Les utilitaires et l'intelligence artificielle pour un système d'aide à la conception en architecture

Michel Léglise, Dominique Caradant, Jean-Pierre Goulette, Patrick Pérez

284

Ecole d'Architecture de Toulouse - Departement de la Recherche
Chemin du Mirail - 31100 - TOULOUSE - FRANCE - Tél. (61) 40 47 28
Laboratoire d'informatique appliquée à l'architecture

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

- 1984 -

Dominique Caradant
Jean Pierre Goulette
Patrick Pérez

RAPPORT FINAL DE RECHERCHE

Rapport final d'une recherche subventionnée par le Ministère de l'Urbanisme et du Logement, Direction de l'Architecture, Secrétariat de la Recherche Architecturale (décision n° 64067 du 6 Août 1984).

AVRIL 1985.

L12a
Ecole d'Architecture de Toulouse - Departement de la Recherche
Chemin du Mirail - 31100 - TOULOUSE - FRANCE - Tél. (61) 40 47 28
Laboratoire d'informatique appliquée à l'architecture

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

- 1984 -

Dominique Caradant
Jean Pierre Goulette
Patrick Pérez

RAPPORT FINAL DE RECHERCHE

Rapport final d'une recherche subventionnée par le Ministère de l'Urbanisme et du Logement, Direction de l'Architecture, Secrétariat de la Recherche Architecturale (décision n° 64067 du 6 Août 1984).

AVRIL 1985.

Ecole d'Architecture de Toulouse.
Département de la Recherche.
Laboratoire Li2A.
Chemin Aristide Maillol. 31057 Toulouse Cedex.
Tél : (61) 40 47 28 poste 223.

Dominique Caradant
Jean-Pierre Goulette
Patrick Pérez

LES UTILITAIRES ET L'INTELLIGENCE ARTIFICIELLE

POUR UN SYSTEME D'AIDE A LA CONCEPTION EN ARCHITECTURE

- 1984 -

RAPPORT FINAL DE RECHERCHE

Rapport final d'une recherche subventionnée par le Ministère de l'Urbanisme et du Logement, Direction de l'Architecture, Secrétariat de la Recherche Architecturale (décision n° 64067 du 6 Août 1984).

Responsable scientifique : Michel Léglise

Li2A, Avril 1985.

SOMMAIRE

--==*==--

INTRODUCTION

Ce rapport se présente en deux parties qui permettent d'explorer les orientations générales de la recherche à travers différents exemples.

Une première approche permettra au lecteur de se familiariser avec les divers domaines d'exploitation du logiciel "Tangram", logiciel mis au point au Laboratoire Li2A dans un autre contexte que celui de la présente recherche. Ce logiciel est un interpréteur type Lisp comportant un éditeur graphique et doté d'une grande puissance, en particulier dans le domaine de l'interactivité.

Cette première manipulation nous conduit par la suite à définir une forme plus évoluée de dialogue avec l'utilisateur. Nous étudions alors une autre génération de langages : les langages orientés objets, en montrant quelle peut être la pertinence de leur utilisation en phase de conception en Architecture.

Deux études sont alors proposées à partir de ces langages, études différentes mais complémentaires, qui montrent à la fois les difficultés et les potentialités de telles utilisations dans un domaine de la connaissance aussi complexe que celui de l'Architecture.

La conclusion porte sur une évaluation des méthodes utilisées et sur les aspects aussi bien positifs que contraignants de telles méthodes.

INTRODUCTION PAR UNE MANIPULATION SIMPLE

Dans ce premier chapitre nous illustrerons les multiples aspects de Tangram à travers un exemple : une manipulation simple et interactive d'éléments d'architecture soumise à un contrôle de cohérence.

Cette première approche permettra au lecteur de se familiariser avec les divers domaines d'exploitation de ce logiciel (langage de programmation interpreté, éditeur graphique, base de données permettant une définition interactive de l'univers de travail) et de saisir ainsi la potentialité de leur symbiose. Le lecteur intéressé par une description plus précise des modes d'utilisation du logiciel pourra se référer à "TANGRAM, manuel de l'utilisateur" <LI2A-85>.

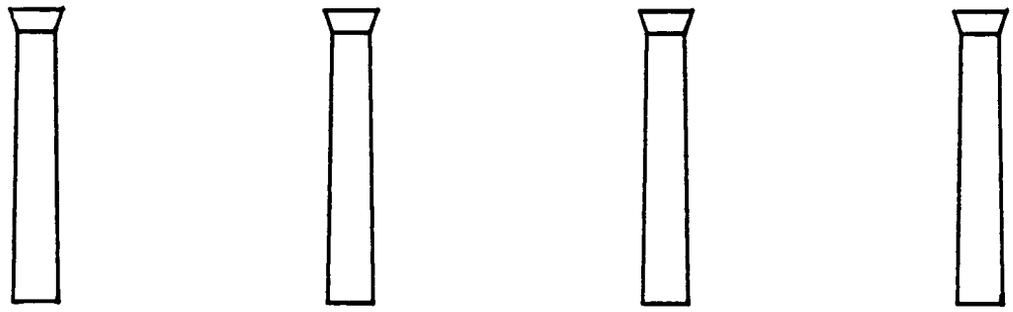
Nous diviserons la présentation de notre étude en plusieurs étapes :

Première étape : constitution d'une bibliothèque d'éléments architecturaux.

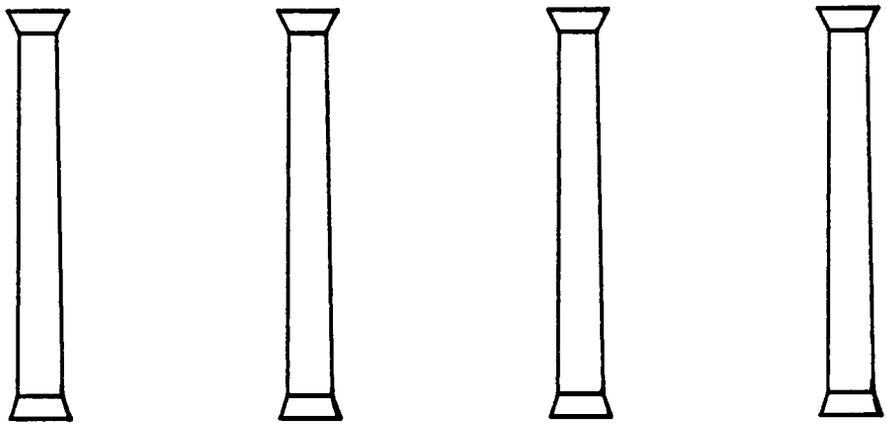
Nous avons défini, à partir de l'éditeur graphique de Tangram un ensemble d'éléments d'architecture en tous points conformes à ceux exposés par J.N.L. Durand dans son précis des leçons d'architecture <Durand-81>. Cette bibliothèque regroupe divers types d'objets, mais, pour la suite de cette étude, nous nous intéresserons plus précisement à un sous-ensemble bien particulier des "soutiens verticaux isolés" : les cinq genres de colonne (nous retiendrons, pour qualifier nos colonnes, le terme de "genre", employé par Durand, qui nous semble plus souple que les dénominations classiques : dorique, ionique, corinthien, toscan et composite).

Outre une description complète des modes de tracé des différents genres de colonne (où chaque grandeur est "soumise" au module), Durand nous livre les règles de syntaxe permettant de composer une répétition de soutiens isolés (dans notre cas, une colonnade) correctement rythmée. Nous dispensons le lecteur d'une description détaillée des règles grammaticales de Durand, nous préférons lui en fournir une illustration sur la page suivante.

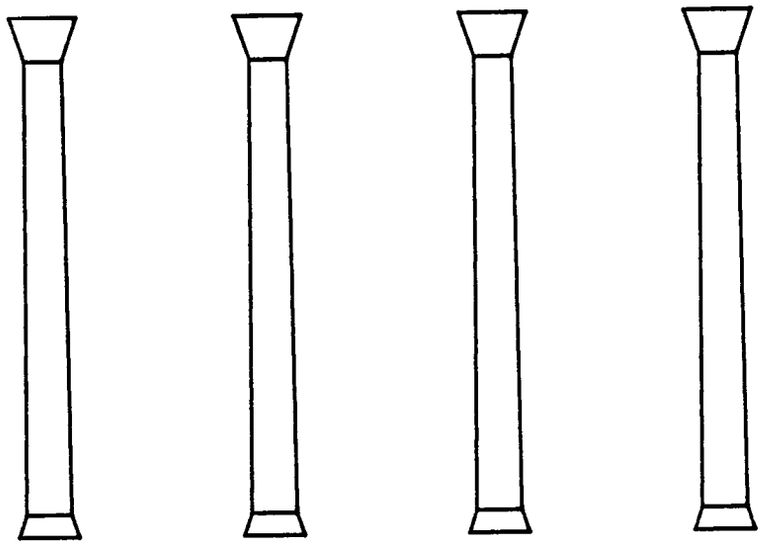
Colonnade formée à partir de colonnes du premier genre :



Colonnade formée à partir de colonnes du troisième genre :



Colonnade formée à partir de colonnes du cinquième genre :



A la vue de ces exemples, une remarque s'impose : notre colonne (quelle qu'elle soit) ne peut simplement procéder d'un amas différentié de pixels sur l'écran (ou de points sur un support quelconque) mais doit, de plus, posséder des propriétés précises permettant de régler ses rapports avec l'ensemble d'une composition. Ceci nous amène naturellement à exploiter les qualités de base de données de Tangram.

Deuxième étape : constitution d'une banque de données relatives aux éléments de notre bibliothèque

Quels types d'information devons nous retenir ?

- En premier lieu, nous formons un schéma relationnel permettant de mettre en correspondance le genre d'une colonne et le nom symbolique du fichier image correspondant. Cette "mini-interface" avec notre bibliothèque nous permettra dorénavant de référencer l'élément colonne simplement par son genre. Nous créons pour cela un atome (sous l'interpréteur Lisp de Tangram) dont la valeur d'une propriété est une liste d'association "genre-nom symbolique" (une structure de donnée plus élaborée serait bien entendu préférable dans le cadre d'une manipulation complexe portant sur un grand nombre d'éléments ; nous en réservons la description pour un prochain chapitre).

Soit donc notre atome "colonnes" :

```
(PUTP 'COLONNES 'DISPONIBLES '(     (1 COL1.PAD)
                                    (2 COL2.PAD)
                                    (3 COL3.PAD)
                                    (4 COL4.PAD)
                                    (5 COL5.PAD)
                               ))
```

- Ensuite nous accrochons une deuxième propriété à notre atome "colonnes" : la valeur de l'entr-axe suivant le genre (les nombres indiqués ont été établis en fonction de l'échelle de dessin de nos colonnes).

```
(PUTP 'COLONNES 'ENTRAX '(      (1 168)
                                (2 156)
                                (3 144)
                                (4 132)
                                (5 120)
                           ))
```

- Enfin nous nous écartons quelque peu des préceptes théoriques de Durand en définissant des seuils de variation pour les entr-axes de chaque genre de colonne (chaque entr-axe peut ainsi prendre un ensemble de valeurs limitées par les bornes des entr-axes des genres immédiatement inférieurs et supérieurs ; les limites de l'entr-axe supérieur du premier genre et de l'entr-axe inférieur du cinquième genre sont pris arbitrairement).

```
(PUTP 'COLONNES 'ENTRAX-DEFORME '(  (1 (162 1000))
                                    (2 (150 162))
                                    (3 (138 150))
                                    (4 (126 138))
                                    (5 (0   126))
                                 ))
```

Nous pouvons ainsi associer la variation des entr-axes des colonnes à un processus continu (par interpolation sur les valeurs du processus discret défini par Durand). Dans la suite de l'étude nous nommerons cette variation des entr-axes : "déformation de la colonnade".

Il nous faut maintenant décrire les actions permettant de manipuler nos éléments graphiques et nos propriétés. Pour cela nous utilisons Tangram comme langage de programmation.

Troisième étape : programmation des utilitaires de manipulation

Tout d'abord nous définissons une procédure permettant de parcourir notre "base de données" et d'associer à une clé (qui sera pour nous le genre) l'information recherchée :

```
(DE ASSOC (CLE BASE)
(COND ((NULL BASE) NIL)
      ((EQ CLE (CAAR BASE))(CADAR BASE))
      (T (ASSOC CLE (CDR BASE)))
      ))
```

D'autres définitions de "Assoc" seront données plus en avant dans cette étude, mais celle-ci nous convient parfaitement pour l'instant.

Ensuite nous implémentons un ensemble de primitives de manipulation de nos éléments :

- Une procédure retournant (par une lecture sur fichier à partir du "catalogue" de notre bibliothèque) l'occurence graphique d'une colonne en fonction du genre (le deuxième paramètre est local à notre procédure et n'a aucune utilité lors de l'appel).

```
(DE RETROUVE (GENRE BIDON)
(COND
((SETQ BIDON(RETR(ASSOC GENRE(GETP 'COLONNES 'DISPONIBLE))))
                BIDON)
(T (PRINT '" Genre non répertorié"))
))
```

- Une procédure permettant d'afficher une série de colonnes espacées par des entr-axes types.

```
(DE REPETE (NOM-COL GENRE NBRE)
(COND ((ZEROP NBRE) NIL)
      (T (APPEND
           (LIST(COP(EVAL NOM-COL)(MULT
                (PADL(ASSOC GENRE (GETP 'COLONNES 'ENTRAX)))
              NBRE) 0))
           (REPETE NOM-COL GENRE (SUB1 NBRE))
      ))))
```

(La primitive "Padl" - incorporée aux primitives de base de Tangram - convertit "une distance en pixels" sur l'écran en une valeur correspondante pour l'editeur graphique - en fonction du zoom sur l'image -. Quant à "Cop", elle effectue une copie de l'élément.)

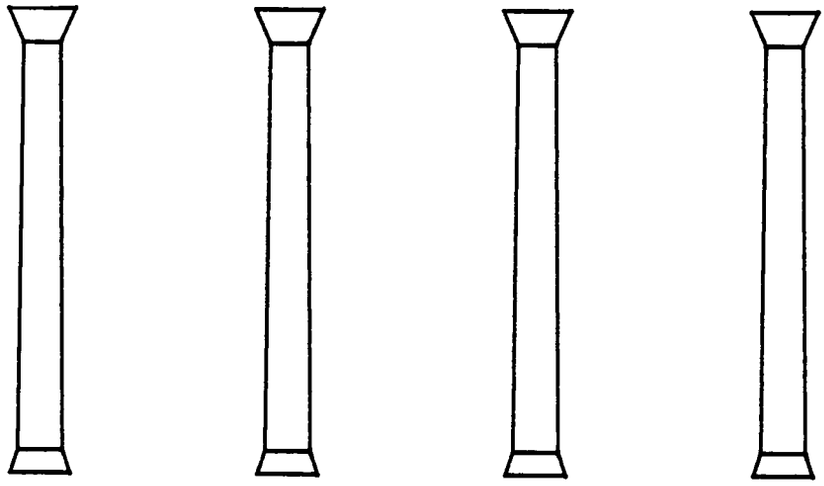
- Une procédure de composition de colonnade (le genre et le nombre de colonnes sont insérés dans la liste de propriétés de l'atome (nom de la colonnade)).

```
(DE COLONNADE ()
(PROG (LOCAL NOM NBRE GENRE)
      (TERPRI)
      (PRIN1 '" Nom de la colonnade : ")(SETQ NOM (READ))(TERPRI)
      (PRIN1 '" Genre : ")(SETQ GENRE (READ))(TERPRI)
      (PRIN1 '" Nombre de colonnes : ")(SETQ N (READ))
      (PUTP NOM 'NOMBRE NBRE)
      (PUTP NOM 'NGENRE GENRE)
      (RETURN
      (SET NOM (REPETE(RETROUVE GENRE NIL) GENRE NBRE)))
))
```

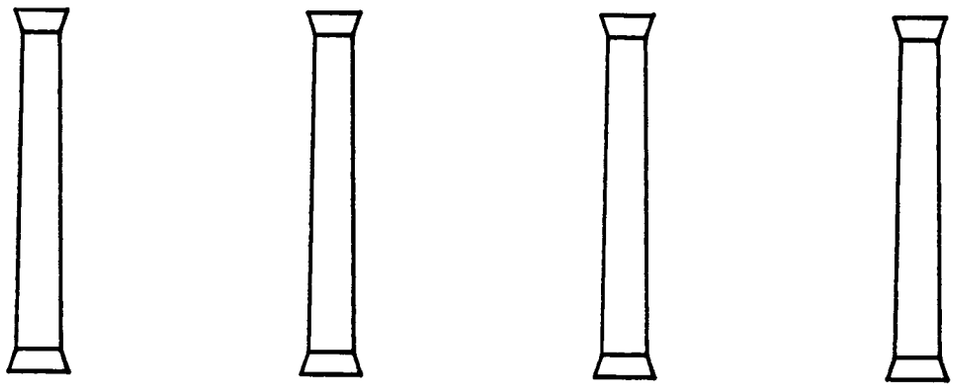
- Une manipulation simple -

Nous proposons au lecteur l'illustration de deux sessions d'activation de "Colonnade" :

```
(COLONNADE)
Nom de la colonnade : COLS1
Genre : 4
Nombre de colonnes : 4
```



```
(COLONNADE)
Nom de la colonnade : COLS2
Genre : 2
Nombre de colonnes : 4
```



Les deux atomes Cols1 et Cols2 ont maintenant pour valeur une liste composée de quatre dessins de colonnes (une demande d'évaluation d'un de ces atomes affichera la colonnade correspondante).

- Deux prédicats permettant de savoir si les colonnes formant une colonnade sont espacées d'un entr-axe type ou d'un entr-axe déformé (compris entre les limites définies dans la seconde étape) :

```
(DE NORME (NOM)
(EQ   (PADL(ASSOC(GETP NOM 'NGENRE)(GETP 'COLONNES 'ENTRAX)))
      (SUB (CAAR(MIMA(CAR(EVAL NOM))))
           (CAAR(MIMA(CADR(EVAL NOM))))))
))
```

(La primitive "Mima" - incorporée aux primitives de base de Tangram - renvoie une liste composée des coordonnées minimales et maximales - encombrement horizontal et vertical - d'un objet graphique).

```
(DE NORMED (NOM)
(PROG (LOCAL S$ G$)
      (SETQ S$ (SUB  (CAAR(MIMA(CAR(EVAL NOM))))
                     (CAAR(MIMA(CADR(EVAL NOM))))))
      (SETQ G$ (ASSOC (GETP NOM 'NGENRE)
                      (GETP 'COLONNES 'ENTRAX-DEFORME)))
      (RETURN (AND   (LESSP (PADL(CAR G$)) S$)
                     (GREATP (PADL(CADR G$)) S$)))
))
```

- Une primitive retournant le genre d'une colonnade.

```
(DE GENRE? ()
(PROG (TERPRI)
      (PRIN1 '" Nom de la colonnade : ")
      (RETURN (GETP (READ) 'NGENRE))
))
```

- Une procédure permettant de déplacer une colonne.

```
(DE BOUG1 (OBJET VALEUR)
(COND ((ZEROP VALEUR) NIL)
      (T (TRAN OBJET (MULT -01 (PADL VALEUR)) 0))
))
```

(La primitive "Tran" - incorporée aux primitives de base de Tangram - active une translation en X et Y - deuxième et troisième paramètres - sur un objet graphique - premier paramètre -).

- Deux procédures pour déplacer toutes les colonnes d'une colonnade.

```
(DE BOUG2 (NOM NBRE M VALEUR)
(COND ((OR (NULL NOM)(ZEROP NBRE)) NIL)
      (T   (BOUG1 (CAR NOM)(MULT VALEUR (SUB M NBRE)))
           (BOUG2 (CDR NOM)(SUB1 NBRE) M VALEUR))
))


(DE BOUG3 (NOM VALEUR)
(PROG (LOCAL X)
      (SETQ X (GETP NOM 'NOMBRE))
      (RETURN (BOUG2(EVAL NOM) X X VALEUR))
))
```

Cette dernière procédure provoque la déformation d'une colonnade sans vérification de cohérence entre le genre de la colonnade et la valeur de l'entr-axe des colonnes. Il nous faut donc implémenter une suite d'actions complémentaires activant "Boug3", et comparant ensuite l'entr-axe des colonnes avec les valeurs permises (par activation de "Norme" et "Normed"). Si la valeur de l'entr-axe se révèle être hors des limites permises, le genre de la colonnade sera modifié en conséquence (ainsi que, bien entendu, le dessin des colonnes).

Nous diviserons cette tâche en deux procédures :

- Une première procédure permettant la lecture des paramètres, l'activation de "Boug3", le test sur les entr-axes et l'activation, le cas échéant, d'une seconde procédure (décrite plus loin).

```
(DE DEFORME ()
(PROG (LOCAL NOM VALEUR)
      (TERPRI)
      (PRIN1 '" Nom de la colonnade : ")(SETQ NOM (READ))
      (TERPRI)
      (PRIN1 '" Valeur de la déformation : ")
      (SETQ VALEUR (READ))
      (BOUG3 NOM VALEUR)
      (RETURN   (COND     ((NORME NOM) '" Entr-axe type")
                          ((NORMED NOM) '" Entr-axe déformé)
                (T   (COND ((LESSP VALEUR 0)
                                (CHG(PLUS1(GETP NOM 'NGENRE))))
                           (T  (CHG(SUB1(GETP NOM 'NGENRE)))))))))
))
```

- Une procédure permettant d'affecter à une colonnade des colonnes d'un genre correspondant à la valeur de l'entr-axe (la liste des objets graphiques représentant la colonnade est modifiée par activation de "Retrouve").

```
(DE CHG (GENRE)
(PROG (EFFACE)
      (PRINT '" Déformation supérieure au seuil")
      (PRIN1  '" Le genre de la colonnade est maintenant  : ")
      (PRINT GENRE)
      (SET NOM (REPETE(RETROUVE GENRE) GENRE (GETP NOM 'NBRE)))
      (PUTP NOM 'NGENRE GENRE)
      (RETURN '" Changement de genre")
))
```

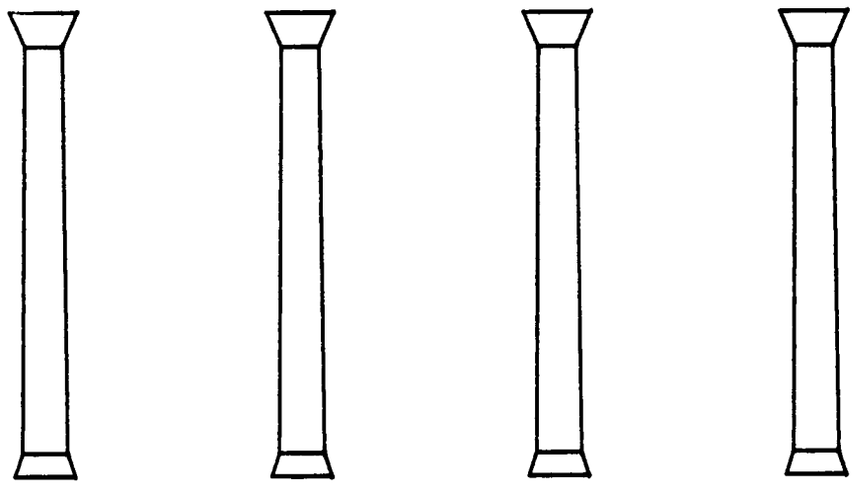
(La primitive "Efface" appartient à Tangram et efface l'écran graphique et texte).

Nous proposons au lecteur l'illustration de quelques exemples d'activation de "Déforme" sur les pages suivantes.

Il faudra apporter une grande attention aux modifications de Types de colonnes, leurs différences ne sautant pas aux yeux lors d'une lecture rapide ; mais nous avons préferé respecter les Types de Durand plutôt que de manipuler des éléments plus différenciés mais ne correspondant pas à un exemple précis tiré d'une théorie architecturale.

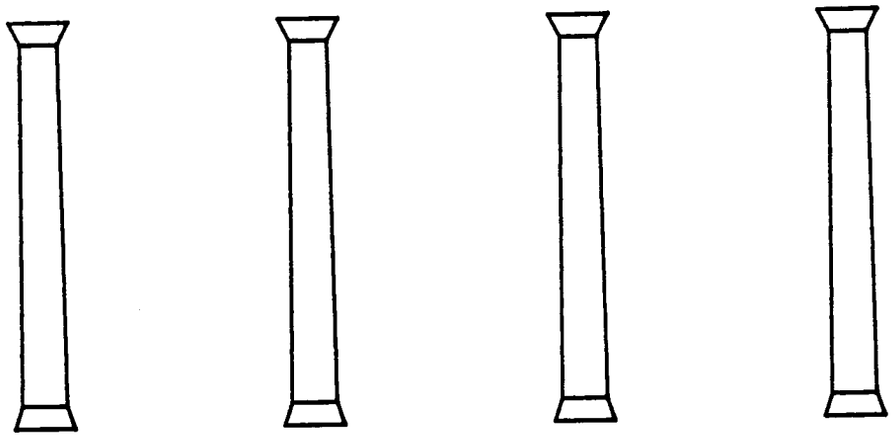
/Activation de "Déforme" sur la colonnade "Cols1" (de genre 4) définie plus haut./

(DEFORME)
Nom de la colonnade : COLS1
Valeur de la déformation : 5
EVAL : Entr-axe déformé



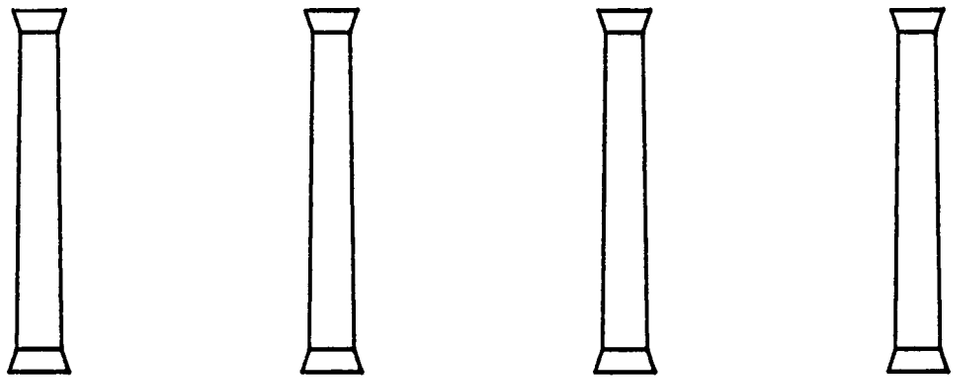
(DEFORME)
Nom de la colonnade : COLS1
Valeur de la déformation : 4

Déformation supérieure au seuil
Le genre de la colonnade est maintenant : 3
EVAL : Changement de genre



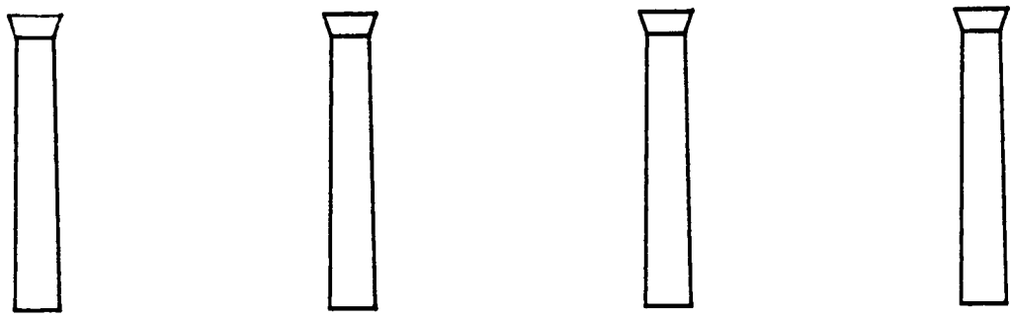
```
(DEFORME)
Nom de la colonnade : COLS1
Valeur de la déformation : 9

Déformation supérieure au seuil
Le genre de la colonnade est maintenant : 2
EVAL : Changement de genre
```



```
(DEFORME)
Nom de la colonnade : COLS1
Valeur de la déformation : 11

Déformation supérieure au seuil
Le genre de la colonnade est maintenant : 1
EVAL : Changement de genre
```



Quatrième étape : conclusion de la première partie

Cette étude préliminaire nous a permis d'explorer rapidement les différentes possibilités d'exploitation de Tangram. La manipulation proposée était simple, ne nécessitait pas d'analyse ou de réflexion préalable et peut être aisément implémentée. Elle nous a permis de définir un premier "passage entre l'acte de Conception et l'Informatique". Toutefois son objet reste très limité. De plus la nécessité de définir dans son intégralité chaque action implémentée (en quelque sorte au "coup par coup") nous impose deux remarques :

- Le langage utilisé est sans aucun doute de "trop bas niveau" pour pouvoir être réellement exploité par un utilisateur non informaticien.

- Il semblerait intéressant de disposer d'un univers de travail plus structuré - nous serions tentés de dire plus "charpenté" - permettant une définition simple des actions et des objets manipulés.

Il nous faut donc étudier des structures de données et des modes d'implementation de procédures plus élaborés susceptibles de répondre d'une manière générale aux problèmes rencontrés dans le cadre de notre recherche. Ceci fera l'objet des chapitres suivants.

INTERET DE L'UTILISATION DES LANGAGES OBJETS

La notion d'Objet (on parlera de Langages Objets, ou orientés objets) présente un avantage considérable sur les systèmes de programmation "classiques", dans un domaine de connaissances aussi complexe que celui de l'Architecture : la réduction de la complexité.
Une des grandes difficultés à élaborer un logiciel en phase de conception architecturale vient du fait que les modes de programmation classiques apportent des solutions à des problèmes non seulement "bien définis", mais de plus dont les réponses sont formellement connues d'avance. Il s'agit alors de mettre en place des mécanismes de résolution spécifiques et systématiques. (Algorithmes).
Il serait illusoire et dangereux (car la contrainte devient insupportable pour l'architecte devant se plier à la machine, et non l'inverse), de tentér ce type d'approche mutilante et réductionniste, alors que rien n'est plus riche, mouvant et complexe que la conception architecturale. (<GOULETTE 84-2>).

La philosophie des langages objets revient à poser le problème à l'envers : Pourquoi ne pas permettre à l'utilisateur de créer au fur et à mesure les objets dont il a besoin (constituant par là-même sa propre base de données), et définissant les actions que les objets peuvent entreprendre, leur typologie (aspects sémantiques), et la description de leurs inter-relations (aspects syntaxiques).

Ainsi, aucun algorithme ne pourra être établi "a priori", en dehors de toute considération sur le type de démarche et d'évolution que l'utilisateur créera, aucun "programme" ne venant figer une évolution imprévisible.
Le rôle du programmeur revient donc à créer une espèce de "boîte à outils", permettant de générer un micro-univers dans lequel une cohérence relationnelle entretiendra le fonctionnement.

On trouvera d'autres arguments dans <CARADANT 84>.

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. La plupart des langages objets admettent une définition métacirculaire, c'est à dire qu'à l'instar de LISP le langage peut s'auto-décrire dans son propre vocabulaire et sa propre sémantique.

- les microlangages que nous proposons ne vont pas aussi loin dans la finesse théorique : ils n'assurent pas eux-mêmes leur complétude (ils seront décrits en LISP). Nous verrons néanmoins que nous n'avons plus affaire uniquement à une structure de représentation mais aussi à de véritables langages.

Les différents types de Langages Objets

On distingue plusieurs types de langages orientés objet :

- Les langages directement issus de SMALLTALK qui sont aussi actuellement à l'étude au sein de LI2A.

- 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 de type "frame" : KRL, FRL, MERING, LOOP, qui s'intéressent plus précisement à la représentation des connaissances.

Nous exposerons dans ce rapport deux types de travaux que nous avons réalisés sur les langages objets de type "frame" à attachements procéduraux.

PREMIER EXEMPLE : UNE ORIENTATION "METHODES"

I). Aperçu général

Trois points de vue ont présidé à la mise en place du logiciel que nous allons décrire :

A)

L'idée que l'Architecture peut, pour une large part, être analysée, décrite et comprise comme un processus combinatoire à la fois synthétique et additif, d'un ensemble d'éléments finis ; combinatoire régie par des groupes de règles transformationnelles et relationnelles associées au type de chaque objet. Nous nommerons ces règles "Méthodes". Si dans le détail le nombre des objets architecturaux peut sembler très grand, il devient à l'inverse assez faible dès que l'on aborde une classification par l'analyse typologique. S'appuyant sur ce point de vue, la recherche qui suit développera quatre concepts principaux:

1: Le type Architectural :

Définit les Méthodes et Contraintes applicables aux objets qui en sont issus. (Attention : le type ne contient pas d'occurence graphique.)

2: Les Méthodes :

ou Actions que les objets peuvent entreprendre.

3: Les Contraintes

que l'on doit respecter sur tout objet. (ex : Proportions géométriques, Modules, Résistance, Ordonnancement etc...). Nous nommerons ici cet ensemble "Modèle".

4: Les Objets :

Occurences des Types.

B)

L'outil un peu particulier dont nous disposons au LI2A, type de LISP graphique connecté à un éditeur deux dimensions, permet l'implantation de ce genre de logiciel (conf : "Une manipulation simple").

C)

La confrontation avec des logiciel connus pour être soit-disant des logiciels de conception, amène bien des questions.

II). Analyse du Programme.

II-1. Les Classes

On désignera par Classes, l'ensemble des éléments du micro-univers ayant pour caractéristiques communes la même structuration des données.
Nous différencierons ici deux classes principales :
- La classe des Objets.
- La classe des Types.

II-1.1. Classe des Objets

Un Objet est une micro-base de données (structurée selon le mode classique des listes de propriétes manipulées par LISP (voir <CARADANT 84>), dont chaque information est conditionnée par un ensemble de contraintes imposées par le Type auquel l'Objet appartient. Grâce à cet ensemble de contraintes qu'il "porte", l'Objet est autonome, il est à la fois 'Procédure' (au sens conventionnel) et base de données, il peut agir sur l'environnement dans lequel il est placé, ou s'auto-modifier.

Tout Objet est donc constitué d'un ensemble de CHAMPS du type:

<CHAMP>::=<Nom de Facette><Valeur>

La Facette est l'identificateur qualitatif d'un Caractère de l'objet. C'est toujours une expression atomique.

La Valeur est le contenu de la facette de l'objet considéré. C'est un atome ou une liste.

Certaines facettes sont automatiquement attribuées aux objets du logiciel, d'autres sont définies par l'opérateur lors de la déclaration du type auquel l'objet se rapporte. (On parlera alors de Champs Modélisés).

On peut donc définir l'objet par:

<Objet>::=<Nom d'objet>
<Champ procedural>
<Champ Type>
<Champs Modélisés>
<Champs utilisés par certaines procédures, non accessibles par l'opérateur ou Objets autres>

II-1.2. Classe des Types

Le Type détermine le rayon d'action et l'existence des Objets qu'il engendre. Il est lui-même issu d'un Type dont il hérite des propriétés. Il est évident que les Types initiaux ont pour propriété d'avoir pour Type eux-mêmes : ils composent la classe particulière des Méta-Types.

Les Facettes du Type sont connues d'avance et déterminées par le logiciel, leurs valeurs sont introduites par l'opérateur lors de la déclaration.

On compte trois Facettes pour chaque Type :

La Facette TYPE :

Elle a pour valeur le NOM du PERE du Type.

La Facette METHODES :

C'est la liste des actions que tous les Objets issus de ce Type pourront entreprendre. Attention : Il y a héritage de Méthodes d'un Type "père" à un Type "fils".

La Facette MODELES :

C'est l'ensemble des contraintes sur les Valeurs des Aspects que les Objets pourront posséder.

II-2. L'interprète des modèles

Tout Objet est composé de champs de propriétés appelés Aspects comme nous l'avons défini plus haut. Les Aspects déterminent donc des Valeurs accrochées aux Objets qui ne sont validées que si les contraintes de Facettes le sont. Ces contraintes s'expriment sous forme de Clauses de Horn. La définition BNF d'un modèle est donc :

<Modèle>::= {<Facette>{ <Clause>}}

<Clause>::= <Identificateur de Type de clause>
[<Expression symbolique LISP>]

<Identificateur de Type de Clause>
::= CONST | VALEUR | PROCEDURE | RESTRICTION | ENSEMBLE |
DEMON | DEFAUT

Où :

VALEUR :

Stipule que la valeur de la facette sera mise à jour de manière externe à l'objet.

CONST :

Permet d'affecter une valeur constante à la facette.

PROCEDURE :

Détermine le fait que la valeur sera déduite ou re-calculée par une procédure.

RESTRICTION :

Est un prédicat contrôlant la valeur entrée.

ENSEMBLE :

Prédicat spécifiant les valeurs autorisées sur la facette.

DEMON :

n'est pas à proprement parler une contrainte, c'est une procédure qui se déclenche à chaque fois que la valeur de l'aspect est touchée.

DEFAUT :

Permet d'initialiser la valeur d'une facette.

Il faut donc pour qu'une valeur sur une facette soit validée que l'ensemble des clauses s'y rapportant soit vrai.
D'où si α, β, γ désignent les paramètres d'une clause, ξ l'évaluation d'une clause et:

$$\forall \text{ Clause}, \xi(\text{Clause}) \in \{\text{Vrai}, \text{Faux}\}$$

Alors on doit avoir :

$$\text{Validation-Facette}(\alpha,\beta,\gamma\ldots) \Leftrightarrow \bigwedge_{i=1}^{n_{clause}} \xi[\text{Clause}(\alpha,\beta,\gamma\ldots)]_i = \text{Vrai}$$

Parmi les fonctions importantes figurent celles qui permettent le chargement des aspects d'un objet.
Les Objets peuvent recevoir des informations ou en fournir de plusieurs manières. Les valeurs des aspects peuvent être données par l'opérateur, par un autre objet, ou peuvent être instanciées.
Dans ce cas on considère qu'elles sont "déduites" à la suite de certaines informations sur l'environnement stimulant des démons ou des procédures.

Voici une première procédure dite de "validation".

```
(DE VALIDE(O ASS VA CL)
(COND((NULL CL) T)
      ((AND(EQ (CAAR CL) 'DEFAUT)(EQ VA NIL)
           (VALIDE O ASS VA (CDR CL)))
           (PUTP O ASS (CADAR CL))T)
      ((AND(EQ(CAAR CL) 'VALEUR)
           (VALIDE O ASS VA (CDR CL)))
           (PUTP O ASS VA)T)
      ((AND(EQ(CAAR CL) 'PROCEDURE)
           (VALIDE O ASS VA (CDR CL)))
         (PUTP O ASS (APPLY (CADAR CL)(LIST O VA)))T)
      ((AND(EQ(CAAR CL) 'ENSEMBLE)
           (VALIDE O ASS VA (CDR CL)))
         (MEMBERP VA (CADAR CL)))
      ((AND(EQ(CAAR CL) 'RESTRICT)
           (VALIDE O ASS VA (CDR CL)))
         (APPLY(CADAR CL)(LIST O VA)))
      ((AND(EQ(CAAR CL) 'DEMON)
           (VALIDE O ASS VA (CDR CL)))
         (APPLY(CADAR CL)(LIST O VA))T)
      (T NIL))))))))
```

Les objets communiquent avec cette procédure au moyen de deux méthodes :

Une fonction pour laquelle un objet voit une de ses facettes instanciée par l'opérateur :

```
(DE RECOIT(X)
(COND((VALIDE (CAR X)(CADR X)(CADDR X)
       (ASSOC (CADR X)
           (GETP (GETP (CAR X) 'PERE)'MOD) )  )
        T)
     (T(PRIN1 (LIST (CAR X)
                    '" NE PEUT RECEVOIR "
                    (CADR X)
                    (CADDR X)))
 ))))))
```

Ou une fonction pour laquelle c'est un objet qui instancie un autre objet :

```
(DE DONNE-A(X)
(COND((VALIDE (CADR X)(CADDR X)(GETP (CAR X) (CADDR X))
           (ASSOC(CADDR    X)
                 (GETP(GETP(CADR   X)   'PERE) 'MOD))))
      (T(PRIN1(LIST (CADR X)
                    '" NE PEUT RECEVOIR "
                    (CADDR X))))))
```

Suivent quatre procédures d'exploration 'classiques' utilisant les héritages :

```
1:
(DE QUI-EST (V)
(QUI V PILEOBJET))

(DE QUI (V L)
(COND((NULL L) NIL)
     ((MEMBERP1 V (PROPLIST(CAR L)))
           (CONS(CAR L)(QUI V (CDR L))))
     (T(QUI V (CDR L)))))


2:
(DE QUI-A (F)
(QUA F PILETYPE))

(DE QUA (F L)
(COND((NULL L) NIL)
     ((MEMBERP1 F (GETP(CAR L) 'MODELE))
           (CONS(GETP(CAR L) 'FILS)(QUA F (CDR L))))
     (T(QUA F (CDR L)))))

3:
(DE EST? (O V)
(COND((NULL O)NIL)
     ((MEMBERP1 V (PROPLIST O))
      T)
     (T(EST? (GETP O 'EST-UN) V))))

4:
(DE QUOI? (O A)
(COND((NULL O) NIL)
     (T(COND((GETP O A)
                (CONS(GETP O A)(QUOI? (GETP O 'EST-UN)
                                      A)))
           (T(QUOI? (GETP O 'EST-UN) A))))))
```

```
(DE MEMBERP1(X L)
(COND((NULL L) NIL)
     ((ATOM L)(EQ X L))
     (T(OR(MEMBERP1 X (CAR L))(MEMBERP1 X (CDR L))))))
```

II-3. L'interprète des méthodes

Son fonctionnement est relativement simple :

Considérons que tout objet est une procédure LISP du type "macro-expression" (procédure ayant la caractéristique de se prendre elle-même pour argument et de réévaluer son résultat);
Chaque Objet cherche alors dans la liste des méthodes de son type si l'action à entreprendre est permise.
Si c'est le cas, il applique la méthode sur lui-même avec la liste des arguments qui lui font suite, en tenant compte du fait que l'argument ":" signifie "évaluer ce qui suit".

Premier exemple :

(Colonne1 reçoit quantité 1)

1: Colonne1 cherche si son type (par exemple : Colonne), connaît la méthode "reçoit".
2: Si oui, l'appel à la méthode "reçoit" est effectué avec pour paramètres : Colonne1 quantité et 1.
3: Si non, on obtient un message du type : Colonne1 ne connaît pas la méthode "reçoit".

Deuxième exemple :

(Colonne1 reçoit quantité : (plus 10 1))

Le processus est identique mais les arguments sont :
Colonne1 quantité et 11, en raison du caractère ":"

Voici le code de la boucle générale et de l'interprète :

```
(DE L1()
(PROG
A$$$
(EVAL(READ))
(GO A$$$))))))
```

La commande de base associée à tout objet :

```
(SETQ METACOM '(MLAMBDA A
(COND((MEMBERP (CADR A)(GETP(GETP(CAR A) 'PERE) 'MET))
            (APPLY(GETP(CADR       A)      'EXPR)
                  (CONS(CAR A)(SUPE(CDDR A))) ) T)
      (T(PRIN1  (LIST (CAR A) '" NE SAIT PAS  FAIRE  "
                      (CADR A)))))

(DE SUPE(X)
(COND((NULL X) NIL)
          ((EQ(CAR   X)  ':)(CONS(EVAL(CADR  X))
                                 (SUPE(CDDR X))))
      (T(CONS(CAR X)(SUPE(CDR X)))))))
```

II-4. Définition des types initiaux

Pour que l'interprétation puisse commencer il est nécessaire de disposer de types et objets initiaux.

En premier lieu l'opérateur lui-même :

```
(PUTPLIST 'JE
          '((MET(CHARGE
                 VALIDE
                 SAUVE
                 DETRUIT
                 VIDE
                 ?
                 DEF-OBJET
                 DEF-TYPE ))

            (PERE JE))
))))

(PUTP 'JE 'EXPR  METACOM)
```

Le Méta-type ou père de tous les types sur toutes les méthodes :

```
(PUTP 'META 'MET '(AFFICHE
                   RECOIT-DESSUS
                   RECOIT-DEDANS
                   PREND-ECHELLE
                   CREE
                   PREND-BASE
                   SE-REPETE-AVEC
                   BOUGE-DE
                   MONTE-DE
                   TES-METHODES
                   RECOIT
                   DONNE-A
                   AJOUTE-A)
)))
```

Puis vient l'écran en tant que tampon graphique :

```
(PUTPLIST 'ECR '((MET(AFFICHE
                      DONNE-A
                      RECOIT))

                 (MOD(
                       (IMAGE(VALEUR))))
))))

(PUTP 'ECRAN 'PERE 'ECR)

(PUTP 'ECRAN 'EXPR METACOM)
```

Les fichiers aussi sont des objets définis :

```
(PUTPLIST 'FI '((MET(RECOIT
                     DONNE-A))

                (MOD((NOM
                      (VALEUR))
                     (IMAGE
                      (PROCEDURE
                       (LAMBDA(X)(RETR(GETP X 'NOM))))))
)))))))))

(PUTP 'FICHIER 'PERE 'FI)

(PUTP 'FICHIER 'EXPR METACOM)
```

Initialisation de la base de données :

```
(SETQ BASEDON '())
(SETQ PILETYPE '())
(SETQ PILEOBJET '())
```

II-5. Définition de quelques méthodes

Premièrement les méthodes de création d'objets et de types :

Module de création d'Objet

```
(DE DEF-OBJET (X)
(PROG (LOCAL W Z)
(TERPRI)
(PRIN1 '" Nom        : ")(SETQ W (READ))(TERPRI)
(SETQ BASEDON (CONS W BASEDON))
(SETQ PILEOBJET (CONS W PILEOBJET))
(PRIN1 '" Type       : ")(SETQ Z (READ))(TERPRI)
(PUTP W 'PERE Z)(PUTP W 'EXPR METACOM)
(PUTP Z 'FILS (CONS W (GETP Z 'FILS)))
(RETURN(PRIN1 '" Objet Défini ")
)))))
```

Module de création de Type

```
(DE DEF-TYPE (X)
(PROG (LOCAL W Y)
(TERPRI)(PRIN1 '" Nom        : ")(SETQ W (READ))
(SETQ BASEDON (CONS W BASEDON))
(TERPRI)
(PRIN1 '" SurType    : ")(SETQ Y (READ))(TERPRI)
(PUTP W 'PERE Y)
(PRIN1  '"  Méthodes    :  ")(PUTP W 'MET
                              (APPEND  (GETP  Y 'MET)(READ)))
(TERPRI)
(PRIN1 '" Modèle     : ")(PUTP W 'MOD (READ))(TERPRI)
(SETQ PILETYPE (CONS W PILETYPE))
(RETURN(PRIN1 '" Type Défini ")
)))))))
```

La définition de méthodes s'applique au travers des utilitaires de base EDIT, PP, TRACE, STEPPER implémentés dans le langage LISP ou faisant partie de notre bibliothèque de base LISP.

Méthodes implémentées à titre d'exemples :

Cette méthode permet d'ajouter la valeur de la facette d'un objet à un autre objet sans interpréter le modèle :

```
(DE AJOUTE-A(X)
 (PUTP (CADR X) (CADDR X)
       (CONS(GETP(CAR X)(CADDR X))
            (GETP(CADR X)(CADDR X)))))
```

Un objet affiche ses méthodes :

```
(DE ? (O)
(PRIN1 (GETP(GETP (CAR O) 'PERE) 'MET)))
```

Pour éliminer un objet de la base de données :

```
(DE DETRUIT(X)
(PROG
(SETQ PILEOBJET (DELETE(CADR X) PILEOBJET))
(PUTP (GETP(CADR X) 'PERE) 'FILS
     (DELETE(CADR X)(GETP(GETP(CADR X) 'PERE) 'FILS)))
(PUTPLIST (CADR X) '())
(SETQ BASEDON
      (DELETE(CADR X)BASEDON)))
(RETURN(SET(CADR X) NIL)))))
```

Sauver la totalité de l'univers si l'opérateur le désire :

```
(DE SAUVE(X)
(PROG
(PRIN1 '" Nom du fichier de la base :  ")
(PUTN (READ))(REWRITEF)
(PUTF (LIST 'SETQ 'BASEDON (LIST 'QUOTE BASEDON)))
(SAVP BASEDON)
(SAVI BASEDON)
(CLOSEF)
(RETURN (PRIN1 '" Terminé "))))))))
```

```
(DE SAVP(X)
(COND((NULL X) NIL)
      (T(PUTF(LIST 'PUTPLIST(LIST 'QUOTE (CAR X))
                           (LIST 'QUOTE(PROPLIST(CAR X)))))
        (SAVP(CDR X)))))))))

(DE SAVI(X)
(COND((NULL X) NIL)
      (T(COND((MEMBERP 'IMAGE(PROPLIST(CAR X)))
                    (STOCK (GETP(CAR X) 'IMAGE)(CAR X))
                    (PUTF(LIST 'PUTP (LIST 'QUOTE (CAR X))
                                     (LIST 'QUOTE 'IMAGE)
                                     (LIST 'RETR (CAR X))))
                    (SAVI(CDR X)))
                (T(SAVI(CDR X))))))))))))))
```

Charger au contraire un univers...

```
(DE CHARGE(X)
(PRIN1(LOAD(CADR X)))))
```

L'objet affiche l'une de ses caractéristiques :

```
(DE AFFICHE(X)
(PRIN1 (GETP (CAR X) (CADR X))))))
```

Une petite méthode graphique permettant d'emboîter deux objets l'un dans l'autre (graphiquement il s'entend) :

```
(DE RECOIT-DEDANS(X)
(TRAN (GETP (CADR X) 'IMAGE)
    (SUB(CAAR(MIMA(GETP(CAR X) 'IMAGE)))
        (CAAR(MIMA(GETP(CADR X) 'IMAGE))))
    (SUB(CADAR(MIMA(GETP(CAR X) 'IMAGE)))
        (CADAR(MIMA(GETP(CADR X) 'IMAGE)))))))))))
```

Une méthode pour positionner deux objets l'un sur l'autre :

```
(DE RECOIT-DESSUS(X)
(COND((EQ(GETP(CAR X) 'ORDRE)(GETP(CADR X) 'ORDRE))
        (SUG2 (GETP(CAR X) 'IMAGE)(GETP(CADR X) 'IMAGE))))))))
      (T(PRIN1  '"  IMPOSSIBLE CES 2 OBJETS NE SONT PAS  DU
                    MEME ORDRE "))))))

(DE SUG2(X Y)
(TRAN Y (SUB(CAAR(MIMA X))(CAAR(MIMA Y)))
        (SUB(SUB(CADAR(MIMA X))(CADAR(MIMA Y)))
        (SUB(CADDR(MIMA Y))(CADAR(MIMA Y))))))))))
```

Pour modifier l'échelle de l'occurence graphique d'un objet

```
(DE PREND-ECHELLE(X)
(ECH(GETP(CAR X) 'IMAGE)(CADR X) 'G)))))))))
```

("Ech" est une fonction de Tangram qui change l'échelle globalement si son troisième paramètre est 'G.)

Un objet peut en créer un autre par "copie" de lui-même :

```
(DE CREE(X)
(PROG
(SET (CADR X) METACOM)
(PUTP (GETP (CAR X) 'PERE) 'FILS
      (CONS (CADR X) (GETP(CAR X) 'PERE)))
(SETQ PILEOBJET (CONS(CADR X) PILEOBJET))
(PUTPLIST (CADR X) (PROPLIST(CAR X)))
(SETQ BASEDON (CONS(CADR X)BASEDON))
(RETURN(PUTP(CADR  X)  'IMAGE
            (COP(GETP(CAR X) 'IMAGE)  0  0)))))
)))
```

Méthode permettant à un objet de répéter son occurence graphique avec pour paramètres le nombre de copies :

```
(DE SE-REPETE-AVEC(X)
(PUTP (CAR X) 'IMAGE
(JONCTION
(APPEND(REP1(GETP(CAR   X)'IMAGE)(CADDR   X)(GETP(CADR   X)
                                                'ENTRAX))
(LIST(GETP(CAR X) 'IMAGE)))))))))))

(DE JONCTION(X)
(COND((NULL (CDR X))(CAR X))
     (T(JOIN (CAR X)(JONCTION(CDR X)))))))

("Join" est une fonction de Tangram qui joint deux éléments en un seul.)

(DE REP1(X Y Z)
(COND((ZEROP Y) NIL)
     (T(CONS(COP X (PADL Z) 0)
        (REP1(COP X (PADL Z) 0)(SUB1 Y) Z))
       )))))))))
```

Méthodes permettant à un objet de bouger en conservant la cohérence géométrique dans laquelle il est situé. (Conservation des rapports et des relations) :

```
(DE BOUGE-DE (X)
(BOUGE-DE1 (CAR X)(CADR X))))))))

(DE BOUGE-DE1(L  Y)
(COND((NULL L) NIL)
      ((ATOM L)
     (COND((GREATP(PLUS(XMAX(GETP L 'IMAGE)) Y)
                    (XMAX(GETP(GETP      L      'EST-CONTENU)
                                  'IMAGE)))
             (BOUGE-DE1(GETP L 'EST-CONTENU)
                       (SUB(PLUS Y (XMAX(GETP L 'IMAGE)))
                           (XMAX(GETP(GETP  L   'EST-CONTENU)
                                     'IMAGE))))
             (TRAN (GETP L 'IMAGE) Y 0))
           ((LESSP(PLUS(XMIN(GETP L 'IMAGE)) Y)
                    (XMIN(GETP(GETP      L      'EST-CONTENU)
                                  'IMAGE)))
             (BOUGE-DE1(GETP L 'EST-CONTENU)
                       (SUB(PLUS Y (XMIN(GETP L 'IMAGE)))
                           (XMIN(GETP(GETP  L   'EST-CONTENU)
                                     'IMAGE))))
             (TRAN (GETP L 'IMAGE) Y 0))
           (T(TRAN(GETP L 'IMAGE)Y 0)))
     (BOUGE-DE1(GETP L 'CONTIENT)Y)
     (BOUGE-DE1(GETP L 'PORTE) Y)
     )
     (T(BOUGE-DE1 (CAR L) Y)
       (BOUGE-DE1 (CDR L) Y))))))))))))))))))
```

Quelques fonctions d'assistance :

```
(DE MEMBERP(X Y)
(COND((NULL Y) NIL)
     ((EQ X (CAR Y)) T)
     (T(MEMBERP X (CDR Y)))))))

(DE XMIN(A)
(CAAR(MIMA A))))

(DE XMAX(A)
(CADR(MIMA A))))))
```

III). Un exemple de session obtenue avec cet interprèteur.

Les commentaires qui ne sont pas dans le dialogue homme-machine sont présentés en caractères gras et annoncent la commande qui suit.

- TANGRAM - V 3/2

1----

Chargement de l'interprète

```
(LOAD 'PATSM.LSP)  Fichier Ok
ASSOC
VALIDE
RECOIT
DONNE-A
QUI-EST
QUI
QUI-A
QUA
EST?
QUOI?
MEMBERP1
L1
(MLAMBDA  A(COND((MEMBERP(CADR  A)(GETP(GETP(CAR  A)(QUOTE
PERE))
(QUOTE   MET)))(APPLY(GETP(CADR  A)(QUOTE  EXPR))(CONS(CAR
A)(SUPE
(CDDR  A))))T)(T(PRIN1(LIST(CAR  A)(QUOTE   NE  SAIT   PAS
FAIRE  )
(CADR A))))))
SUPE
JE
META
ECR
ECRAN
FI
FICHIER
DEF-OBJET
INTERDEF
DEF-TYPE
AJOUTE-A
?
DETRUIT
SAUVE
SAVP
SAVI
CHARGE
AFFICHE
```

```
RECOIT-DEDANS
RECOIT-DESSUS
SUG2
PREND-ECHELLE
CREE
SE-REPETE
JONCTION
REP1
BOUGE-DE
BOUGE-DE1
MEMBERP
XMIN
XMAX

EVAL : PATSM.LSP
    2----
```

Définition du type COLONNE avec respect des contraintes classiques

```
(JE DEF-TYPE)
 Nom       : COLONNE
 SurType   : META
 Methodes  :  NIL  Elle hérite des méthodes de son sur-Type
 Modèle    :
((IMAGE(RESTRICT(LAMBDA(X Y)(EQ(GETP X 'RAPPORT)
                                (RDIV(GETP X 'HAUTEUR)
                                     (GETP X 'LARGEUR)))))
        (VALEUR)
        (DEMON(LAMBDA(X Y)(PROG
                        (PUTP X 'HAUTEUR (HO Y))
                        (PUTP X 'LARGEUR (LO Y))
                        (RETURN(PUTP  X 'ENTRAXE(MULT
                                     (GETP X 'ENTRAXEP)
                                     (GETP   X 'LARGEUR))
                        )))))
 )
 (ORDRE(DEMON(LAMBDA(X Y)(PROG
                        (PUTP X 'RAPPORT
                              (CAR(ASSOC Y
                                 '((DORIQUE 6)
                                   (IONIQUE 9)
                                   (CORINTHIEN 10))
                        )))
                       (RETURN
                       (PUTP X 'ENTRAXEP
                              (CAR(ASSOC Y '((DORIQUE 4,5)
                                             (IONIQUE 3)
                                             (CORINTHIEN 2,5)
                              ))))))))
     (VALEUR)
     (ENSEMBLE(DORIQUE IONIQUE CORINTHIEN))))))))))))
```

```
 Type défini

EVAL :  COLONNE
Note   :   La méthode ci-dessus aurait pu être chargée
directement depuis un fichier

    3----

 Ce que je sais faire

 (JE ?) (CHARGE VALIDE SAUVE DETRUIT VIDE ?  DEF-OBJET DEF-
TYPE)
EVAL :  T

    4-----

 Définition d'un Objet

 (JE DEF-OBJET)

  Nom         :  COL1
  Type        :  COLONNE
  Objet Défini

EVAL :  T

    5----

 Ce que sait faire la colonne COL1

 (COL1 ?)
(AFFICHE RECOIT-DESSUS RECOIT-DEDANS PREND-ECHELLE
CREE PREND-BASE SE-REPETE ? BOUGE-DE MONTE-DE RECOIT
DONNE-A AJOUTE-A)

EVAL :  T

    6----

 Instanciation d'une des facettes par l'opérateur

 (COL1 RECOIT ORDRE DORIQUE)

EVAL :  T

    7----

 L'information a été enregistrée

 (COL1 AFFICHE ORDRE) DORIQUE
EVAL :  T
```

8----

En fonction de l'ordre, l'entraxe type est instancié (en diamètres)

```
(COL1 AFFICHE ENTRAXEP) 04,5
EVAL :  T
```

9----

...Le rapport hauteur/base aussi

```
(COL1 AFFICHE RAPPORT) 06
EVAL :  T
```

10----

Instanciation de la facette image au moyen d'un élément graphique sur fichier. Celle-ci est acceptée par l'interprète après vérification de l'adéquation avec l'ensemble de contraintes

```
(COL1 RECOIT IMAGE : (RETR 'COL1)))

EVAL :  T
```

11----

On efface l'écran

```
(EFFACE)

EVAL :  T
```

12----

Recherche de l'image

```
(COL1 AFFICHE IMAGE)
```

```
EVAL :  T
```

13----

Quelques propriétés instanciées par les "DEMONS" lors de l'instanciation de l'image

```
 (COL1 AFFICHE HAUTEUR) 4992
EVAL :  T
```

14----

```
 (COL1 AFFICHE ENTRAXE) 4032
EVAL :  T
```

15----

Controle du domaine d'application des méthodes sur un objet

```
 (COL1 MODIFIE RAPPORT)
 (COL1 NE CONNAIT PAS LA METHODE MODIFIE)
EVAL : NIL
```

16----

Un objet en crée un autre par duplication de lui-m~me

```
 (COL1 CREE COLONNADE)
EVAL:  T
```

17----

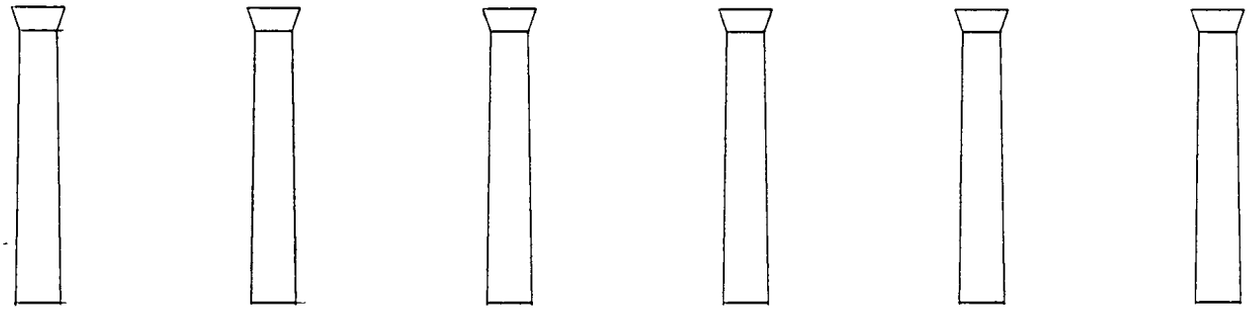
On peut construire graphiquement la colonnade, puisque tout l'objet est cohérent et son entr-axe a été mis à jour par les "DEMONS"

```
 (COLONNADE SE-REPETE 6)

EVAL:  T
```

```
   18----

 (COLONNADE AFFICHE IMAGE)
```



```
EVAL :  T

   19----

 Controle de modèle...

 (COLONNADE RECOIT COULEUR ROUGE)
     (COLONNADE NE CONNAIT PAS COULEUR ROUGE)
EVAL:  NIL

   20----

 Tout est mis à jour 'en ligne'

 (COLONNADE AFFICHE LARGEUR) 25088
EVAL :  T

   22----

(EXIT  On sort de TANGRAM

*****************************************************************

 NOMBRE DE CELLULES TANGRAM UTILISEES    :  8.61000E+03
 NOMBRE DE SESSIONS TRAITEES             :           22
 NOM DU FICHIER HISTORIQUE (si demandé)  : HISTPAGE.LSP

*****************************************************************
FIN TANGRAM A : 15:32:40

EVAL :  T
```

DEUXIEME EXEMPLE : UNE ORIENTATION "BASE DE DONNEES"

I). Présentation des langages à structure de Frames

Nous présentons ici les spécificités d'un autre type de langage que nous avons implémenté. 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>).

I-1. 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 : certaines primitives permettront néanmoins de rendre des objets privilégiés non modifiables par le système.

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.

Les différentes possibilités de la structure déclarative :

- 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.

I-2. 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" 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.

L'aspect déclaratif et procédural de la structure sera mis en évidence un peu plus loin, néanmoins le lecteur peut constater dès à présent que la valeur par défaut peut être vue comme un cas particulier de l'aspect procédural de la structure.

II). Mise en oeuvre de notre prototype

Nous avons repris l'exemple donné dans <CARADANT-84>, écrit sur APPLE II que nous avons porté sur BFM-186 en y incluant un éditeur et surtout en améliorant le système de représentation jusqu'à en faire un langage. Ceci nous a permis sur des exemples architecturaux simples de voir les problèmes générés par une approche hiérarchisée des connaissances, quelles soient déclaratives ou procédurales. Nous avons ensuite donné une puissance accrue aux objets en leurs incluant la presque totalité des structures de contrôle.

La mise en place des exemples liés au domaine de l'architecture nous a permis de valider notre approche et de donner les améliorations nécessaires pour permettre son utilisation en CAO en architecture.

Le langage est écrit en LISP et profite de toutes les possibilités de ce langage (le source LISP de notre interpréteur est donné en annexe).

II-1. Structure générale

Nous ne donnerons ici que l'essentiel, le lecteur pourra se reporter à <CARADANT-84> si nécessaire.

La structure que nous avons retenue est basée sur l'utilisation des listes d'association. Une liste d'association est une liste d'éléments, chacun pouvant être identifié et retrouvé dans la liste avec ce que l'on appelle une clé.

Grâce à cette structure, la frame apparaît au plus haut niveau comme ceci :

(<nom frame> (<nom propiété1> ...) (<nom propriété2> ...)
...)

C'est une liste de propriétés dont les clés sont les noms de propriétés. Au niveau en dessous nous avons :

(<nom propriété> (<aspect1> ...) (<aspect2> ...) ...)

C'est une liste d'association dont les clés sont les aspects de la propriété. Elles permettent de retrouver les aspects VALEUR, DEFAUT, SI-BESOIN, SI-AJOUT et SI-DETRUIT, dont on précisera l'utilisation ultérieurement .

Au niveau le plus bas nous avons :

(< nom aspect> (<valeur1>) (<valeur2>) ...)

Ce qui donne comme structure générale de la frame:

```
(<nom frame> (<propriété1> (<aspect1> (<valeur1>)
                                      (<valeur2>)
                                       .
                                       . )
                           (<aspect2> (<valeur1>)
                                       .
                                       . )
             (<propriété2>   (<aspect1>   (<valeur1>)
...)...)
                              .
                              . )
```

La solution retenue pour stocker la frame d'un objet a été de l'accrocher à la liste de propriétés de l'objet sous la propriété "FRAME".

II-2. Les interfaces avec l'environnement

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

II-2.1. Les interfaces courantes

- Création des frames

PUT-F

* PUT-F place les informations dans la structure.

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

- **Consultation des frames**

GET-F

* **GET-F** ramène les informations de la structure.

Le chemin d'accès passe par une frame, une propriété, et un aspect.

- **Destruction des frames**

DEL-F

* **DEL-F** détruit des informations dans la structure.

Il faut donner un chemin de type frame-propriété-aspect.

II-2.2 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**

GET-F-H

* **GET-F-H** ramène les informations de la structure.

L'utilisateur donne un chemin d'accès de la forme frame-propriété. GET-V-H 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

PUT-F-H

* PUT-F-H place les informations dans la structure.

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

DEL-F-H

* DEL-F-H détruit les informations.

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

II-2.3. Procédures de gestion des héritages

Nous avons un certain nombre de procédures de gestion des héritages :

-Procédures de consultation

F-CLASSES

F-CLASSES ramène tous les pères d'un objet, à tous les niveaux dans toutes les branches.

F-FILS

F-FILS ramène tous les fils d'un objet .

- Gestion du réseau des hiérarchies

La gestion du réseau créé par la propriété EST-UN est réalisée par les objets eux-mêmes, c'est à dire que la détection des bouclages (qui peuvent être générés si l'on admet la structure réseau) est réalisée par l'objet origine de toute la hiérarchie (nous l'appellerons LIEU). Nous verrons aussi qu'il gère la plupart des structures de contrôle du système.

- Les utilitaires annexes

Un certain nombre d'utilitaires a pour but de gérer le dialogue Homme-Machine et les interfaces fichiers.

Nous ne décrirons pas en détail l'éditeur car là n'est pas notre propos, néanmoins il faut savoir que celui-ci présente les différentes options sous formes de menus et qu'il est modifiable par l'utilisateur (il est, lui aussi, écrit en LISP).
Le système peut sauver une frame sous forme de fichier de caractères directement manipulable par un éditeur de type WORDSTAR. Ce fichier est lu directement par LISP pour lui permettre de construire la structure de données correspondante. Le sauvetage et le chargement d'une frame ne touche que les objets contenus dans cette frame (nous avons choisi de sauver toute la hiérarchie d'un objet pour lui permettre d'être à nouveau cohérent lors de son chargement).

Les utilitaires d'impression permettent d'écrire les frames sous un format PRETTY-PRINT, au sens LISP : c'est sous cette forme que seront stockés les fichiers.

III). Mise en place d'un exemple

La manipulation que nous avons mise en place n'a pas la prétention de montrer toutes les possibilités et limites de la structure que nous proposons. Elle a pour but, à travers un exemple simple, de montrer la démarche que sous-tend l'utilisation de ce genre de système, l'analyse de la pertinence de ces systèmes intégrés à un systeme de CAO sera étudiée dans la suite de nos recherches.

Cet exemple tend à montrer l'utilisation possible de ce type de langage objet pour la validation dynamique de la cohérence d'un projet.

Dans ce cas particulier il s'agit d'un projet ayant des contraintes très précises : les contraintes sont données, soit par la réglementation en vigueur, soit d'une manière très générale par les desiderata du concepteur : ils seront donc gérés dynamiquement par celui-ci. C'est à dire qu'il pourra, à tout instant, changer ses contraintes de manière interactive, (nous voyons ici la notion de base de données dynamique chère à tous les concepteurs de systèmes de CAO).

Les structures de contrôle étant elles aussi portées par les objets, certaines modifications peuvent être fatales au système (suppression du test de bouclage dans les héritages par exemple).

Nous avons choisi d'étudier la réalisation d'une école : nous ne ferons qu'effleurer cet objectif, le projet étant bien trop vaste pour être étudié complètement dans le cadre de cet exemple.

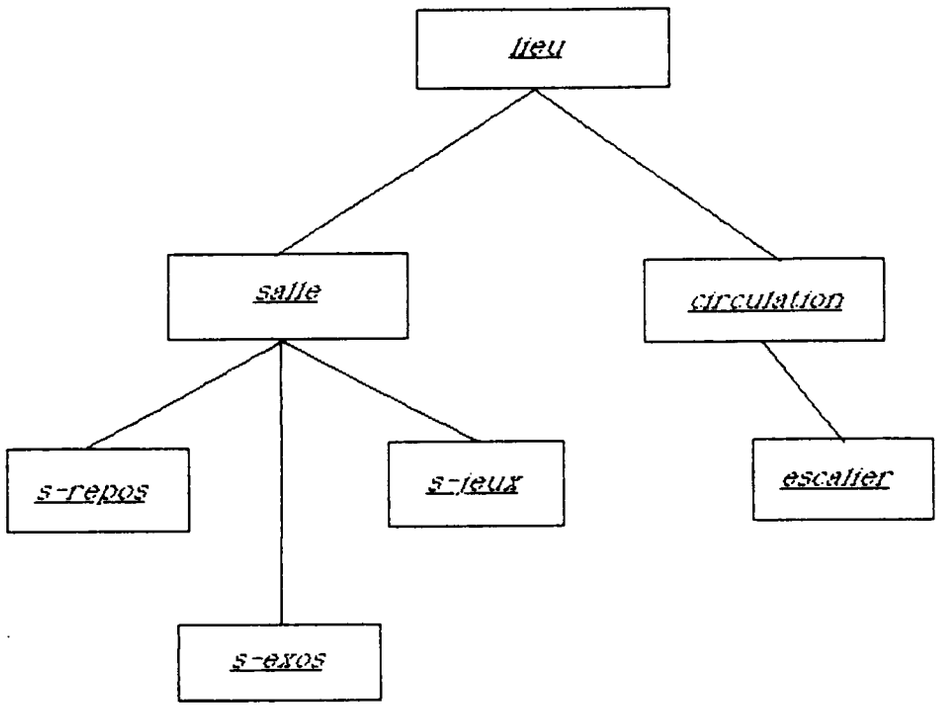
Nous allons néanmoins essayer de montrer comment l'approche que nous proposons peut aider l'architecte en phase de conception : l'aide semblera peut-être bien modeste pour une étude en situation professionnelle, mais cette tentative peut être généralisée dans des situations beaucoup plus complexes si les moyens humains et matériels sont disponibles.

- Une orientation "base de données" -

La structure de représentation que nous proposons est basée sur une approche hiérarchique des objets que manipule l'architecte au plus haut niveau de la conception : les espaces.

III-1. Structure hiérarchisée des espaces

Elle peut se mettre sous la forme :



III-2. Les structures de représentation de chaque espace

Nous donnerons les représentations telles quelles sont créées par le système, c'est à dire en LISP sur un fichier de type caractères. Le listing source complet de ce prototype se trouve en annexe. Nous ne donnerons ici, pour ne pas alourdir le texte, que les définitions des Frames.

La structure de plus haut niveau : l'espace LIEU

L'espace LIEU est un espace abstrait qui porte les structures de contrôle de tous les autres objets. Sa frame se présente ainsi :

```
(PUTQQ LIEU FRAME
(LIEU (PERE-DE (VALEUR (ESCALIER)
                       (CIRCULATION)
                       (SALLE)))
      (HAUTEUR (SI-AJOUT ((VERIF-LIM 200 1000))
                         ((VAL-UNIQUE))))
      (LARGEUR (SI-AJOUT ((VAL-UNIQUE))))
      (LONGUEUE (SI-AJOUT ((VAL-UNIQUE))))
      (SURF-VITRE (SI-AJOUT ((VAL-UNIQUE))))
      (SURFACE (SI-AJOUT ((VAL-UNIQUE))))
      (NB-DE-MARCHE (SI-AJOUT ((VAL-UNIQUE))))
      (RAMPE (SI-AJOUT ((VAL-UNIQUE))))
      (COMMUNIQUE-AVEC (SI-AJOUT ((SYMETRIQUE))))))
```

PERE-DE est une propriété qui permet à un objet de retrouver tous ses fils.

Ex LIEU a pour fils : ESCALIER CIRCULATION SALLE

VAL-UNIQUE est une procédure LISP qui permet de ne garder que la dernière valeur pour une propriété qui ne peut pas en avoir plusieurs :

Ex : LONGUEUR LARGEUR HAUTEUR SURFACE ... pour un volume quelconque.

SYMETRIQUE est une procédure qui permet de créer automatiquement dans la frame le symétrique d'une propriété

Ex :
SALLE-DE-JEUX COMMUNIQUE-AVEC VALEUR SALLE-A-MANGER

crée automatiquement à cause du méta-objet LIEU :

SALLE-A-MANGER COMMUNIQUE-AVEC VALEUR SALLE-DE-JEUX

Les espaces types de haut niveau : SALLE CIRCULATION

```
(PUTQQ SALLE FRAME
(SALLE (EST-UN (VALEUR (LIEU)))
       (HAUTEUR (SI-AJOUT ((VERIF-LIM 250
                                      350))))
       (LARGEUR (SI-AJOUT ((VERIF-LIM 400
                                      700))))
       (LONGUEUR (SI-AJOUT ((VERIF-LIM 500
                                       800))))
       (SOL (DEFAUT (MATERIAUX-SOUPLES))
            (SI-AJOUT ((VERIF-SET LINO PLASTIC MOQUETTE))))
       (COMMUNIQUE (VALEUR (EXTERIEUR)))
       (OCCULTATION (DEFAUT (PARE-SOLEIL)))
       (SITUATION (VALEUR (R-C-H))
                  (SI-AJOUT (INTERDIT)))))
)
```

```
(PUTQQ CIRCULATION FRAME
(CIRCULATION (EST-UN (VALEUR (LIEU)))
             (LONGUEUR (SI-AJOUT ((VERIF-LIM 0
                                             20))))
             (RAMPE (SI-AJOUT ((VERIF-LIM 0
                                          10))))
             (LARGEUR (DEFAUT (250))
                      (SI-AJOUT ((VERIF-LIM 150
                                            300))))))
)
```

VERIF-LIM est une procédure qui autorise uniquement les valeurs comprises entre les bornes passées en paramètres :

Ex : la hauteur d'une salle est comprise entre 250 et 350.

VERIF-SET est une procédure qui vérifie qu'une valeur est bien dans l'ensemble des valeurs données en paramètre :

Ex : le sol d'une salle a pour valeur par défaut MATERIAUX-SOUPLES et ses valeurs font partie de l'ensemble (MOQUETTE LINO PLASTIC).

INTERDIT est une procédure qui interdit l'action qui est en cours

Note :

L'utilisation dans une hiérarchie de l'aspect VALEUR a pour fonction de rendre obligatoire certaines valeurs dans les propriétés où il est employé :

> Dans notre exemple toutes les SALLES communiquent au moins avec l'extérieur. Elles peuvent communiquer avec d'autres espaces mais on retrouvera toujours la valeur EXTERIEUR lorsque l'on interrogera un objet relié à cette frame.
>
> La situation d'une salle est en R-C-H et c'est la seule possible car tout ajout de valeurs est interdit.

Les objets générateurs de niveau inférieur

Nous retrouvons ici tous les objets qui vont nous servir à générer les différents espaces qui composent une école. Nous retrouvons les différents types de salles telles qu'elles sont définies par les cahiers des chages courants. Chaque type portera alors ses propres contraintes.

Nous avons généré par l'objet SALLE :

les salles de jeux	:	S-JEUX
les salles d'exercices	:	S-EXO
les salles de repos	:	S-REPOS

```
(PUTQQ S-JEUX FRAME
(S-JEUX (EST-UN (VALEUR (SALLE)))
        (COMMUNIQUE (DEFAUT (S-EXO S-MANGER))
                    (SI-AJOUT ((AJOUT TERR-JEUX)))))

(PUTQQ S-EXO FRAME
(S-EXO (EST-UN (VALEUR (SALLE)))
       (SURFACE (SI-AJOUT ((VERIF-LIM 50
                                      70)))))

(PUTQQ S-REPOS FRAME
(S-REPOS (EST-UN (VALEUR (SALLE)))
         (VOLUME (SI-AJOUT ((NVERIF-SET MEZZANINE))))

         (COMMUNIQUE (VALEUR (S-EXO-P)))))
```

Nous avons généré à partir de CIRCULATION

```
Les escaliers          :    ESCALIER
Les circulations       :    CIRCULATION,
```

une frame pouvant être un générateur de frames terminales ou pas.

```
(PUTQQ ESCALIER FRAME
(ESCALIER  (EST-UN (VALEUR (CIRCULATION)))
           (LARGEUR (DEFAUT (200))
                    (SI-AJOUT ((VERIF-LIM 150
                                          300))))
           (NB-DE-MARCHE (SI-AJOUT ((VERIF-LIM 3
                                               15))))
           (VOLEE (VALEUR (DROITE))
                  (SI-AJOUT ((INTERDIT))))
           (CONTRE-MARCHE (VALEUR (AVEC)))
          ))
```

Nous ne détaillons pas ici toutes les frames utilisées, celles-ci réagissant comme les précédentes : il faut néanmoins constater que les objets de plus bas niveau sont les plus précis et qu'ils affinent les contraintes et les valeurs par défaut.

III-3. Un exemple de session obtenu avec ce prototype :

Nous avons, à l'aide de la frame mise en place, "rentré" un petit projet pour une école : les contrôles de cohérence dynamiques se sont passés comme prévu et les interrogations de la structrure de données en différents instants de la manipulation ont permis de retrouver les valeurs par défaut que nous avions données, ou les valeurs nouvellement introduites.
La session suivante donne un aperçu concret de ce que nous venons de dire.

Les commentaires qui ne sont pas dans le dialogue Homme-Machine sont présentés en caractères gras et annoncent la commande qui suit.

Menu des ordres possibles tel qu'il se présente au chargement de l'éditeur ; il peut être appellé à chaque instant grâce à la touche M.

*** Editeur de Frame Version 1.2 Dominique Caradant LI2A-85

O P T I O N S

F EDIT FRAME
A AJOUT PROP
K SUPRIME PROP
H DONNE HERITAGE
E EXPLORE CLASSES
T TYPE OBJET
V RECHERC VALEUR
I IMPRIME FRAME
S SAUVE FRAME
R RAMENE FRAME
U UNIVERS
M MENU
X EXIT

L'univers de l'éditeur après chargement de l'environnement présenté en exemple :

Entrez vos Choix: U

--> (S-JEUX SALLE S-EXO LIEU CIRCULATION ESCALIER)

- Une orientation "base de données" -

Création d'un objet :

Entrez vos Choix: F

Frames ? : -Nom- -Propriété- -Aspect- -Valeur-
: ESCALIER1 EST-UN VALEUR ESCALIER

--> ESCALIER

Ramener la propriété d'un objet :

Entrez vos Choix: E

Consulte Frames ? : -Nom- -Propriété-
: ESCALIER1 LARGEUR

--> (200)

Ajouter une propriété :

Entrez vos Choix: A

Ajoute Frames ? : -Nom- -Propriété- -Aspect- -Valeur-
: ESCALIER1 LARGEUR VALEUR 250

--> 250

Entrez vos Choix: E

Consulte Frames ? : -Nom- -Propriété-
: ESCALIER1 LARGEUR

--> (250)

L'ajout de la propriété suivante est interdite par un attachement procédural dans la frame ESCALIER :

Entrez vos Choix: A

Ajoute Frames ? : -Nom- -Propriété- -Aspect- -Valeur-
: ESCALIER1 NB-DE-MARCHE VALEUR 2

--> Votre Valeur n'est pas adaptée les limites sont : 3 ; 15
--> Cette modification est interdite à cause de : ESCALIER

Quel est notre nouvel univers ? :

Entrez vos Choix: U

--> (S-JEUX SALLE S-EXO LIEU CIRCULATION ESCALIER ESCALIER1)

```
                    Entrez vos Choix: A

Ajoute Frames ? : -Nom- -Proprieté- -Aspect- -Valeur-
                : ESCALIER1 LARGEUR VALEUR 350

--> Votre Valeur n'est pas adaptée les limites sont : 150 ; 300
--> Cette modification est interdite à cause de : ESCALIER
```

Pour détruire une propriété particulière dans une frame : ici l'attachement procédural précédent :

```
                    Entrez vos Choix: K

Supprime Frames ? : -Nom- -Proprieté- -Aspect- -Valeur-
                  : ESCALIER LARGEUR SI-AJOUT (VERIF-LIM 150 300)
```

Impression du contenu d'une Frame :

```
                    Entrez vos Choix: I

Nom(s) de(s) frames : : ESCALIER

(ESCALIER (EST-UN (VALEUR (CIRCULATION)))
          (LARGEUR (DEFAUT (200)))
          (NB-DE-MARCHE (SI-AJOUT ((VERIF-LIM 3
                                               15))))
          (VOLEE (VALEUR (DROITE))
                 (SI-AJOUT ((INTERDIT))))
          (CONTRE-MARCHE (VALEUR (AVEC)))
          (PERE-DE (VALEUR (ESCALIER1)))))
```

On va voir maintenant que la frame CIRCULATION empêche aussi l'introduction de la valeur 350 (la frame CIRCULATION est plus haute dans la hiérarchie) :

```
                    Entrez vos Choix: A

Ajoute Frames ? : -Nom- -Proprieté- -Aspect- -Valeur-
                : ESCALIER1 LARGEUR VALEUR 350

--> Votre Valeur n'est pas adaptée les limites sont : 150 ; 300
--> Cette modification est interdite à cause de : CIRCULATION
```

Voyons les héritages...

```
                    Entrez vos Choix: H

Consulte Classes de(s) Frame(s) ? : : (ESCALIER)

Les Pères sont : (ESCALIER CIRCULATION LIEU)
Les Fils sont : (ESCALIER ESCALIER1)
```

Suppression d'un attachement procédural :

```
                Entrez vos Choix: K

Supprime Frames ? : -Nom- -Proprieté- -Aspect- -Valeur-
                  : CIRCULATION LARGEUR SI-AJOUT (VERIF-LIM 150 300)
```

Imprimons le résultat pour vérifier la modification effectuée :

```
                Entrez vos Choix: I

Nom(s) de(s) frames : : CIRCULATION

(CIRCULATION (EST-UN (VALEUR (LIEU)))
             (LONGUEUR (SI-AJOUT ((VERIF-LIM 0
                                             20))))
             (RAMPE (SI-AJOUT ((VERIF-LIM 0
                                          10))))
             (LARGEUR (DEFAUT (250)))))
```

Avec l'ajout suivant, la valeur est enregistrée et l'ancienne valeur est détruite (à cause de l'attachemment procédural de LIEU : largeur est une valeur unique) :

```
                Entrez vos Choix: A

Ajoute Frames ? : -Nom- -Proprieté- -Aspect- -Valeur-
                : ESCALIER1 LARGEUR VALEUR 350

--> L'ancienne valeur est detruite
--> 350
```

Vérifions :

```
                Entrez vos Choix: I

Nom(s) de(s) frames  : ESCALIER1

(ESCALIER1 (EST-UN (VALEUR (ESCALIER)))
           (LARGEUR (VALEUR (350))))
```

Les trois échanges suivants montrent que COMMUNIQUE est une propriété symétrique :

```
                Entrez vos Choix: A

Ajoute Frames ? : -Nom- -Proprieté- -Aspect- -Valeur-
                : ESCALIER1 COMMUNIQUE VALEUR S-JEUX-1

--> S-JEUX-1
```

```
                    Entrez vos Choix: I

Nom(s) de(s) frames :  ESCALIER1

(ESCALIER1 (EST-UN (VALEUR (ESCALIER)))
           (LARGEUR (VALEUR (350)))
           (COMMUNIQUE (VALEUR (S-JEUX-1)))))

                    Entrez vos Choix: I

Nom(s) de(s) frames : S-JEUX-1

--> (S-JEUX-1 (COMMUNIQUE (VALEUR (ESCALIER1)))))
```

Quel est l'aspect valeur de ESCALIER pour la propriété VOLEE ? :

```
                    Entrez vos Choix: E

Consulte Frames ? : -Nom- -Propriété-
                  : ESCALIER1 VOLEE
--> (DROITE)
```

Et c'est la seule valeur possible :

```
                    Entrez vos Choix: A

Ajoute Frames ? : -Nom- -Propriété- -Aspect- -Valeur-
                : ESCALIER1 VOLEE VALEUR CIRCULAIRE

--> Cette modification est interdite à cause de : ESCALIER
```

Test du maintien de la cohérence de la hiérarchie effectué par l'objet LIEU (attachement procédural à la propriété EST-UN de l'objet LIEU) :

```
                    Entrez vos Choix: A

Ajoute Frames ? : -Nom- -Propriété- -Aspect- -Valeur-
                : LIEU EST-UN SI-AJOUT (TESTBOUCLE)

--> (TESTBOUCLE)

                    Entrez vos Choix: A

Ajoute Frames ? : -Nom- -Propriété- -Aspect- -Valeur-
                : CIRCULATION EST-UN SI-AJOUT ESCALIER1

--> Vous essayez de générer un bouclage dans les héritages.
--> l'héritage courant est :  (ESCALIER1 ESCALIER CIRCULATION
LIEU)
```

Etc...

CONCLUSION

Nous avons exploré quelques uns des modes de structuration de l'univers de travail proposés par les techniques de programmation issues des Langages Orientés Objets.

Nous avons successivement :
- Mené une manipulation simple permettant de respecter une méthode (bien décrite) par l'architecte J.N.L. Durand. Ceci n'a pu être fait que parce que :
  * nous disposions de l'outil "Tangram" développé par ailleurs (langage LISP possédant des accès graphiques interactifs permettant de visualiser en deux dimensions des propriétés graphiques d'un objet)
  * J.N.L. Durand a décrit d'une manière non ambigüe la caractérisation des entr-axes de colonnades.

Nous avons alors remarqué que si la définition interactive de nos objets et nos procédures était un atout majeur dans la libre expression de nos méthodes, cet atout se payait par la nécessité d'un dialogue homme-machine en LISP. Ce langage est trop difficile à manier pour un utilisateur non averti. Nous avons aussi compris qu'il vaudrait mieux disposer d'un univers d'objets plus structuré pour pouvoir manipuler plus simplement, et de manière cohérente, nos objets. Notre intérêt s'est alors porté sur les langages orientés objets.

- Ecrit un premier prototype, orienté "méthodes" (au sens des langages objets) pour explorer cette orientation.

Nous avons alors un sur-interpréteur qui permet d'une part de s'affranchir des arcanes de LISP et qui permet à un utilisateur architecte de se servir du prototype, l'exemple de session présenté le montrant clairement. De plus, pour ce premier prototype, nous avons résolument tourné nos démonstrations vers une visualisation graphique de nos éléments, pour montrer en particulier que les actions entreprises pouvaient être saisies immédiatement dans les modes de représentation utilisés en architecture. On trouvera peut-être l'exemple assez banal et assez éloigné de la pratique professionnelle, mais il s'agit là d'un premier pas qui nous semble porteur de grandes promesses.

- Ecrit un deuxième prototype, toujours en LISP, orienté, lui, dans une optique "base de données". Ce prototype est différent du premier en ce qu'il n'utilise ni les mêmes concepts, ni même les mêmes définitions que le premier, mais il représente une autre approche : celle qui permet de structurer un univers d'objets en fonction de certaines règles et contraintes, et de faire que toute action entreprise sur ou avec ces objets respecte la cohérence de cet univers en permanence. Le sur-interprète est par ailleurs un autre exemple de ce qui peut être fait pour la dialogue homme-machine. Il travaille plutôt sous forme de menus, ce qui rend sa présentation sur papier un peu plus délicate. Par ailleurs, nous avons volontairement dépouillé tous les objets de l'univers manipulé par ce deuxième protoype de leurs aspects graphiques, et ceci pour faire porter l'accent uniquement sur les problèmes de cohérence.

Ces trois exemples nous autorisent à conclure sur la très grande richesse des applications possibles de l'Intelligence Artificielle en phase de conception en Architecture. Les langages orientés objets, appliqués aux deux derniers exemples, semblent encore plus prometteurs.

Force nous est cependant de reconnaître que nous ne voyons encore que la partie émergée de l'iceberg, et ceci nous amène à faire le point sur l'ensemble de notre démarche. Nous avons dès à présent à notre disposition l'ensemble des outils informatiques nécessaires pour notre approche. Bien sûr, ces outils devront être affinés, voire même redéfinis, mais l'effort doit à présent porter sur la pertinence de notre approche sur le plan architectural. C'est par une analyse des objets de l'architecture et de la démarche architecturale (ces deux axes devant être développés simultanément) que nous pourrons valider ou non nos travaux.

Nous savons évidemment que nous nous attaquons là à un problème très difficile, mais la maîtrise des outils informatiques nous rejette - enfin - vers l'objet principal de nos préoccupations : l'Architecture.

Nous envisageons notamment d'exploiter les résultats de notre recherche dans l'élaboration d'une "ébauche" de système expert orienté objets en CAAO. Nous pensons en effet que l'association de la "structuration Objets" et des techniques de résolution de l'Intelligence Artificielle nous permettra de fournir un environnement souple et interactif pour la manipulation d'éléments d'architecture dans un souci de contrôle de cohérence.

ANNEXE

****

******** GESTION DES FRAMES **********

****

Ce programme est une interface interactive qui permet d'utiliser les procédures de gestion des frames. Il permet dans le réseau créé par celles-ci de retrouver toutes les propriétés rattachées à une certaine classe d'objets. Le lien EST-UN permet de créer ces différentes hiérarchies. Les différents aspects reconnus par le système sont :

VALEUR , DEFAUT , SI-BESOIN , SI-AJOUT , SI-DETRUIT

ceux-ci ne sont pas limitatifs ; l'utilisateur peut donc en introduire d'autres à volonté.

Pour voir l'effet de chaque aspect, se reporter à la définition de chaque procédure ; toutefois il est à remarquer que les valeurs des aspects : SI-BESOIN, SI-AJOUT, SI-DETRUIT sont des procédures dont la forme d'appel a pour allure :

(FONCT ARG1 ARG2 ... ARGn)

Cette forme est exploitée par FUNCALL.

Nous ne donnons pas ici le listing de l'éditeur proprement dit car cela prendrait trop de place. Il nous semble d'ailleurs préférable que l'accent soit mis sur le coeur du système de frames et non sur les différents interfaces possibles avec l'utilisateur.

Procédure de parcours de l'OBLIST : elle ramène tous les objets vérifiant le prédicat PRED (il doit lui-même ramener un objet).

```
(DEFUN PARCOURT (LAMBDA (PRED LST)
  (SETQ LST (OBLIST))
  (SETQ UNIVERS NIL)
  (LOOP
    ((NULL LST) UNIVERS)
    (SETQ UNIVERS (APPEND UNIVERS (PRED (POP LST)))) ) ))
```

****

```
(DEFUN FRAMEP (LAMBDA (OBJ)
 (COND  ((GET OBJ 'FRAME)
    (LIST OBJ) ) )))
```

****

```
(DEFUN UNIVERS (LAMBDA NIL
(REVERSE  (PARCOURT 'FRAMEP)) ))
```

****

Procédure qui type un objet en vue de le rendre intouchable. C'est à dire que l'on ne pourra pas écrire ou détruire les informations dans la frame de l'objet après cette opération.

```
(DEFUN TYPE (LAMBDA (OBJ)
(FLAG OBJ 'TYPE))))

(DEFUN UNTYPE (LAMBDA (OBJ)
(REMFLAG OBJ 'TYPE))))
```

****

ASSOC-F est la procédure de plus bas niveau pour la manipulation des frames. Si l'information se trouve dans la A-LST considérée alors on la ramène, sinon on ajoute de manière HARD cette information dans la A-LISTE.
Ceci supprime le problème de l'écriture dans une fermeture (au sens LISP du terme) car toutes les manipulations sont réalisées de manière HARD et immédiatement.

```
(DEFUN ASSOC-F (LAMBDA (CLEE A-LST)
  (COND
    ((ASSOC CLEE (CDR A-LST)))
    (T (CADR (RPLACD (LAST A-LST) (LIST (LIST CLEE)))))  )
))
```

****

PUT-F est la procédure qui met des informations dans une frame. Si la frame n'existe pas il la crée et il se sert de ASSOC-F pour générer dynamiquement le chemin d'accès à l'information ; si la frame est protégée ou si l'aspect n'est pas autorisé, PUT-F génère une erreur.

```
(DEFUN PUT-F (LAMBDA (FRA SLO FAC VAL)
(COND  ((FLAGP FRA 'TYPE)
          (THROW 'ERR (ERREUR 11)) )
       ((NOT (MEMBER FAC LSTFAC))
         (THROW 'ERR (ERREUR 31)) )
       ((MEMBERP VAL (GET-F FRA SLO FAC)) NIL)
         (ASSOC-F  VAL  (ASSOC-F  FAC  (ASSOC-F  SLO   (FGETFRAME
                   FRA))))
 VAL ))))
```

****

GET-F ramène l'information présente dans la frame.

```
(DEFUN GET-F (LAMBDA (FRA SLO FAC)
  (MAPCAR (CDR (ASSOC FAC
    (CDR (ASSOC SLO (CDR (GET FRA 'FRAME))))))) 'CAR) ))
```

****

DEL-F : Détruit toujours de manière HARD l'information présente dans la frame. Quelle que soit la profondeur, seule l'information considérée est détruite. La structure est par là même préservée.

```
(DEFUN DEL-F (LAMBDA (FRA SLO FAC VAL SLOS FACS VALS INFS)
     (COND   ((FLAGP FRA 'TYPE) (ERREUR 11))
             (T (SETQ SLOS (FGETFRAME FRA))
                 (SETQ FACS (ASSOC SLO (CDR SLOS)))
                 (SETQ VALS (ASSOC FAC (CDR FACS)))
                 (SETQ INFS (ASSOC VAL (CDR VALS)))
                 (HARDEL INFS VALS)
                 (COND ((NULL (CDR VALS)))
                       (T (HARDEL VALS FACS) ) ))
                 (COND ((NULL (CDR FACS))
                       (T( HARDEL FACS SLOS) ) ))
                   (NOT (NULL INFS)) ))))
```

****

LES PROCEDURES QUI MANIPULENT LES HERITAGES

****

GET-F-I explore le réseau créé par les différentes hiérarchies pour ramener tout ce qu'il y a dans l'aspect VALEUR de la frame. Elle ramène une liste (au sens LISP du terme) de TOUTES LES VALEURS qui sont trouvées dans la hiérarchie.

```
(DEFUN GET-F-I (LAMBDA (FRA SLO CLASSES RESULT)
  (SETQ CLASSES (F-CLASSES FRA))
  (LOOP
    ((NULL CLASSES) RESULT)
    (SETQ  RESULT  (APPEND RESULT (GET-F (CAR CLASSES)  SLO
          'VALEUR)))
    (POP CLASSES) ) )))
```

****

GET-F-H fait comme GET-F-I dans un premier temps mais ensuite il explore pour chaque noeud du réseau l'aspect DEFAUT et SI-BESOIN ; il arrête l'exploration à la première valeur trouvée ou calculée.

```
(DEFUN GET-F-H (LAMBDA (FRA SLO CLASSES RESULTAT)
  (SETQ CLASSES (F-CLASSES FRA))
  (LOOP
    ((NULL CLASSES) RESULTAT)
    ((SETQ RESULTAT (OR (GET-F-I FRA SLO)
          (GET-F (CAR CLASSES) SLO 'DEFAUT)
          (MAPCAR  (GET-F  (CAR  CLASSES)  SLO  'SI-BESOIN)
                   '(LAMBDA (E)  (APPLY E NIL) )) )))
    (POP CLASSES) ) ))
```

****

PUT-F-H est une fonction qui insère une information en exécutant les procédures présentes dans l'aspect SI-AJOUT de toutes les frames du réseau hiérarchique.

ACTION** est une variable globale qui autorise finalement l'insertion. Elle est obligatoire à cause de MAPC (ACTION** est manipulée par la procédure INTERDIT).

DEL** est une variable globale qui contient l'ancienne valeur à détruire si cette action est finalement autorisée (DEL** est manipulée par la procédure VAL-UNIQUE).

```
(DEFUN PUT-F-H (LAMBDA (FRA SLO FAC VAL)
 (COND ((PUT-F FRA SLO FAC VAL)
        (DEL-F FRA SLO FAC VAL)
         (MAPC (F-CLASSES FRA) '(LAMBDA (E)
            (MAPC (GET-F E SLO SI-AJOUT) 'FUNCALL) ))
       ((NOT (NULL ACTION**))
           (DEL-F FRA SLO FAC DEL**)(SETQ DEL** NIL)
           (PUT-F FRA SLO FAC VAL) )
       (T (SETQ ACTION** T) ) ))
```

****

DEL-F-H détruit une propriété et vérifie dans le réseau si des attachements procéduraux ne sont pas mis en action dans ce cas à travers l'aspect SI-DETRUIT.

```
(DEFUN DEL-F-H (LAMBDA (FRA SLO FAC VAL )
   (COND ((DEL-F FRA SLO FAC VAL)
                  (MAPC  (F-CLASSES FRA) '( LAMBDA (E)
                    (MAPC   '(GET-F   E  SLO   'SI-DETRUIT)
                              'FUNCALL))))
              (T VAL)))))
```

FGETFRAME ramène la frame si elle existe et la crée sinon.

```
(DEFUN FGETFRAME (LAMBDA (FRA)
  (COND
    ((GETP FRA 'FRAME))
    (T (PUTP FRA 'FRAME (LIST FRA))) ) ))
```

****

FREPLACE transfère toute une liste de propriétés d'une frame dans l'autre.

```
(DEFUN FREPLACE (LAMBDA (FRA1 FRA2 SLO)
  (RPLACD (ASSOC-F SLO (FGETFRAME FRA1))
          (CDR (ASSOC-F SLO (FGETFRAME FRA2))))
 ))
```

****

F-CLASSES ramène tous les pères d'un objet et ceci dans tout le réseau.

```
(DEFUN F-CLASSES (LAMBDA (FRA QUEUE TEMPO CLASSES)
  (SETQ QUEUE (LIST FRA))
  (LOOP
    ((NULL QUEUE)
      (REVERSE CLASSES) )
    ( ((NOT (MEMBER (CAR QUEUE) CLASSES))
       (SETQ CLASSES (CONS (CAR QUEUE) CLASSES)) ) )
    (SETQ TEMPO (GET-F (CAR QUEUE) 'EST-UN 'VALEUR))
    (SETQ QUEUE (CDR QUEUE))
    (SETQ QUEUE (APPEND QUEUE TEMPO)) ) ))
```

****

F-FILS ramène tous les fils d'un objet.

```
(DEFUN F-FILS (LAMBDA (FRA QUEUE TEMPO CLASSES)
  (SETQ QUEUE (LIST FRA))
  (LOOP
    ((NULL QUEUE)
      (REVERSE CLASSES) )
    ( ((NOT (MEMBER (CAR QUEUE) CLASSES))
       (SETQ CLASSES (CONS (CAR QUEUE) CLASSES)) ) )
    (SETQ TEMPO (GET-F (CAR QUEUE) 'PERE-DE 'VALEUR))
    (SETQ QUEUE (CDR QUEUE))
    (SETQ QUEUE (APPEND QUEUE TEMPO)) ) ))
```

I-PUT-F : Sert lors de la création d'un objet pour initialiser l'arbre des hiérarchies.

```
(DEFUN I-PUT-F (LAMBDA (FRA SLO FAC VAL)
  ((EQUAL SLO 'EST-UN )
   (PUT-F VAL 'PERE-DE FAC FRA)
      (PUT-F FRA SLO FAC VAL) )
    (PUT-F FRA SLO FAC VAL))))
```

****

Un FUNCALL un peu personnalisé pour exécuter les attachements procéduraux :

```
(DEFUN FUNCALL (LAMBDA FORME (APPLY (CAAR FORME) (CDAR FORME))))
```

****

Imprime tous les pères et tous les fils d'un objet.

```
(DEFUN IMPHERIT (LAMBDA (FRA)
  (PRIN1 "Les Pères sont : ")(PRINT (F-CLASSES FRA))
  (PRIN1 "Les Fils sont : ") (PRINT (F-FILS FRA)) ))))
```

****

HARDEL est une procédure qui détruit de manière physique un atome dans une liste, à tous les niveaux.

```
(DEFUN HARDEL (LAMBDA (AT LST TEMPO)
  ((ATOM LST) NIL)
  ((EQ AT (CAR LST))
    ((ATOM (CDR LST)) NIL)
    (RPLACA LST (CADR LST))
    (RPLACD LST (CDDR LST))
    (HARDEL ATOM LST) )
  (SETQ TEMPO LST)
  (LOOP
    ((ATOM (CDR TEMPO)) LST)
    (((EQ AT (CADR TEMPO))
      (RPLACD TEMPO (CDDR TEMPO)) )
    (POP TEMPO) ) ))))
```

- Annexe -

Gestion des erreurs du système :

```
(DEFUN ERREUR (LAMBDA (N)
 (APPLY (CADR (ASSOC N LSTERR)))))
```

****

LES PROCEDURES APPELEES PAR LES FRAMES

DANS LES ATTACHEMENTS PROCEDURAUX

****

On peut à l'aide de l'éditeur passer des procédures avec des paramètres si les structures des procédures appelées le permettent.

Cette procédure apparaît alors sous la forme :

(Fonction Paramètre1 Paramètre2 ..... ParamètreN)

Il faut savoir que l'évaluation de ces procédures se fera dans l'environnement au moment de l'appel de celles-ci : les variables FRA (nom de la frame), SLO (nom de la propriété), FAC (nom de la facette) et VAL (nom de la valeur) sont les mêmes que celles définies lors de l'appel.

Si les fonctions de SI-AJOUT ramènent T dans la variable ACTION**, elles autorisent l'action d'insertion, si elles ramènent NIL elles l'empêchent.

De même la valeur de la variable DEL** sera enlevée de la frame lors de l'appel de PUT-F-H (en cas de valeur unique, par exemple, il faut détruire l'ancienne valeur).

LES FONCTION DE VERIFICATION DES DOMAINES DE VALIDITES

******* Pour des nombres *******

Appartient :

```
(DEFUN VERIF-LIM (LAMBDA (MIN MAX)
((NOT (TESVAL MIN MAX) (INTERDIT))))))
```

N'appartient pas :

```
(DEFUN NVERIF-LIM (LAMBDA (MIN MAX)
(( (TESVAL MIN MAX)) (INTERDIT))))))
```

******* Pour des atomes ********

Appartient :

```
(DEFUN VERIF-SET (LAMBDA X
  ((MEMBER VAL X))
  (INTERDIT) ))
```

N'appartient pas :

```
(DEFUN NVERIF-SET (LAMBDA X
  ((NOT (MEMBER VAL X)))
  (INTERDIT) ))
```

****

******* Utilitaires *********

****

Demande à l'utilisateur de rentrer une information dans le prototype :

```
(DEFUN DEMANDE (LAMBDA (ATM)
  (PRINT '"Je ne peux pas répondre à cette question")
  (PRIN1 '" Veux-tu donner une valeur (O / N) à : ")
  (PRIN1 FRA) (SPACES 2) (PRIN1 SLO)(SPACES 2)(PRIN1 FAC)
(SETQ ATM (READ)) ((EQ 'O ATM)
  (PUT-F-H FRA SLO FAC (READ)) ))) ))
```

Ajoute d'autorité une information sous la propriété (l'information est passée en paramètre de la fonction AJOUT) :

```
(DEFUN AJOUT (LAMBDA (OBJ)
(TERPRI)
(PRIN1 "Je vais ajouter la valeur : ")
 (PRIN1 OBJ)
 (PRIN1 " à la propriété : ")
(PRIN1 SLO)
 (PRIN1 " pour l'objet : ")
(PRINT FRA)
(PUT-F FRA SLO 'VALEUR OBJ)
(SETQ ACTION** T)))
```

Pour les tests :

```
(DEFUN TESVAL (LAMBDA (MIN MAX)
(((NULL MIN )(SETQ MIN MIN**)))
(((NULL MAX) (SETQ MAX MAX*)))
((AND (GREATERP MAX VAL )(LESSP MIN VAL)) T)
(PRIN1 "Votre Valeur n'est pas adaptée, les limites sont : ")
(PRIN1 MIN) (PRIN1 " ; ") (PRINT MAX)
 'NIL ))))
```

****

Force le système à créer le symétrique de la propriété :

```
(DEFUN SYMETRIQUE (LAMBDA (ATM)
  ((NULL ACTION**))
  ((NULL ATM)
    (PUT-F VAL SLO FAC FRA) )
  (PUT-F VAL ATM FAC FRA) ))
```

****

Demande au système de ne garder qu'une seule valeur de la propriété. Il met la valeur à détruire dans la variable DEL** :

```
(DEFUN VAL-UNIQUE (LAMBDA (TEMPO)
  ((NULL ACTION**))
  ((SETQ TEMPO (GET-F FRA SLO FAC))
    (PRINT " L'ancienne valeur est détruite ")
    (SETQ DEL** (CAR TEMPO)) )
  (SETQ ACTION** T) )))
```

Interdit au système d'effectuer cette opération en mettant à NIL la variable ACTION** :

```
(DEFUN INTERDIT (LAMBDA ()
((NULL ACTION**))
(PRIN1 "Cette modification est interdite à cause de : ")
(PRINT E)
(SETQ ACTION** NIL ))))
```

Teste le bouclage dans les héritages :

```
(DEFUN TESTBOUCLE (LAMBDA (SLO)
  ((MEMBER FRA (F-CLASSES VAL))
    (THROW 'ERR (ERREUR 41)) ) ))
```

```
**********************************************************

      LES DEFINITIONS DES VARIABLES DU PROGRAMME

**********************************************************
```

Liste des aspects autorisés (on peut naturellement en rajouter d'autres) :

```
(SETQ LSTFAC '(VALEUR DEFAUT SI-AJOUT SI-BESOIN SI-DETRUIT))
```

Liste des erreurs :

```
(SETQQ LSTERR
 ((11 (LAMBDA NIL
      (PRINT  "Cette  objet est un type et  est  protegé  en
destruction et écriture")))
  (31 (LAMBDA NIL
      (PRINT  "mauvais type de facette il faut choisir  dans :")
      (PRINT LSTFAC) ))
   (41 (LAMBDA NIL
      (PRINT  "Vous essayez de générer un bouclage dans  les
héritages.")
        (PRIN1 " l'héritage courant est :")
        (PRINT (F-CLASSES VAL))) )) ))
```

Valeurs initiales des variables globales :

MIN* = 0, MAX* = 1000, ACTION** = T, DEL** = NIL.

BIBLIOGRAPHIE

Borow D. G.

"An overview of KRL". In "Cognitive Science", 1977 .

Charniac E.

"Artificial Intelligence Programming". Lawrence Erlbraum Associates Publishers, 1980.

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.

Cayrol M.

"Le Langage LISP", CEPADUES-EDITION, 1983.

Durand J.N.L.

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

Farreny 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, LI2A, 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

- Bibliographie -

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-85

J.P. Goulette & P. Pérez "Tangram, Manuel de l'utilisateur", LI2A, Note interne à paraître.

Mac-Carty J.

"Lisp 1.5 Programmers Manual". The M.I.T. Press Cambridge Mass, 1962.

Minsky M.

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

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.