienter nos recherches vers la définition d'un système ne possédant, d'une manière interne, qu'une représentation par élément, mais permettant au concepteur d'utiliser diverses

représentations pour un même élément. Un tel système devra donc être capable de reconnaître un élément à travers ses diverses représentations (par le biais d'informations de type contextuel et non iconographique), et de lui associer les attributs correspondants. Cet objectif de recherche nous a amenés à l'étude des mécanismes de reconnaissance de formes issus des techniques de résolution de l'intelligence artificielle. Un protoype d'un tel système à été mis au point au laboratoire LI2A sous la forme d'un "mini système expert" visant à réduire la complexité des problèmes de communication homme-machine dans les logiciels de CAO en architecture.

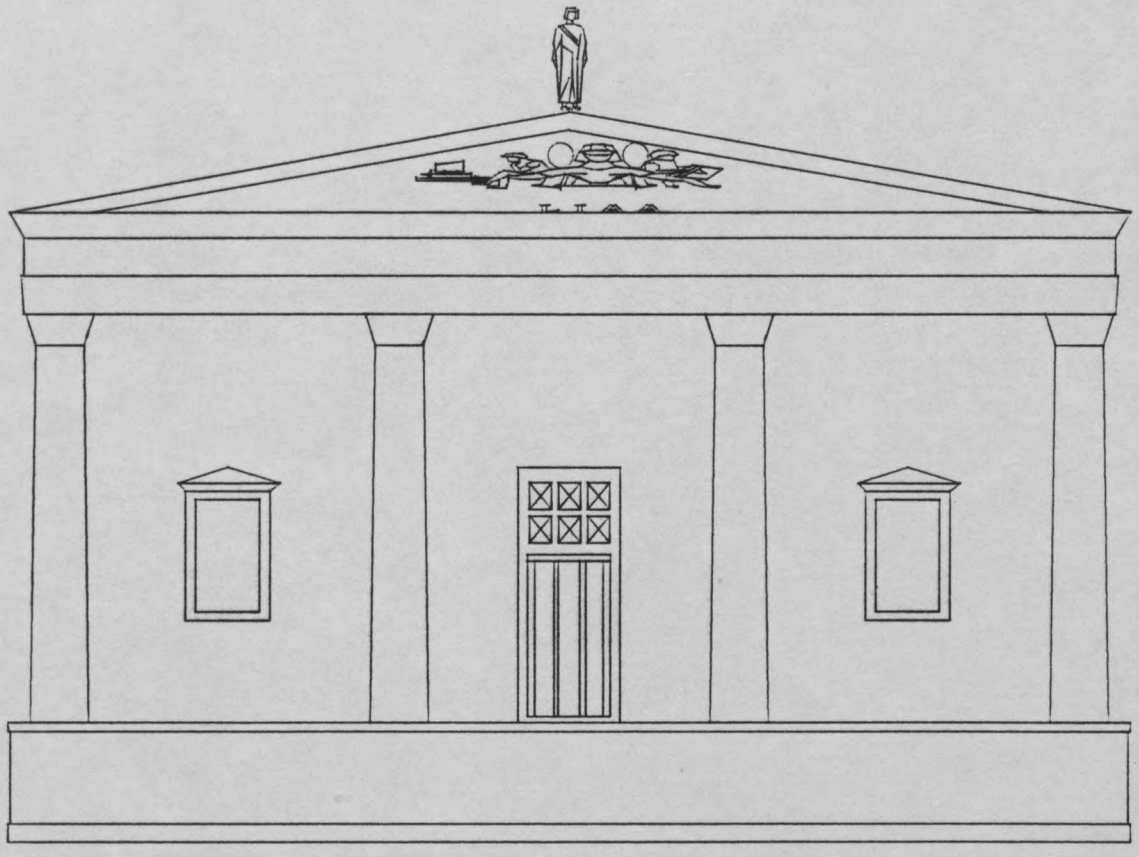
Ce logiciel, que nous avons nommé "Menon", nous a de plus permis de tester sur ce prototype les mécanismes d'inférences liées aux systèmes experts.

1.5 Le logiciel Menon

1.5.1 Le point de vue architecture

Situons tout d'abord le cadre architectural dans lequel opère Menon. Nous avons volontairement limité les possibilités d'analyse de Menon à un cas très particulier : les façades des porches de l'architecture classique française telles qu'elles ont été définies par J.N.L. Durand dans son traité du début du XIX° siècle (DURAND-1981). Cette limitation nous semblait nécessaire à la réalisation d'un prototype simple pouvant constituer le point de départ d'une recherche future visant à étendre les possibilités d'analyse de Menon. Ainsi notre étude, menée aussi bien sur le plan architectural qu'informatique (un équilibre des préoccupations qu'il est parfois difficile de maintenir...) était exempte de toute complexité particulière susceptible de polariser notre attention vers les difficultés de l'implémentation informatique. Nous avons donc choisi un problème simple, mais nous l'avons posé en termes d'architecte sans nous soucier de l'implémentation future. Il s'en suit que le logiciel Menon, s'il répond à nos objectifs d'architecte, ne constitue certainement pas un paradigme des techniques de résolution de l'informatique. Optimiser et étendre les possibilités du logiciel Menon, par l'étude comparative des techniques de programmation de l'intelligence artificielle (évaluées en termes de performance par rapport au but architectural recherché), fera l'objet d'une étude future.

Nous posons donc le problème en termes d'architecte. Les éléments d'architecture rentrant dans la composition d'un porche sont : les colonnes, le ou les entablements (dans le cas de deux rangés de colonnes superposées), le fronton, le tympan, la statue (posée au sommet du fronton), les croisées, les portes et, éventuellement le piedestal (les colonnes du porche pouvant reposer directement sur le socle de l'édifice). Un porche est donc défini par combinaison de ces éléments (la combinaison minimum regroupant deux colonnes et un entablement). Nous donnons un exemple (pris parmi d'autres) d'un tel porche :



Il nous faut ensuite définir les règles permettant de reconnaître les éléments constitutifs du porche. Nous avons choisi d'orienter tout d'abord le processus d'analyse de scènes vers la reconnaissance des éléments porteurs et des éléments portés (que nous nommerons éléments soutenus dans la suite de ce texte). La première étape de l'analyse sera donc d'établir (et de mémoriser) les relations porteur et soutenu entre tous les éléments.

Nous définissons avant tout une méta-règle générale : une règle ne s'applique qu'aux éléments non encore reconnus. Ainsi, les éléments totalement identifiés étant soustraits au mécanisme d'inférence, certaines règles pourront procéder par élimination.

Nous avons défini la relation porteur comme suit :

Règle 1
Si l'élément étudié est dessous un autre élément et
est en contact avec cet élément
Alors l'élément étudié est porteur de cet élément.

De même nous avons défini la relation soutenu :

Règle 2
Si l'élément étudié est dessus un autre élément et
est en contact avec cet élément
Alors l'élément étudié est soutenu par cet élément.

La pertinence de ces deux règles est bien entendu limitée au cas précis qui nous intéresse... Nous admettrons, pour la suite du processus, que ces relations constituent une base de données propre à chaque élément et disponible pour la définition d'autres règles.

Nous avons maintenant suffisament d'informations pour pouvoir orienter le processus vers la reconnaissance des éléments colonnes. En effet, l'élément colonne est (parmi l'ensemble des éléments définis ci-dessus) le seul qui est à la fois porteur et "debout" (très allongé dans le sens vertical si l'on préfère). Donc :

Règle 3
Si l'élément étudié est porteur (quelque soit l'élément
porté) et
est debout
Alors l'élément étudié est une colonne

Il nous semble maintenant naturel de définir les règles relatives aux deux éléments qui encadrent verticalement la colonne : le piedestal et l'entablement. L'un porte les

colonnes, l'autre est soutenu par elles. Un petit problème se pose dans le cas d'une superposition de colonnes : l'élément d'articulation entre les deux rangées de colonnes porte bien des colonnes mais n'est pas un piedestal. Nous posons donc deux conditions à la reconnaissance d'un élément piedestal :

Règle 4
Si l'élément étudié est porteur d'au moins une colonne et n'est pas soutenu par une colonne
Alors l'élément étudié est un piedestal.

Nous pouvons alors définir l'entablement et le fronton :

Règle 5
Si l'élément étudié est soutenu par au moins une colonne
Alors l'élément étudié est un entablement.

Règle 6
Si l'élément étudié est soutenu par l'entablement et couvre tout l'entablement
Alors l'élément étudié est un fronton.

Viennent ensuite les définitions des règles relatives au tympan et à la statue :

Règle 7
Si l'élément étudié est au-dessus de l'entablement et est situé dans les limites verticales d'un fronton
Alors l'élément étudié est un tympan.

Règle 8
Si l'élément étudié est soutenu par un fronton
Alors l'élément étudié est une statue.

Il nous reste maintenant à établir les règles de définition des deux autres types d'élément : les portes et les croisées. Nous décidons de définir une règle intermédiaire : à ce stade du processus, les éléments restant à reconnaître et situés au-dessus du piedestal et (si celui-ci est absent de la composition) au-dessous de l'entablement, sont des ouvertures.

Règle 9
Si l'élément étudié est au-dessus d'un piedestal ou est au-dessous de l'entablement
Alors l'élément étudié est une ouverture.

La porte sera caractérisée par le fait qu'elle "découpe" le mur jusqu'au niveau où repose les colonnes (jusqu'au socle ou jusqu'au piedestal) :

Règle 10
Si l'élément étudié est une **ouverture** et
est au **même niveau inférieur qu'une colonne**
Alors l'élément étudié est une **Porte.**

Les croisées seront définies comme les ouvertures restantes (les portes sont éliminées par la méta-règle générale) :

Règle 11
Si l'élément étudié est une **ouverture**
Alors l'élément étudié est une **croisée.**

L'ensemble des ces règles constitue bien entendu une des multiples possibilités pour la définition (et la reconnaissance) des éléments rentrant dans la composition de notre porche. Notre souci n'étant pas la recherche de l'exhaustivité mais la réalisation d'un prototype, nous nous contenterons des onze règles définies ci-dessus pour articuler notre processus d'analyse de scènes.

1.5.2 Le point de vue informatique

Il nous faut maintenant concevoir et réaliser un moteur d'inférence capable d'utiliser l'ensemble des règles que nous venons de définir. Nous avons plusieurs contraintes :

- Le moteur d'inférence doit pouvoir manipuler des variables (ordre 1)

- La conclusion de l'activation d'une règle doit être mémorisée dans une base de données propre à chaque élément (cette conclusion pouvant être utilisée ultérieurement par d'autres règles).

- Les règles doivent être employées dans l'ordre dans lequel elles ont été définies.

- Un élément reconnu devra être ôté de la liste des éléments à reconnaître (certaines règles procédant par élimination).

De plus il nous faut bien entendu écrire l'ensemble des procédures permettant l'établissement des relations (nous emploierons dans la suite de ce texte le terme "slot" par référence aux langages objets) entre éléments : "dessous", "en contact avec", "dessus", "debout", "couvre tout", "situé dans les limites de", "au même niveau inférieur que". Nous utiliserons, pour écrire notre logiciel, le langage de programmation Tangram (interprète Lisp interfacé avec un editeur graphique 2D ; voir LI2A-1984). Celui-ci nous permet en effet de définir et de manipuler des atomes graphiques, tout en étant capable de fournir certaines informations sur le positionnement et la géométrie de la représentation graphique de ces atomes.

Nous regrouperons l'ensemble des éléments constituant le porche à étudier dans une liste d'atomes (un atome par élément) possédant chacun une propriété graphique mémorisant le dessin de l'élément. De plus nous attacherons à ces atomes une expression fonctionnelle lui permettant de constituer et d'explorer "lui-même" sa propre base de données pour répondre à toute requête concernant celle-ci. Nous donnons le squelette de cette expression :

Si la réponse à la requête est présente dans la base de données de l'élément,
Alors renvoyer cette réponse
Sinon, Si la requête concerne un slot,
Alors
    Calculer la réponse (par le biais de la définition fonctionnelle du slot),
    Insérer cette réponse dans la base de données de l'élément (à la propriété "slot"),
    Renvoyer cette réponse
Sinon, renvoyer une liste vide.

Ainsi les slots de chaque élément ne seront calculés qu'au plus une fois. Par exemple, une requête du type
(<ELEMENT-1> DESSOUS)
retournera toujours une liste d'éléments situés en dessous de ELEMENT-1 (celui-ci sera donc dessus chaque élément de la liste retournée). De plus les différents slots de chaque élément ne seront calculés que si nécessaire (c'est à dire si une requête concernant ce slot est nécessaire à l'activation d'une règle ; les règles définissent les colonnes n'utilisant que les slots "dessous", "dessus", "en contact avec", "debout", les autres slots seront ignorés pour les éléments ayant été reconnus comme colonnes.).

Le résultat de l'activation d'une règle sera, au même titre que le résultat d'un calcul de slot, introduit dans la base de données de l'élément. Une requête du type
(<ELEMENT-1> PORTEUR)
pourra donc être formulée par le système (ou l'opérateur) après l'activation de la règle 1 et renverra la liste (éventuellement vide) des éléments portés par ELEMENT-1.

Le moteur d'inférence en lui-même consiste simplement à appliquer chaque règle sur chaque élément non encore reconnu (création de requête pour chaque condition de la règle et soumission de cette requête à chaque élément). L'ordre des règles est bien entendu respecté, le moteur procède par chaînage avant, en régime irrévocable, dans le cadre d'une logique monotone (le dessin n'évoluant pas pendant l'activation des règles, chaque propriété établie reste valide). Nous n'étudierons pas précisement le reste du système, nous préférons fournir, en annexe, le listing commenté du source de Menon, écrit en Lisp Tangram (cf. ci-dessus).

Dans ce prototype, nous avons exploité le résultat de l'analyse en implémentant des procédures permettant de remplacer le dessin de chaque élément ébauché par l'utilisateur par le tracé exact de l'élément défini en bibliothèque. Ainsi, l'opérateur dessine tout d'abord une ébauche de son porche, active le mécanisme de reconnaissance des éléments et peut ensuite obtenir un dessin exact de son porche (dessin composé à l'aide des éléments définis en bibliothèque ; ces éléments sont placés automatiquement sur le dessin à la position et aux dimensions des éléments ébauchés).

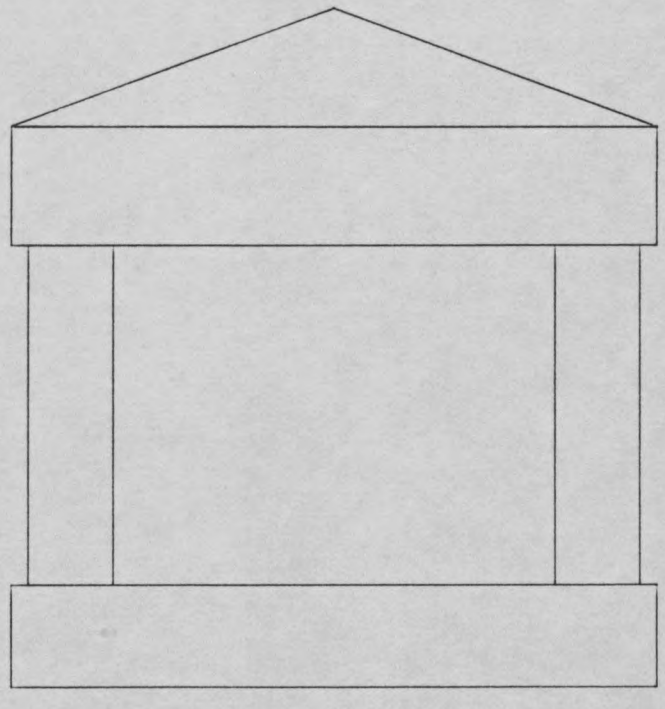
Une des utilisations possibles de cet utilitaire pourrait être son intégration dans l'éditeur d'un programme de DAO, cet éditeur devenant ainsi "intelligent".

1.5.3 Exemples d'utilisation

Nous présenterons deux exemples d'utilisation. Les commentaires seront placés entre ";".

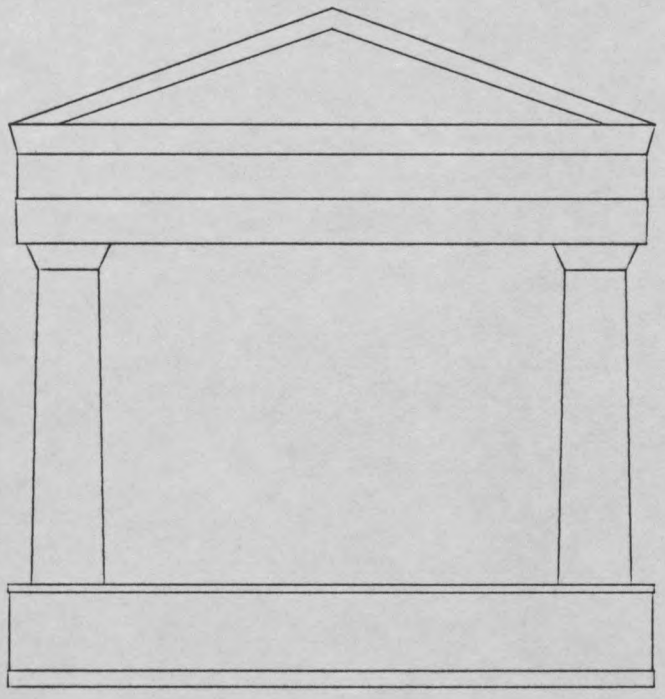
- Premier exemple

```
;chargement du programme;
(LOAD 'MENON.LSP)
Eval :  Fichier Chargé

;définition des éléments constituant le porche.
L'utilisatuer accéde à l'éditeur graphique pour dessiner
chaque élément;
(DEFELT)

Eval : T
;nous présentons l'ébauche finale;
```



```
(EXPERTISE)
ON Y VA...
(R1(DESSOUS ?)ET(TOUCHEY ?)(-> PORTEUR ?))
R1 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R2(DESSUS ?)ET(TOUCHEY ?)(-> SOUTENU ?))
R2 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R3(PORTEUR)ET(DEBOUT)(->T COLONNE))
TIENS, UNE COLONNE !!!
TIENS, ENCORE UNE COLONNE !!!
R3 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R4(PORTEUR COLONNE)ET(NON SOUTENU COLONNE)(->T PIEDESTAL))
TIENS, UN PIEDESTAL !!!
R4 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R5(SOUTENU COLONNE)(->T ENTABLEMENT))
TIENS, UN ENTABLEMENT !!!
R5 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R6(SOUTENU ENTABLEMENT)ET(TAILLEX ENTABLEMENT)(->T FRONTON))
TIENS, UN FRONTON !!!
R6 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...

Eval : T

;affichage des résultats de l'expertise;
(AFFICHE)

;le programme efface l'écran, puis redessine les éléments un
par un en affichant leur nom;
FRONTON
PIEDESTAL
ENTABLEMENT
COLONNE
COLONNE

Eval : T

;chargement des fichiers bibliothèque. Calcul du
positionnement et des dimensions des nouvelles
représentations graphiques;
(AU-PROPRE)

;les deux dessins du porche (l'ébauche et la représentation
exacte), sont superposés;
Eval : T
```

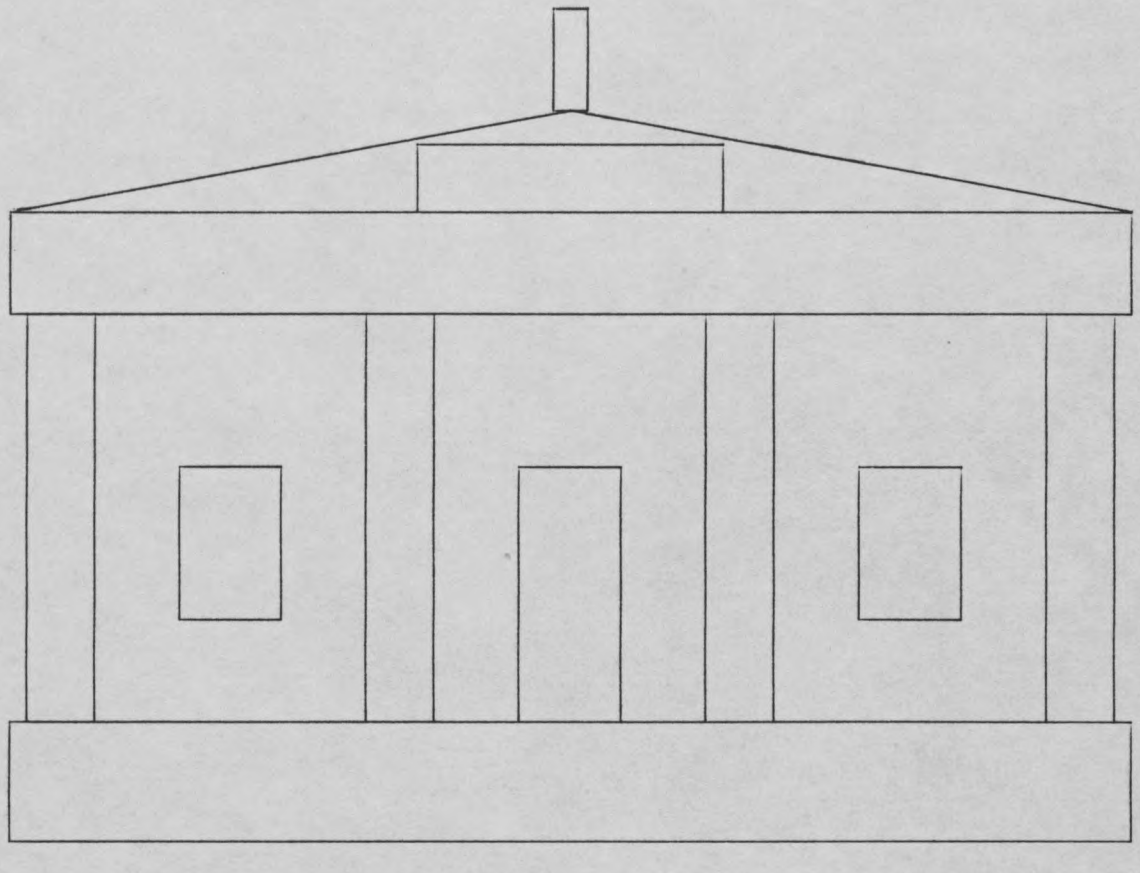
```
;Affichage du nouveau dessin du porche, élimination de
l'ébauche;
(RENDU)

Eval : T
```



- Deuxième exemple

(DEFELT)

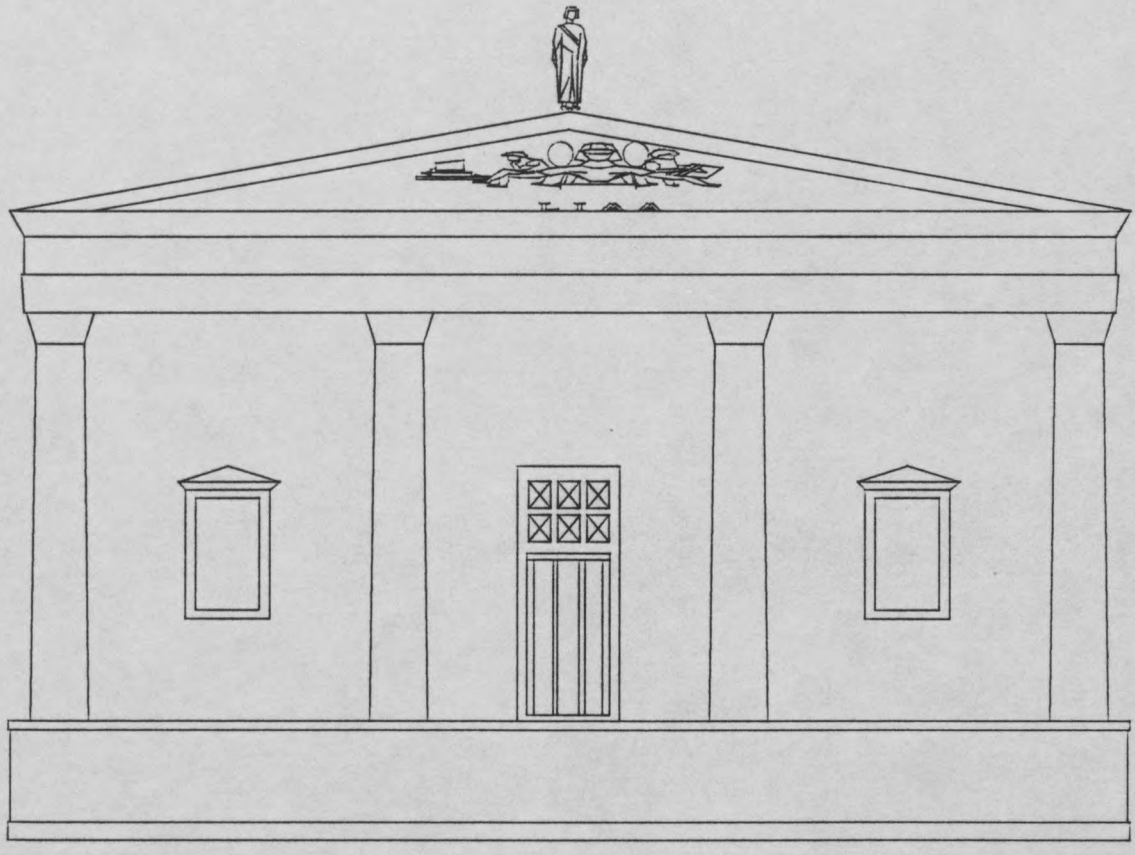
Eval : T



```
(EXPERTISE)
ON Y VA...
(R1(DESSOUS ?)ET(TOUCHEY ?)(-> PORTEUR ?))
R1 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R2(DESSUS ?)ET(TOUCHEY ?)(-> SOUTENU ?))
R2 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R3(PORTEUR)ET(DEBOUT)(->T COLONNE))
TIENS, UNE COLONNE !!!
TIENS, ENCORE UNE COLONNE !!!
TIENS, ENCORE UNE COLONNE !!!
TIENS, ENCORE UNE COLONNE !!!
R3 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R4(PORTEUR COLONNE)ET(NON SOUTENU COLONNE)(->T PIEDESTAL))
TIENS, UN PIEDESTAL !!!
R4 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R5(SOUTENU COLONNE)(->T ENTABLEMENT))
TIENS, UN ENTABLEMENT !!!
R5 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R6(SOUTENU ENTABLEMENT)ET(TAILLEX ENTABLEMENT)(->T FRONTON))
TIENS, UN FRONTON !!!
R6 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R7(DESSUS ENTABLEMENT)ET(DANSY FRONTON)(->T TYMPAN))
TIENS, UN TYMPAN !!!
R7 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R8(SOUTENU FRONTON)(->T STATUE))
TIENS, UNE STATUE !!!
R8 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R9(DESSUS PIESDESTAL)OU(DESSOUS ENTABLEMENT)(-> OUVERTURE T))
R9 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R10(OUVERTURE)ET(NIVEAUBAS COLONNE)(->T PORTE))
TIENS, UNE PORTE !!!
R10 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...
(R11(OUVERTURE)(->T CROISEE))
TIENS, UNE CROISEE !!!
TIENS, ENCORE UNE CROISEE !!!
R11 APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...

Eval : T


(AU-PROPRE)

Eval : T
```

(RENDU)

Eval : T



1.6 Conclusion

Le lecteur a pu voir les différentes techniques de l'IA dans le processus de conception : les Systèmes Experts et les Systèmes de Gestion de Bases de Données sémantiques. Les SGBD sont d'une aide précieuse pour la gestion de la cohérence du projet, les systèmes experts, eux, s'intéressent plus particulièrement à la proposition de solutions. Ces deux aspects de l'IA présentent pourtant des limites :

-Les SGBD sémantiques basés sur le modèle Entité-Relation sont trop pauvres en a-priori culturels simples et nécessitent la mise en place d'un modèle de résolution général beaucoup trop lourd souvent basé sur la logique du premier ordre. Dans ce cas l'approche déclarative pure est trop limitée.

-Les systèmes experts classiques posent le problème de la représentation des connaissances pour un logiciel performant en CAO. Dans un domaine comme l'architecture, où tout objet est "typé" culturellement, cette représentation est très importante.

La clé de toute approche est sans doute dans la représentation de la connaissance, qui, souvent dans les systèmes experts, est aussi importante qu'un principe de résolution ou une heuristique puissante. Nous étudions de plus près la représentation des connaissances dans le chapitre suivant.

2. LA REPRESENTATION DES CONNAISSANCES
LE LOGICIEL "KOSMIGOL"

2.1 Introduction

La représentation des connaissances est une des branches de l'intelligence Artificielle qui connaît à l'heure actuelle de grands développements.

Ce domaine pose deux questions :

- Quelles sont les catégories conceptuelles nécessaires à l'établissement d'une base de connaissance ?

- Quelles sont les formulations informatiques les plus adéquates pour représenter ces catégories conceptuelles ?

La première question fait l'objet d'une réflexion sur la conceptualisation, en distinguant les différentes entités susceptibles d'intervenir dans notre connaissance du monde et les liens statiques et dynamiques qui associent ces concepts entre eux ou qui permettent d'en déduire de nouveaux (objets, inférences, méta-connaissances, etc) et sur l'utilisation qui est faite de cette connaissance : traitement des connaissances (ce que l'on dénomme souvent sous le terme général de raisonnement), accès aux informations (ou processus de remémoration) et acquisition de nouvelles données (les mécanismes d'apprentissage).

Cette réflexion fait partie du thème "Architecture" de notre recherche qui doit etre développée parallèlement à toute appproche informatique pour valider celle-ci dans le domaine que nous nous sommes fixés.

La deuxième question, et c'est celle qui nous intéresse dans cette partie, concerne les moyens informatiques mis en oeuvre pour représenter et manipuler ces concepts.

2.2 Les concepts abstraits et les objets concrets.

Si, à la base, les système de représentation des connaissances manipulent des objets concrets, il est très intéressant de se servir d'un modèle de type abstrait pour représenter les différentes entités, les objets devenant des occurences de ce type abstrait. On augmente alors la généralisation du modèle, et l'on peut raisonner avec des variables.

A ce niveau apparaît quelque chose de très important pour notre réflexion à venir: **la notion de type.**

> "Nous pensons que la notion de type est d'une grande utilité dans l'aide à la conception, elle permet au concepteur de rester à un niveau constant de complexité lors de la réalisation du projet. Le type lié à la notion d'élément permet d'introduire la notion de règle du jeu, propre aux structuralistes combinatoires : "L'attitude des structuralistes combinatoires adopte dès le départ une attitude relationnelle selon laquelle ce qui compte n'est ni l'élément, ni un tout s'imposant comme tel, sans que l'on puisse préciser comment, mais des relations entre les éléments, autrement dit les procédés ou processus de composition, dont les lois sont celles du système".
> Piaget in le Structuralisme <DUPLAY-83>.

2.3 La représentation des connaissances

La représentation des connaissances grâce à la logique des prédicats du 1er ordre est une méthode qui repose sur la logique formelle. Elle permet de représenter aussi bien les connaissances factuelles sous forme d'assertions que les règles à l'aide du symbole d'implication <CARADANT-84>.

Les caractéristiques de ce type de Système de Représentation des Connaissances viennent directement de la nature formelle du système de résolution et de son indépendance par rapport au domaine; c'est un système complet : une base de connaissance associée à un système de traitement.

Les inconvénients ont la même origine : le principe de résolution impose un temps de calcul très important si la base est un peu grande et demande beaucoup d'unifications. La seconde difficulté vient du fait qu'un grand nombre

d'informations sont très difficiles à représenter avec la logique des prédicats.

En effet, les objets, dans ces types de représentations, sont si simples que la plupart des structures complexes ne peuvent pas être représentées facilement : par exemple quand le domaine contient des objets, chacun avec un grand nombre de propriétés, incluant des relations avec les autres objets, il est souvent plus facile de collecter toutes ces propriétés dans la description d'un seul objet complexe.

2.4 Les réseaux sémantiques

2.4.1 Principe

Développés à partir de 1968, (Quillian-68, Raphael-68) les réseaux sémantiques se composent, pour un informaticien, de sommets (représentant les objets, et plus généralement des concepts), et d'arcs (des liens les unissant sous forme de relation).

Les liens les plus courants sont :

-sorte-de :
qui relie un élément à la classe située au-dessus du concept en question.

-partie-de et partie :
qui désigne un objet faisant partie d'un autre objet.

2.4.2 Raisonnement avec la connaissance

Comme pour les autres mécanismes de représentation des connaissances, la puissance des réseaux sémantiques vient de la capacité du système à raisonner avec cette connaissance. Plusieurs techniques sont alors possibles :

1-la recherche d'intersection (intersection search (Quillan-68)) qui permet de répondre aux questions du type : quelle est la relation entre X et Y.

2-la recherche procédurale qui utilise des procédures de recherche spécifiques pour répondre à des questions particulières. La procédure est alors chargée d'explorer l'arbre sémantique pour en ramener les informations nécessaires.

3-le filtrage, qui permet une mise en correspondance d'un filtre contenant une variable pour répondre à des questions précises.

Mais l'on remarque quelques problèmes : lorsque l'ensemble des sommets et des relations devient important la recherche des relations croît de manière exponentielle si l'on n'utilise pas des heuristiques puissantes qui alourdissent alors le système.

Pour pallier les inconvénients des réseaux sémantiques, ont été développées des structures de représentation plus sophistiquées, qui sont plus que des structures de données : les frames.

2.5 Les frames ou schémas

2.5.1 Principe

L'idée des schémas, issue d'un article de Minsky (MINSKY-75), est basée sur le fait que nous avons dans notre pensée toute une série de schémas, ou stéréotypes, qui représentent notre passé et l'idée que l'on se fait d'une situation, d'un objet et des gens. Cet a'priori nous aide à réfléchir, à raisonner sur une nouvelle situation : nous sélectionnons une 'stucture générale que nous complétons par des détails de la situation actuelle.

Le terme de frame soutend l'existence d'un grand nombre de représentations par liaisons :

objet*attribut*valeur.

Dans un premier temps les schémas peuvent apparaître comme des supers réseaux sémantiques, mais ils sont en réalité beaucoup plus que cela et méritent une approche plus approfondie. Il sont en effet les premiers intégrant conjointement : caractère déclaratif (d'où la ressemblance avec un enregistrement dans une base de données) et procédural, par les procédures rattachées à SI-BESOIN et à DEFAUT. La possibilité d'héritage de propriétés est assurée par les propriétés de type SORTE-DE qui permettent de recevoir les propriétés du père.

2.5.2 Comment raisonner avec les frames

Les frames sont très intéressantes car elles peuvent inférer des faits, non déjà observés, d'une situation particulière.

- Les frames peuvent contenir beaucoup d'informations sur les différents aspects des objets qu'elles peuvent décrire, aspects observés ou non.

- Les frames décrivent une instance typique d'un concept, mais si un aspect particulier n'est pas respecté cela peut entraîner une action particulière : une chaise qui est supposée avoir quatre pieds, peut en avoir trois, mais doit être fixée au sol.

Marvin Minsky a donné d'autres méthodes pour pouvoir inférer des frames à partir de l'observation de la situation courante :

- Filtrage avec une partie de la frame pour la selectionner et vérifier s'il n'y a pas d'incompatibilité avec les autres propriétés.

- Sélection d'un concept général et descente dans ses fils pour voir s'il n'y a pas de conflits possibles...

2.6 La représentation des objets archictecturaux

L'introduction d'un système intelligent, basé sur une représentation à l'aide de frames, en tant qu'aide à la conception architecturale, peut se faire de plusieurs manières :

Les frames, en tant que structures de représentation complexes, peuvent servir à décrire les objets architecturaux, de manière à permettre au système de prendre en charge certaines cohérences, que nous qualifierons de bas niveau : Une chambre doit toujours avoir une fenêtre, une porte, une surface minimale (là apparaît le caractère procédural).

Pour valider cette première approche des structures de frames nous avons implémenté un prototype qui sur un exemple précis nous a permis de voir les avantages et les inconvénients de notre point de vue. Le prétexte en est le suivant :

Si la naissance des langages artificiels a permi à l'homme de communiquer avec la machine, il ne faut pas oublier que ceci s'est fait en faveur de la machine et donc, au détriment de l'homme. Il est indéniable que la science informatique progressant, des domaines de plus en plus complexes et difficiles à maîtriser, telle la C.A.O, sont l'objet d'une demande considérable de la part de plus en plus de professionnels divers : systèmes d'expertises, de consultation assistée, systèmes de gestions de données, etc...

Mais un problème crucial se fait de plus en plus pesant: celui du langage. La communication homme-machine doit se faire dans le langage de l'homme, pour que n'importe quel type d'utilisateur puisse introduire ses spécifications, ses règles et réseaux, ses métarègles, structures de connaissances etc...

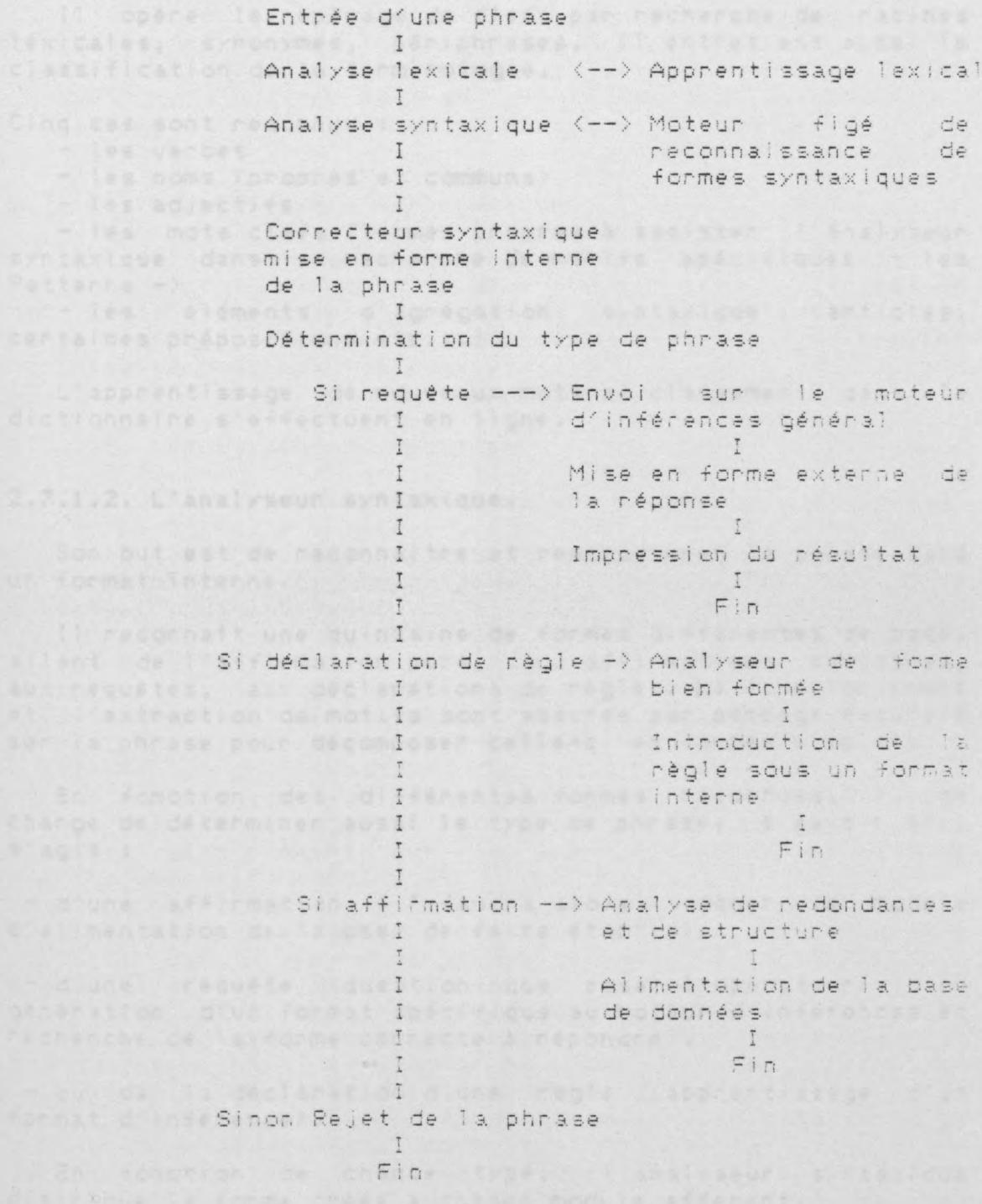
C'est pourquoi les analyseurs de langage naturel deviennent partie intégrante des systèmes de C.A.O. Nous avons tenté une approche de problème en définissant et implémentant le programme Kosmigol.

2.7 Le programme Kosmigol

Il se définit comme un moteur d'inférence d'ordre 1, chapeauté par un analyseur de langage naturel.

La principale problématique qui a conditionné la mise en oeuvre de ce système était d'étudier la possibilité pour un non-informaticien de déclarer un environnement et un ensemble de règles d'inférences architecturales en langage naturel.

La structure générale du programme peut être grossièrement décrite comme suit:



2.7.1. Analyse des différents constituants.

2.7.1.1. L'analyseur lexical.

Il opère le repérage de clefs par recherche de racines lexicales, synonymes, périphrases. Il entretient aussi la classification de la terminologie.

Cinq cas sont reconnus :
- les verbes
- les noms (propres et communs)
- les adjectifs
- les mots clefs (termes propres à assister l'analyseur syntaxique dans sa recherche de motifs spécifiques - les Patterns -)
- les éléments d'agrégation syntaxique (articles, certaines prépositions etc...)

L'apprentissage de nouveaux mots et classement dans le dictionnaire s'effectuent en ligne.

2.7.1.2. L'analyseur syntaxique.

Son but est de reconnaitre et restructurer la phrase dans un format interne.

Il reconnaît une quinzaine de formes différentes de base, allant de l'affirmation pure, aux affirmations composées, aux requêtes, aux déclarations de règles. Le fonctionnement et l'extraction de motifs sont assurés par pompage récursif sur la phrase pour décomposer celle-ci en formes simples.

En fonction des différentes formes reconnues, il se charge de déterminer aussi le type de phrase, à savoir s'il s'agit :

- d'une affirmation (il faudra alors invoquer le module d'alimentation de la base de faits établis).

- d'une requête (question que pose l'opérateur d'où génération d'un format spécifique au moteur d'inférences et recherche de la forme correcte à répondre).

- ou de la déclaration d'une règle (apprentissage d'un format d'inférence).

En fonction de chaque type, l'analyseur syntaxique distribue la forme créée à chaque module afférent.

2.7.1.3. L'analyseur de règles.

Il assure de la cohérence de la règle par rapport au moteur d'inférences. Recherche du type lexical des variables dans le contexte, des formes indéfinies par pré-déclarations, etc...

Une fois cette analyse effectuée avec succès, il transforme la règle en format interne et la stocke.

2.7.1.4. L'analyseur d'affirmations.

Il est chargé de maintenir la cohérence de la base de faits lors de l'introduction d'un nouveau fait. Elimination des formes redondantes, des formes conflictuelles par sous-définition ou sur-définition de données précédemment déclarées, cohérence lexicale, etc...

2.7.1.5. L'analyseur de requêtes.

Il lui revient une tache complexe et délicate : générer une forme correcte à inférer, lancer la forme dans le moteur d'inférences, récupérer le résultat et remodeler le résultat pour obtenir une réponse congrue (le plus possible !), à ce qu'attend l'utilisateur.

Ce dernier point est le plus complexe à réaliser et à maîtriser. L'analyseur syntaxique offre déja un pré-filtrage d'assistance à l'analyseur de requêtes, mais bien des cas persistent pour lesquels la réponse n'est pas adaptée à l'attente de l'utilisateur. En effet, donner une réponse correcte à une question dont la formulation est souvent très incomplète (très vague !) par rapport à l'état de la base, signifie généralement que l'on a une 'idée' de la réponse avant d'opérer la recherche.

Une approche en ce sens est réalisée dans le programme Kosmigol par l'introduction des 'formats' et des 'liens'. Chaque lien relationnnel porte non seulement les éléments qu'il lie dans une base de données propre, (principe d'éclatement de la base pour accélérer les recherches et introduire une structuration par concept d'Objet Relationnel), mais encore, le format, la structure de sa base. Chaque objet-lien sait donc comment est construite sa propre base de données y compris si sa base est déductible par inférence (là encore c'est l'objet qui porte la règle de dérivation). On découvre ici une très grande différence avec nombre de systèmes experts dans lesquels, la base de faits est un amas indifférencié (sauf à l'analyse) de listes.

C'est pourquoi le programme Kosmigol possède de très bons temps de réponse dans ses phases d'inférence et d'analyse. Celui-ci peut donc à tout moment avoir une 'idée formelle' de la réponse à une requette dès que le lien sémantique dans la question est découvert. Ce point de vue du 'lien' nous amènera par la suite à formuler une nouvelle acception du terme 'verbe'. En effet, dans tout ce qui suit, un 'verbe' sera pour nous, un lien relationnel entre plusieurs objets.

2.7.1.6. Le moteur d'inférences.

Il est du type chaînage arrière, ordre 1, permettant les modes trace et apprentissage par introduction en faits établis des sous-buts prouvés.

Une règle est une disjonction de conjonctions de prédicats, chaque prédicat pouvant être à son tour dérivable.

Tout prédicat considéré comme improuvable, annule la conjonction toute entière. Le résultat retourné est issu de l'unification de toutes les disjonctions.

La valeur de tout prédicat peut être calculée de quatre manières différentes :

P , Non P, P est dérivable, P est demandable.

Les atomes littéraux représentant des variables doivent impérativement débuter par le point d'interrogation.

2.7.2. Utilisation.

Après cette brève description interne, illustrons le fonctionnement de cet analyseur par quelques exemples concrets :

Décrivons de manière très élémentaire et uniquement structurelle (cette limitation n'est aucunement induite par les capacités du programme, mais par souci d'une exposition claire du problème et de la manière dont celui-ci est résolu par le le système), les principaux constituants architecturaux d'un temple dorique au moyen de règles exprimées dans un langage tout à fait naturel :

Si l'élément est sur le sol alors l'élément est l'euthynteria.

Si l'élément est debout alors l'élément est une colonne.

Si l'élément porte un élément2 et que l'élément2 est une colonne alors l'élément est le stylobate.

Si l'élément est le stylobate ou l'élément est l'euthynteria ou l'élément est porté par un élément2 et l'élément2 est l'euthyntéria alors l'élément est une assise.

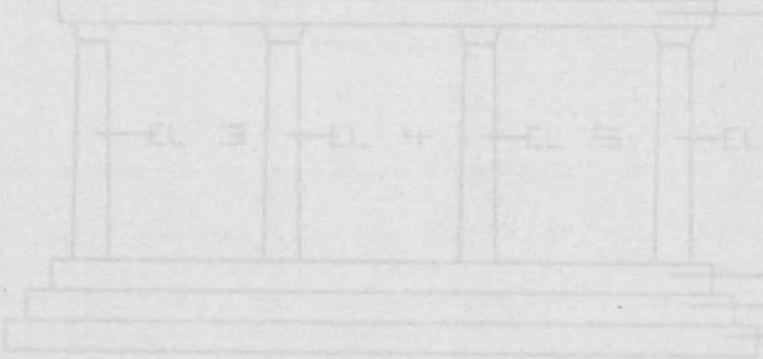
Si l'élément est porté par un élément2 et l'élément2 est une colonne alors l'élément est une architrave.

Si l'élément est porté par un élément2 et l'élément2 est une architrave alors l'élément est une frise.

Si l'élément est une frise ou l'élément est une architrave alors l'élément est un entablement. (Si nous voulons dire que c'est la conjonction de l'architrave et de la frise qui fait l'entablement, et que l'architrave en est le composant principal, nous dirons plutôt :

Si l'élément est une architrave et un élément2 est une frise alors l'élément est un entablement).

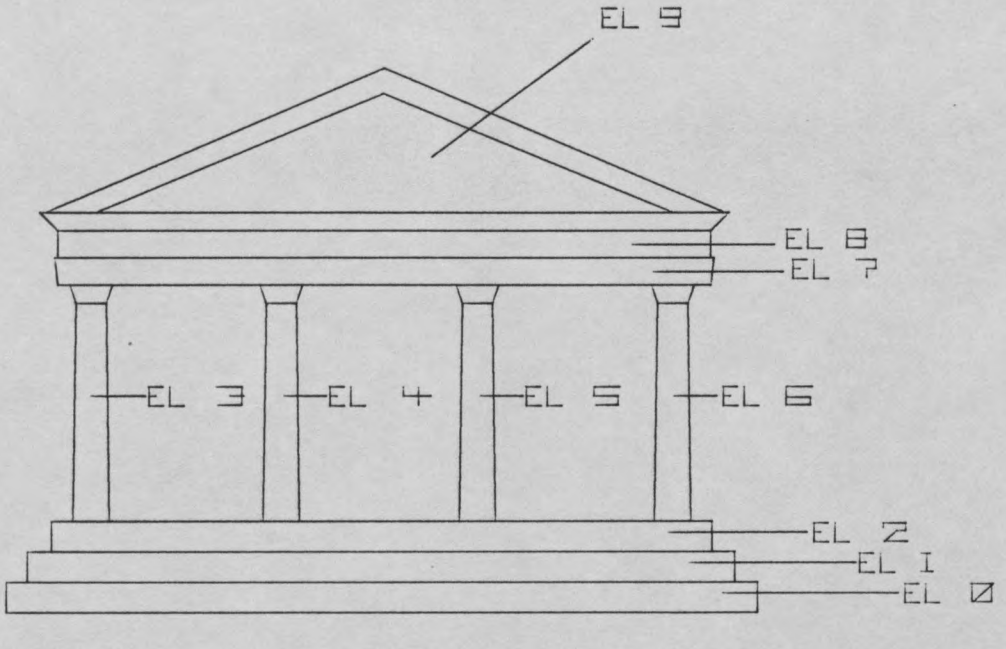
Si l'élément est porté par un élément2 et l'élément2 est un entablement alors l'élément est un fronton.



Nous voyons au moyen de cette description très sommaire, que les seules informations de base utilisées sont le fait qu'un élément soit porté par un autre élément et le fait qu'un élément soit debout. La relation 'porte' peut s'exprimer par la règle :

Si l'élément est porté par un élément2 alors l'élément2 porte l'élément.

Opérons donc la description du temple ci-dessous.



Le programme possède bien entendu un dictionnaire architectural de base dont voici le contenu :

Vocables d'articulation :

```
AVOIR LE LA LES DE DES D UN UNE DU L A ONT EST ETRE SONT
```

Dictionnaire :

```
?ADJECTIF ?NOMCOMMUN ?VERBE POURQUOI QUI QUE COMMENT EST-CE-
QUE  COLONNE  DEBOUT STYLOBATE ARCHITRAVE FRISE  ENTABLEMENT
FRONTON LARMIER CORNICHE FUT CHAPITEAU DORIQUE ECRIRE  EGALE
ASSISE SI ALORS ET OU EL1 EL9 ?A ?B ?C ?D ?E EL0 EL8 EL7 EL6
EL5 EL4 EL3 EL2 SOL ADJECTIFS REGLE

REVOIR équivalences BYE CIAO SALUT
DETRUIT      "      DETRUI ENLEVE SUPPRIME
MONTRE       "      MONTRE AFFICHE ECRIT
BASE         "      'BASE DONNEE'  DONNEE
NOMCOMMUNS   "      NOM COMMUN
VERBES       "      VERBE
SUR          "      SUR 'PORTE PAR' 'APPUYE SUR' 'POSE SUR'
SOUS         "      DESSOUS PORTE PORTEUR
SEMBLE       "      'DOIT ETRE' RESSEMBLE PARAIT
EUTHYNTERIA  "      'ASSISE REGLAGE'
```

On remarquera la très faible importance de ce dictionnaire de départ. Ceci n'est nullement une contrainte du fait que le système peut accroître son dictionnaire dynamiquement.

2.7.3 Un exemple de session Kosmigol

Note : Tout ce qui suit le point d'interrogation a été tapé par l'opérateur. Les phrases doivent être encadrées par des parenthèses. Les variables débutent toujours par un point d'interrogation. Les commentaires sont entre chevrons.

```
 ? : (EL0 EST SUR LE SOL)
 c'est bon

 ? : (EL1 EST SUR EL0)
OK one more !

 ? : (EL2 EST PORTE PAR EL1)
OK one more !
```

```
 ? : (EL3 ET EL4 SONT PORTES PAR EL2)
Ok one more !
Ok one more !
 ? : (EL5 ET EL6 SONT SUR EL2)
Ok one more !
Ok one more !

 ? : (EL7 EST PORTE PAR EL3 ET EL4)
Ok one more !
Ok one more !

 ? : (EL7 EST SUR EL5)
Ok one more !

 ? : (EL7 EST SUR EL6)
Ok one more !

 ? : (EL8 EST PORTE PAR EL7)
Ok one more !

 ? : (EL9 EST PORTE PAR EL8)
Ok one more !

(Il est à noter que ces informations de base peuvent être
introduites automatiquement par un tout petit programme
d'analyse structurelle de scène.)

 ? : (MONTRE LA BASE DE DONNEES DE PORTE PAR)
((BD (EL9 EL8) (EL8 EL7) (EL7 EL6) (EL7 EL5) (EL7 EL4) (EL7
EL3) (EL6 EL2) (EL5
EL2) (EL4 EL2) (EL3 EL2) (EL2 EL1) (EL1 EL0) (EL0 SOL)))

 ? : (QUI EST SUR EL6 ?)
 Je ne comprends pas "?"
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (?)
 Je ne comprends toujours pas : (?)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 2
Reponse   - (EL7)

 ? : (QUE PORTE EL2 ?)
Reponse   - FAUX

 ? : (AFFICHE LA REGLE PORTE)
NIL

 ? : (SI ?A EST PORTE PAR ?B ALORS ?B PORTE ?A)
 c'est correct, regle introduite
```

```
 ? : (QUE PORTE EL2 ?)
Reponse   - (EL6 EL5 EL4 EL3)
 ? : (EL3 ET EL4 SONT DEBOUT)
 c'est bon
Ok one more !

 ? : (EL5 ET EL6 SONT AUSSI DEBOUT)
 Je ne comprends pas AUSSI
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (AUSSI)
 Je ne comprends toujours pas : (AUSSI)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 2
Ok one more !
Ok one more !

 ? : (QUI EST DEBOUT ?)
Reponse   - (EL6 EL5 EL4 EL3)

 ? : (POURQUOI EL6 EST IL DEBOUT)
 Je ne comprends pas IL
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (IL)
 Je ne comprends toujours pas : (IL)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 2
->(DEBOUT EL6)-/-T
Reponse   - VRAI

 ? : (IMPRIME LA BASE DE DONNEES DE DEBOUT)
 Je ne comprends pas IMPRIME
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (AFFICHE)
                              ( Imprime est pris ici comme
                              synonyme local à la requête)

((BD (EL6) (EL5) (EL4) (EL3)))

 ? : (SI ?A EST DEBOUT ET ?A PORTE ?B ALORS ?A EST UNE
COLONNE)
 c'est correct, regle introduite

 ? : (QUI EST UNE COLONNE ?)
Reponse   - (EL6 EL5 EL4 EL3)
```

```
 ? : (POURQUOI EL3 EST IL UNE COLONNE ?)
->(DEBOUT EL3)-/-T
->(SUR ?B EL3)-/-((?B EL7))
->(SOUS EL3 ?B)-/-((?B EL7))
->(COLONNE EL3)-/-T
Reponse   - VRAI

 ? : (DONC EL3 EST SUR EL7 ?)
 Je ne comprends pas DONC
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (DONC)
 Je ne comprends toujours pas : (DONC)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 1
 De quel mot ? EST-CE-QUE
Reponse   - FAUX

 ? : (EST-CE-QUE EL3 EST SOUS EL7)
Reponse   - VRAI

 ? : (DONC EL3 PORTE EL7 ?)
Reponse   - VRAI

 ? : (SI ?A PORTE ?B ET ?B EST UNE COLONNE ALORS ?A EST LE
 STYLOBATE)
 c'est correct, regle introduite

 ? : (QUI EST LE STYLOBATE ?)
Reponse   - (EL2)

 ? : (POURQUOI EL2 EST IL LE STYLOBATE ?)
->(SUR ?B EL2)-/-((?B EL6 EL5 EL4 EL3))
->(SOUS EL2 ?B)-/-((?B EL6 EL5 EL4 EL3))
->(DEBOUT EL6)-/-T
->(SUR ?B EL6)-/-((?B EL7))
->(SOUS EL6 ?B)-/-((?B EL7))
->(COLONNE EL6)-/-T
->(DEBOUT EL5)-/-T
->(SUR ?B EL5)-/-((?B EL7))
->(SOUS EL5 ?B)-/-((?B EL7))
->(COLONNE EL5)-/-T
->(DEBOUT EL4)-/-T
```

```
->(SUR ?B EL4)-/-((?B EL7))
->(SOUS EL4 ?B)-/-((?B EL7))
->(COLONNE EL4)-/-T
->(DEBOUT EL3)-/-T
->(SUR ?B EL3)-/-((?B EL7))
->(SOUS EL3 ?B)-/-((?B EL7))
->(COLONNE EL3)-/-T
->(STYLOBATE EL2)-/-T
Reponse    - VRAI

 ? : (QUI EST L EUTHYNTERIA ?)
Reponse    - FAUX

 ? : (MONTRE MOI LA REGLE DE L EUTHYNTERIA)
 Je ne comprends pas MOI
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (MOI)
 Je ne comprends toujours pas : (MOI)
 Est-ce : 1  un nouveau synonyme
          2  un element de syntaxe
          3  un nouveau mot
 ? : 2
NIL

 ? : (SI ?A EST PORTE PAR LE SOL ALORS ?A EST L EUTHYNTERIA)
 c'est correct, regle introduite

 ? : (QUI EST L EUTHYNTERIA ?)
Reponse    - (EL0)

 ? : (IMPRIME LA REGLE DE L ASSISE DE REGLAGE)
 Je ne comprends pas IMPRIME
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (IMPRIME)
 Je ne comprends toujours pas : (IMPRIME)
                ( Cette fois, le terme IMPRIME va être
                introduit définitivement dans le dictionnaire
                comme étant un synonyme de MONTRE)
 Est-ce : 1  un nouveau synonyme
          2  un element de syntaxe
          3  un nouveau mot
 ? : 1
 De quel mot ? MONTRE
((?A) ((SUR ?A SOL)))

 ? : (SI ?A EST UNE EUTHYNTERIA OU ?A EST UN STYLOBATE OU
BIEN
?A EST PORTE PAR ?B ET ?B EST UNE EUTHYNTERIA ALORS ?A EST
UNE ASSISE)
 c'est correct, regle introduite
```

```
 ? : (QUI EST UNE ASSISE ?)
Reponse   - (EL0 EL2 EL1)

 ? : (AFFICHE MOI LA REGLE DE L ASSISE)
((?A)  ((EUTHYNTERIA  ?A))  ((STYLOBATE ?A))  ((SUR  ?A  ?B)
(EUTHYNTERIA ?B)))

 ? : (DONNE MOI LES VERBES QUE TU CONNAIS)
 Je ne comprends pas DONNE
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (DONNE)
 Je ne comprends toujours pas : (DONNE)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 1
 De quel mot ? MONTRE
(CONNAI  COLONNE CHAPITEAU FUT EUTHYNTERIA  ASSISE  CORNICHE
LARMIER FRONTON
ENTABLEMENT  FRISE ARCHITRAVE STYLOBATE EGALE ECRIRE  DEBOUT
SEMBLE SOUS SUR
?VERBE)

 ? : (POURQUOI EL2 EST IL UNE ASSISE ?)
->(SUR ?B EL2)-/-((?B EL6 EL5 EL4 EL3))
->(SOUS EL2 ?B)-/-((?B EL6 EL5 EL4 EL3))
->(DEBOUT EL6)-/-T
->(SUR ?B EL6)-/-((?B EL7))
->(SOUS EL6 ?B)-/-((?B EL7))
->(COLONNE EL6)-/-T
->(DEBOUT EL5)-/-T
->(SUR ?B EL5)-/-((?B EL7))
->(SOUS EL5 ?B)-/-((?B EL7))
->(COLONNE EL5)-/-T
->(DEBOUT EL4)-/-T
->(SUR ?B EL4)-/-((?B EL7))
->(SOUS EL4 ?B)-/-((?B EL7))
->(COLONNE EL4)-/-T
->(DEBOUT EL3)-/-T
->(SUR ?B EL3)-/-((?B EL7))
->(SOUS EL3 ?B)-/-((?B EL7))
->(COLONNE EL3)-/-T
->(STYLOBATE EL2)-/-T
->(SUR EL2 ?B)-/-((?B EL1))
->(ASSISE EL2)-/-T
Reponse   - VRAI

 ? : (QUI EST L ASSISE DE REGLAGE ?)
Reponse   - (EL0)
```

```
 ? : (POURQUOI EL0 EST IL LE STYLOBATE ?)
->(SUR ?B EL0)-/-((?B EL1))
->(SOUS EL0 ?B)-/-((?B EL1))
Reponse    - FAUX

 ? : (QUI EST LE STYLOBATE ?)
Reponse    - (EL2)

 ? : (QUI PORTE)
Reponse    - (EL8 EL7 EL6 EL5 EL4 EL3 EL2 EL2 EL2 EL2 EL1 EL0
SOL)

 ? : (QUI PORTE LE SOL?)
Reponse    - FAUX

 ? : (QUE PORTE LE SOL ?)
Reponse    - (EL0)

 ? : (QUI SONT DES COLONNES ?)
Reponse    - (EL6 EL5 EL4 EL3)

 ? :  (SI  ?B PORTE ?A ET ?B SONT DES COLONNES ALORS ?A EST
UNE ARCHITRAVE)
 c'est correct, regle introduite

 ?  : (SI ?A EST PORTE PAR ?B ET ?B EST UNE ARCHITRAVE ALORS
?A EST UNE FRISE)
 c'est correct, regle introduite

 ? : (AFFICHE LA REGLE DE L ARCHITRAVE)
((?A) ((SOUS ?B ?A) (COLONNE ?B)))

 ? : (DONNE LA REGLE DE LA FRISE)
((?A) ((SUR ?A ?B) (ARCHITRAVE ?B)))

 ? :  (SI  ?A EST UNE FRISE OU BIEN ?A EST  UNE  ARCHITRAVE
ALORS ?A EST UN
ENTABLEMENT)
 c'est correct, regle introduite

 ? : (MONTRE LA REGLE DE L ENTABLEMENT)
((?A) ((FRISE ?A)) ((ARCHITRAVE ?A)))
```

```
 ? : (QUI COMPOSE L ENTABLEMENT ?)
 Je ne comprends pas COMPOSE
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (COMPOSE)
 Je ne comprends toujours pas : (COMPOSE)
 Est-ce : 1  un nouveau synonyme
          2  un element de syntaxe
          3  un nouveau mot
 ? : 2
Reponse   - (EL8 EL7)
```

< Remarquer que définir COMPOSE comme élément de syntaxe permet de ne pas le retenir comme signifiant sinon comme forme copule. Ce qui n'empêche pas de l'inclure dans le dictionnaire d'où le comportement de Kosmigol à la session suivante >

```
 ? : (POURQUOI EL8 COMPOSE-T-IL L ENTABLEMENT)
 Je ne comprends pas COMPOSE-T-IL
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (COMPOSE)
->(SUR EL8 ?B)-/-((?B EL7))
->(SUR EL7 ?B)-/-((?B EL6 EL5 EL4 EL3))
->(SOUS ?B EL7)-/-((?B EL6 EL5 EL4 EL3))
->(DEBOUT EL6)-/-T
->(SUR ?B EL6)-/-((?B EL7))
->(SOUS EL6 ?B)-/-((?B EL7))
->(COLONNE EL6)-/-T
->(DEBOUT EL5)-/-T
->(SUR ?B EL5)-/-((?B EL7))
->(SOUS EL5 ?B)-/-((?B EL7))
->(COLONNE EL5)-/-T
->(DEBOUT EL4)-/-T
->(SUR ?B EL4)-/-((?B EL7))
->(SOUS EL4 ?B)-/-((?B EL7))
->(COLONNE EL4)-/-T
->(DEBOUT EL3)-/-T
->(SUR ?B EL3)-/-((?B EL7))
->(SOUS EL3 ?B)-/-((?B EL7))
->(COLONNE EL3)-/-T
->(ARCHITRAVE EL7)-/-T
->(FRISE EL8)-/-T
->(SUR EL8 ?B)-/-((?B EL7))
->(SOUS ?B EL8)-/-((?B EL7))
->(ENTABLEMENT EL8)-/-T
Reponse   - VRAI
```

```
? : (SI ?A EST SUR LA FRISE ALORS ?A EST UN FRONTON)
c'est correct, regle introduite

? : (LE FRONTON EST TRIANGULAIRE)
Je ne comprends pas TRIANGULAIRE
Donnez-moi un synonyme ou périphrase, ou corrigez
Entre parentheses svp : (TRIANGULAIRE)
Je ne comprends toujours pas : (TRIANGULAIRE)
Est-ce :  1  un nouveau synonyme
          2  un element de syntaxe
          3  un nouveau mot
? : 1
De quel mot ? TRIANGLE
Je ne trouve pas le mot dont vous donnez le synonyme
Je ne comprends pas TRIANGULAIRE
Donnez-moi un synonyme ou périphrase, ou corrigez
Entre parentheses svp : (TRIANGLE)
Je ne comprends toujours pas : (TRIANGLE)
Est-ce :  1  un nouveau synonyme
          2  un element de syntaxe
          3  un nouveau mot
? : 3

Quel type ?
     1  un verbe
     2  un adjectif
     3  un nom propre ou commun
     4  une clef (conjonction etc...)
? : 2
Je ne comprends pas TRIANGULAIRE
Donnez-moi un synonyme ou périphrase, ou corrigez
Entre parentheses svp : (TRIANGULAIRE)
Je ne comprends toujours pas : (TRIANGULAIRE)
Est-ce :  1  un nouveau synonyme
          2  un element de syntaxe
          3  un nouveau mot
? : 1
De quel mot ? TRIANGLE
c'est bon
```

```
 ? : (LA COLONNE EST CYLINDRIQUE)
 Je ne comprends pas CYLINDRIQUE
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (CYLINDRIQUE)
 Je ne comprends toujours pas : (CYLINDRIQUE)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 3

 Quel type ?
      1  un verbe
      2  un adjectif
      3  un nom propre ou commun
      4  une clef (conjonction etc...)
 ? : 2
 c'est bon

 ? : (LA COLONNE EST AUSSI CANNELEE)
 Je ne comprends pas CANNELEE
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (CANNELE)
 Je ne comprends toujours pas : (CANNELE)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 3

 Quel type ?
      1  un verbe
      2  un adjectif
      3  un nom propre ou commun
      4  une clef (conjonction etc...)
 ? : 2
Ok one more !
 c'est bon

 ? : (COMMENT EST LA COLONNE ?)
Reponse   - (CANNELE CYLINDRIQUE)

 ? : (QUI EST CANNELE)
Reponse   - (COLONNE)

 ? : (DONNE MOI LES NOMS COMMUNS QUE TU CONNAIS)

(COLONNE  CHAPITEAU  FUT EUTHYNTERIA ?A ?B ?C ?D  ?E  ASSISE
CORNICHE  LARMIER  FRONTON  ENTABLEMENT  FRISE  ARCHITRAVE
STYLOBATE EL0 EL9 EL8 EL7 EL6 EL5 EL4 EL3 EL2 SOL EL1 ?NOMCOMMUN)
```

```
< Remarquer que certains 'mots' peuvent être à la fois liens
et éléments dans la base.  D'où des status à la fois 'verbe'
et 'nom commun'>

 ? : (LA COLONNE EST COMPOSEE DU FUT ET DU CHAPITEAU)
 c'est bon
Ok one more !

 ? : (DE QUOI EST COMPOSEE LA COLONNE ?)
 Je ne comprends pas QUOI
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (QUOI)
 Je ne comprends toujours pas : (QUOI)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 1
 De quel mot ? QUE
Reponse   - (CHAPITEAU FUT)

 ? : (DE QUOI EST COMPOSE L ENTABLEMENT)
   - FAUX

 ? : (L ENTABLEMENT EST COMPOSE DE LA FRISE ET DE L
ARCHITRAVE)
Ok one more !
Ok one more !

 ? : (LE SOCLE EST COMPOSE DE L EUTHYNTERIA ET DU STYLOBATE)
 Je ne comprends pas SOCLE
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (SOCLE)
 Je ne comprends toujours pas : (SOCLE)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 3

 Quel type ?
      1  un verbe
      2  un adjectif
      3  un nom propre ou commun
      4  une clef (conjonction etc...)
 ? : 3
Ok one more !
Ok one more !

? : (LE SOCLE EST AUSSI COMPOSE D UNE ASSISE)
Ok one more !
```

```
? : (LE FRONTON EST COMPOSE DE LA CIMA ET DU TYMPAN)
Je ne comprends pas CIMA
Donnez-moi un synonyme ou périphrase, ou corrigez
Entre parentheses svp : (CIMA)
Je ne comprends toujours pas : (CIMA)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 3

 Quel type ?
      1  un verbe
      2  un adjectif
      3  un nom propre ou commun
      4  une clef (conjonction etc...)
 ? : 3
 Je ne comprends pas TYMPAN
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (TYMPAN)
 Je ne comprends toujours pas : (TYMPAN)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 3

 Quel type ?
      1  un verbe
      2  un adjectif
      3  un nom propre ou commun
      4  une clef (conjonction etc...)
 ? : 3
Ok one more !
Ok one more !

 ? : (DE QUOI EST COMPOSEE LA BASE ?)
Reponse   - FAUX

 ? : (DE QUOI EST COMPOSE LE SOCLE ?)
Reponse   - (ASSISE STYLOBATE EUTHYNTERIA)

 ? : (QUI RESSEMBLE A UNE COLONNE ?)
Reponse   - (EL6 EL5 EL4 EL3)
```

```
 ? : (A UN FRONTON ?)
 Essai de tournure elliptique avec :
(QUI SEMBLE FRONTON)
        < Ici le système ne peut analyser la requête comme
        une requête de type 'normal'. Il cherche donc s'il
        ne peut l'analyser au travers d'une tournure
        elliptique. La recherche fut concluante !>

Reponse   - (EL9)

 ? : (A UNE ASSISE)
 Essai de tournure elliptique avec :
(QUI SEMBLE ASSISE)
Reponse   - (EL0 EL2 EL1)

        .....
                < En particulier, définition de la règle
                SEMBLE>.....

 ? : (ECRIT LA REGLE SEMBLE)
((?A ?C) ((EGALE ?C EUTHYNTERIA) (SUR ?A SOL)) ((EGALE ?C
STYLOBATE) (SOUS ?A
?B) (SEMBLE ?B COLONNE)) ((EGALE ?C ASSISE) (SEMBLE ?A
EUTHYNTERIA)) ((EGALE ?C
ASSISE) (SEMBLE ?A STYLOBATE)) ((EGALE ?C ASSISE) (SUR ?A
?B) (SEMBLE ?B
EUTHYNTERIA)) ((EGALE ?C COLONNE) (DEBOUT ?A)) ((EGALE ?C
ARCHITRAVE) (SUR ?A
?B) (SEMBLE ?B COLONNE)) ((EGALE ?C FRISE) (SUR ?A ?B)
(SEMBLE ?B ARCHITRAVE))
((EGALE ?C FRONTON) (SUR ?A ?B) (SEMBLE ?B FRISE)) ((EGALE
?C ENTABLEMENT) (
SEMBLE ?A ARCHITRAVE)) ((EGALE ?C ENTABLEMENT) (SEMBLE ?A
FRISE)))

 ? : (SI ?A EST PORTE PAR ?B ET ?B SEMBLE ETRE ?C ALORS ?A
EST AU-DESSUS DE ?C)
 Je ne comprends pas AU-DESSUS
 Donnez-moi un synonyme ou périphrase, ou corrigez
 Entre parentheses svp : (AU-DESSUS)
 Je ne comprends toujours pas : (AU-DESSUS)
 Est-ce :  1  un nouveau synonyme
           2  un element de syntaxe
           3  un nouveau mot
 ? : 3
```

```
 Quel type ?
      1 un verbe
      2 un adjectif
      3 un nom propre ou commun
      4 une clef (conjonction etc...)
 ? : 1
 c'est correct, regle introduite

 ? : (QUI EST AU-DESSUS DE L ARCHITRAVE)
Reponse  - (EL8)

 ? : (DU FRONTON ?)
 Essai de tournure elliptique avec :
(QUI AU-DESSUS FRONTON)
Reponse    - FAUX
 ? : (QUE PORTE EL0)
Reponse    - (EL1)

 ? : (EL2)
 Essai de tournure elliptique avec :
(QUE SOUS EL2)
Reponse    - (EL6 EL5 EL4 EL3)

 ? : (COMMENT EST LA COLONNE ?)
Reponse    - (CANNELE CYLINDRIQUE)

 ? : (LE FRONTON ?)
 Essai de tournure elliptique avec :
(COMMENT FRONTON)
Reponse    - (TRIANGLE)

 ? : (AU REVOIR ET MERCI)
CIAO!!!
```

On a pu observer la souplesse d'utilisation du système Kosmigol (le texte ci-dessus étant une copie exacte de la console de l'ordinateur). Celui-ci reconnaît donc les périphrases, les synonymes, les conjonctions, les tournures elliptiques (dans une certaine mesure), et un certain nombre de motifs syntaxiques parmi lesquels :

```
rs::= <groupe sujet>
rs::= <groupe sujet> ETRE <groupe adjectif>
rs::= <groupe sujet> <verbe>
rs::= <groupe sujet> <verbe> <groupe complement>
rs::= <groupe sujet> ET <groupe sujet> <verbe>
rs::= <groupe sujet> <verbe> <groupe complement> ET
      <groupe complément>
<groupe complément>::= <groupe nominal>
<groupe sujet>::= <groupe nominal>
<groupe nominal>::= <nom> I <groupe adjectif> <nom> I
                    <nom> <groupe adjectif> I <groupe
                    adjectif> <nom> <groupe adjectif>
```

A ces motifs doivent être ajoutés toutes les formes reconnaissables par des mots clefs (commandes) ou des conjonctions (interrogation, déclaration de règles etc...).
Il apparaît que le développement d'un analyseur syntaxique beaucoup plus complet et une interface entre le système et un éditeur graphique pourraient apporter une souplesse encore plus grande à l'utilisateur.

Nous avons donc pu valider le postulat selon lequel un architecte non informaticien peut introduire ses propres règles dans une machine pour autant que celle-ci contienne un minimum d' "intelligence". Il y a là un ensemble de débouchés certains dans le domaine de l'enseignement assisté par ordinateur et dans les voies qui devront être suivies pour constituer les futurs outils de conception assistée par ordinateur.

Note : Le source en Lisp du système Kosmigol est fourni en annexe.

3. UN LANGAGE A STRUCTURE DE FRAME ET ORIENTE OBJETS : LE LANGAGE "FROC"

3.1 Rappels

Nous présentons ici d'abord les spécificités du langage que nous avons implémenté dans la première phase de notre étude. Pour des renseignement complémentaires se référer à <LI2A-84>. Les lignes générales se retrouvent dans un certain nombre de langages semblables et sont communes à ces langages (par exemple FRL <ROBERTS-77> ou KRL <BOROW-77>).

Le côté déclaratif

Les objets élémentaires du langage sont bâtis sur le même modèle : un nom et un ensemble de propriétés rattachées à ce nom.

La notion de classe

La notion de classe à proprement parler n'existe pas, chaque objet pouvant être le générateur d'un objet terminal, ou d'une autre classe.

L'héritage

La gestion des héritages est assurée par le lien de type "EST-UN" : cette propriété prédéfinie génère un héritage entre les différents objets.

3.1.1 Les différentes possibilités de la structure déclarative :

Le coté déclaratif

aspect "VALEUR"

Chaque objet hérite des valeurs des propriétés de tous ses pères. Il s'agit de ramener toutes les VALEURS présentes dans la hiérarchie. Elles sont supposées être, dans ce cas, des valeurs obligatoires.

aspect "DEFAUT"

L'héritage de la valeur par défaut se fera par exploration de la hiérarchie : celle-ci se présentera souvent sous la forme d'un réseau. Si l'objet n'a pas de valeur on va rechercher la première valeur par défaut

présente dans la hiérarchie, et cette valeur sera alors la valeur de la propriété pour l'objet.

Le côté procédural

Le côté procédural de la frame est mis en place par le même type de hiérarchie. Il est accroché à certains "aspects" (appelés encore dans ce cas démons) de la structure déclarative des objets. Ce sont les modifications opérées sur les propriétés : à la consultation, à l'ajout, ou à la modification de valeurs, qui lancent l'exécution d'une procédure. Nous sommes en présence d'une programmation dirigée par les données.

3.1.2 Les interfaces avec l'environnement

Nous définissons ici les procédures qui sont les interfaces utilisateur de notre langage.

Les interfaces courantes

- Création des frames
- Consultation des frames
- Destruction des frames

L'utilisateur donne un chemin de type frame-propriété-aspect.

Les interfaces hiérarchisées

Ces interfaces utilisent l'arbre hiérarchique (qui est en fait un réseau) mis en place par la relation EST-UN ; elles poursuivent les recherches et les contrôles dans cette hiérarchie.

- Consultation

L'utilisateur donne un chemin d'accès de la forme frame-propriété. Le système explore déjà l'aspect VALEUR dans toute la hiérarchie, toutes les valeurs ramenées sont gardées car l'aspect VALEUR est prioritaire à tous les niveaux (nous verrons dans les exemples l'utilisation de cette propriété). L'aspect DEFAUT (valeur par défaut) et l'aspect SI-BESOIN (qui peut lancer une procédure) sont ensuite inspectés mais la recherche s'arrête à la première valeur trouvée ou calculée.

Création

S'il y a un aspect SI-AJOUT dans une des frames liées à l'objet auquel on veut accrocher une propriété, alors le système exécute la fonction qui s'y trouve, avant de placer l'information (validation dynamique de la cohérence). Ce style de fonction introduit à la notion de "démons", bien connus en IA. Ils ont pour but d'assurer la cohérence d'une base de faits (au sens IA du terme), indépendemment du reste du système. Nous verrons que cela nous permet d'accrocher des structures de contrôle puissantes à chaque objet.

Destruction

S'il y a un aspect SI-DETRUIT elle opère de la même manière que pour la création, en vérifiant les effets de bord de la destruction de l'information. Nous avons toujours ici la notion de "démons".

3.1.3 Mise en oeuvre de notre premier prototype

On consultera la définition de ce premier prototype dans (LI2A-85).

Celui-ci nous a permis sur des exemples architecturaux simples, de soulever les problèmes générés par une approche hiérarchisée des connaissances (déclaratives ou procédurales) :

Si l'approche hiérarchisée convient bien à la spécialisation des concepts :

un mur en béton est une specialisation du concept MUR,

un couloir ou un escalier sont une spécialisation du concept de circulation,

il n'en est pas de même pour le concept d'"agrégat". Une pièce est formée de plusieurs entités :

mur , fenêtres, portes, planchers, plafond ... qui "héritent" d'un certain nombre d'attributs de cette pièce mais dans ce cas nous n'avons plus le concept de spécialisation par rapport à la chambre mais un concept d'"échelle" ou de "point de vue" par rapport à celle-ci.

3.1.4 Les nouvelles formes d'héritage

Pour pallier cette lacune de la démarche hiérarchisée des Frames certains chercheurs proposent d'étendre la notion d'héritage et de créer des héritages de types "sélectifs" (OURGERDIL-85). Ces héritages sélectifs permettent de prendre en compte le fait que les murs vont "hériter" de la

hauteur de la pièce mais ceci au prix d'une complexification inacceptable de la sémantique du système :

L'héritage sélectif est géré par les objets même s'il est déclaré dans des classes : on voit apparaître la notion de sur-objet et de sous-objet qui se développe parallèlement à la notion de sur-classe et de sous-classe ce qui fait perdre son "typage" à la notion d'objet et de classe.
Dans ces systèmes les utilisateurs peuvent avoir quelques difficultés à se faire une "image mentale" un peu réaliste du fonctionnement de ceux-ci ; un objet peut en effet avoir :

-une classe dont il est l'instance
-un sur-objet dont il hérite sélectivement
-des surclasses dont il hérite de manière hiérarchique classique
-des surclasses dont il hérite de manière sélective avec une exploration hiérarchique.

Cette approche, si elle a le mérite de poser correctement le problème de la gestion des héritages qui ne sont pas déduits par une spécialisation d'un concept, nous semble beaucoup trop compliquée à gérer par un utilisateur qui, rappelons le, doit pouvoir introduire ses propres concepts et ses propres objets dans la machine.

Ce problème qui est sans doute l'un des plus difficiles à résoudre en CAO nous à conduit vers les "langages orientés objets".

3.2 Les langages objet

La différence fondamentale entre les systèmes classiques et les systèmes orientés objets réside dans l'approche réellement modulaire des structures de représentation et de contrôle de ces derniers : dans les langages objets tout est porté par les objets : connaissance d'eux-mêmes, de leur environnement, mais aussi du comportement général de cet environnement.

3.2.1 Les différents types de Langages Objet

On distingue plusieurs types de langages orientés objet :

- Les langages directement issus de SMALLTALK (GOLBERG-81).
- Les langages d'acteurs : PLASMA, ACT1, FORMES, qui sont quelques peu différents surtout dans leur comportement dans la gestion des héritages : ils sont certainement plus souples à ce niveau que les

précédents.
- Les langages objet de type "frame" : MERING, KOOL, LRO qui intègrent un système complexe de représentation des connaissances à un système de communication par envoi de messages.

Voici par exemple quatre axiomes de base suffisant à décrire le langage SMALLTALK :

A1- Toute entité SMALLTALK manipulable est un objet
Smalltalk unifie la notion de données et de procédures. Le concept objet vise à donner une représentation unique à l'ensemble des informations manipulées par le langage. Le principe est de représenter toutes les entités du langage de la plus simple à la plus complexe par une base de données et une base de procédures.
Un objet Smalltalk sera donc un booléen, un caractère, un nombre, mais aussi une fenêtre sur l'écran, un fichier...

A2- Tout objet est une instance d'une classe
Le principe est de regrouper à l'intérieur d'une classe les objets aux comportements similaires.
Une classe est donc une desciption INTENTIONNELLE d'un ensemble d'objets.
La description intentionnelle d'un objet est modélisée dans une classe à l'aide d'attributs et de descripteurs d'attributs.

A3- Tout objet est ACTIVE à la reception d'un MESSAGE
La transmission de MESSAGES entre OBJETS est le seul moyen de contrôle dont l'utilisateur du langage dispose.
Un message peut être vu comme une demande d'actions.
Un message est capté par un objet à l'aide d'un SELECTEUR DE MESSAGE (message parsing) associé à une méthode.
L'ensemble des couples (sélecteurs, méthodes) est appelé dictionnaire des méthodes.
Les messages peuvent etre de tous types : dessine pour une tortue, incrémente pour un nombre, prend-hauteur pour un colonnade...

A4- Toute classe est sous-classe d'une ou plusieurs autres classes (la classe initiale étant sa propre mère)
Ce principe de HIERARCHIE permet le réaliser le princice d'HERITAGE.
Une classe héritière de ses super-classe hérite de ses dictionnaires de méthodes associées à ses tables de sélecteurs de message et des champs de ses classes supérieures. Le modèle SMALLTALK-80 (GOLDBERG 83) permet aussi l'héritage des variables de classe c'est à dire des attributs qui sont constants pour toutes les instances de la classe.

3.2.2 La transmission de messages dans les Langages Orientés Objet

Le contrôle du langage s'effectue grâce à un mécanisme d'envois de messages. Bien que n'étant pas un langage parallèle, MERLING II introduit cependant, avec la notion de message retardé et la possibilité de programmer par continuation, des mécanismes de contrôles quasi-parallèles.

Les messages sont eux aussi des entités, généralement de durée éphèmère, qui se représentent dans le langage lui-même. Ainsi, la transmission :

(toto :age (- 34) est équivalente à

```
(message =) ()
emetteur =  toplevel          ; s'il est envoyé au top-level
receveur =  (toto :age)       ; le chemin vers le receveur
sélecteur = (-                ; le sélecteur de la méthode
arguments = (34)              ; son ou ses arguments
```

Chaque message est transformé lors de la lecture en une entité qui s'analyse elle même et détermine de quel type de message il s'agit. Il est possible, durant cette phase d'effectuer un certain nombre de vérifications syntaxiques et sémantiques et/ou de construire une forme évaluable, équivalente sur le plan fonctionnel, mais plus performante.

Ensuite, lors de l'évaluation, le message est activé, et passé en argument à l'évaluateur de la méthode.

Dans d'autres langages, comme PLASMA, il existe principalement deux types de messages :

- Les messages applicatifs classiques, dont le résultat de l'évaluation est renvoyé à l'émetteur du message. Pour reprendre la terminologie PLASMA, nous les intitulerons "messages à continuation implicite".

- Les messages qui désignent le récepteur du message, en faisant appel à la continuation : il s'agit donc de messages à continuation explicite.

Messages retardés

Il est souvent intéressant de pouvoir décrire des messages dont l'activation dépend d'une condition. Dans une optique de communication par envois de messages, cela revient à décrire des mécanismes d'évaluation quasi-parallèle.

Le message retardé est introduit en MERLING II pour simuler des activités concurrentes. Ceci est à rapprocher des mécanismes "d'évaluation paresseuses" (lazy evaluation) telle qu'on la rencontre en programmation fonctionnelle.

Définition :

> Un message retardé est un message dont le contenu ne sera évalué que si une condition liée à un évènement particulier est satisfaite.

La fonction des messages retardés est de construire des systèmes d'inférences à l'aide d'entités, de réflexes et d'envois de messages, dont le fonctionnement sera déclenché par un évènement initial tel que le placement de valeurs dans des attributs. Il s'agit donc essentiellement de produire un effet de bord qui conduit à la réalisation d'un système de propagation, dont la structure dépend essentiellement de la forme de la condition (la partie WHEN du message).

Il s'agit là de la description d'une inférence (une opération déclarative), qui sera compilée sous la forme d'un réseau de propagation (dont le fonctionnement est procédural).

3.2.3 La représentation des connaissances sous le formalisme objet

Le modèle logique renferme toutes les notions fondamentales autour desquelles s'articulent les connaissances stricto sensu. Ces entités logiques sont : la notion de classe, d'objet, de relation, d'unicité et de pluralité, de nombre, etc...

Généralement, ces informations font partie du langage de représentation : elles sont le fondement de toute construction ultérieure.

Les techniques de représentation employées s'articulent autour de l'opposition entre "formalisme objet" et "formalisme relationnel ou prédicatif".

Dans le formalisme relationnel un objet n'existe pas en tant que tel, mais seulement comme participant à un ensemble d'énoncés dispersés dans la base. Le travail de raisonnement consiste alors à manipuler ces énoncés afin de déduire d'autres énoncés. Dans ce cadre, il n'est jamais question d'objets, mais de ce qui peut être dit sur ces objets : les langages PLANNER et PROLOG se rangent dans cette catégorie.

Le modèle logique articulé autour du formalisme objet suppose un certain nombre d'informations de base :

a) Les éléments de base de la connaissance peuvent être décrits sous la forme d'entités (au sens large du terme) caractérisées par leurs propriétés ou attributs.
Par exemple :

Un objet est défini par sa couleur, sa taille, sa forme, son poids, sa substance, son utilité, etc...
Une personne est définie par son nom, son état civil, sa profession, son âge, son poids, ses occupations, ses goûts, ses désirs, son passé, ses intentions, etc...
Une action est décrite par l'agent, le thème, le lieu et le temps, l'instrument, la source et la destination de cette action, etc..
Un scénario est représenté par ses acteurs, les lieux et les objets concernés, la condition pour qu'il ait lieu, l'ensemble des évènements et l'état résultant du déroulement du scénario, etc...

b) Chaque attribut est caractérisé par un certain nombre de valeurs explicites et implicites, contraintes, etc...
Par exemple :

L'agent de l'action vendre doit être une personne ou une entreprise.
L'âge d'une personne est un nombre entier positif.

c) A ces attributs peut être associée la manière explicite de trouver les informations manquantes, ou ce qui doit être effectué lors de l'ajout d'une donnée à cet attribut. Souvent, ces informations sont placées sous la forme de démons, c'est à dire de procédures qui sont activées lors d'un évènement intervenant à l'un des attributs.

d) Il existe des types génériques, les concepts, dont les individus, les objets, les actions,... du monde réel sont des concrétisations, lesquels sont intitulés les représentants (ou instances) de ces types.

e) Il est possible de caractériser un type particulier en termes de types plus généraux, et donc de créer un ordre général ou partiel dans la spécialisation des concepts. En d'autres termes, il est possible de dire que des concepts sont plus généraux que d'autres. Les concepts plus spécifiques comportent plus de propriétés, et les contraintes sur leurs attributs sont plus restrictives que pour les concepts situés plus haut dans la hiérarchie.

3.2.4 Limites de ces langages

Les langages orientés objets tels que SMALLTALK (GOLDBERG-83), OBJVLISP (COINTRE-84) proposent un formalisme qui s'apparente aux schémas et permettent donc de représenter, et surtout de manipuler des connaissances, grâce aux points suivants :

- Structuration de la base par différenciation entre classes et instances
- Champs locaux pouvant s'apparenter aux attributs
- Intégration des caractères liés à la représentation et au traitement des connaissances grâce à l'emploi des méthodes.

Cependant il leur manque un certain nombre de caractéristiques que nous avons mentionnées (valeurs par défaut, démons, contraintes), et qui de ce fait les rendent inaptes à la représentation des connaissances dans des domaines à fortes tendances inférentielles.

Ces insuffisances nous ont amenés à développer notre propre langage, que nous avons nommé "Froc".

3.3 Le langage "Froc"

Le langage FROC que nous avons implémenté permet de manipuler des structures très puissantes : il allie la puissance déclarative des FRAMES à la modularité des structures de contrôle des langages orientés objets.

Il permet comme nous le verrons aux travers d'un exemple de validation de traiter avec une sémantique très simple les problèmes des héritages héarchiques (donc dûs à la spécialisation d'un concept) et celui des héritages de cohérence (dûs à la définition d'un concept regroupant un certain nombre d'objets de base).

3.3.1 La structure du langage

Nous avons implémenté dans le langage les deux structures de représentation d'objets définies dans le paragraphe 3.2 :

- Un interprète de type SMALLTALK qui permet de gérer les envois de messages entre les différents objets.

- Une structure de représentation des données de type FRAMES qui permet de gérer aussi bien le côté déclaratif que le coté procédural (nous avons repris en partie la structure de l'interprète de notre premier prototype) < LI2A-85 >.

L'interprète de type SMALLTALK

Cet interprète, qui reprend les travaux qui ont été faits à l'université de PARIS-IV par Pierre COINTE autour de OBJVLISP et par Isabelle BORNE autour de MICRO-SMALLTALK, est fondé sur la définition de SMALLTALK 76. C'est à dire que notre modèle n'admet pas le multi-héritage au niveau de la recherche des méthodes accrochées à une classe.

La notion de classe et d'instance est par contre bien différencié comme en SMALLTALK-80 :

Le générateur de classes est la classe METACLASS qui génére toutes les classes et qui supporte les méthodes que seules les classes peuvent réaliser : création d'instances de classe, attachement de variable de classes, de démons ....

Les objets terminaux ou instances de classe ne peuvent recevoir les méthodes de classe, mais héritent de toutes les méthodes qui ont été définies dans le pére de tous les objets du syatème : l'objet "OBJET". Nous retrouvons la grande pureté des langages objets dont la sémantique est ainsi respectée.

La gestion des différentes notion d'héritage

- L'héritage des methodes dans l'interprète SMALLTALK

Les objets sont, comme dans l'ensemble des langages objets, les entités du langage qui contrôlent la reception et l'interprétation d'un message qui leurs est envoyé. La recherche de la méthode correspondante commence dans la classe générative de l'objet ; si cette recherche échoue dans cette classe, l'objet receveur continue celle-ci dans l'arbre des surclasses de cette classe. Si la méthode n'est pas trouvée un message d'erreur est envoyé au système (un autre choix, tel que ne rien faire, pourrait fort bien être pris). Dans le cas contraire la méthode est interprétée et le résultat est renvoyé à l'objet qui a envoyé ce message. Notre logiciel ne traitant pas pour l'instant ces envois dans la syntaxe du langage telle qu'elle apparaît à l'utilisateur au plus haut niveau, le passage des résutats sont réalisés dans une syntaxe proche de celle de LISP.

- L'héritage dans la structure de frames

Pour voir le fonctionnement de l'héritage dans la structure des FRAMES se référer au paragraphe 3.1 Il faut surtout retenir que les héritages dans les FRAMES permettent la gestion des multi-héritages : une classe peut avoir pour les gestions évoluées des cohérences plusieurs classes pères (nous sommes alors en présence d'un réseau).

3.2.2 Syntaxe des instructions les plus courantes :

- Messages de classe

< class CREE objet x1 x2 x3 ... >
Crée un objet instanstance de la classe "class" avec pour "valeur d'attribut" x1 x2 x3

<class FILS?>
Ramène toutes les instances de la classe "class"

<class CHAMPS? >
Permet de voir les attributs de la classe "class"

< class <- prop valeur >
Donne à la propriété "prop" la valeur "valeur" pour tous les objets de cette classe (si cette propriété n'est pas déjà spécifiée dans les objets terminaux).

< class DETRUIT prop valeur >
Détruit les variales de classe de la propriété "prop".

< class ? PROP >
Ramène la variable de classe de la propriété "prop" la recherche est en fait une recherche de type hiérarchique dans l'arbre des frames pères de la classe "class".

<class AT-PROC prop facette valeur>
Sert à générer des attachements procéduraux de type SI-AJOUT, SI-BESOIN, SI-DETRUIT dans la classe "class".

<class DET-PROC prop facette valeur>
Détruit un attachement procédural.

< class HERA >
Ecrit les objets ou classes qui sont "au-

dessus" ou "au-dessous" de la classe "class" dans les arbres hiérarchiques des frames.

<class HSTK >
Ecrit les objets ou classes qui sont "au-dessus" ou "au-dessous" de la classe "class" dans les arbres hiérarchiques de l'interprète SMALLTALK.

< class MENU? >
Ramène l'ensemble des méthodes de la classe "class".

- Messages pour les classes des gestionnaires de fichiers

<files SAUVE>
Permet de sauver une classe d'objets SMALLTALK. Tous les éléments de la classe sont sauvés.

<files RAMENE fichier suffixe>
Ramène le fichier "fichier.suffixe".

- Messages pour les objets terminaux

< objet <- prop valeur>
Permet de réaliser la liaison d'un attribut à une certaine valeur. Avant de faire cette liaison, l'objet regarde dans sa frame (c'est à dire dans tout l'héritage, au sens FRAME du terme) si cette valeur est valide. Ce contrôle est assuré par le lien SI-AJOUT.

<objet ? prop >
Demande à un objet la valeur de l'attribut "prop". Si cette valeur existe dans l'objet, elle est ramenée, sinon la recherche se fait dans la frame de l'objet (ceci permet d'avoir des variables de classe, des valeurs par défaut, ou des valeurs calculées du type SI-BESOIN)

<objet EST? >
Ramène la classe d'un objet.

<objet REPETE nombre fonction>
Force l'objet à répéter la fontion "fonction" un nombre "nombre" de fois.

4. EXEMPLE D'IMPLEMENTATION D'UN PROTOTYPE EN LANGAGE FROC

4.1 Introduction

L'analyse que nous pouvons faire à ce stade de notre recherche ne peut être complète. Nous avons concience de travailler dans un domaine nouveau qui nécessite un travail de longue haleine et demande la participation de tous : chercheurs en informatique et en architecture. C'est de cette pluridisciplinarité que dépend la réussite de notre approche.

L'intéret du langage que nous avons implémenté par rapport aux autres langages qui ont été proposés pour gérer le projet architectural réside dans l'approche très "pure" que nous avons recherché :

> Tout ce nous allons voir dans l' exemple que nous proposons est réalisable (tout au moins dans sa forme visible) avec des langages plus conventionnels. Il n'en est pas de même pour le fond : nous proposons surtout une méthodologie pour la réalisation des futurs systèmes de CAO en architecture. Il ne s'agit plus pour nous de faire un programme de CAO mais de permettre, dans un moyen terme, au concepteur de disposer d'un réel outil d'aide à la "conceptualisation".

Il est certain que la Programmation Orienté Objet ne va pas résoudre tous les problèmes que posent l'introduction de l'ordinateur dans le processus de conception. C'est cependant, actuellement, le seul outil qui respecte les systèmes de conception employés par les architectes.

Les exemples que nous avons implémentés ont pour but de permettre au lecteur de se familiariser avec l'approche objet et la programmation par envoi de messages. Ils montreront aussi les nouvelles possibilités offertes par l'intégration de la programmation par envoi de messages à l'intérieur d'une structure de données de type "FRAMES".

Les potentialités de ces nouveaux systèmes et leurs adéquations avec notre approche de la conception est un des points clées que nous souhaitons explorer plus avant dans nos recherches futures ; toutefois les premiers résultats

que nous avons obtenus sont très prometteurs et sont pour nous un outil incomparable pour valider les parti-pris, les choix et les objectifs qui ont été les nôtres depuis le début de notre recherche.

4.2 Exemple de manipulation d'objet architecturaux

Cet exemple à pour but de montrer que l'approche objet permet de manipuler de manière très simple et très cohérente les différents type d'"héritages " qui sont présents dans le projet architectural. Nous avons pris un exemple de l'architecture classique car la notion de "concept" et de "spécialisation" y est très forte ce qui va nous permettre de montrer que pour nous ces deux types d'héritages doivent être gérés différemment.

Les "objets" que nous utilisons par la suite sont ceux qui sont présentés au paragraphe 1.5 pour le logiciel Menon, et sont donc ceux qui peuvent constituer un porche classique.

4.2.1 Les héritages dûs à l'utilisation d'envois de messages

Nous partons sur le principe que le concepteur travaille à un certain niveau de définition qui se situe dans un premier temps au niveau du porche. Il manipule dans ce cas un objet ayant une sémantique très précise, toute manipulation à cette "échelle" doit être répercutée au niveau inférieur suivant un certain nombre de règles définies par le concepteur, dans notre exemple les règles sont des règles de composition de l'architecture classique.

Nous ne sommes pas en présence d'un objet "porche" spécialisé en d'autres objets, mais devant un concept défini par le savoir faire de l'architecte comme étant l'assemblage d'objets plus élémentaires suivant un certain nombre de règles. Toutes les techniques modernes de conception par éléments suivent cette approche et des manipulations autour de l'introduction de démarches équivalentes dans la conception industielle ou semi-industrielle sont en cours de réalisation.

Prenons un exemple de "méthode" de manipulation : la manipulation de la hauteur du porche :

La hauteur d'un "macro-élément " comme le porche conditionne la hauteur des différents éléments le constituant. Il influence en effet la hauteur de la colonnade, de la base, de l'entablement, et du fronton. La méthode PRENDS-HAUTEUR qui est définie dans la classe des

porches "PORCHE*" doit donc tenir compte de cela et envoyer des messages aux objets élémentaires pour réaliser ce maintien de cohérence.

L'utilisateur peut donc définir la classe PORCHE*

```
! METACLASS CREE PORCHE*
        (SURCLASSE METACLASS)
        (CHAMPS HAUTEUR COLONNADE BASE FRONTON ENTABLE) !

! PORCHE* APPREND  PRENDS-HAUTEUR
   ° (X) (RELIE HAUTEUR X)
         (SEND (TOI '? 'FRONTON) '(- 'HAUTEUR (QUOTIENT X 6))
         (SEND (TOI '? 'BASE) '(- 'HAUTEUR (QUOTIENT X 6))
         (SEND (TOI '? 'ENTABLE) '(- 'HAUTEUR (QUOTIENT X 6))
         (SEND (TOI '? 'COLONNADE) '(- 'HAUTEUR (QUOTIENT X 2)) $!
```

C'est la classe PORCHE* qui va permettre à l'objet spécifique PORCHE1, créé par l'utilisateur de manière dynamique durant la conception, de maintenir sa cohérence.

Si PORCHE1 reçoit comme hauteur 400 alors il va envoyer une suite de messages comme :

```
(SEND (TOI '? 'FRONTON) '(- 'HAUTEUR (QUOTIENT X 6))
```

Ce message se traduit par : TOI (l'objet qui est le recepteur actuel) se demande quel est son FRONTON et envoie ensuite à cet objet l'ordre de modifier sa hauteur : ici le FRONTON va prendre comme hauteur 400 divisé part 6, donc 66. Les sous-concepts de porche vont donc changer de hauteur quant on a fait une manipulation sur l'objet PORCHE1.

On peut généraliser cette démarche en forçant un concept à se modifier lors du changement des valeurs des objets le constituant :

Par exemple pour colonnade :

```
 ! METACLASS CREE COLONNADE*
        (SURCLASSE METACLASS)
        (CHAMPS HAUTEUR) !

! COLONNADE* APPRENDS PRENDS-HAUTEUR
   ° (X)(RELIE HAUTEUR X)
         (SEND (TOI '? PARTIE-DE) '(- 'HAUTEUR (TIMES X 2)) $
```

On remarque que la colonnade envoie au concept dont elle fait partie (relié par le lien PARTIE-DE) un message qui le force à modifier sa hauteur. Il est à remarquer aussi que le

même message PRENDS-HAUTEUR n'est pas interprété par les deux classes PORCHE* et COLONNADE* de la même manière. Ceci peut être généralisé à tous les objets qui composent ce concept.

```
! METACLASS CREE ENTABLE*
      (SURCLASSE METACLASS)
      (CHAMPS HAUTEUR) !

! ENTABLE APPRENDS PRENDS-HAUTEUR
   ° (X) (RELIE HAUTEUR X)
        (SEND (TOI '? PARTIE-DE) '(- 'HAUTEUR (TIMES X 6)) $ !

! METACLASS CREE FRONTON*
      (SURCLASSE METACLASS)
      (CHAMPS HAUTEUR) !

! FRONTON* APPRENDS PRENDS-HAUTEUR
   ° (X) (RELIE HAUTEUR X)
      (SEND (TOI '? 'PARTIE-DE) '(- 'HAUTEUR (TIMES X 3)) $ !


! METACLASS CREE BASE*
      (SURCLASSE METACLASS)
      (CHAMPS HAUTEUR) !

! BASE* APPRENDS HAUTEUR
   ° (X) (RELIE HAUTEUR X)
        (SEND (TOI '? 'PARTIE-DE) '(- 'HAUTEUR (TIMES X 6)) $
```

4.2.2 Les héritages dûs à la spécialisation

Ces héritages sont ceux que nous avons vus grâce à notre premier prototype basé sur une structure de FRAMES. La possibilité d'introduire ces héritages maintenant classiques avec les héritages de type méthodes en font un outil nouveau. Dans notre exemple nous l'avons introduit de manière très simple : pour voir toutes les possibilités de ces genres de systèmes le lecteur se référera à <LI2A-85>.

Voici une utilisation possible dans notre langage : les attachements procéduraux nous permettent de créer de manière automatique le symétrique de l'appartenance à un concept : si l'on déclare que FRONTON1 est un sous-objet de PORCHE1 alors le système rajoute automatiquement le fait que FRONTON1 est une PARTIE-DE PORCHE1 (ceci est dû à un aspect SI-AJOUT classique).

```
! PORCHE* AT-PROC COLONNADE SI-AJOUT '(SYMETRIQUE PARTIE-DE)'
```

exemple d'un prototype écrit en Froc

```
! PORCHE* AT-PROC FRONTON SI-AJOUT '(SYMETRIQUE PARTIE-DE)!

! PORCHE* AT-PROC BASE SI-AJOUT '(SYMETRIQUE PARTIE-DE) !

! PORCHE* AT-PROC ENTABLE SI-AJOUT '(SYMETRIQUE PARTIE-DE) !
```

De la même manière un objet peut regarder dans ses pères s'il a le droit de prendre une certaine valeur pour une propriété particulière :

```
! COLONNADE* AT-PROC HAUTEUR SI-AJOUT
                    '(VERIF-LIM 100 500 INTERDIT) !
```

Si la valeur de la hauteur de la colonne n'est pas comprise entre 100 et 500 alors l'action est interdite.

Le système actuel n'intègre pas le graphique mais celui-ci est en cours d'implémentation et va permettre de visualiser les modifications opérées sur les objets durant une session d'utilisation.

Le programme est écrit en LISP sous BFM-186. Le source, vu sa taille importante, ne figure pas dans ce rapport, mais peut être obtenu auprès du Laboratoire Li2A.

4.3. Exemple de session

Nous présentons ici un exemple de session utilisant le prototype décrit ci-dessus.
Les ordres donnés à l'interprète par l'utilisateur sont précédés du caractère "!"

```
! FILES CREE GEST-FICHIER !
GEST-FICHIER

! GEST-FICHIER RAMENE EXEMPLE STK !
```

La classe FILES a créé un gestionnaire de fichier GEST-FICHIER qui nous a servi ensuite à ramener le fichier EXEMPLE.STK (qui regroupe ce que nous avons vu au chapitre précédent).

```
! PORCHE* CHAMPS? !
(HAUTEUR COLONNADE BASE FRONTON ENTABLE)
Les champs de la classe PORCHE*

! PORCHE* CREE PORCHE1 !
PORCHE1
```

Instanciation des différents champs de l'objet PORCHE1

Par message simple :

```
 ! PORCHE1 <- COLONNADE COLONNADE1 !
COLONNADE1

 ! PORCHE1 ? COLONNADE !
COLONNADE1
```

Ou par message composé avec un receveur constant (caractère &) :

```
 ! PORCHE1 <- BASE BASE1 & <- FRONTON FRONTON1 & <- ENTABLE
ENTABLE1 !
ENTABLE1

 ! :PORCHE1 !
(NIL COLONNADE1 BASE1 FRONTON1 ENTABLE1)
```

Quelques tests pour déterminer la classe d'un objet :

```
 ! PORCHE1 EST COLONNADE* !
NIL

 ! PORCHE1 EST PORCHE* !
T
```

Un message composé avec un changement de receveur (caractère $) :

```
 ! FRONTON* CREE FRONTON1 $ BASE* CREE BASE1 $ COLONNADE*
CREE COLONNADE1 !
COLONNADE1

 ! ENTABLE CHAMPS? !
ENTABLE n'est pas un objet STK pour : ! ENTABLE CHAMPS? !
```

C'était une erreur de frappe...

```
 ! ENTABLE* CHAMPS? !
(HAUTEUR)
```

On crée un objet et on remplit ses champs dans le même temps :

```
! ENTABLE* CREE ENTABLE1 50 !
ENTABLE1
```

Tous les objets qui interviennent dans le concept PORCHE* sont maintenant définis

```
! ENTABLE1 ? HAUTEUR !
50
! COLONNADE1 ? HAUTEUR !
NIL
```

Elle n'a pas encore reçu de valeur.

On manipule le concept PORCHE1 en lui donnant une hauteur :

```
 ! PORCHE1 PREND-HAUTEUR 400 !
 400
 ! PORCHE1 ? HAUTEUR !
400
```

Mais nous avons une influence directe sur les éléments qui composent ce concept :

```
! COLONNADE1 ? HAUTEUR !
200
 ! FRONTON1 ? HAUTEUR !
66
! BASE1 ? HAUTEUR !
66
! ENTABLE1 ? HAUTEUR !
66
```

Aucun de ces objets n'a été manipulé directement ; l'affectation de leur hauteur est uniquement due à l'objet PORCHE* qui a envoyé un message à tous ses éléments quand il a reçu l'ordre de changer sa hauteur.

```
! COLONNADE1 EST? !
COLONNADE*
```

On manipule maintenant directement l'objet COLONNADE1 :

```
! COLONNADE1 HAUTEUR 300 !
300
! PORCHE1 ? HAUTEUR !
600
```

On remarque que l'effet inverse s'est bien produit : le PORCHE1 a bien changé de hauteur par envoi de message de l'objet le composant. Il est à remarquer ici que le message HAUTEUR n'a pas été interprété de la même manière par les

deux classes PORCHE* et COLONNADE*.

Nous allons réaliser maintenant un attachement procédural sur la classe des porches :

```
 ! PORCHE1 AT-PROC HAUTEUR SI-AJOUT '(VERIF-LIM 500 1000
INTERDIT) !
AT-PROC Methode inconnue de : PORCHE1
```

On ne peut naturellement pas faire d'attachements procéduraux sur les objets terminaux.

```
 ! PORCHE* AT-PROC HAUTEUR SI-AJOUT '(VERIF-LIM 500 1000
INTERDIT) !
(VERIF-LIM 500 1000 INTERDIT)

 ! PORCHE1 <- HAUTEUR 400 !
HAUTEUR Action interdite à cause de : PORCHE*
```

C'est bien ce que l'on a déclaré la ligne précédente :

```
 ! PORCHE1 ? HAUTEUR !
600
```

On enlève cet attachement procédural :

```
! PORCHE* DET-PROC HAUTEUR SI-AJOUT '(VERIF-LIM 500 1000
INTERDIT) !

! COLONNADE1 ? HAUTEUR !
300

! COLONNADE1 <- HAUTEUR 550 !
HAUTEUR Action interdite à cause de : COLONNADE*
```

Pourquoi ?

```
 ! COLONNADE* ECFRA !
    (COLONNADE* (EST-UN (VALEUR (METACLASS)))
                (HAUTEUR (SI-AJOUT ((VERIF-LIM 100
                                               500
                                               INTERDIT))))
                (PERE-DE (VALEUR (COLONNADE1)))))
La classe générative des colonnades à bien un aspect SI-
AJOUT

! COLONNADE1 <- HAUTEUR 400 !
400

 ! PORCHE1 HAUTEUR 1200 !
HAUTEUR Action interdite à cause de : COLONNADE*
```

Encore un envoi de message de la part de PORCHE1 : ici c'est un de ses éléments, COLONNADE1, qui a signalé que l'action était impossible.

Nous allons maintenant créer des classes dynamiquement :

```
! METACLASS CREE COLONNADE-SPECIALE*
    (SURCLASSE COLONNADE*)
    (CHAMPS NOMBRE-COLONNE ENTRAXE) !
COLONNADE-SPECIALE*

! COLONNADE-SPECIALE* APPREND LARGEUR?
  ' () (TIMES (PLUS NOMBRE-COLONNE 1) ENTRAXE)) §
```

La méthode LARGEUR? est écrite ici à la LISP car elle ne fait pas intervenir d'envoi de message.

```
 ! COLONNADE-SPECIALE* CREE COLONNADE2 !
COLONNADE2

! COLONNADE-SPECIALE* CHAMPS? !
(HAUTEUR NOMBRE-COLONNE ENTRAXE)
```

Il a hérité du champ HAUTEUR de sa surclasse COLONNADE*

```
 ! COLONNADE2 <- ENTRAXE 150 !
150

 ! COLONNADE2 <- NOMBRE-COLONNE 6 !
6

 ! COLONNADE1 LARGEUR? !
LARGEUR? Methode inconnue de : COLONNADE1
```

C'est normal car elle a été définie dans sa sous-classe.

```
 ! COLONNADE2 LARGEUR? !
1050
```

On continue :

```
 ! PORCHE* CREE PORCHE2 !
PORCHE2

 ! PORCHE2 <- COLONNADE COLONNADE2 !
COLONNADE2

 ! COLONNADE2 ? HAUTEUR !
NIL
```

```
 ! COLONNADE2 HAUTEUR 300 !
300

 ! PORCHE2 ? HAUTEUR !
600

 ! COLONNADE2 <- HAUTEUR 600 !
HAUTEUR  Action interdite à cause de : COLONNADE*
```

Toujours COLONNADE* dans les héritages :

```
 ! COLONNADE2 HFRA !
Les Pères sont :  (COLONNADE2 COLONNADE-SPECIALE* COLONNADE*
METACLASS OBJET)
Les Fils sont : (COLONNADE2)
```

COLONNADE-SPECIALE est aussi une spécialisation d'une autre classe :

```
 ! COLONNADE-SPECIALE* AT-PROC EST-UN VALEUR SPECIALE* !
SPECIALE*

 ! COLONNADE2 HFRA !
Les Pères sont :  (COLONNADE2 COLONNADE-SPECIALE* COLONNADE*
SPECIALE* METACLASS OBJET)
```

Une classe de plus dans l'héritage de la FRAME. Nous avons en effet ici un multihéritage.

```
 ! BYE !
Salut merci de votre visite
```

CONCLUSION

Nous orientons donc actuellement nos recherches vers les systèmes experts orientés objets. Dans ce genre de systèmes les règles de productions ne jouent plus qu'un rôle secondaire, la notion de moteur d'inférence devient moins importante : les mécanismes de déclenchement des règles ne sont plus centraux comme dans EMYCIN, TROPIC et autres. Au contraire, ils deviennent délocalisés, se perdent dans la masse des connaissances, pour prendre la forme de simples réflexes sensibles à la modification de telle ou telle donnée.

La distinction entre connaissances déclaratives, d'une part, et mécanisme de raisonnement, d'autre part, si elle demeure préservée, ne constitue pas pour autant une division en ce qui concerne le déroulement effectif du programme. Les systèmes construits autour de langages orientés objets, permettent d'établir des bases de connaissances importantes, et de garder cependant une modularité et des performances qui en font les rivaux directs des systèmes de règles de productions.

Notre programme futur recouvre un certain nombre de thèmes et axes de recherche, qui concernent aussi bien le champ de l'informatique que celui de l'architecture. Il faut considérer le contenu des thèmes que nous aborderons comme un ensemble homogène qui aboutirait à la définition et la réalisation à terme d'un système intelligent d'assistance à la conception. Il s'agira pour nous de manipuler à travers un langage des objets à haut niveau sémantique, comme nous venons de montrer qu'il est possible de faire. Bien sûr, l'intégration de capacités graphiques devra être faite, en particulier dans le langage Froc. Ainsi, bien au-delà de la simple manipulation de traits et de lignes, de 'dessins', notre but est de créer un outil capable de gérer des connaissances, des règles et des relations : l'expression graphique d'un 'objet' n'est plus alors que l'une de ses nombreuses occurences - certes privilégiée - à un moment determiné, dans un contexte précis.

Par ailleurs, les nouvelles générations de langages informatiques tels que Froc ont introduit de nouveaux concepts : ces langages définissent la notion de Classe, qui regroupe tous les objets obéissant de la même manière à l'envoi de messages identiques. Chaque objet hérite de manière directe ou indirecte d'attributs et de méthodes issus de sa classe générative. Ce système de représentation

permettra-t-il de définir de manière cohérente les caractéristiques des "éléments" architecturaux et des "méthodes" (ou actions) qu'ils sont capables d'interpréter ? La question fondamentale est alors de savoir si un système informatique bâti sur l'approche "objets" peut permettre la description de certains types de manipulations architecturales. Le travail que nous présentons ici apport déjà quelques éléments de réponse à cette question qu'il faudrait examiner maintenant sous d'autres angles.

En approfondissant les notions d'échelle dans la pratique conceptuelle de l'architecte, nous pensons pouvoir cerner d'une manière plus précise les niveaux pertinents permettant de réaliser des associations d'objets et/ou d'actions. La classification de ces objets ou actions permettrait l'établissement d'une base de connaissances qui pourrait être vide au départ, et progressivement remplie par l'utilisateur. Ceci lui permettrait de conserver sa propre démarche et concourrait à préserver ses savoir-faire.

Enfin, les systèmes experts en architecture sont étroitement liés à la répresentation des connaissances qui conditionne les performances du système. Il est donc primordial que la structure choisie pour représenter ces connaissnaces soit organisée de manière à permettre une exploitation maximale. Ainsi, à chaque niveau de représentation, à chaque échelle, à chaque niveau conceptuel, pourraient être associées des procédures d'expertises spécifiques. On pourrait ainsi envisager, dès les premières étapes de l'esquisse, aborder des problèmes tels que la définition approximative des coûts, l'évaluation progressive du comportement thermique du bâtiment, le respect des prospects d'urbanisme, etc...

BIBLIOGRAPHIE

Borne I. — Micro-Smalltalk pas-à-pas. Draft juin 84.

Borow D. G. — "An overview of KRL". In "Cognitive Science", 1977 .

Charniac E. — "Artificial Intelligence Programming". Lawrence Erlbraum Associates Publishers, 1980.

Cayrol M. — "Le Langage LISP", CEPADUES-EDITION, 1983.

Cointe P. — Une extension de VLISP par les objets. Science of computer Programming, 161 North Holland 1984

Cullen Y. — "Expert Systems in Architectural and Planning Education" Proceeding of the eCAADe, BRUSSEL 1983.

Durand J.N.L. — "Précis des leçons d'architecture données à l'Ecole Royale Polytechnique". Ed. Verlag Dr. Alfons Uhl, Nördlingen, 1981.

Dugerdil P. — Une méthodologie orientée objet pour la représentation des connaissances en C.A.O Architecture G.R.T.C. Marseille 1985

Duplay M. — "Methode illustrée de conception architecturale." Editions du Moniteur 1893.

Fareny H. — "LISP". Collection A B C des langages. Ed. Masson, 1984.

Goldberg A. — "SMALLTALK-80". Addison Welsley Pub, 1983.

Goulette J. P. — "Adaptation du noyau de Projet sur micro-ordinateur 16 bits en PASCAL". Rapport final d'une recherche financée par le Ministère de la Recherche Architecturale, LIZA, Juin 1984.

Goulette J. P. "Le dessin d'Architecte et l'Informatique". Travail personnel de fin d'études. Ecole d'Architecture de Toulouse, Novembre 1984

Latombe J.C. "Artificial intelligence in computer aided design". in Allen (Ed), CAD systems Amsterdam, North-Holland 1979.

LI2A-83 "Les utilitaires et l'intelligence artificielle pour un système d'aide à la conception en Architecture" Rapport final d'une recherche financée par le Secrétariat de la Recherche Architecturale. LI2A Juin 1983.

LI2A-84 "Les utilitaires et l'intelligence artificielle pour un système d'aide à la conception en Architecture" Rapport final d'une recherche financée par le Secrétariat de la Recherche Architecturale. LI2A Avril 1985.

LI2A-84
Caradant D. "Les utilitaires et l'intelligence artificielle pour un système d'aide à la conception en Architecture". Rapport final de recherche. LI2A, Juin 1984.

LI2A-85
J.P. Goulette
P. Pérez "Tangram, Manuel de l'utilisateur", in "Aides Intelligentes au dessin d'Architecte", Rapport final de la Convention AdI 84/935, LI2A, Sept. 85.

Mac-Carty J. "Lisp 1.5 Programmers Manual". The M.I.T. Press Cambridge Mass, 1962.

Michell T.M. "Data Base Management Systems and Expert Systems for CAD. CAD Systems Framework. North-Holland Publishing Company IFIP, 1983.

Minsky M. "A Framework for Representing Knowledge". in The Psychology of Computer Vision, McGraw-Hill, NY, 1975.

Raphael B. "A computer Program for Semantic Information Retrievial". in Semantic Information Proceedings, M.I.T Press, Cambridge, Mass 1968.

Rich E. A. "Users modelling via stereotypes". Cognitive Science, Vol 3, 1979.

Rich E. A. "Artificial Intelligence". Ed. McGraw-Hill in AI, 1983.

Roberts R. "The FRL Manual". MIT, AI Report, 1977.

Van Melle W. "The EMYCIN Manual". Technical Report, Heuristic Programming Project, Stanford University, 1981.

Waterman
Hayes-Roth "Pattern-Directed Inferences Systems". Ed Academic Press, 1977.

Weizenbaum "Symetric List Processor". Communication of ACM, April 1964.

Winston P. H. "LISP". Ed Addison-Wesley, Reading, Mass., 1981.

ANNEXES

Annexe 1 : le source du logiciel Menon

```
;       Logiciel Menon. Version 1.0.
        (C) Jean-Pierre Goulette / LIZA 1985


        L'expression fonctionnelle attachée à chaque atome
        constituant le porche. Permet de constituer et
        d'explorer la base de données de chaque élément.
;
(SETQ MACROG '(MLAMBDA LM$
(COND   ((SETQ AUX$ (GETP (CAR LM$)(CADR LM$)))(LIST 'CDR 'AUX$))
        ((MEMBER (CADR LM$) METATC)
          (PUTP (CAR LM$) (CADR LM$)
                (CONS 'VU (SETQ AUX$ (APPLY (CADR LM$) (LIST (CAR LM$)))
                                    )))
                'AUX$)
        (T NIL))))
))))))))))
%
(SETQ VU 'VU)
%

;       Procédure permettant la définition interactive des
        éléments par l'utilisateur. La représentation graphique
        de l'élément est attachée à la propriété IMA.
;
(DE DEFELT()
(PROG   (LOCAL G)
LOOP    (SETQ G (GENSYM))
        (PUTP G 'IMA (CREOBJET))
        (PUTD G MACROG)
        (PUSH G BASEG)
        (TERPRI)(PRIN1 '" Un autre ? : ")
        (COND   ((NULL (EQ (READCH) 'O))(RETURN T)))
        (GO LOOP)))))))))
%

(SETQ BASEG NIL)
%

;       Défintion des slots et des mécanismes de mise à jour
        de ces slots. La fonction MINMAX appliquée à une
        représentation graphique renvoie les coordonnées du
        rectangle circonscrit à l'élément.
;
```

```
(DF DEFSLOT L$
((PUSH (CAR L$) METATC)
(PUTD (CAR L$) (CONS 'LAMBDA (CDR L$)))
(CAR L$))))))))))))))

%
(SETQ METATC NIL)
%

(DE CONSTRUIT (M B PRED)
(PROG   (LOCAL RESULT)
        (SETQ RESULT NIL)
LOOP    (COND   ((NULL B)(RETURN RESULT))
                ((APPLY PRED (LIST (MINMAX (GETP (CAR B) 'IMA))))
                        (PUSH (CAR B) RESULT))
                (T NIL))
        (POP B)
        (GO LOOP))))))))))))

%
(DEFSLOT DESSOUS (G)
(CONSTRUIT (MINMAX (GETP G 'IMA))(DELETE G BASEG) 'SOUSP))))))))))
%

(DE SOUSP (M2)
(GREATEQP (CADAR M)(CADDR M2))))))))
%

(DEFSLOT DESSUS (G)
(CONSTRUIT (MINMAX (GETP G 'IMA))(DELETE G BASEG) 'SURP)))))))
%

(DE SURP (M2)
(LESSEQP (CADDR M)(CADAR M2))))))))))
%

(DE LESSEQP (A B)
(OR (EQ A B)(LESSP A B)))))))
%

(DEFSLOT TOUCHEY (G)
(CONSTRUIT (MINMAX (GETP G 'IMA))(DELETE G BASEG) 'TOUCHEYP)))))))
%

(DE TOUCHEYP (M2)
(OR (EQ (CADDR M)(CADAR M2))
        (EQ (CADAR M)(CADDR M2))))))))))
%
```

```
(DEFSLOT DEBOUT (G)
(PROG    (LOCAL M)
         (SETQ M (MINMAX (GETP G 'IMA)))
         (RETURN (COND    ((GREATP(SUB (CADDR M)(CADAR M))
                                   (SUB (CADR M)(CAAR M)))
                           '(T))
                          (T NIL)))))))))))))
%

(DEFSLOT DANSY (G)
(CONSTRUIT (MINMAX (GETP G 'IMA))(DELETE G BASEG) 'DANSYP)))))))))
%

(DE DANSYP (M2)
(AND     (GREATEQP (CADAR M)(CADAR M2))
         (LESSEQP (CADDR M)(CADDR M2)))))))))))
%

(DEFSLOT TAILLEX (G)
(CONSTRUIT (MINMAX (GETP G 'IMA))(DELETE G BASEG) 'TAILLEXP)))))))))))
%
(DE TAILLEXP (M2)
(AND     (EQ (CAAR M)(CAAR M2))
         (EQ (CADR M)(CADR M2)))))))))))
%
(DEFSLOT NIVEAUBAS (G)
(CONSTRUIT (MINMAX (GETP G 'IMA))(DELETE G BASEG) 'NVBP)))))))
%
(DE NVBP (M2)
(EQ (CADDR M)(CADDR M2)))))))
%

;        Les règles permettant la reconnaissance des éléments
         définis par l'utilisateur (ces règles utilisent les
         slots définis ci-dessus).
         Le signe ->T indique que la conclusion est une propriété
         terminale (l'élément est totalement identifié).
;
(SETQ REGLES '(
     (R1 (DESSOUS ?) ET (TOUCHEY ?) (-> PORTEUR ?))
     (R2 (DESSUS ?) ET (TOUCHEY ?) (-> SOUTENU ?))
     (R3 (PORTEUR) ET (DEBOUT) (->T COLONNE))
     (R4 (PORTEUR COLONNE) ET (NON SOUTENU COLONNE) (->T PIEDESTAL))
     (R5 (SOUTENU COLONNE) (->T ENTABLEMENT))
     (R6 (SOUTENU ENTABLEMENT) ET (TAILLEX ENTABLEMENT) (->T FRONTON))
     (R7 (DESSUS ENTABLEMENT) ET (DANSY FRONTON) (->T TYMPAN))
     (R8 (SOUTENU FRONTON) (->T STATUE))
     (R9 (DESSUS PIEDESTAL) OU (DESSOUS ENTABLEMENT) (-> OUVERTURE T))
     (R10 (OUVERTURE) ET (NIVEAUBAS COLONNE) (->T PORTE))
     (R11 (OUVERTURE) (->T CROISEE)))))))))))))
%
```

```
;       Le coeur du système exploitant les règles et permettant,
        par inférence, de constituer la base de données de
        chaque élément. Le mécanisme est réitéré jusqu'à la
        sémantisation totale de l'élément (ou l'épuisement des
        règles si l'élément n'est pas reconnu).
        Une trace des actions entreprises par le système permet
        de suivre pas à pas le déroulement de l'expertise.
;
(DE EXPERTISE ()
(PROG   (LOCAL BR BG)
        (SETQ BR REGLES)
        (SETQ BG BASEG)
        (TERPRI)(PRINT '"ON Y VA...")
LOOP    (COND   ((OR (NULL BR)(NULL BG))(RETURN T))
                (T(PROG (LOCAL BG2)
                        (SETQ BG2 BG)
                        (PRINT (CAR BR))
                LOPP    (COND   ((NULL BG2)(RETURN T))
                                (T(APPLIQUE(CDAR BR)(CAR BG2))))
                        (POP BG2)
                        (GO LOPP))))
        (PRINT (CAAR BR))
        (PRINT '"APPLIQUEE, ON RAMASSE LES CELLULES INUTILES...")
        (POP BR)
        (GC)
        (GO LOOP)))))))
%

(DE APPLIQUE (R G)
(PROG   (LOCAL ARG)
        (SETQ ? BASEG)
LOOP    (COND   ((EQ (CAR R) 'OU)(SETQ ? BASEG))
                ((OR (NULL R)(NULL ?))(RETURN NIL))
                ((EQ (CAR R) 'ET)NIL)
                ((EQ (CAAR R) '-))(PUTP G (CADAR R) (CONS 'VU
                                  (COND ((EQ (CADDAR R)T)'(T))
                                        (T(EVAL (CADDAR R))))))
                                        (RETURN T))
                ((EQ (CAAR R) '->T)(PUTP G 'TERMINAL (CADAR R))
                                        (SETQ BG (DELETE G BG))
                                        (INDIQUE (CADAR R))
                                        (RETURN T))
                (T (INFERE)))
        (POP R)
        (GO LOOP)))))))))
%

(DE CADDAR (X)(CAR (CDDAR X)))))))))))
%
```

```
(DE INFERE ()
(COND   ((EQ (CAAR R) 'NON)
                (SETQ ? (NULL (MEMP (CADDAR R)
                                    (TERM (EVAL (LIST G (CADAR R))))))))
        (T (SETQ ARG (EVAL (LIST G (CAAR R))))
                (COND   ((EQ (CADAR R) '?)(SETQ ? (INTER ? ARG)))
                        ((CADAR R)(SETQ ? (MEMP (CADAR R)(TERM ARG))))
                        (T (SETQ ? ARG)))))))))
%

(DE TERM (L)
(PROG   (LOCAL AUX RESULT)
        (SETQ RESULT NIL)
LOOP    (COND   ((NULL L)(RETURN RESULT))
                ((SETQ AUX (GETP (CAR L) 'TERMINAL))
                        (PUSH AUX RESULT))
                (T NIL))
        (POP L)
        (GO LOOP))))))))))
%

(DE MEMP (A L)
(COND   ((MEMBER A L)T)
        (T NIL))))))
%

(DE INTER (L1 L2)
(COND   ((OR (NULL L1)(NULL L2))NIL)
        ((EQ L1 T) L2)
        ((EQ L2 T) L1)
        (T(PROG (LOCAL RESULT)
                (SETQ RESULT NIL)
           LOOP (COND   ((NULL L1)(RETURN RESULT))
                        ((MEMP (CAR L1) L2)(PUSH (CAR L1)RESULT))
                        (T NIL))
                (POP L1)
                (GO LOOP))))))))
%

;       Procédure indiquant à l'utilisateur qu'un élément
        a été totalement identifié. Elle est utilisée pour
        imprimer une "trace" de l'expertise.
;
(DE INDIQUE(O)
(PROG   (LOCAL NO)
        (SETQ NO (GETP O 'N))
        (PRIN1 '"TIENS,")
        (COND   ((EQ NO 0)NIL)
                (T(PRIN1 '"ENCORE")))
        (PUTP O 'N (PLUS1 NO))
        (COND   ((EQ (GETP O 'MF) 'F)(PRIN1 '"UNE"))
```

```
                (T(PRIN1 '"UN")))
        (PUTP 0 'N (PLUS1 NO))
        (PRIN1 0)
        (RETURN (PRINT '" !!!")))))))))))
%

(PUTPLIST 'COLONNE '((MF F)(N 0))))))
%
(PUTPLIST 'PIEDESTAL '((MF M)(N 0)))))
%
(PUTPLIST 'ENTABLEMENT '((MF M)(N 0)))))))
%
(PUTPLIST 'FRONTON '((MF M)(N 0))))))
%
(PUTPLIST 'TYMPAN '((MF M)(N 0))))))
%
(PUTPLIST 'STATUE '((MF F)(N 0)))))))
%
(PUTPLIST 'CROISEE '((MF F)(N 0))))))
%
(PUTPLIST 'PORTE '((MF F)(N 0)))))
%

;       Affiche les résultats de l'expertise. Le dessin est
        éffacé, chaque élément réapparait à l'écran avec sa
        dénomination terminale.
;
(DE AFFICHE()
(PROG   (LOCAL BBG)
        (SETQ BBG BASEG)
        (EFFACE)
LOOP    (COND   ((NULL BBG)(RETURN T))
                (T(PRINT (GETP (CAR BBG) 'IMA))
                  (PRINT (GETP (CAR BBG) 'TERMINAL))
                  (READCH)))
        (POP BBG)
        (GO LOOP))))))))
%

;       Permet d'obtenir un dessin " au propre". Le système
        charge en mémoire, à partir de fichiers en
        bibliothèque, une représentation graphique exacte
        de l'élément, et la superpose à l'élément ébauché
        par l'utilisateur. La nouvelle représentation graphique
        est mise précisement à la position et aux proportions
        de l'élément ébauché.
;
```

```
(DE AU-PROPRE ()
(PROG   (LOCAL BG)
        (SETQ BG BASEG)
LOOP    (COND   ((NULL BG)(RETURN T))
                (T(REMPLACE(CAR BG)(MINMAX(GETP(CAR BG) 'IMA)))))
        (POP BG)
        (GO LOOP)))))))
%
(DE ASSOCIE (A)
(COND   ((EQ A 'COLONNE)        'COL1.PAD)
        ((EQ A 'PIEDESTAL)      'PIED1.PAD)
        ((EQ A 'ENTABLEMENT)    'ENTAB1.PAD)
        ((EQ A 'FRONTON)        'FRONT.PAD)
        ((EQ A 'TYMPAN)         'TYMP.PAD)
        ((EQ A 'STATUE)         'STAT.PAD)
        ((EQ A 'PORTE)          'PORTE.PAD)
        ((EQ A 'CROISEE)        'FEN.PAD)
        (T NIL)))))))))
%
(DE REMPLACE (A M)
(PROG   (LOCAL M2 O)
        (SETQ M2 (MINMAX (SETQ O(RETROUVE(ASSOCIE (GETP A 'TERMINAL))))
                         ))
        (DIMENSION (DIMENSION O          (RDIV
                                (SUB (CADDR M)(CADAR M))
                                (SUB (CADDR M2)(CADAR M2)))
                                'Y)
                   (RDIV
                   (SUB (CADR M)(CAAR M))
                   (SUB (CADR M2)(CAAR M2)))
                   'X)
        (SETQ M2 (MINMAX O))
        (RETURN (PUTP A 'ELT
                (TRANSLATE O
                        (SUB (CAAR M)(CAAR M2))(SUB (CADAR M)(CADAR M2))
))))))))
%
;       Efface l'écran est dessine le porche à l'aide des
        éléments extraits de la bibliothèque.
;
(DE RENDU ()
(PROG   (LOCAL BBG)
        (SETQ BBG BASEG)
        (EFFACE)
LOOP    (COND   ((NULL BBG)(RETURN T))
                (T(PRIN1 (GETP (CAR BBG) 'ELT))))
        (POP BBG)
        (GO LOOP)))))))))))))
%
END ; fin de programme ;
%
```

Annexe 2 : le source du logiciel Kosmigol

```
;*********************************************

;     Système expert Kosmigol
;       (c) Patrick Perez LI2A

;*********************************************

; le pattern-matcher
;---------------------------
;  Le  matcher  de l univers des constantes sur la  base  de
;donnée de chaque lien.
; On introduit un pattern et une liste de clauses constantes
; Si l unification est possible sur une clause, la fonction
; ramene T sinon nil
(DE MATCHDIRECT(X L)
(COND((NULL L)NIL)
     ((OR(EQUAL X (CAR L))(MATCHDIRECT X (CDR L)))))))

; Le matcher acceptant les variables sur la base de donnée
; d un objet. Si l unification est possible la fonction
; ramene  la  liste  des listes de  variables  avec  les
; constantes correspondantes dans les clauses
(DE MATCHVAR(X L I RES)
(SETQ RES NIL)
(SETQ I(CORRES X (CAR L)))
(COND((NULL L)NIL)
     (RES(MERGE I (MATCHVAR X (CDR L))))
     ((MATCHVAR X (CDR L))))))

(DE CORRES(L1 L2)
(COND((AND(NULL L1)(NULL L2))(SETQ RES T)NIL)
     ((OR(NULL L1)(NULL L2))NIL)
     ((EQUAL (CAR L1)(CAR L2))(CORRES(CDR L1)(CDR L2)))
     ((EQ(CAR(UNPACK(CAR   L1)))  '?)
        (CONS(LIST(CAR  L1)(CAR L2))
             (CORRES(CDR L1)(CDR L2))))))))))))
```

```
; Le moteur d inferences
;-------------------------

; unification d un predicat
(DE TRACER(PROP REP)
(COND((AND REP TRAC)(PRIN1 '->)
      (PRIN1 PROP)(PRIN1 '-/-)(PRINT REP)REP)
     (T REP))))))

(DE UNIFIEPRED(P I)
(COND((NULL P)T)
     ((EQ(CAR P) 'NOT)(COND((UNIFIEPRED(CDR P))NIL)
                           (T(TRACER P T))))
     ((EQ(CAR P) 'ASK)(COND((UNIFIEPRED(CDR P)))
                           (T(PRIN1 (CDR P))(PRIN1 '" ? :")
                             (SETQ I(LECTURE))
                             (COND((AND I(PURE(CDR P)))
                                   (EVAL(CONS '-> (CDR P)))
                                      I))
                            )))
     ((PURE P)(COND((MEMBER(CAR P) MOTS-RESERVES)
                    (TRACER P(EVAL P)))
                   ((TRACER P(MATCHDIRECT (CDR P)
                                (GET(CAR P)'BD))))
                   ((AND(SETQ I(GETD(CAR P)))
                      (UNIFIE(SUBSTITUTION(CAR I)
                                (CDR P)(CDR I)
                      )))(TRACER P T))))
     ((AND(SETQ I (GET(CAR P)'BD))
        (TRACER P(MATCHVAR(CDR P)I))))
     ((SETQ I (GETD(CAR P)))
      (TRACER P(GARDE(CDR P)
                 (UNIFIE(SUBSTITUTION(CAR I)
                           (CDR P)(CDR I))))
))))))))))))))))

; unification d une clause
; disjonction de conjonctions de predicats
(DE UNIFIE(X PL)
(COND((NULL X)NIL)
     ((SETQ PL (UNIFIECONJ(CDAR X)(UNIFIEPRED(CAAR X))))
        (UNIONL PL (UNIFIE(CDR X))))
     ((UNIFIE(CDR X)))))))

; unification d une conjonction de predicats
(DE UNIFIECONJ(X I J)
(COND((NULL X)I)
     ((EQ I T)
      (UNIFIECONJ(CDR X)(UNIFIEPRED(CAR X))))
     ((AND I(SETQ J(UNIFIECONJ(SUBTETE I(CDR X))
                  (UNIFIEPRED(SUBTETE I(CAR X))))))))
```

```
          (UNIONL(UNIONL(TETES    I)J)
                 (UNIFIECONJ X(DIMINA I))))
     (I(UNIFIECONJ X (DIMINA I))))))))))))))))))))))))

;---------------------------
; utilitaires d unification
;---------------------------
(DE GARDE(L P I)
(COND((OR(NULL P)(NULL L))NIL)
     ((AND(EQ(CAR(UNPACK(CAAR L))) '?)
          (SETQ I (MEMBER(CAAR L)P)))
      (CONS I(GARDE(CDR L)P)))
     ((GARDE(CDR L)P))))))))
 )))))

(DE MEMBER(X L)
(COND((NULL L)NIL)
     ((EQ(CAR L)X)L)
     ((ATOM(CAR L))(MEMBER X (CDR L)))
     ((MEMBER X (CAR L)))
     ((MEMBER X (CDR L)))))))))

(DE PURE(L)
(COND((NULL L)T)
     ((AND
       (COND((ATOM(CAR   L))
             (NOT(EQ(CAR(UNPACK(CAR   L)))'?)))
            (T T))
       (PURE(CDR L))))))))))))

(DE SUBST(X Y L R)
(COND((NULL L)R)
     ((ATOM(CAR L))(COND((EQ(CAR L)X)
                         (RPLACA L Y)(SUBST X Y (CDR L)R))
                        (T(SUBST X Y(CDR L)R))))
     (T(SUBST X Y (CAR L)R)
       (SUBST X Y (CDR L)R))))))))

(DE SUBSTITUTION(X Y Z)
(COND((NULL X)Z)
   (T(SUBSTITUTION(CDR X)(CDR Y)(SUBST(CAR X)(CAR Y)Z Z)
))))))))))

(DE UNIONL(L1 L2 I)
(COND((EQ L2 T)L1)
     ((NULL L1)L2)
     ((SETQ I(MEMBER(CAAR L1)L2))
      (CONS(CONS(CAAR L1)(UNION (CDAR L1)(CDR I)))
           (UNIONL(CDR L1)(ENLEVE I L2))))
     (T(CONS(CAR L1)(UNIONL(CDR L1)L2))))))))))))))))
```

```
(DE ENLEVE(L E)
(COND((NULL E)NIL)
     ((EQUAL L (CAR E))(CDR E))
     (T(CONS(CAR E)(ENLEVE L (CDR E))))))))))

(DE UNION(L1 L2)
(COND((NULL L1)L2)
     ((NOT(MEMBER(CAR L1)L2))
      (CONS(CAR L1)(UNION(CDR L1)L2)))
     (T(UNION(CDR L1)L2)))))))

(DE SUBTETE(X L)
(COND((NULL X)L)
     (T(SUBTETE(CDR X)(SUBSTV(CAAR X)(CADAR X)L))))))))))

(DE DIMINA(L)
(COND((OR(NULL L)(NULL(CDDAR L)))NIL)
     ((CONS(CONS(CAAR L)(CDDAR L))(DIMINA(CDR L)))))))))))

(DE SUBSTV(A B L)
(COND((NULL L)NIL)
     ((ATOM L)(COND((EQ L A)B)
                   (T L)))
     ((CONS(SUBSTV A B (CAR L))
        (SUBSTV A B (CDR L))))))))))

(DE MERTETE(A L)
(COND((OR(NULL L)(EQ L T))A)
     ((APPEND(TETES A)L)))))))

(DE TETES(L)
(COND((NULL L)NIL)
     ((CONS(LIST (CAAR L)(CADAR L))(TETES(CDR L)))))))

(DE MERGE(A B)
(COND((NULL A)B)
     ((NULL B)A)
     ((EQ A T)B)
     ((EQ B T)A)
     ((CONS(APPEND(CAR    A)(CDAR    B))(MERGE(CDR    A)(CDR
B)))))))))))


;--------------------------------
; environnement et utilitaires
;--------------------------------

; introduction d une regle ou clause d unification

(PUTD 'REGLE '(NLAMBDA X
(PUTD (CAR X)(CONS(CADR X)(CADDR X)))
```

```
(PUT (CAR X) 'PATTERN (TRPAT1(CADR X)(CADDR X)))
(PRINT " c'est correct, regle introduite ")
(CAR X))))))))))))

(DE TRPAT1(L F)
(COND((NULL L)NIL)
     (T(CONS(TRPAT2(CAR L)F)(TRPAT1(CDR L)F))))))))
(DE TRPAT2(A F)
(COND((NULL  F)(PRINT  '"  Erreur  :  Tentative  de  Post-
declaration de constantes")
           (THROW 'ERREUR))
     ((MEMBER A '(?ADJECTIF ?VERBE ?NOMCOMMUN))A)
     ((TRPAT3 A(CAR F)))
     ((TRPAT2 A (CDR F))))))))))

(DE TRPAT3(A F)
(COND((NULL F)NIL)
     ((TRPAT4 A (CAR F)))
     ((TRPAT3 A (CDR F))))))))
(DE TRPAT4(A F I)
(COND((MEMBER A F)
        (SETQ I (DIFFERENCE(LENGTH
                  (MEMBER A (REVERSE(CDR F))))1))
                (NTH I (FORMO(CAR F))))))))))))))

; introduction d un fait ou constante d unification sur la
;  base  de  donnee d un lien (atome porteur d une  base  de
; donnees)

(DE -> X
(COND((NULL(GET(CAR X) 'BD))(PRINT " c'est bon ")
      (PUT(CAR X) 'BD (CONS (CDR X)(GET(CAR X) 'BD))))
     (T(SETQ INTZ (CORRECI(CDR X)(CAR X)))
       (COND((SETQ ERT (MATCHDIRECT INTZ (GET(CAR X) 'BD))))
             (T(SETQ ERT(MATCHVAR INTZ (GET(CAR X) 'BD))))))

     (COND
     ((AND ERT (LESSP(LENGTH INTZ)(LENGTH (CDR X))))
      (PRINT " ATTENTION : perte de definition de l objet")
      (PRINT " par elimination des champs sur-definis")
      (PRIN1 INTZ)(PRIN1 "  -> EX:  ")(PRINT (CAR(GET(CAR X)
'BD)))
      (PRIN1 ERT)(PRINT " REJET")(TERPRI))
     ((AND(GREATERP(LENGTH INTZ)(LENGTH (CDR X)))
       ERT)
      (PRINT " ATTENTION : perte de definition de l objet")
      (PRINT " par collision sur champs sous-definis")
      (PRIN1 INTZ)(PRIN1 "  -> EX: ")(PRINT (CAR(GET(CAR X)
       'BD)))
      (PRIN1 ERT)(PRINT " REJET")(TERPRI))
     (ERT NIL)
```

```
      (T(PRINT "Ok one more ! ")
        (PUT(CAR X) 'BD (CONS INTZ (GET(CAR X)'BD))))))))))
))))))))

(DE CORRECI(L AT I)
(SETQ I (FORMO AT))
(COND((NULL I)L)
     ((COMPAREC L I))))))))

; declaration d une fonction LISP dans le moteur d inference
(DF RESERVE X
(COND((NOT(MEMBER(CAR X)MOTS-RESERVES))
      (SETQ MOTS-RESERVES (CONS(CAR X) MOTS-RESERVES))))
(PUT (CAR X) 'PATTERN (CADR X))
(EVAL(CONS 'DF (CONS(CAR X)(CDDR X)))))))))

(DE RESET()
(SETQ VOCABULAIRE '(
(?ADJECTIF(?ADJECTIF))
(?NOMCOMMUN(?NOMCOMMUN))
(?VERBE(?VERBE))
(POURQUOI(POURQUOI))
(QUI (QUI))
(QUE(QUE))
(COMMENT(COMMENT))
(EST-CE-QUE(EST-CE-QUE))
(SI(SI))(ALORS(ALORS))(ET(ET))(OU(OU))
))
(SETQ  VOCABIDON '(LE LA LES DE DES D UN UNE DU L A ONT  EST
ETRE SONT))
(SETQ OBJETS NIL)
(SETQ REGLES NIL)
(SETQ ADJECTIF '(?ADJECTIF))
(SETQ GENV 0)
(SETQ VERBE '(?VERBE))
(SETQ NOMCOMMUN '(?NOMCOMMUN))
(SETQ MOTS-RESERVES NIL)
(RECLAIM)))))
```

```
;      PHASE D ANALYSE LEXICALE
;************************************************
(DE TRADUIT(X)
(SETQ X (DELETELIST X VOCABIDON))
(INTRO1 X)))))))
(DE INTRO1(X F Z)
(COND((OR(EQ X T)(NULL X))())
     ((SETQ    F    (TROUVEEQ    X    VOCABULAIRE))(CONS(CAR
      F)(INTRO1(CDR F))))
     (T (PRIN1 " Je ne comprends pas ")(PRINT (CAR X))
      (PRINT   "  Donnez-moi un synonyme  ou  périphrase,  ou
      corrigez  ")
      (PRIN1 " Entre parentheses svp : ")
      (COND((SETQ F(TROUVEEQ(APPEND(DELETELIST
                    (SETQ Z(LECTURE))
                   VOCABIDON)(CDR X))
                   VOCABULAIRE))
            (CONS(CAR F)(INTRO1(CDR F))))
           (T(PRIN1  '" Je  ne comprends  toujours  pas  :
")(PRINT Z)
            (PRINT '" Est-ce :  1  un nouveau synonyme")
            (PRINT '"           2  un element de syntaxe")
            (PRINT '"           3  un nouveau mot")(PRIN1
'" ? : ")
            (SETQ F (LECTURE))
            (COND((EQ F '1)(PRIN1 '" De quel mot ? ")
                           (SETQ F (LECTURE))
                      (SETQ   VOCABULAIRE(AJOUSYN   F   Z
VOCABULAIRE)))
                  ((EQ  F  '2)(SETQ  VOCABIDON(APPEND   Z
VOCABIDON)))
                  (T (TERPRI)
                     (PRINT '" Quel type ?")
                     (PRINT '"       1  un verbe")
                     (PRINT '"       2  un adjectif")
                     (PRINT  '"        3  un nom  propre  ou
commun")
                     (PRINT     '"          4     une     clef
(conjonction etc...)")
                     (PRIN1 '" ? : ")(SETQ F (LECTURE))
                     (COND((EQ  F  1)(SETQ VERBE  (CONS(CAR
Z)VERBE)))
                           ((EQ F 2)(SETQ  ADJECTIF(CONS(CAR
Z)ADJECTIF)))
                           ((EQ F 3)(SETQ NOMCOMMUN
                                          (CONS(CAR
Z)NOMCOMMUN))))
                     (SETQ VOCABULAIRE
                      (CONS     (CONS(CAR    Z)(LIST    Z))
VOCABULAIRE)))
            (TRADUIT X)))))))))))))))))))))))))))))))
```

```
(DE AJOUSYN(M S L)
(COND((NULL  L)(PRINT '" Je ne trouve pas le mot  dont  vous
donnez le synonyme")
                NIL)
     ((EQ  (CAAR  L) M)(CONS(CONS M (CONS S  (CDAR  L)))(CDR
L)))
     (T(CONS(CAR L)(AJOUSYN M S (CDR L))))))))))))))

;***PHRASE SUR PATTERN
(DE MATCH(F P R)
(SETQ R (LENGTH P))
(COND((EQ(LENGTH F) R)(EQUALMAX F P))
     ((EQUALMAX (DECORT R F)P)(NTH F(PLUS R 1)))))))))

(DE DECORT(N L)
(COND((EQ N 1)(LIST(CAR L)))
     ((CONS(CAR L)(DECORT(DIFFERENCE N 1)(CDR L)))))))))))

;***PHRASE SUR ENSEMBLE DE PATTERN
(DE MATCH1(F ENS)
(COND((NULL ENS)())
     ((MATCH F(CAR ENS)))
     ((MATCH1 F (CDR ENS)))))))

;***TROUVE LES SEQUENCES EQUIVALENTES
(DE TROUVEEQ(F VOCA A B)
(COND((NULL VOCA)())
     ((SETQ A (MEMBERL (CAR F)(CAR VOCA)))
      (COND((SETQ B (MATCH1 F A)) (CONS (CAAR VOCA) B))
           ((TROUVEEQ F (CDR VOCA)))))
     ((TROUVEEQ F (CDR VOCA)))))))))

(DE MEMBERL(X L)
(COND((IN X (CDR L))(CDR L)))))

(DE IN(X L)
(COND((NULL L)())
     ((ATOM L)(EQPO(UNPACK X)(UNPACK L)))
     ((OR(IN X (CAR L))(IN X (CDR L)))))))))))))))

(DE DELETELIST (F V)
(COND((NULL F)())
     ((MEMBER(CAR F)V)(DELETELIST (CDR F) V))
     ((CONS(CAR F)(DELETELIST(CDR F) V)))))))))))))))
```

```
(DE DELETE(X L)
(COND((NULL L)NIL)
     ((ATOM(CAR L))(COND((EQ(CAR L)X)(DELETE X (CDR L)))
                        (T(CONS(CAR  L)(DELETE X  (CDR
L))))))
     (T(CONS(DELETE X (CAR L))
             (DELETE X (CDR L)))))))))
(DE EQUALMAX(L1 L)
(COND((AND(NULL L1)(NULL L))T)
     ((OR(NULL L1)(NULL L))NIL)
     ((EQPO (UNPACK(CAR L1))(UNPACK(CAR L)))
      (EQUALMAX(CDR L1)(CDR L)))))))))))))
(DE EQPO (A1 A2)
(COND((NULL A2)T)
     ((NULL A1)())
     ((AND(EQ(CAR     A1)(CAR     A2))(EQPO(CDR     A1)(CDR
A2)))))))))))
;************************************
; PHASE D'ANALYSE SYNTAXIQUE
;************************************
(DE MATCHSYN(P DA)
(COND((AND(NULL P)(NULL DA))T)
     ((AND(NULL  DA)(NULL(CDR  P))(EQ(CAR  (UNPACK(CAR  P)))
'*)))
               (SET(PACK(CDDR(UNPACK(CAR P))))NIL)T)
     ((OR(NULL P)(NULL DA))NIL)
     ((EQ(CAR P)(CAR DA))(MATCHSYN(CDR P)(CDR DA)))
     ((AND(LISTP(CAR P))(MATCHSYN(CDR P)(CDR DA))(MEMBER(CAR
DA)(EVAL(CADAR P))))
               (SET(CAAR P)(CAR DA))T)
     ((EQ(CAR(UNPACK(CAR P)))'?)(MATCHSYN(CDR P)(CDR DA)))
     ((EQ(CAR(UNPACK(CAR P))) '*)
               (COND((AND(EQ(CADR(UNPACK(CAR P))) '?)
                         (MATCHSYN(CDR P)(CDR DA)))
                         (SET(PACK(CDDR(UNPACK(CAR  P))))(CAR
DA))T)
                    ((EQ(CADR(UNPACK(CAR P))) '*)
                         (COND((MATCHSYN(CDR P)DA)
                                   (SET(PACK(CDDR(UNPACK(CAR
P))))
                                        NIL)T)
                              ((MATCHSYN(CDR P)(CDR DA))
                                   (SET(PACK(CDDR(UNPACK(CAR
P))))
                                        (LIST(CAR DA)))T)
                              ((MATCHSYN P(CDR DA))
                                   (SET(PACK(CDDR(UNPACK(CAR
P))))
                                        (CONS(CAR DA)
                                          (EVAL(PACK(CDDR(UNPACK(CAR
P)))))))T))))))))))))))))))
```

```
(DE MAJR(L)
(COND((ATOM(CAR L))(DELETE '-> (LIST(LIST L))))
     ((ATOM(CAAR L))(DELETE '->(LIST L)))
     (T (DELETE '-> L))))))))))

(DE CONSTRUITOU(A B)
(COND((ATOM(CAR A))(CONSTRUITOU(LIST A)B))
     ((ATOM(CAR B))(CONSTRUITOU A(LIST(LIST B))))
     ((AND(ATOM(CAAR A))(ATOM(CAAR B)))(CONS A(LIST B)))
     (T(CONS A B))))))))))))

(DE CONSTRUITET(A B)
(COND((ATOM(CAR B))(CONSTRUITET A (LIST B)))
     (T(CONS A B)))))))

(DE ANASYN(X  TEST A B C D E F G H I J)
(COND((NULL X)NIL)
     ((MATCHSYN '(POURQUOI )*B)X)
      (SETQ TRAC T)(ANASYN (CONS 'EST-CE-QUE B)))
     ((MATCHSYN '(SI )*B ALORS )*C (D VERBE) )*E)X)
      (LIST 'REGLE D (APPEND(ANASYN C)(ANASYN E))
        (MAJR(ANASYN B))))
      ((MATCHSYN '(QUI (A VERBE) )*B)X)
      (LINEA(LIST '? A '< '?NOMCOMMUN '> (ANASYN B))))
     ((MATCHSYN '(QUI (A ADJECTIF))X)
      (LINEA(LIST '? A '< '?NOMCOMMUN '>)))
     ((MATCHSYN '(QUE (A VERBE) )*B)X)
      (LINEA(LIST '? A (ANASYN B) '< '?NOMCOMMUN '>)))
     ((MATCHSYN '(COMMENT (A NOMCOMMUN))X)
      (LINEA(LIST '? A '< '?ADJECTIF '>)))
     ((MATCHSYN '(EST-CE-QUE )*A (B VERBE) )*C)X)
      (LINEA(LIST '? B (ANASYN A)(ANASYN C))))
;NC ET NC VB Q
    ((MATCHSYN   '((A  NOMCOMMUN)ET(B  NOMCOMMUN)(C   VERBE)
)*D)X)
      (EVAL(LINEA(LIST '-> C A (SETQ I(ANASYN D)))))
      (LINEA(LIST '-> C B I)))
;Q VB NC ET NC
    ((MATCHSYN   '()*D   (C  VERBE)(A   NOMCOMMUN)   ET   (B
NOMCOMMUN))X)
      (EVAL(LINEA(LIST '-> C (SETQ I(ANASYN D))A)))
      (LINEA(LIST '-> C I B)))
;Q OU Q
    ((MATCHSYN '()*A OU )*B)X)
      (CONSTRUITOU(ANASYN A)(ANASYN B)))
;Q ET Q
     ((MATCHSYN '()*A ET )*B)X)
      (CONSTRUITET(ANASYN A)(ANASYN B)))
;Q VB Q
     ((MATCHSYN '()*A (B VERBE) )*C)X)
      (LINEA(LIST '-> B (ANASYN A)(ANASYN C))))
```

```
;Q NC Q
     ((MATCHSYN '()*A (B NOMCOMMUN) )*C)X)
      (SETQ A (ANASYN A))(SETQ C(ANASYN C))
      (ANAETRE B A)(ANAETRE B C)
      (COND(TEST(LINEA(LIST '-> A C B)))
            (T(LINEA(LIST A C B)))))
;Q ADJ Q
     ((MATCHSYN '()*A (B ADJECTIF) )*C)X)
      (LINEA(LIST(ANASYN A) B (ANASYN C))))

; verbe
     ((MATCHSYN '((A VERBE)) X) X)

     (T (PRIN1 '" Syntaxe trop complexe sur :")(PRIN1 X)
       (PRINT " Desole!!!!")(THROW 'ERREUR)))))))))

(DE ANAETRE(O L)
(COND((NULL L)NIL)
     (T(EVAL(LIST '-> O (CAR L)))
       (EVAL(LIST '-> (CAR L) O))
       (ANAETRE O (CDR L)))))))

(DE LINEA(X)
(COND((NULL X)NIL)
     ((NULL(CAR X))(LINEA(CDR X)))
     ((ATOM(CAR X))(CONS(CAR X)(LINEA(CDR X))))
     ((APPEND(CAR X)(LINEA(CDR X)))))))))

(DE DIALOG(DIA)
(CATCH 'ERREUR)
(TERPRI)(SETQ TRAC NIL)
(PRIN1 '" ? : ")(DIAL (TRADUIT(LECTURE)))
))))))))))

(DE DIAL(X)
(COND((MATCHSYN '()*A REVOIR )*A) X)(PRINT 'CIAO!!!))
     ((MATCHSYN  '(DETRUIT )*B (A VERBE)) X)(RPLACD  A
NIL)(PRINT "c'est fait")
                              (DIALOG))
     ((MATCHSYN '(MONTRE )*B BASE )*B (A VERBE))X)(PRINT(CDR
A))(DIALOG))
     ((MATCHSYN      '()*B      NOMCOMMUNS      )*B)X)(PRINT
NOMCOMMUN)(DIALOG))
     ((MATCHSYN '()*B VERBES )*B)X)(PRINT VERBE)(DIALOG))
     ((MATCHSYN       '()*B      ADJECTIFS      )*B)X)(PRINT
ADJECTIF)(DIALOG))
     ((MATCHSYN     '(MONTRE     )*B    REGLE     )*B     (A
VERBE))X)(PRINT(GETD A))(DIALOG))
     ((AND(MATCHSYN '((A NOMCOMMUN))X)
          (MATCHSYN '()*C (B NOMCOMMUN) )*D) PHRASEP))
```

```
      (PRINT " Essai de tournure elliptique avec :")
      (DIAL(PRINT(SUBSTV B A PHRASEP))))
      (T(SETQ PHRASEP X)(EVAL(ANASYN X T))(DIALOG))))))))))

(DE LISTP(X)(NULL(ATOM X))))))))

;*************************************
;     CORRECTION DE STRUCTURE
;*************************************

; formation du motif lié à une regle
(DE FORMO(AT)
(COND((FORMOTIF(CAR(GET AT 'BD))))
     ((FORMOTIF(GET AT 'PATTERN)))
     (T NIL)))))))

; formation du motif
(DE FORMOTIF(PP)
(COND((NULL PP)NIL)
     (T(CONS(STSY(CAR PP))(FORMOTIF(CDR PP))))))))))))))
)))))

(DE STSY(X)
(COND((MEMBER X ADJECTIF)'?ADJECTIF)
     ((MEMBER X NOMCOMMUN)'?NOMCOMMUN)
     ((MEMBER X VERBE) '?VERBE)
     (T '*UNDEF))))))))))

; comparateur et reconstruction de requete
(DE COMPAREC(PH PAT)
(COND((AND(NULL PAT)(NULL (CDR PH))(EQ(CAR PH) '))) '()))
     ((NULL PAT)NIL)
     ((NULL PH)PAT)
     ((OR(EQ(CAR    PH)    '()(EQ(CAR    PH)    ')))(CONS(CAR
PH)(COMPAREC(CDR PH)PAT)))
     ((OR(EQ (CAR PH)(CAR PAT))
       (EQ(STSY(CAR PH))(CAR PAT)))
           (CONS (CAR PH)(COMPAREC(CDR PH)(CDR PAT))))
     ((AND(EQ(STSY(CAR PH)) '?ADJECTIF)
       (EQ (CAR PAT) '?NOMCOMMUN))
            (COMPAREC(CDR PH)PAT))
     ((AND(EQ(STSY(CAR PH)) '?NOMCOMMUN)
       (EQ(CAR PAT) '?ADJECTIF))
        (CONS '?ADJECTIF (COMPAREC PH (CDR PAT))))
      (T    (PRINT    '"    Impossible    à    analyser")(THROW
'ERREUR))))))))))))))
```

```
(DE ? X
(SETQ RE11(UNIFIEPRED(CONS(CAR X)
      (ENLEVCHEV(COMPAREC(CDR X)(FORMO(CAR X)))))))
(PRIN1 "Reponse ")(REPONEP REP RE11)T))))))))

(DE ENLEVCHEV(LEF FEF)
(COND((NULL LEF)NIL)
     ((EQ (CAR LEF) '()(SETQ REP NIL)(ENLEVCHEV(CDR LEF)T))
     ((EQ (CAR LEF) '))(SETQ REP(REVERSE REP))(ENLEVCHEV(CDR
LEF)NIL))
     (FEF (SETQ GENV(PLUS GENV 1))
        (SETQ REP (CONS(PACK(LIST(CAR LEF)GENV))REP))
       (CONS(PACK(LIST(CAR   LEF)GENV))(ENLEVCHEV(CDR   LEF)
FEF)))
     (T (CONS(CAR LEF)(ENLEVCHEV(CDR LEF)))))))))))

(DE REPONEP(PRE LRE IRE)
(COND
     ((EQ LRE T)(PRINT '"  - VRAI"))
     ((EQ LRE NIL)(PRINT '"  - FAUX"))
     ((NULL PRE)NIL)
     (T(SETQ   IRE   (MEMBER(CAR   PRE)LRE))(PRIN1   '"     -
")(PRINT(CDR IRE))
       (REPONEP(CDR PRE)LRE))
     )))))))


;*******************************
; SAUVETAGE DU DICTIONNAIRE
;*******************************

(DE SAUVE(X)
(WRS X SXX)
(PRINT (LIST 'SETQ 'VOCABIDON '"'" VOCABIDON))
(PRINT (LIST 'SETQ 'VOCABULAIRE '"'" VOCABULAIRE))
(PRINT (LIST 'SETQ 'VERBE '"'" VERBE))
(PRINT (LIST 'SETQ 'NOMCOMMUN '"'" NOMCOMMUN))
(PRINT (LIST 'SETQ 'ADJECTIF '"'" ADJECTIF))
(PRINT '(RDS))
(WRS)
T))))))

(DE LECTURE()
(PRINT (READ))))))
```