# Communications entre les systèmes de CAO et les systèmes experts à bases de connaissances en bâtiment dans un environnement d'intelligence artificielle

Georges Sarkis

# THESE

présentée à

## l'ECOLE NATIONALE DES PONTS ET CHAUSSEES

Pour l'obtention du grade de **DOCTEUR de l'ENPC**
spécialité: **Sciences et Techniques du Bâtiment**

par

**Georges SARKIS**

# Communications entre les Systèmes de CAO et les Systèmes Experts à Bases de Connaissances en Bâtiment dans un Environnement d'Intelligence Artificielle

Soutenue le Lundi 10 Février 1992

| | | |
|---|---|---|
| Jury: | MM. Albert DUPAGNE | Président |
| | Jacques RILLING | Rapporteur |
| | Jean MAURETTE | Rapporteur |
| | Louis LARET | Directeur de Thèse |
| | Alain CONSTANS | Examinateur |

*à mes parents*
*Boulos et Amale*

*à mes frères*
*Fadi et Dany*

*à tous ceux que j'aime*

*Je tiens à exprimer tous mes remerciements et ma plus vive gratitude*

*à mes parents pour tous leurs sacrifices*

*à Monsieur Jacques RILLING sans qui la présente thèse n'aurait jamais démarré ni abouti*

*à Monsieur Jean MAURETTE pour avoir accepté la responsabilité d'être rapporteur scientifique de ma thèse*

*à Monsieur Albert DUPAGNE qui me fait l'honneur de présider le jury*

*à Monsieur Alain CONSTANS qui me fait l'honneur de faire partie du jury*

*à Monsieur Louis LARET et Madame Anne-marie DUBOIS avec qui j'ai pu avoir de nombreux échanges scientifiques et amicaux.*

*Je remercie d'autre part toutes les personnes qui ont contribué au bon déroulement de ce travail, et notament*

*Monsieur Michel RUBINSTEIN directeur du CSTB à Sophia Antipolis pour m'avoir admis dans le centre de recherche*

*Madame Nicole BENSOUSSAN pour son aide dans le travail administratif*

*Madame Françoise COUDRET pour son efficacité qui a rendu l'énorme travail bibliographique possible*

*Madame Edith FRUGIER pour avoir facilité le contact avec l'Ecole Nationale des Ponts et Chaussées.*

*Je remercie aussi tous ceux qui travaillent au CSTB à Sophia Antipolis pour leur accueil chaleureux qui a facilité mon intégration dans l'établissement.*

*Je remercie surtout Mme Madeleine BAZALGETTE pour son assistance technique, Mme Linda TROCCAZ pour avoir facilité les communications et Monsieur Thierry MOTTE pour sa serviabilité.*

*Je remercie finalement la direction de recherche de l'ENPC qui m'a accordé la bourse d'étude pour la réalisation de cette thèse.*

# Liste des figures

# Résumé

Les données géométriques sont une partie très importante des informations en bâtiment et leur intégration dans une base de données n'est pas évident.

Le dessin est le langage le plus expressif utilisé par les concepteurs; les systèmes de CAO jouent un rôle très important en étant des interfaces d'utilisateurs très interactives et des outils de saisie graphiques.

La plupart de ces systèmes ont une représentation de données de bas niveau, pauvre en sémantique. Des approches existantes essaient de créer un modèle de données riche en sémantique implémenté sous forme d'une base de données; cette base de données devrait alors être couplée avec la base géométrique du système de CAO.

Comment faut-il faire le lien entre une base de données et un sytème de CAO?
Comment assurer la maitenance et la cohérence dans les deux bases?
Est-il nécessaire d'avoir deux bases de données?

## Types de données

La structure de données pour la description d'un projet de bâtiment est très complexe, et la quantité d'informations concernant un projet est énorme. Un formalisme pour exprimer un modèle standard compatible avec tous les points de vue de tous les acteurs n'est pas simple.

Quelles sont les classes à choisir? Quels types d'attributs et de relations faut-il inclure dans le modèle?

Peut-on penser à un noyau qui ne contient que les composants physiques du bâtiment?

Pour décrire complètement l'environnement d'un projet de bâtiment, il faut gérer des informations de natures différentes:

-des informations statiques valables pour un ensemble de projets (tel que composants, structures de couches,...)

-des informations à caractère réglementaire liées à l'environnement du projet et à l'usage futur du bâtiment.

-des informations à caractère dynamique évoluant du fait des reformulations itératives du projet effectuées par le concepteur ou du fait de l'intervention des modules de calcul ou évaluateurs.

-des informations sémantiques sur le comportement des objets manipulés.

Il faut noter que ce sont les informations à caractère dynamique qui posent le plus de problèmes en termes de représentation, de restitution et de mise à jour pour le maintien de cohérence après les modifications.

## Modeleurs intelligents et données géométriques

En utilisant les ordinateurs, une représentation des données appropriée devrait être utilisée pour la modélisation du projet.

Les informations à caractère dynamique présentées ci-dessus représentent la plus grande partie du projet à modéliser. Elles représentent aussi la partie qui peut être facilement modélisée et saisie à l'aide d'outils de CAO.

Pour cela , on a développé une approche expérimentale basée sur un système de CAO intégré; ce système a une base de données géométrique simple et pauvre en sémantique mais il est équipé d'un ensemble de fonctions d'analyse géométrique et spatiale.

Les éléments géométriques sont des formes simples de base ayant une texture; les textures utilisées sont vide, plein et transparent. Ces textures sont les seules distinctions entre les éléments géométriques de la base du modeleur. Le vide est utilisé pour représenter les espaces intérieurs et extérieurs, le plein pour représenter les enveloppes (murs, plafonds, planchers,...), et le transparent pour les ouvertures.

Aucune autre information sémantiquement plus riche n'est ajoutée. Le projet n'est donc décrit que par un ensemble de volumes (vides, pleins ou transparents) positionnés dans un système d'axes appelé "univers". Ainsi, des informations sémantiques tel que mur, porte, cloison, linteau,... ne sont ni saisies ni mentionnées.

D'autre part, des analyseurs sont implémentés en une couche superposée au noyau géométrique. Ces fonctions sont capables de reconnaitre et d'interpréter différentes notions de haut niveau tel que murs, cloisons, communications,...

Le principe de ces fonctions est de pouvoir décrire comment une information de haut niveau peut être décrite par des informations de bas niveau (i.e. vide, plein ou transparent) et essayer par la suite d'identifier ces descriptions en scannant la base de données géométrique du modèle.

Par exemple, la fonction qui peut identifier l'espace intérieur d'un local, une fois lancée, commence par chercher les éléments de textures "vide" et qui sont contigus. L'ensemble de ces éléments sera par la suite testé pour voir s'il est envloppé par une couche de volumes "plein" ou "transparent". La fonction retourne, si l'espace est fermé, l'ensemble vide ainsi assemblé. La fonction prend comme argument un point de l'espace à partir duquel la recherche commence.

Les algorithmes utilisés sont une version simplifiée pour démontrer la faisabilité de fonctions assez "intelligentes" pour identifier des notions riches en sémantiques du monde architectural à partir de données géométriques simples.

Des méthodes plus complexes et plus efficaces peuvent être utilisées pour améliorer la puissance de résolution de telles fonctions. Le tracé de rayons est un exemple de méthode mathématique et géométrique qui peut scanner efficacement les données géométriques simples à la recherche d'informations de haut niveau utiles pour le raisonnement.

Bien sûr, des attributs simples comme la hauteur ou la surface d'un local peuvent aussi être implémentés par des fonctions qui analysent le modèle géométrique.

On était concerné dans notre étude par une démonstration sur les espaces intérieurs d'un projet. On a donc développé une couche de fonctions au dessus d'un modeleur intégré qui sont capables d'identifier les espaces intérieurs et les enveloppes d'un local, les cloisons de séparation, les parois extérieures, les ouvertures, les communications, et quelques attributs géométriques (surface, hauteur, volume,..).

Le principe de fonctions d'analyse peut être généralisé sur d'autres domaines concernant d'autres acteurs que l'architecte comme l'ingénieur de structure ou le thermicien ou l'acousticien. Ils auront chacun leurs propres fonctions qui peuvent extraire du modèle géométrique les informations qui les intéressent.

## Conclusion

L'intégration de données géométriques de cette façon est simple et efficace. On n'est amené ni à enrichir la base de données du modeleur par des classes d'objets riches en sémantique, ni à construire un autre base de données contenant ces classes pour qu'elle soit couplée plus tard avec l'outil de CAO.

Avec les fonctions d'analyse implémentées, l'identification des informations est faite en appliquant dynamiquement des fonctions sur la base de données géométrique, la mise à jour de ces informations est donc réalisée automatiquement (du moins pour les informations géométriques) du fait de la mise à jour des données graphiques.

On croit aussi que cette approche simplifie l'utilisation de systèmes de CAO, car la description du projet revient à décrire la volumétrie et les entités spatiales. Le concepteur n'a pas à s'occuper à saisir des entités sémantiques tel que murs, cloisons, ou des notions plus complexes tel que coins, intersections, ou même des notions spécialisées tel que linteau ou ponts thermiques,...

Chaque acteur aura ses propres analyseurs capables d'identifier dynamiquement à l'intérieur des données gémétriques simples les informations de haut niveau dont il a besoin.

Cette méthode assure aussi la modularité et l'extensibilité du système.

Le plus grand inconvénient d'un système tel que décrit avant, reste le temps de calcul des fonctions (calcul d'intersection, d'adjacence, dinclusion,...). On croit cependant que ce facteur peut être amélioré en optimisant les algorithmes et les codes sources, et en utilisant des machines performantes.

# Chapitre 1

# Introduction

## 1.1 Cadre général

Les outils informatiques commencent à jouer un rôle important dans le travail des entreprises en apportant un gain de productivité surtout dans les domaines des tâches mécaniques et répétitives.

Les outils de CAO et de DAO font partie de ces outils qui ont permis de faire évoluer certains aspects du travail dans les entreprises. On a vu ainsi une automatisation des tâches de dessin: on passe des tables à dessin, de l'équerre, et des crayons, à l'utilisation des écrans, de la souris, des tablettes à digitaliser et des traceurs.

Cet apport reste cependant limité à la phase de production de plans ou de présentation du projet.

D'autre part les percées récentes de l'informatique, notament en intelligence artificielle ont permis de mettre en oeuvre des systèmes qui sont capables de représenter une connaissance ou une méthodologie exploitable dans un travail de conception.

En bâtiment, le problème de conception est pluridisciplinaire; plusieurs experts (architectes, ingénieurs, techniciens, acousticiens...) doivent coopérer et négocier afin de produire un résultat cohérent.

Notre étude consiste à définir un outil d'aide à la conception multi-acteurs en bâtiment, où l'a machine jouera un rôle non seulement dans les phases finales du projet, mais aussi dès la première phase de spécifaction et de conception.

On utilisera pour cela un environnement avancé d'intelligence artificielle alliant une représentation orientée objet à des mécanismes d'organisation et de contrôle du raisonnement.

Le travail portera surtout sur le couplage de systèmes experts à bases de connaissances en bâtiment (pour le raisonnement) avec des outils de CAO (pour la saisie et l'interaction graphique).

Ce travail de thèse consistera en une contribution à la spécification des éléments de base d'un prototype de cet outil. La spécification sera afinée à l'aide d'une maquette logicielle.

## 1.2 Description du problème

Un environnement d'intelligence artificielle en informatique impose une représentation adéquate des données, des informations et de la connaissance liées à un problème donné pour pouvoir apporter ou proposer des solutions et de les évaluer.

On essaiera dans notre thèse de définir la structure et l'architecture de modèles de données et de modèles de connaissances dans le domaine du bâtiment.

Les informations et les données manipulées sont de types variés, mais les informations géométriques prennent un poids considérable; ceci nous amène à définir le rôle que devrait jouer un outil de CAO dans un environnement d'aide à la conception surtout que le dessin est un moyen de simulation des problèmes et des solutions; c'est à la fois pensée opératoire et modèle réduit d'un espace concret pour le partag e des l'information.

Les outils de CAO existants achoppent sur différents aspects, tel qu'on le verra dans les chapitres suivants. On essaiera alors de définir le prototype de ce qu'on qualifiera d'outils de CAO intelligents.

On essaiera par la suite de les coupler à des systèmes experts qui eux modéliseront les connaissances et les méthodologies de conception et d'évaluation d'un projet.

## 1.3 Plan et avancement de la thèse

Dans une première phase, on a étudié l'état de l'art des différentes techniques et outils utilisés dans l'environnement prévu.

On a exploré le rôle des outils de CAO dans la représentation des données, les approches de modélisation, et les échanges de données. Cette étude est basée sur des outils existants, pour lesquels on a essayé d'exposer les limites de leurs états actuels, et les orientations de la recherche pour les outils du futur dits "intelligents".

Puis on a approfondi la connaissance des différents environnements et techniques de l'intelligence artificielle pour la représentation de la connaissance, le raisonnement et le stockage de données; on a essayé d'évaluer certains modèles utilisés dans des problèmes de conception. On a noté de même les limites des approches existantes pour l'application qu'on envisage ainsi que les évolutions déjà proposées.

La deuxième phase qui forme le coeur de la thèse, est une analyse théorique de l'approche que l'on propose; on a détaillé le rôle de l'outil de CAO dans la représentation des données, celui des systèmes experts et des bases de connaissances pour les problèmes de conception ; et finalement le moyen de couplage des deux univers.

Notre travail de recherche ne se limite pas à la spécification d'un outil mais aussi porte sur la réalisation d'une maquette logicielle. Ainsi, en parallèle de notre étude, on a essayé d'implémenter les idées théoriques proposées pour prouver leur faisabilité et leur originalité. Ceci nous a permis d'affiner le modèle de notre étude et d'étayer certaines idées.

Une démonstration présente l'outil graphique réalisé, les systèmes experts implémentés et le couplage des deux environnements dans le domaine de la conception en architecture des espaces intérieurs.

## 1.4 Déroulement de la thèse

Le choix de l'environnement de développement s'est porté sur un système multi-expert SMECI. Celui-ci est basé sur un langage orienté objet pour la représentation de la base de faits ; à ces objets sont associés des méthodes écrites en LE-LISP et des bases de règles pour le raisonnement écrites en langage SMECI, proche du langage naturel.

Le travail est matérialisé sous la forme d'une maquette logicielle où sont implémentés des exemples simples de conception relatifs au bâtiment (saisie, stockage et restitution d'informations, création et positionnement d'un local en fonction de contraintes,...) qui mettent en oeuvre les idées proposées.

## 1.5 Le présent document

La thèse compte sept chapitres:

I - Introduction
II - CAO et architecture
III - Conception et IA
IV - Couplage CAO-IA
V - Réalisation logicielle
VI - Applications
VII - Conclusions

Ces chapitres essaient d'expliciter un certain nombre d'idées acquises, déduites et proposées; Le problème de conception étant très vaste et général, la thèse ne prétend pas apporter des réponses à tous les aspects. Certains problèmes ne seront pas traaités de façon détaillée car ils nécessiteront à eux seuls un travail de thèse complet.

On s'est contenté alors de souligner ces problèmes, de donner quelques détails en vue de montrer l'intérêt et la généralité de notre approche en la considérant comme une base solide permettant une extension et une intégration de nouveaux concepts assz facilement.

Ce document présente aussi une partie de la bibliographie sur laquelle était basée notre étude.

# Chapitre 2

# CAO et architecture

Dans ce chapitre, nous étudierons tout particulièrement le problème du dessin et des systèmes de CAO. Pour cela nous commencerons par une partie qui décrit les méthodes classiques et le rôle du dessin avant l'informatisation; puis nous étudierons plus en profondeur les systèmes de CAO et leurs utilisations en architecture; nous analyserons et évaluerons les apports et les limites de ces systèmes. Nous exposerons en bref les essais et les orientations des recherches qui sont mis en oeuvre pour faire évoluer ces systèmes de CAO vers des systèmes plus "intelligents".

## 2.1 Avant la CAO les méthodes classiques

### 2.1.1 introduction

La perception du monde par les hommes est une tâche intelligente. La capacité d'exprimer au moins une portion de cette perception est essentiel pour notre appréciation de la connaissance des gens et leur aptitude à partager et à devélopper cette connaissance.

### 2.1.2 Rôle du dessin et des outils graphiques

La manifestation de la connaissance des gens dépend de la structure des expressions; l'arithmétique est un formalisme familier pour exprimer la connaissance, le texte l'est aussi en termes de structures formelles et les dessins qui sont en général informels.[BIJL A., 86]

La conception s'exprime au moyen d'un outil fonctionnel que nous appelons "dessin".
Le croquis, l'épure, le rendu apparaissent comme la forme écrite du langage de conception architecturale.[QUINTRAND P. 85]

Pour la résolution des problèmes, une coordination intense est nécessaire; elle concerne différents niveaux de l'activité cognitive [LEBAHAR J.C. 85]:

-coordination de points de vues
-coordination dimensionnelle
-coordination logique des conditions et des hypothèses

Elle concerne aussi différents niveaux de méthodes:

-coordination plan, coupe, axonométrie,...
-trames
-catalogues techniques

-connaissance en construction

Le dénominateur commun de toutes ces formes de coordination est "l'espace".

Il est nécessaire de représenter spatialement un objet architectural pour qu'il existe. L'architecte utilise les plans pour simuler, fixer et communiquer sa représentation de l'objet architectural. Le dessin permet ainsi la production de "modèles réduits" homomorphes au système réel qu'ils représentent.

Le dessin est un moyen de simulation des problèmes et des solutions; c'est un système privilégié de traitement de l'information spatiale. Cette adéquation de l'expression graphique aux problèmes que se pose l'architecte pendant le projet provient principalement du statut même de cette expression: elle est à la fois pensée opératoire et modèle réduit d'un espace concret [QUINTRAND P. 85].

Le dessin est le moyen du concepteur pour partager l'information. Par suite, la compréhension du dessin parait nécessaire pour le devéloppement de la théorie de la conception [OXMAN R. et AL 91].

Le dessin exprime les divers états de la représentation du problème depuis un état initial d'une représentation partielle à un état final permettant une interprétation en terme matériel sur un chantier.

### 2.1.3 Résolution graphique et représentation

La simulation graphique est le fruit de l'interaction entre l'architecte et son problème de conception; le travail de l'architecte consiste en une approche spatiale selon divers points de vue: en plan, en coupe ou en élévation; chaque point de vue ajoutant de l'information aux autres [LEBAHAR J.C. 83].

Le plan est presque toujours l'espace de référence et le lien permettant à l'architecte de préciser les diverses parties de l'objet à concevoir; l'architecte s'appuie souvent sur des outils opératoires tel que axes et trames pour délimiter des points de références.

En architecture, la représentation en deux dimensions correspond aux projections sur des plans horizontaux ou verticaux pour obtenir les plans, coupes et façades. Les projections ne sont pas de simples coupes de l'objet réel mais une représentation schématique avec des conventions plus ou moins précises; le dessin n'est pas une simple traduction graphique de volumes ou d'entités géométriques; le modèle doit être interprété non seulement du point de vue géométrique mais aussi fonctionnel, relationnel ... [LEBAHAR J.C. 85]

Les plans s'apparentent à des schémas de représentation mais des schémas qui se veulent réalistes.

### 2.1.4 Le rôle du dessin dans la communication

L'établissement des plans décrivant le bâtiment permettent le dialogue entre l'architecte et son client et l'architecte et ses collaborateurs (plans côtés, coffrages, feraillages, ...)

L'établissement du modèle de construction sera caractérisé par l'encodage et le décodage des plans d'architectes visant à lever toute ambiguité dans l'interprétation des plans.

La communication est d'autant plus favorisée que le code graphique est clair, que le modèle graphique est analogue à l'objet, et que l'ambiguité est résolue par recoupement d'informations graphiques redondantes.

### 2.1.5 Conclusion

Le dessin joue un rôle important dans le processus de conception architecturale. C'est un outil de résolution de problèmes et de représentation pour des fins de communication.

L'architecte progresse dans son problème de la forme initiale esquissée au dessin final restituant la solution du problème posé en des termes adéquats au besoin de sa réalisation concrète sur le chantier.

De là l'informatisation de ce processus, par la réalisation de systèmes de CAO devrait garantir l'interaction entre le concepteur et son projet tout au long du processus de conception.

## 2.2 La CAO en architecture

### 2.2.1 Introduction

Un système de CAO est efficace s'il permet la construction d'une maquette numérique de l'objet architectural en cours de conception depuis l'esquisse aux plans d'exécution tout en facilitant les conditions d'accès à l'outil tant du point de vue des opérations de manipulation que du langage du spécaliste pour l'interaction homme / machine lors de la manipulation des données morphologiques du projet et lors de la restitution des informations sous forme de dessins et d'images pour la simulation et la communication.

### 2.2.2 La modélisation géomètrique

La modélisation est le passage de l'objet concret réel à une représentation interne dans la machine sous une forme compréhensible du point de vue informatique [QUINTRAND P.85].

Les outils traditionnels de CAO ont été devéloppés initialement pour aider le dessinateur. Les structures de données utilisées pour le modèle sont souvent celles utilisées pour la visiualisation à l'écran (points, lignes, arcs,...) [WOODBURRY R. 91], [OXMAN R. et AL 87].

Par suite la modélisation devient restreinte à la représentation des objets du point de vue de leurs propriétés géomètriques et non pas du point de vue de leurs propriétés relationnelles ou fonctionnelles. Seule la forme des objets est prise en compte.

Cette modélisation peut être bidimensionelle ou tridimesionnelle [GARDAN Y. SOHNOUNE M. 90], [PEROCHE B. et AL 90].

### 2.2.2.1 La modélisation bidimensionelle

Sous cette forme l'ordinateur ne connait que des vues planes de l'objet. Ainsi le contrôle de cohérence entre un plan et la façade correspondante n'est pas géré par l'outil.

Ces outils permettent de manipuler des points, lignes, segments de droites, polygones, cercles ou autres entités planes, permettant de construire des entités 2D plus complexes.

Ce genre de modeleurs est surtout utilisé comme outil de DAO, pour le dessin et le rendu et de façon moins efficace en conception.

### 2.2.2.2 La modélisation tridimensionelle

C'est le modèle qui nous interesse le plus dans notre étude; dans ce cas l'objet est représenté dans ces trois dimensions. Cette approche est plus réelle vu que les objets manipulés par les architectes sont essentiellement tridimensionnels.

on distingue trois types de modélisation tridimensionelle:

**Modèle en fil de fer**

le système ne conserve de l'objet modélisé que les sommets et les arêtes. La face n'est pas connue. Les objets ainsi modélisés sont transparents et ne permettent pas l'effacement des lignes cachés. De même il est impossible de calculer l'interpénétration de deux volumes. Il n'y a aucune identification du vide et du plein.

C'est un modèle ambigu avec risques de non sens.

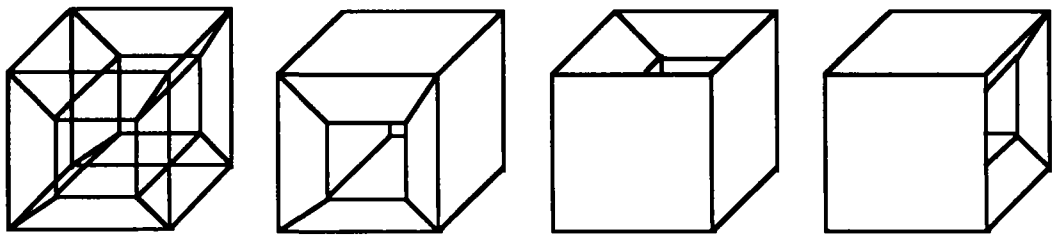


*fig. 2.1 plusieurs interprétations possibles pour un même modèle en fil de fer*

**Modèle surfacique**

Ce modèle permet de définir des surfaces complexes: surfaces de révolution, surfaces de Bézier, B-spline... Les calculs géométriques sont toujours difficiles. Le volume n'est pas défini car le système ne connait que les surfaces sans savoir où se trouve la matière.

Ce modèle est intéressant dans les applications où seule la surface est importante (conception de formes, commande numérique ...).

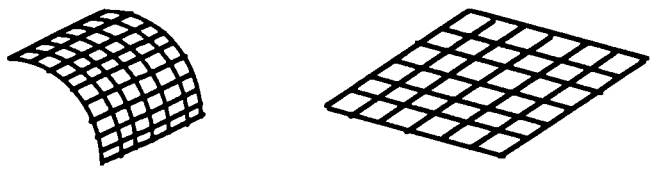


*fig. 2.2 surfaces de type z = f(x, y)*

**Modèle solide**

Les outils de CAO qui manipulent ce modèle offrent des primitives de création de volumes et sont appelés modeleurs volumiques. Le système connait dans ce cas l'objet totalement. Il pourra traiter correctement les parties cachées. Il permet la modélisation d'objets complexes, reconnait l'intérieur et l'extérieur de l'objet, les notions de matière (plein et vide). Il permet le calcul de grandeurs géométriques et les opérations booléennes sur ces volumes: union, intersection, soustraction.

Les modèles solides utilisés couramment en CAO sont les suivants:

- le modèle par les limites
- le modèle par arbre de construction

Ces différents modèles sont basés sur les notions suivantes:

- **le modèle par les limites** (ou B-Rep: Boundary Representation): dans ce modèle le système conserve la peau de l'objet et sait de quel côté est la matière. Un tel modèle comprend en général des informations géométriques (coordonnées des sommets, équations des faces ..), des informations topologiques (façon dont sont reliées les informations géométriques) et des informations annexes (couleurs des faces).

Il suffit par artifice d'orienter les sens de parcours des contours limitant les faces pour distinguer intérieur et extérieur.

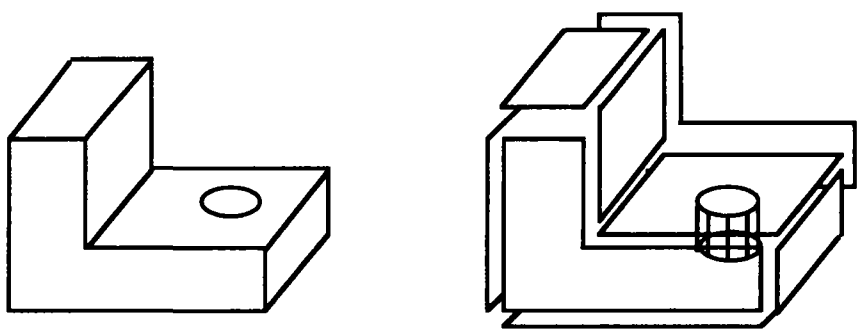


*fig. 2.3 solide décomposé en facette (BREP)*

- **le modèle par arbre de construction** (CSG: Constructive Solid Geometry): ce modèle est appelé ainsi parce qu'il peut être représenté par un arbre. En général, on trouve aux feuilles de l'arbre des objets primitifs paramétrables et aux noeuds des opérations. A chaque noeud correspond un objet,

même si celui-ci n'est pas réellement calculé; En fait plutôt que de parler d'arbre de construction on peut parler de "conservation de l'historique". Il y a une volonté de conserver une information "générique". Ils sont assez souvent limités aux opérations boléennes et ne prennent pas forcément en compte tous les types d'objets.

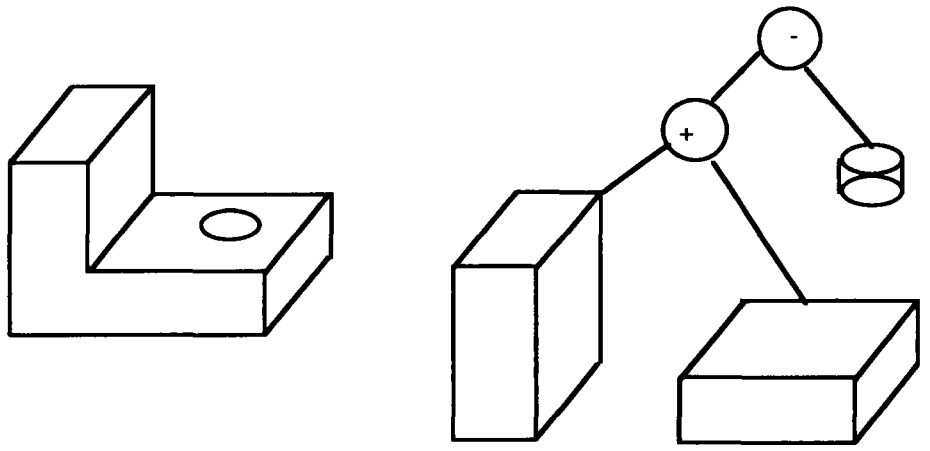



*fig. 2.4 solide décomposé en volumes élémentaires (CSG)*

Un arbre de construction peut avoir pour feuilles des demi-espaces: un demi-espace est défini par une relation qui partitionne l'espace en deux demi-espaces possédant une frontière commune:

$$D = \{ x, y, z \,/\, f(x, y, z) >= 0\}$$

L'opération fondamentale, qui consiste à établir l'appartenance d'un point à un solide, est alors extrêmement simple au niveau des feuilles, dans la mesure où il suffit de reporter les coordonnées du point dans l'équation f(x, y, z): si la valeur obtenue est positive, le point appartient au demi-espace considéré; sinon, le point appartient au complémentaire du demi-espace considéré. On peut également décrire facilement les objets de base tels que parallélépipèdes, cylindres, cônes ... à l'aide d'intersections booléennes de demi-espaces.

-**L'approche spatiale**: c'est un modèle très peu utilisé dans les systèmes industriels; il s'agit de diviser l'espace en "petits" éléments et de spécifier pour un solide donné, si chaque élément est rempli ou non. Pour atteindre une bonne précision, il est nécessaire de diviser l'espace en éléments suffisament petits.

La modélisation par arbres octaux (Octree) permet de minimiser le nombre d'informations à conserver.Elle consiste à considérer des divisions de l'espace global en huit, puis chacun des sous-espaces en huit récursivement jusqu'à ce que l'une des conditions suivantes soit remplie:

- le sous-espace considéré ne contient aucune partie de l'objet à modéliser: il est vide et il est donc inutile de continuer à le subdiviser.
- le sous-espace considéré est complètement contenu dans l'objet modélisé: il est plein et il est donc inutile de continuer à le subdiviser.
- le sous-espace considéré est de la taille minimale admise ( il correspond au petit élément): il est soit complètement dans l'objet (il est plein), soit

complètement en dehors de l'objet (il est vide), soit partiellement dans l'objet; dans ce dernier cas, on peut appliquer plusieur stratégies, par exemple, décider que si le volume du sous-espace considéré compris dans l'objet est supérieur à la moitié du volume total du sous espace, il est traité comme s'il était plein (sinon, il est vide).

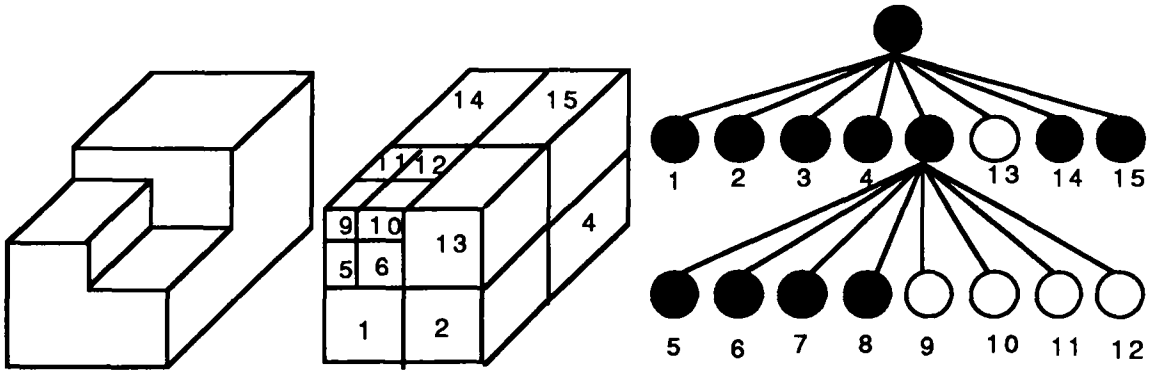



*fig. 2.5 solide représenté par un arbre octal (OCTREE)*

La modélisation spatiale est souvent utilisée pour faciliter et accélérer des algorithmes (arbres octaux, volumes englobants ...).

### 2.2.3 Fonctions graphiques

Les logiciels dits de CAO sont généralement dotés de divers capacités de traitements:

-création d'objets
-édition d'objets existants
-transformations (translation, rotations...)
-changement d'échelle
-zoom
-visualisation en perspective
-lignes cachées, rendu

A partir des données graphiques communiquées à l'ordinateur, le système permet d'éditer, avec plus ou moins de facilité les parties graphiques (plans, façades, coupes, perspectives...) [WEEK D. 91].

La phase de saisie étant toujours la plus difficile, elle condamne l'utilisation professionnelle courante en conception, et limite l'utilisation de ces outils à des pratiques de saisie et de dessin une fois la conception prête.

### 2.2.4 Représentation des données

Le modèle ou la maquette numérique virtuelle qui est la représentation informatique du projet dans la machine ne tient compte, en général, que des informations géométriques: coordonnées de points, topologie des formes représentées... mais ne tient pas compte des relations entre les différents objets [QUINTRAND P. 85].

on peut distinguer deux types de relations:

- relations de liaisons: un composant peut être lié à un autre par différentes relations; par exemple un sanitaire est lié à un mur pour permettre le raccordement aux arrivées d'eau et à l'évacuation.

- relations hiérarchiques: un objet peut être décomposé en sous objets, par exemple, une porte peut être décomposé en une partie fixe et une partie mobile; la partie mobile elle-même est décomposable en constituants élémentaires...

### 2.2.5 Manipulation de données sémantiques

Dans la plupart des systèmes de CAO existants, les notions non géomètriques ne sont pas traitées directement (caractéristiques technologiques, relations entre objets,...)
Ces informations peuvent être qualitatives (types de matériaux) ou des informations quantitatives (nombre d'éléments de tel ou tel type) ou des informations sémantiques ou relationnelles (les parois de la cuisine, la cuisine est adjacente au séjour...)

La maquette numérique du projet dans le système informatique ne contient que des informations géométriques et ne tient pas compte des informations structurelles et sémantiques.

Certains outils essaient de remédier à ce manque en associant à des entités géométriques des informations non géométriques sous forme d'attributs de type de chaine de caractères [TROUSSE B. 89].

### 2.2.6 Approches de modélisation

En architecture la modélisation contient:

- des espaces architecturaux
- des éléments constructifs qui composent le bâtiment

L'espace architectural est formé de vide limité par des plans . Par contre les éléments constructifs sont des volumétries pleines assemblées et qui peuvent être très complexes formées par fusion, différence ou intersection de volumétries simples élémentaires.

L'espace architectural ne peut être conçu commme un énorme volume de matière dans lequel on percerait des galeries et des pièces, mais plutôt comme un espace vide que l'on partitionne.

D'autre part, la prise en compte de modèles géométriques complexes est encore très difficile (pièces composées d'objets simples ou complexes et de sufaces gauches).

En général la démarche conceptuelle fait appel à la décomposition hiérarchique et la structuration arborescente des éléments constitutifs du projet. Par suite on peut distinguer deux types de démarches pour la modélisation [QUINTRAND P. 85]:

- la démarche ascendante: elle consiste en l'élaboration à partir des constituants physiques élémentaires des blocs plus importants pour aboutir aux objets complexes souhaités

-la démarche descendante: ou analytique qui consiste à définir des spécifications, niveau par niveau, en partant de l'objet et en descendant vers les constituants élémentaires.
cette démarche permet de réduire la complexité du problème et de pouvoir travailler avec des informations incomplètes et enfin d'avoir la possibilité de décomposer le problème en sous problèmes plus élémentaires.

Souvent ces deux démarches sont imbriquées; certains logiciels de CAO fonctionnent suivant la démarche ascendante, d'autres suivant la démarche descendante, mais la méthodologie mixte reste presque impossible.

Ainsi en démarche ascendante l'objet complexe généré n'a aucune existence qui permette de l'exploiter. D'autre part, dans la démarche descendante, les composants n'existent pas en dehors d'une représentation graphique.

**Exemple**

Supposons que l'on effectue la saisie d'un local suivant une démarche descendante c'est à dire on saisit le volume du local puis plus tard on définit l'espace intérieur par soustraction du premier volume:

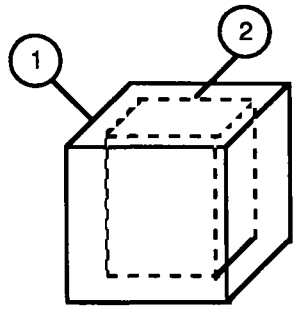



*fig. 2.6 saisie d'un local par soustraction d'un volume vide à un volume plein*

En CSG, le résultat est stocké en deux volumes, et en BREP on a les facettes des deux volumes.

Le volume saisi représente non seulement le volume du local, mais aussi celui des murs, du plafond,...

Dans ce cas, aucune des deux représentations ne permet d'avoir des informations sur les composants implicites du local saisi.

Supposons maintenant que la saisie s'effectue par assemblage, en assemblant les différentes parois du local:

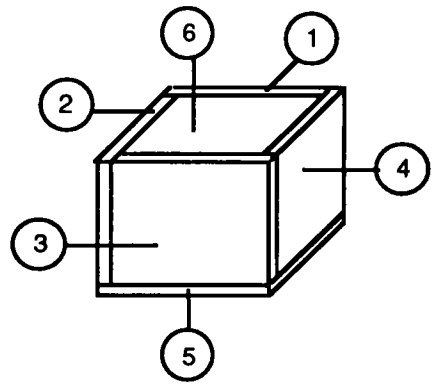



*fig. 2.7 saisie d'un local par assemblage des parois*

En CSG, on aura six volumes, et en BREP on aura les facettes des six volumes.
Dans les deux cas le local en lui-même n'existe pas sous une forme exploitable pour pouvoir en tirer des informations tel que surface, hauteur,...

### 2.2.7 Création des versions ou historique

L'architecte est amené souvent à choisir parmi différentes alternatives sans savoir à priori quelle sera la meilleure solution et éventuellement à revenir sur son choix.
Par suite l'élaboration et le stockage de versions différentes ou de historiques de construction sont souvent facilités; mais il n'y a aucune assistance de la part du système pour vérifier la qualité d'une solution, ou sa conformité à certaines règles ou contraintes.

### 2.2.8 Echange de données

Les logiciels de CAO proposent des algorithmes pour convertir leur représentation interne des entités CAO dans des représentations normalisées (I.G.E.S., S.E.T...) ou devenues standard par l'utilisation (D.X.F.) pour faciliter les échanges avec d'autres logiciels utilisés par des personnes différentes.

## 2.3 Logiciels existants

### 2.3.1 CAO et saisie graphique

Il existe des outils de CAO à fonctionnalités différentes, et leur performance ne peut s'analyser qu'au regard des utilisations attendues et par rapport à un cadre économique d'exploitation; tel logiciel pourrait s'avérer très performant pour les uns, et totalement inadapté pour les autres.

Cependant on peut dire qu'aucun des systèmes n'est adapté aux premières phases de conception du projet, là où la mesure est encore incertaine; par contre tous apportent une aide plus ou moins importante à l'instrumentation du projet donc au dessin et au rendu final sous différentes formes (plans d'exécution, vues perspectives de démonstration...).

### 2.3.2 Utilitaires couplés ou intégrés

Les domaines d'application des utilitaires couplés ou intégrés sont très variés, même si certains secteurs sont très mal couverts au bénéfice d'autres secteurs surchargés [EASTMAN C. 91].

On peut trouver des logiciels de calcul de structures (béton armé, charpente métallique ou en bois, structures tendues,...), des logiciels d'estimation, de devis et métré, de thermiques ...

En thermique, par exemple, on peut trouver des logiciels permettant le calcul du coefficient K, ou du coefficient G, ou B, des apports solaires et des déperditions.

Certains de ces logiciels sont indépendants du système de CAO et offrent un moyen d'être couplés avec celui-ci pour récupérer des données en paramètres de sortie, d'autres contiennent eux-mêmes leur propre outil de saisie graphique ou bien sont intégrés dans un système de CAO déjà existant [CARADANT D. 87].

### 2.3.3 Limites de ces utilitaires

un défaut commun à tous ces logiciels est qu'ils sont souvent très différents, et chacun ayant sa propre représentation interne du projet (fichiers ASCII, modèle géométrique...), et aussi sa propre philosophie de saisie ou de décomposition du projet.

L'utilisateur ayant introduit les éléments de son volume pour dessiner une perspective, il devra les entrer une nouvelle fois, et peut être sous une autre forme, pour le calcul du coefficient G; la normalisation des échanges entre programmes d'origines différents n'est pas bien définie.

Pour les outils de CAO qui intégrent des utilitaires de calcul, ceux-ci ne seront actionnés qu'après avoir terminé la phase de conception, et l'efficacité de tels produits est contestable surtout au niveau de l'interaction entre l'utilisateur ou le concepteur et le système ainsi que l'interaction du module de calcul et du système de CAO; souvent le système se contente d'évaluer et de calculer et les propositions de modifications restent très limitées.

### 2.3.4 Limites des logiciels de CAO

- les outils de CAO sont des outils géométriques très sophistiqués et performants, cependant ils sont parfois d'usage compliqué et n'offrent pas toutes les fonctionnalités souhaitées dans la phase de conception.

- l'entrée des données est une tâche laborieuse pour la description des objets tridimensionnels. Cette phase est coûteuse en temps et propice aux erreurs du fait de l'interaction fréquente de l'utilisateur.

- la prise en compte de modèles géométriques complexes reste encore difficile.

- les notions non géométriques ne sont pas traités directement. On ne peut considérer les modèles actuels comme des maquettes virtuelles; la gestion des informations non géométriques est à un stade primaire non évolutif. Il faut pouvoir associer aux éléments géométriques des informations annexes et les utiliser pour des recherches, des tris, ou des éditions.

- les modeleurs n'ont aucune connaissance explicite de concepts fondamentaux (parois d'un local, locaux communiquants..), ou de concepts de relations ou contraintes (être adjacent à, être parallèle à...); ces relations ne peuvent pas être spécifiées explicitement par l'utilisateur.

- la gestion de la cohérence est insuffisante; par exemple, un élément de bibliothèque peut être formé par d'autres éléments de bibliothèque, l'utilisateur doit restaurer la cohérence manuellement lors de la modification d'un constituant.

- Il est impossibile de mettre en oeuvre une méthodologie de conception mixte (démarche ascendante et démarche descendante); la transformation dynamique de la structure d'un assemblage est souvent impossible avec les outils de CAO classiques alors que cela est souvent indispensable.

- il n'y a aucune assistance de la part du système pour vérifier la qualité d'une solution, ou sa conformité à certaines règles ou contraintes.

- certains logiciels sont fermés et par suite très rigides dans leur évolution et leur utilisation; il faut en effet avoir des langages de commandes qui à l'insu de langages de programmation permettent le pilotage de ces logiciels de l'extérieur, pour extraire, ajouter ou éditer les informations sans avoir à passer par l'interface du logiciel.

- certains produits sont difficiles d'emploi par manque de procédures interactives, et les interfaces Homme/Machine sont très peu évoluées.

En réalité, l'analyse des systèmes de CAO mis à la diposition des architectes montre qu'ils traitent pour la plupart uniquement de dessin assisté par ordinateur; ils ne s'interessent qu'au phase de rendu, de production de plans, de pièces écrites, perspectives de présentation, mais tout ce qui a rapport à la conception reste très élémentaire.

## 2.4 Vers une CAO intelligente

### 2.4.1 Evolution des recherches en CAO

Les environnements de CAO des futures générations pourront être manipulés et guidés par des concepts de plus haut niveau. Les principales recherches en CAO visent essentiellement à offrir une modélisation plus riche des entités de CAO manipulées.

- l'intégration de représentations multiples
- la conception paramétrique et variationnelle
- l'intégration du modèle objet

Nous détaillerons par la suite chacun de ces aspects; par contre, les approches de systèmes intégrés où les systèmes de CAO empruntent des principes à l'intelligence artificielle et aux systèmes de gestion de bases de données ne seront abordées que dans le chapitre suivant.

#### 2.4.1.1 L'intégration de représentations multiples

Les paquets de programmes dédiés à différentes applications avec des aspects fonctionnels différents sont en pleine évolution: on peut citer les programmes pour l'éclairage, l'analyse d'énergie, l'acoustique, le terrassement, l'aménagement, le design d'espaces intérieurs... [CAMARATA S. et AL 84]

Chacun de ces programmes a besoin de ses propres données en entrée et génère ses propres données en sortie.

Plus le nombre de représentations nécessaires est grand, plus le temps de conception est plus grand. De même la coordination entre différentes représentations est de la responsabilité du concepteur.

Avec des méthodes de conception courante, on définit les éléments et on les compose dans les dimensions multiples de leur interaction (géométrique, structurelle, électrique, acoustique, ...) en utilisant différentes représentations [EASTMAN C. 91].

Les éléments sont quand à eux définis dans des détails variés; en première phase de conception, des éléments simples sont utilisés, qui seront plus tard détaillés et décomposés en éléments multiples.

Par exemple, les murs sont définis à certaines échelles et étapes de la conception comme des éléments monolithiques, et plus tard, ils seront définis par des assemblages de composants.

Dans les systèmes de CAO actuels, chaque représentation d'un élément est définie et gérée séparement par le concepteur; ceci au frais de grands efforts de traduction et de coordination. on peut effectivement se douter du coût de la coordination des changements dans les phases finales.

Des essais sur une autre génération de modeleurs de bâtiment sont en cours; cette génération est différentes des systèmes de CAO conventionnels, et fait intervenir parfois des pratiques qui ne sont pas les mêmes que les méthodes manuelles de conception.

La conception se fait par l'assemblage d'ensembles d'objets ( à ne pas confondre avec la programmation objet). Un objet contient la description d'une entité physique, le modèle 3D, différentes propiétés de dessin et de matériaux, et contient aussi d'autres descriptions alternatives de la partie nécessaire pour les différentes représentations.

par exemple une porte est un objet qui peut avoir plusieurs sortes de représentations graphiques dépendant du contexte de dessin:

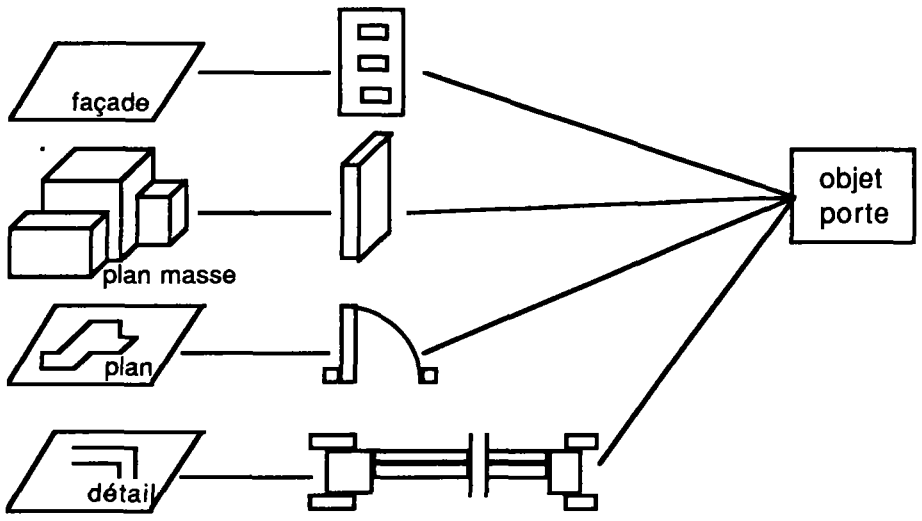



*fig. 2.8 exemple de représentations multiples d'une porte*

Un tel modeleur de bâtiment a deux structures hiérarchiques principales:

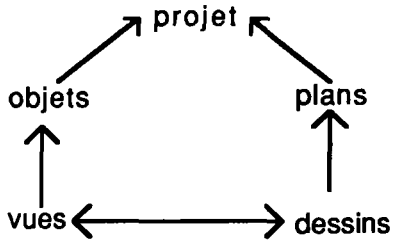



*fig. 2.9 structure hiérarchique d'un projet*

D'un côté un projet est la composition d'objets avec des aspects multiples, de l'autre côté, on a une collection de plans et de vues qui représentent le projet.

Certains outils de CAO nouveaux permettent la modélisation par objets. Ils donnent la possibilité de gérer différentes vues d'un même objet, qui diffèrent beaucoup au niveau des détails.
Cette génération de système de CAO permet un niveau d'automatisation nouveau;

On peut toutefois essayer de s'approcher de ces concepts de manipulation de représentation et d'objets moyennant certaines astuces comme l'utilisation de calques (layers) ou l'utilisation d'éléments de bibliothèques mais cela n'est pas très simple à gérer et présente un supplément de travail et une méthode d'utilisation sophistiquée.

### 2.4.1.2 La conception paramètrique et variationnelle

Jusqu'à présent, les outils de CAO tridimensionnels, permettent de construire des modèles géométriques de dimensions données (formes et cotes) et de créer entre eux des relations spécifiques (de contact, d'assemblage...).

L'utilisateur qui aborde la conception préliminaire d'un nouveau projet ne sait pas toujours avec certitude quelle configuration géométrique conviendra. Pendant cette phase dynamique de la conception les outils de modifications qu'offrent les systèmes actuels ne lui procurent pas la souplesse suffisante.

D'autre part, les systèmes n'associent pas automatiquement les contraintes d'ingénierie avec les géométries (parallèle, perpendiculaire,...).

De même, pour concevoir un nouveu modèle, on peut réutiliser les données d'un ancien, et faire des révisions incrémentales.

Ce contexte a engendré les méthodes de CAO paramétriques et variationnelles qui sont pour la plupart appliquées en CAO mécanique mais peuvent être adaptées à la CAO en bâtiment [GARDAN Y. SINGER D. et AL 90], [COLETTE P. 87], [LIGHT R.A. et AL 82].

Ces systèmes définissent des relations entre les variables comme des équations qui doivent être satisfaites. Des changements de ces variables entrainent toute une série de modifications de sorte que l'ensemble d'équations reste satisfait.

Ces nouveaux logiciels ont un point en commun: la capacité de mise à jour automatique d'une forme géomètrique lorsqu'on modifie ses cotes à l'écran.

Pour ce type de travail, le concepteur spécifie d'entrée les données de sa conception en termes de contraintes géométriques (parallèle, perpendiculaire, angles, tangences,...) et de relations (adjacent, àdroite,...) exprimées dans le logiciel par des équations.

Le système maintiendra, chaque fois que l'on modifie un des paramètres, les relations imposées. Pour cela il est équipé d'un moteur d'inférence géométrique (technique emprunté à l'intelligence artificielle).

Entre conception paramètrique et variationnelle la différence est la suivante:

- avec un système paramètrique, le concepteur dispose en librairie d'un ensemble prédéfini et limité de contraintes géométriques qu'il pourra combiner et appliquer aux formes dessinées à l'écran. Le concepteur doit savoir à l'avance les données et les interrelations qu'ultérieurement il veut modifier.

- avec un système variationnel, le concepteur n'a pas à spéculer sur les contraintes géométriques et leurs combinaisons; il peut les modifier et en ajouter en cours de travail. Le système s'accomode du couplage des contraintes géométriques avec les équations d'ingénierie n'autorisant alors que les modifications réalisables.

#### 2.4.1.3 Intégration de l'orienté-objet

Certains devéloppeurs d'outils de CAO ont essayé de construire des systèmes autour du concept "objet"; la programmation orienté objet étant à la base de ces systèmes [AISH R. 90].

Un noyau graphique est construit à l'aide d'un langage orienté objet. Au dessus du noyau se superposent les différentes couches du système graphique: noyau exécutif, noyau de visualisation, couches applications (mécanique, architecture, ...).

L'approche objet permet au système de profiter d'une simplification dans l'implémentation de représentations multiples. Les éléments de conception, en tant qu'objets, savent comment se dessiner dans un plan à une échelle donnée ou dans un plan de détail. L'utilisateur peut éditer n'importe laquelle des représentations.

Les informations peuvent être classées dans des structures hiérarchisées facilitant leurs manipulations.

On peut imaginer, par exemple, les colonnes et les poutres appartenir à la classe d'éléments de structure linéaires, et les voiles et dalles appartenir à la classe des éléments de structure planes; les fenêtres et les portes appartiennent à la classe ouverure ...

Le système devient par suite flexible et extensible. Différents types d'éléments peuvent être ajoutés et utilisés sans que les anciens soient affectés.

on peut citer dans ce paragraphe des systèmes qui utilisent le concept d'objet sans que la programmation objet soit le moyen. L'approche objet est pensée comme une façon de structurer les données et de réutiliser le travail déjà fait même en y ajoutant quelques petites modifications.

## 2.5 Conclusion

- les systèmes de CAO actuels sont incapables de représenter des informations de haut niveau d'objets architecturaux et d'incorporer ce genre d'informations dans un processus de raisonnement.

- les outils graphiques appelés systèmes de CAO ont été devéloppés initialement pour aider le dessinateur. Leur évolution a porté essentiellement jusqu'à présent sur l'amélioration des fonctionnalités graphiques et sur la modélisation géométique. Ceci est dû au fait que les algorithmes et la structuration des données sont extrêmement complexes.

- les structures utilisées pour le modèle sont souvent celles utilisées pour la visualisation à l'écran (points, lignes, arcs, ...); les meilleurs systèmes ont une base de donées relationnelle pour stocker des informations non géométriques (des relations fonctionnelles,...) mais sans lien direct avec la géométrie.

- il n'y a pas une relation simple entre un objet à concevoir et sa représentation interne. Le même objet peut être représenté par différentes façons suivant le point de vue du concepteur.

- l'architecture ne se limite pas à une description numérique du bâtiment. Elle comprend aussi des concepts, des idées, des jugements et des expériences. Ceux-ci ne sont pas facilement simulés par des représentations procédurales; les architectes utilisent de la connaissance sur les objets, les événements et les pocessus et utilisent de la connaissance déclarative qui ne peut être décrite que symboliquement.

- une des limites des systèmes de CAO est qu'ils sont incapables de représenter et de manipuler la connaissance déclarative non algorithmique ou faire du raisonnement symbolique.

- La conception est un acte où doivent coopérer des domaines différents, d'où la nécessité d'un environnement intégré facilitant la communication et le partage des informations.

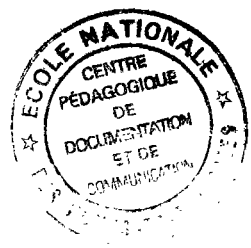
Les recherches en intelligence artificielle ont ouvert le chemin pour l'acquisition et la représentation de la connaissance sous une forme traitable par les machines.

On vise une nouvelle génération d'outils de CAO où les systèmes de gestion de base de données et les systèmes experts d'aide à la conception permettent la génération d'outils intégrés ou plutôt un environnement modulaire qui apportera une assistance intelligente au concepteur non pas une automatisation complète de la conception.

On présentera dans le chapitre suivant les différentes techniques informatiques de l'intelligence artificielle et des systèmes de gestion de bases de données ainsi que leur apport à la tâche de conception.

# CHAPITRE 3

# Conception et IA




Dans ce chapitre, nous essayons de mettre en valeur les aspects de la conception en insistant sur la conception en architecture. Par la suite nous étudierons les différents environnements et techniques de l'intelligence artificielle pour la représentation de la connaissance, le raisonnement et l'échange de données; nous donnerons quelques exemples de modèles utilisés en conception. En dernière partie nous détaillerons les démarches existantes pour l'élaboration de systèmes intégrés c'est à dire d'environnement d'intelligence artificielle adapté aux problèmes de conception et le rôle des outils de CAO dans ces approches.

## 3.1 La conception

### 3.1.1 Généralités

La conception est un terme très difficile à définir, et essayer d'élaborer un modèle de conception est une tâche difficile d'autant plus que ce modèle reste subjectif et chacun possède sa vision et son approche [SRIRAM D., STEPHANOPOULOS G. et AL 89].

La conception peut être vue comme un processus de résolution de problèmes ou d'application de concepts (comme en ingénierie), ou de création et d'invention (comme en architecture), ou même un acte de modification du préexistant ou "redesign" [BEHESHTI R. 90].

la conception est une problématique au sein d'un certain environnement pour trouver une solution.

il existe plusieurs approches pour la conception en fonction du domaine ou du concepteur ou même de la méthodologie appliquée [BEHESHTI R. 90], [CHEN C.C et AL 90], [SHAVIV E. 85], [STEINBERG L. 87], [TARKKO O. 91]:

- décomposition en buts et sous buts
- planification d'actions
- raisonnement par analogie
- essais erreur
- apprentissage
- propagation de contraintes

Cependant, quelle que soit la méthode utilisée, on peut identifier trois étapes fondamentales:

- analyse du problème
- génération de solutions

- critique et évaluation des solutions

La difficulté de modéliser la conception par des schémas classiques vient du fait que ces trois phases ne sont pas indépendantes et surtout non séquentielles [KAHKONEN K. et AL 87].

### 3.1.2 Les composantes de la conception

Les notions manipulées en conception peuvent être classées sous forme d'espace de conception. On peut identifier les espaces suivants [LAWSON B. 80]:

- l'environnement: c'est l'environnement intérieur et extérieur de l'objet à concevoir. En architecture, par exemple, la fonction et les dimensions d'un local font partie de l'environnement intérieur du projet, tandis que le site et l'ensoleillement font partie de l'environnement extérieur du projet.
Cet environnement peut être considéré comme un ensemble de contraintes qui s'imposent tout au long de la conception; ils guideront la recherche de solution dont on vérifiera la conformance à la fin du processus [DE GARRIDO L.A. 89].

- la représentation: un problème doit être représenté d'une certaine façon pour pouvoir le résoudre. De même la solution sera élaborée suivant une certaine représentation aussi [CARRARAG. et AL 90], [OXMAN et AL 91], [SHAVIV E. 85], [VIGNARD P. 86], [WOODBURY R. 91].
C'est un point fondamental dans notre thèse, car on essaiera de définir une façon de représenter les données, la connaissance, et les solutions qui sont rencontrées dans un problème de bâtiment.

- le modéle de raisonnement: un concepteur pour aboutir à un résultat final suit un modèle de raisonnement (itération, essai-erreur, ...); il peut même changer de mode de raisonnement au sein d'une même conception, d'où la difficulté de modéliser ce raisonnement dans des systèmes de conception automatique [MONTALBAN M. 87].

- *les choix: il existe deux types de choix:*

  - les choix d'action
  - les choix de valeurs

Le concepteur peut choisir l'action à exécuter pour continuer quand plusieurs options se présentent (concevoir l'espace jour avant l'espace nuit ou l'inverse), comme il peut imposer une valeur à une variable dont il connait le domaine de variations des valeurs (épaisseur d'un isolant).

- *l'évaluation: pour prendre une décision, choisir tel ou tel action, ou* s'arrêter si on atteint une solution valable, le concepteur doit pouvoir évaluer la solution proposée, la tester vis à vis des spécifications [CARRARA et AL 90], [FERRIES et AL 88], [KAHKONEN et AL 87].
Les critères d'évaluation sont difficiles à établir surtout qu'ils évoluent au cours de la conception.

### 3.1.3 Les modes de conception

Comme on avait déjà cité, il existe plusieurs modes ou approches pour la conception; nous détaillerons un peu plus certains modes:

- l'itération: De nombreuses itérations sont nécessaires tout au long du projet. Le contenu des descriptions du problème changent à chaque itération.

- l'essai-erreur: Les données du problème ne sont pas toujours toutes connues et l'expert doit faire des choix pour les paramètres manquants [ANDRE J.M. 86], [DE GARRIDO L.A. 89], [TARKKO O. 91].
Le paradigme générer, tester et modifier illustre bien ce genre de raisonnement:

  - générer: des hypothèses, des choix sont générés durant cette phase.

  - tester: les hypothèses sont testées si elles sont complètes ou valides selon tel ou tel situation.

  - modifier: durant cette phase on peut réparer les erreurs mises en évidence par le test pour pouvoir repartir dans un nouveau cycle.

### 3.1.4 La conception en architecture

La conception en architecture est un processus d'invention qui conduit à la création des formes en réponse à une fonction à assurer. elle peut signifier en ingénierie l'incorporation de concepts adéquats [QUINTRAND P. 85].

Trois grandes phases apparaissent dans ce processus:

- Le diagnostic architectural (ou la représentation du problème): au cours de cette phase l'architecte traduit les données du programme et de l'existant en informations qui vont amorcer un début de solution.

- La formalisation et la résolution graphique du problème: dans cette phase le dessin tient un rôle principal car il est à la fois preuve et moyen. Il est un modèle de représentation du problème et de la solution et un moyen de simulation [DE GARRIDO L.A. 89].

- L'établissement de modèle de construction: c'est la mise au propre du projet pour des fins de communication.

### 3.1.5 Conclusion

On peut souligner dans le processus de conception architectural les aspects suivants:

- la complexité du processus de conception et de production architectural et les problèmes que posent le transfert des informations.

- la nature des problèmes architecturaux peu ou pas modélisables.

- le rôle majeur de la pratique graphique dans le processus de conception.

-*le caractère multidisiplinaire de la conception; plusieurs acteurs de* spécialités différentes interviennent pour résoudre ou vérifier; le partage des données et l'approche non séquentiel des interventions sont des points essentiels.

*Au delà de fonction de représentations pour des fins de communcation, le dessin* architectural est un outil de résolution de problème.

Réaliser un système de CAO a permis jusqu'à présent d'informatiser la phase de saisie et d'instrumentation.

Peut-on compter sur les logiciels de CAO pour l'informatisation du processus de conception?

Peut-on raisonner avec du dessin?

## 3.2 L'intelligence artificielle

C'est l'étude des moyens qui permettent à une machine d'être intelligente.

### 3.2.1 Environnement et techniques de l'IA

Une machine pourrait être intelligente si elle contient de la connaissance et si la machine pourrait raisonner à l'aide de cette connaissance.

Différentes techniques ont été utilisées pour faciliter la représentation de la connaissance et l'implémentation des techniques de raisonnement [FORNARINO M. et AL 90], [GERO J.S. 85], [NARAT V. 86], [OXMAN R. et AL 91], [VIGNARD P. 86].

Nous noterons dans ce domaine les techniques suivantes:

- la représentation procédurale
- la représentation par la logique
- les langages à objets et les réseaux sémantiques
- les règles de production
- les bases de données

Nous détaillerons par la suite la représentation par la logique, les langages à objets, les systèmes experts basés sur les règles de production, et les bases de données.

#### 3.2.1.1 La représentation par la logique

deux types classiques de représentation en logique sont utilisés en intelligence artificielle:

- le calcul propositionnel
- le calcul des prédicats du premier ordre

le calcul propositionnel manipule des propositions dont la caractéristique principale est qu'elles peuvent prendre deux valeurs: VRAI ou FAUX. Une proposition exprime un fait [FORNARINO M. et AL 90], [VIGNARD P. 86].

Les propositions sont reliées entre elles par le connecteurs de la logique: ou, et, non, =>, <=>, et deviennent des formules bien formées.

Le calcul des prédicats étend la logique des propositions grâce à l'introduction de variables et des quantificateurs universels "s'il existe" et "quelle que soit".

La logique permet de représenter la connaissance sous forme de règles d'inférences.
La règle du modus ponens
SI A et (A => B) alors B
est à la base des systèmes de production ou des systèmes experts à chainage avant.
L'unification est utilisée pour déterminer l'ensemble des substitions possibles des variables d'une expression.

### 3.2.1.2 Les langages à objet

les techniques de l'intelligence artificielle devraientt offrir:

- une modélisation plus riche
- une plus grande structuration des entités du domaine
- une meilleure cohérence des données

Les langages procéduraux sont faibles en structures de données, et lourds pour la mise en oeuvre de connaissances déclaratives non algorithmiques.

Les langages à objet apportent la notion d'objet en tant que structure dynamique ayant un comportement autonome grâce aux messages et procédures qui lui sont attachés [BAILLY C. et AL 87], [MASINI G. et AL 89], [ROCHE C. et AL 89].

la programmation orientée objet repose sur les concepts suivants:

- objet
- classe
- méthode
- héritage

- Un objet regroupe une connaissance statique (champs) et une connaissance dynamique (méthodes).

- Une classe définit la structure (champs) et le comportement (méthodes) d'un ensemble d'objets. C'est une abstraction ou un modèle générique qui permet l'instantiation d'objets similaires appelés instances de classe.

- les méthodes ou comportement sont invoquées par envoi de messages entre objets.

- l'héritage permet à des sous classes définies par spécialisation de leurs classes mères d'hériter de celles-ci.
l'héritage peut être simple (une sous classe hérite d'une seule classe mère) ou multiple (une sous classe peut hériter de plusieurs classes mères).

| | | | |
|---|---|---|---|
| CLASSE: point3D | | | point-1 |
| x: type réel | | | x = 0 |
| y: type réel | - - > | instance | - - > y = 0 |
| z: type réel | | | z = 10. |

METHODES
initialiser
afficher

Ce style de programmation apporte:

- modularité
- extensibilité
- facilité de modélisation

### 3.2.1.3 Les systèmes experts

les systèmes experts sont constitués de trois parties principales [FARRENY H. 1985], [PINSON S. 81]:

- la base de fait: elle contient l'ensemble des faits que le programme a pu déduire à un instant donné.

- une base de règles: elle représente la connaissance qu'a le système dans un domaine particulier. Les règles sont en général du type:
condition -> action
Une règle est déclenchée si elle correspond à une situation courante.

- un moteur d'inférence: c'est le coeur du système; il peut fonctionner selon trois schémas types:

  - chainage avant, on dit alors que le raisonnement est guidé par les données; dans ce cas une règle est activable si sa condition est satisfaite.

  - chainage arrière, le raisonnement est alors guidé par les buts; dans ce cas une règle est activable si sa partie action correspond au but recherché.

  - chainage mixte, dans ce cas le raisonnement peut évoluer tantôt en chainage avant , tantôt en chainage arrière en fonction de sa stratégie et de ces éléments de contrôle.

Le moteur peut être d'ordre 0, 0+ ou 1; dans le premier cas il manipule des propositions qui ne contiennent pas de variables; dans le dernier cas les propostions contiennent des variables et le moteur procède par unification pour déterminer les règles activables.

une séquence de cycle élémentaire peut être:

-déterminer les règles et les faits pertinents au moyen d'unification
-choisir parmi les règles applicables celles qui sont déclenchables
-exécuter ces règles et mettre à jour la base de faits

Dans ces systèmes la connaissance est séparée des données, la représentation est modulaire et facile à lire, les structures de contrôle sont faciles à spécifier et à modifier. Elles peuvent être implémentées sous forme de métarègles, cependant si le nombre de règles devient très important, le système perd de son efficacité [DELCAMBRE B. 87].

Certains systèmes décomposent la connaissance en plusieurs modules ou bases de règles souvent appelées sources de connaissance. Dans ce cas certains systèmes reposent sur une décomposition récursive du but à atteindre en sous buts sous forme d'arbre de tâches; d'autres sont à architecture de blackboard et le contrôle se fait par le moyen d'agenda où sont classées à tout instant les sources de connaissances par ordre de priorité de déclenchement.

#### 3.2.1.4 Les bases de données en IA

Une base de données est un ensemble structuré de données enregistrées sur des supports accessibles par la machine [MIRANDA S. et AL 90].

Un système de gestion de base de données permet à des utilisateurs même concurrents de manipuler (insérer, modifier, rechercher ...) efficacement des données contenues dans une base de données en employant des connaissances de bas niveau (dépendances fonctionnelles, relations simples, fonctions d'agrégats,...).

Les SGBD évoluent vers des systèmes plus avancés; on trouve les SGBD orientés objet qui intégrent le même concept objet que les langages à objet ce qui assure la manipulation d'objets complexes et dynamiques [GARDARIN G. et AL 90].

D'autres voies visent à intégrer des techniques de la programmation logique pour aboutir à des bases de données déductives qui permettent de gérer les connaissances autour de la base de données.

Les programmes d'intelligence artificielle actuels n'utilisent pas encore les systèmes de SGBD car ils manipulent très peu de données. Cependant quand la manipulation d'une grande quantité de connaissances est nécessaire, l'utilisation d'un SGBD devient impérative.

Dans un processus de conception, la gestion des informations relatives au projet et l'échange de ces informations entre les divers participants sont deux tâches qui peuvent être gérées efficacement par un système de gestion de bases de données.

Dans un système de CAO en particulier, la difficulté d'utiliser les SGBD provient du fait que les entités manipulées sont complexes, dynamiques, géométriques et possédant des représentations multiples.

### 3.2.2 L'utilisation de l'IA en conception

#### 3.2.2.1 Les systèmes experts en conception

Des systèmes d'aide à la conception sont apparus dans plusieurs domaines tel que l'électronique, le génie civil la mécanique,...; chacun de ces systèmes privilégient un aspect de la conception dans le domaine: aspect fonctionnel, aspect structurel, ou autres [MONTALBAN M. 87].

En électronique, par exemple, CRITTER est un système expert en conception de VLSI; il raisonne sur des descriptions de circuits digitaux en utilisant une représentation déclarative. Il privilégie l'aspect fonctionnel des composants et les spécifications du circuit.

En génie civil, SPEX est un exemple de système expert d'aide à la conception de composants de structure de bâtiment. Le processus se décompose en deux phases:

-génération de contraintes
-satisfaction des contraintes

En mécanique, PRIDE traite la conception comme la recherche d'un ensemble de solutions possibles, puis la vérification de ces solutions; le processus est décomposé en sous tâches structuré en forme d'arbre. Chaque tâche dispose de plusieurs méthodes pour s'exécuter.
Il manipule aussi des contraintes qui sont des objets structurés. Ces contraintes sont testées après l'exécution du but qui les contient.

Parmi d'autres systèmes on peut citer SPEX [GARRET J.H. 86], FLODER [KARAKATSANIS A.G. 85], CADOO [ANDRE J.M. 86], LOOS [FLEMMING U. 90], PREDIKT [OXMAN R. et AL 87],...

#### 3.2.2.2 Limites de ces systèmes

La modélisation géométrique de l'espace dans tous ces systèmes est souvent éludée ou réduite à un problème à deux dimensions [TROUSSE B. 89].

Ces systèmes n'ont pas l'ouverture suffisante pour s'interfacer facilement avec les logiciels graphiques de CAO pour raisonner sur des objets géométriques.

La conception nécessite l'intervention de différents acteurs (experts humains, systèmes experts, programmes de calculs et d'évaluation,...). Il faut prévoir une architecture de coopération qui permet de partager et de faire communiquer des messages.

Le dessin joue un rôle très important dans la démarche de conception traditionnelle; il faut pouvoir l'utiliser en intégrant des logiciels de CAO à l'environnement de conception.

## 3.3 Essais existants: les systèmes intégrés

### 3.3.1 Introduction

Un environnement d'assistance à la conception doit apporter des possibilités de stockage et de restitution d'informations, des possibilités de calcul, d'inférences, d'interactions graphiques, de maintien de cohérence, des fonctions de création et de modification,...

Pour le stockage et la récupération d'informations on essaiera d'étudier l'apport des langages à objets et des systèmes de gestion de bases de données.

Pour le raisonnement, le maintien de cohérence et les fonctions de création on pensera aux environnements d'intelligence artificielle et aux systèmes experts.

Pour l'interactivité , elle est essentiellement graphique, car le dessin est la représentation préférée des concepteurs.

Un couplage entre des outils de CAO et des environnements d'intelligence artificielle parait alors très adapté à la programmation d'une assistance à la conception.

### 3.3.2 Types de données en bâtiment

Pour décrire complètement l'environnement d'un projet de bâtiment, il faut gérer des informations de natures différentes [GUENA F. 87]:

-des informations statiques valables pour un ensemble de projets (tel que composants, structures de couches,...)

-des informations à caractère réglementaire liées à l'environnement du projet et à l'usage futur du bâtiment.

-des informations à caractère dynamique évoluant du fait des reformulations itératives du projet effectuées par le concepteur ou du fait de l'intervention des modules de calcul ou évaluateurs.

-des informations sémantiques sur le comportement des objets manipulés.

Il faut noter que ce sont les informations à caractère dynamique qui posent le plus de problèmes en termes de représentation, de restitution et de mise à jour pour le maintien de cohérence après les modifications.

En utilisant les ordinateurs, une représentation des données appropriée devrait être utilisée pour la modélisation du projet. Cette représentation ne tient pas compte seulement de la description géométrique du bâtiment, mais aussi de son contexte, des contraintes ou des spécifications [DANNER W. 87, 88], [PAPAMICHAEL K. et AL].

#### 3.3.2.1 Données géométriques

Essayons de décomposer un bâtiment d'un point de vue géométrique:

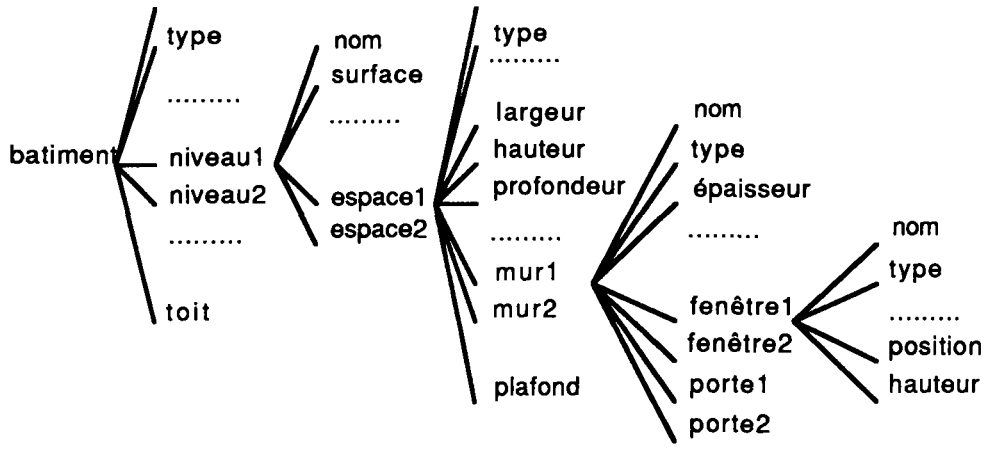



*fig. 3.1 Les données géométriques*

Cette décomposition hiérarchique du bâtiment met en évidence des entités du bâtiment qui sont de caractère très dynamique au niveau de la conception comme la largeur de la pièce ou la position de la fenêtre.

Ces objets ont une existence physique géométrique.

### 3.3.2.2 Données de contexte

D'autre part, on peut décomposer le bâtiment en un ensemble de variables de contexte:

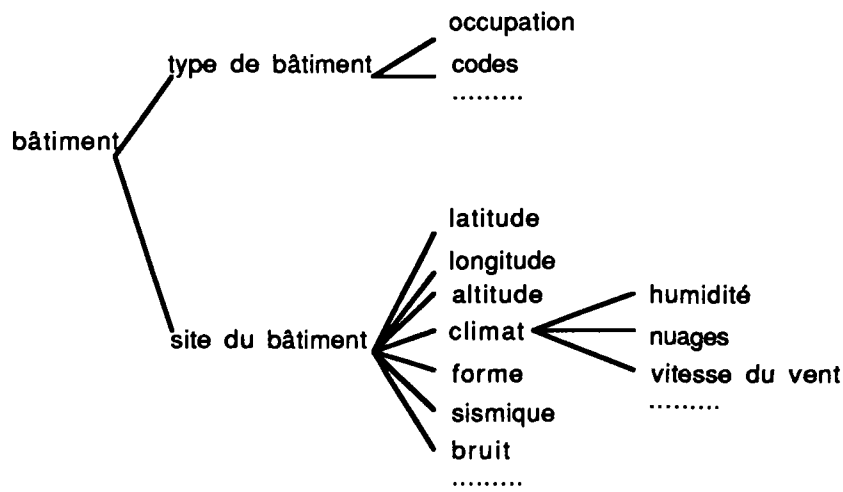



*fig. 3.2 Les données de contexte*

Ces variables sont stables et varient très rarement durant la phase de conception; ils dépendent du projet mais sont définis un fois pour toute.

Ces variables ne sont pas choisies par le concepteur, mais lui sont plutôt imposées comme la direction du vent ou l'orientation du site.

Ces entités sont pour la plupart des entités abstraites moins facilement représentables par des objets géométriques.

### 3.3.2.3 Contraintes et critères de performance

On peut voir aussi le bâtiment comme une solution satisfaisant certaines contraintes et remplissant des critères de performance:

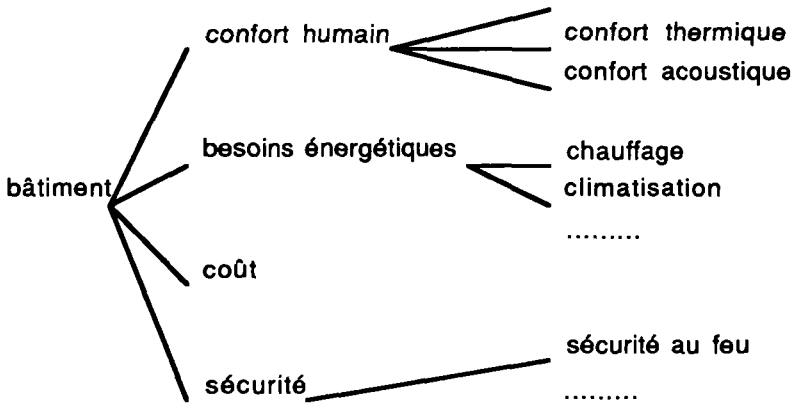



*fig. 3.3 les critères de performances*

Pour ces variables, le concepteur n'essaie pas d'imposer des valeurs spécifiques à atteindre, par contre il essaie de minimiser ou de maximiser ou d'optimiser en définissant des bornes supérieures (maximum), ou inférieures (minimum) ou les deux à la fois (intervalles). Ces critères peuvent servir pour la conception ou pour la vérification d'un projet. Elles peuvent dépendre de la géométrie du projet.

### 3.3.3 Rôle des outils de CAO

Tous ces types d'informations sont manipulés durant la conception; Il faut pouvoir les modéliser sous une forme informatique afin de permettre à la machine de les stocker et de les restituer facilement.

Les bases de données paraissent des outils adaptés pour gérer ces types complexes et différents; l'approche objet permet en plus de bien les structurer.

Les informations géométriques sont une part très importante des informations dans le bâtiment. Leur intégration dans une base de données n'est pas une tâche évidente [KOHLER et AL 90].

Quel rôle peut jouer un outil de CAO et quel type d'informations peut-il modéliser?

Comment faut-il faire le lien entre les bases de données géométriques d'un outil de CAO et une base de données centrale?

faut-il avoir deux bases différentes?

Il ne faut pas s'intéresser, au début, à l'implémentation d'informations de bas niveau (points, lignes, sommets,...),. Il faut plutôt se concentrer sur le modèle conceptuel ou le modèle de haut niveau riche en sémantique.

### 3.3.4 Base de données objet et bases de données géométriques

L'information nécessaire pour décrire un bâtiment est très compliquée, et la quantité d'informations nécessaires est gigantesque. Le nombre des objets (mur, cloison, local, fenêtres,..) est très élevé de l'ordre de dizaines même centaines de

milliers. Gérer ces objets reste une tâche très difficile [BJORK B.C. 90], [GUENA F. 87].

On pourrait penser à un formalisme pour exprimer un modèle standard qui serait compatible avec autant de vues différentes possibles; Quelles sont les classes à choisir, et quels sont les champs à inclure?

Des classes de type bâtiment, local, paroi,... avec les champs qui les décrivent (nombre de locaux, dimensions,...) paraissent les plus évidents.

Une base de données, si elle existe devrait être saisie à l'aide d'interface conviviale, simple et expressive; le premier choix pour saisir des entités à caractère géométriques serait un modeleur de CAO.

Cependant comme on l'a déjà vu, les modeleurs de CAO achoppent sur les points suivants:

-les structures de données utilisées sont souvent celles utilisées pour la visualisation à l'écran (points, lignes,...) sans aucune connaissance des concepts de plus haut niveau tel que local ou paroi.

-outre l'insuffisance sémantique des modèles utilisés, les outils de CAO ont tendance à forcer l'utilisateur dans une approche de conception ascendante (ou assemblage) et fonctionnent très mal en démarche descendante (ou définir des objets complexes puis définir les composants au fur et à mesure de l'avancement du projet).

-les outils de CAO sont incapables de manipuler des objets incomplets utiles en phase préliminaire de conception.

### 3.3.4.1 Première solution: coupler un modeleur de CAO avec une base de données objet

Des approches existantes consistent à construire une base de données objet où des classes de type local, paroi ou bâtiment sont définies pour pouvoir stocker les informations sémantiques du projet [DUBOIS A.M. et AL 90].

classe: Local
- bâtiment
- locaux-adjacents
- liste de parois
- volume
- surface-local
- surface-parois
- fonction
- topologie
- ....

Cependant, la saisie étant toujours effectuées par les concepteurs à l'aide d'outils de CAO, le point le plus délicat de cette approche serait le couplage entre la base de données objet et le modeleur de CAO.

En effet un couplage dans les deux sens s'impose pour assurer la cohérence entre les deux bases et répercuter les actions appliquées à une des deux bases sur l'autre [AUTRAN J. et AL 89].

**Couplage dans le sens base de données objet, outil de CAO**

Si on instancie un objet local dans la base objet, il faut créer la représentation graphique de ce local (parallélépipède ou prisme,...) dans la base du modeleur de CAO.

Dans ce sens, le couplage peut être fait en prévoyant pour chaque objet défini dans la base à objet une description de sa géométrie (topologie, dimensions,...). Il suffit alors que le modeleur de CAO jouent le rôle d'un filtre et se contente d'afficher la base objet sous forme géométrique par l'intermédiaire d'un langage de commande qui pilote l'outil de CAO [MACULET R. 90], [ROSENMAN M.A. 90], [TROUSSE B. 89].

**Couplage dans le sens outil de CAO, base de données objet**

Le passage entre le modeleur de CAO et le schéma conceptuel dans la base de données objet est un point plus délicat. En effet, si les classes bâtiment, local et paroi sont définies, ces classes devront être instanciées en nombre d'objets correspondant au projet et leurs champs devraient être remplis en fonction de la saisie graphique.

Qui va construire ces objets?
Qui va remplir les champs?
Qui va maintenir la cohérence et mettre à jour la base après modifications?

Ce n'est sûrement pas la tâche de l'utilisateur, car c'est une tâche fastidieuse et source d'erreurs.

D'autre part, il n'y a pas une relation simple entre un objet réel et sa représentation géométrique. Le même objet peut être représenté de différentes façons suivant le point de vue du concepteur.

La correspondance entre les objets géométriques et les objets de la base à objets n'est pas simple à trouver (une ligne peut être une paroi, un local, un coin,...) et l'interaction avec le module de CAO pour la création ou la modification ou juste pour la désignation risque de présenter des problèmes pour la répercussion sur la base à objets.

### 3.3.4.2 Deuxième solution: enrichir la base de données du modeleur par des entités riches en sémantiques

Dans ce cas les entités riches en sémantique font partie de la base du modeleur; donc au lieu de manipuler des lignes, des arcs ou des polygones, on manipulera des parois et des locaux [GARDAN Y. et AL 90], [AISH R. 90].

Un projet contient dans sa description des informations non géométriques aussi, on prévoit d'enrichir la base du modeleur par des classes représentant ces informations, ou on couple avec une autre base indépendante.

Le problème d'entretien des deux bases est plus simple car les données géométriques dynamiques qui changent souvent seront manipulées par l'outil de CAO

alors que les autres informations plus statiques seront gérées par la base de données objet (tel que vitesse du vent ou climat)

Le maintien de cohérence à l'intérieur de la base de données géométriques du modeleur n'est pas une tâche évidente.

En effet si l'utilisateur prend la fonction "creer local" par exemple, l'objet local serait instancié mais aussi il faut instancier tous les objets parois du local, les objets cloisons, remplir les champs de locaux adjacents, les objets ouvertures, les objets coins, ...

Et il faut répercuter toute modification sur n'importe quel élément pour maintenir la cohérence totale dans la base de données.

### 3.3.5 Limites de ces approches

La description d'un projet à l'aide d'objets instances de classes (local, parois,...) s'apparente au langage naturel: "un bâtiment contient dix appartements, chaque appartement contient 5 pièces, la première pièce est orienté nord-sud,..."

C'est vrai que sous forme de classes et d'objets les informations sont plus structurées, mais peut-on représenter et contrôler des géométries complexes avec du langage naturel? (intersections de volumes, inclusions, appartenances,...)

Peut-on modéliser simplement un projet complexe? (le nombre d'objets croit rapidement et la cohérence devient difficile à gérer)

Faut-il avoir deux bases à entretenir et à coupler, celle des objets et celle du modeleur?

Certaines informations, d'ordre géométrique, existent dans la base de données du modeleur, il suffit de savoir les extraire; pourquoi les traduire dans une autre base de données? jusqu'à quelle limite peut-on éviter la redondance en exploitant le modèle géométrique?

# CHAPITRE 4

# Couplage CAO-IA

Ce chapitre expose la partie théorique de la thèse. On commence par décrire les méthodes classiques utilisées traditionnellement pour souligner quelques points remarquables concernant les données géométriques simples; nous essaierons après d'illustrer le concept de fonctions d'analyse à travers des exemples très didactiques. Puis nous présentons notre approche, les principes de bases, les moyens de réalisation, les points forts et les points faibles, ainsi qu'une extension du modèle proposé pour une utilisation plus générale par l'implémentation de contraintes.

## 4.1 Les méthodes classiques

### 4.1.1 Le dessin moyen de communication

Le dessin est un message émis par l'architecte et reçu par l'architecte et par les autres participants à l'acte de conception. Le message graphique sera interprété pour en tirer les différentes sortes d'informations [LEBAHAR J.C. 83].

Le dessin parait donc le moyen du concepteur pour partager l'information, surtout que la conception fait intervenir plusieurs acteurs et repose énormément sur la communication des idées.

Les idées sont concrétisées par le dessin de plans cotés, et commentés, de coupes de façades ou autres vues de détails nécessaires pour la compréhension du projet.

### 4.1.2 Codage et décodage du dessin

Analysons plus profondément cette phase de communication:

Si l'architecte passe un plan décrivant le projet à l'ingénieur, ce qui est mis en commun peut être décomposé en deux parties:

-le dessin d'un plan (accompagné éventuellement de descriptifs).
-La connaissance nécessaire pour lire un plan.

Il ne faut pas confondre cette connaissance avec l'expertise propre de l'ingénieur, c'est plutôt le moyen de décoder le plan.

Un plan est un codage spécial de l'information, et le décodage est nécessaire pour extraire cette information.

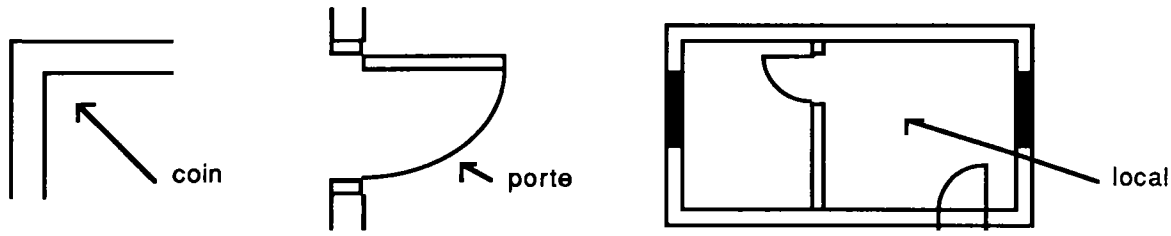



*fig. 4.1 Les codes d'un dessin de bâtiment*

Ainsi l'homme essaie de décoder un plan, ou plutôt de l'interpréter pour en tirer les informations dont il a besoin.

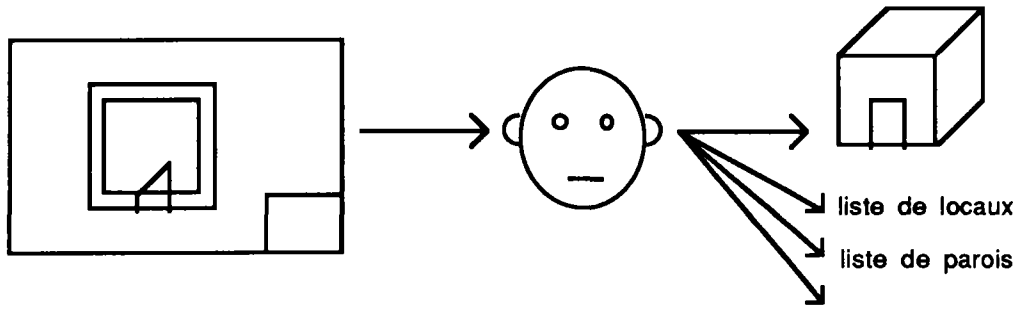



*fig. 4.2 Le décodage d'un dessin*

Un plan contient beaucoup d'informations sémantiques sur le projet codées en termes d'entités géométriques (lignes, arcs, hachures,...)

L'interprétation des informations à partir des entités géométriques peut changer d'un acteur à un autre suivant ses points de vues ou son intérêt dans le projet;

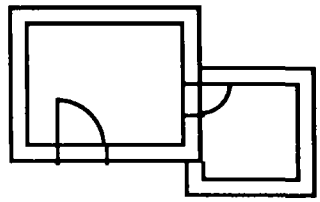


*fig. 4.3 Plan simple*

Si un architecte interpréte ce plan, il verra des locaux communicants, des parois, des cloisons, des portes, des surfaces de déplacement,...

Par contre, un ingénieur de structure sera attiré par les coins, les intersections de murs qui représentent des endroits potentiels pour l'implantation de ces poteaux, et il verra l'espace du local comme la portée entre nu d'appuis pour ses poutres.

### 4.1.3 Conclusion

L'informatisation des moyens de conception a porté surtout sur l'instrumentation de la phase de dessin du projet; cependant, la partie d'interprétation d'entités géométriques en entités plus riches en sémantique n'a toujours pas été implémentée.

Pour modéliser cette nouvelle information, on va ajouter à la classe d'objet cube les champs au dessus et au dessous qui contiendront respectivement l'objet au dessus et l'objet au dessous s'ils existent.

Cependant qui va remplir ces champs et comment?

De même la donnée ajoutée va augmenter la taille des objets et la quantité d'informations devient tout de suite énorme pour peu de connaissances.

A chaque déplacement d'un objet dans la scène il faut remettre à jour tous les champs touchés.

On pense alors à équiper l'objet cube par une méthode de vérification mathématique, de lancer de rayon par exemple, pour pouvoir vérifier s'il y a ou non un objet au dessus ou en dessous.

La fonction sera déclenchée à chaque appel au champ correspondant; on peut même aller plus loin en éliminant le champ et en transformant la méthode en une requête qui ramène l'objet au dessus quand elle est envoyée en message à l'objet.

La base de données devient moins volumineuse, plus dynamique et sa mise à jour plus simple car il suffit de mettre à jour les entités géométriques sans s'inquiéter du reste.

On va essayer d'étendre cette notion de fonction pour qu'elle puisse ramener des objets complexes qui n'ont pas été stockés.

Essayons une autre manipulation de la scène avant: plaçons le cube 1 et le cube 2 au même niveau puis plaçons le prisme de façon à reposer sur les deux cubes.

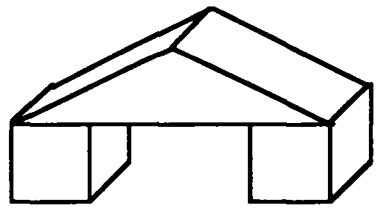


*fig. 4.6 Une deuxième vue de la scène*

On peut tout de suite remarquer qu'un portique s'est formé, une information sémantiquement riche.

Pour stocker cette nouvelle information, on est de nouveau tenté de créer une classe portique qui décrit un portique; cependant qui va instancier cette classe, et quand?

N'oublions pas aussi que le nombre et la nature d'interprétations potentielles d'informations géométriques complexes est généralement difficile à prédire.

En effet, différents experts regardant la scène verront l'objet complexe et l'interpréteront à leur manière [BILLET A. et AL 87], [CAMMARATA S. et AL 84], [WING J. et AL 85].

Un expert qui s'intéresse à assembler verra la scène de la façon suivante:

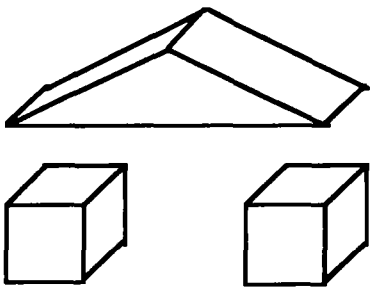


*fig. 4.7 Scène vue par assemblage*

alors qu'un expert qui s'intéresse à découper verra la scène de la façon suivante:

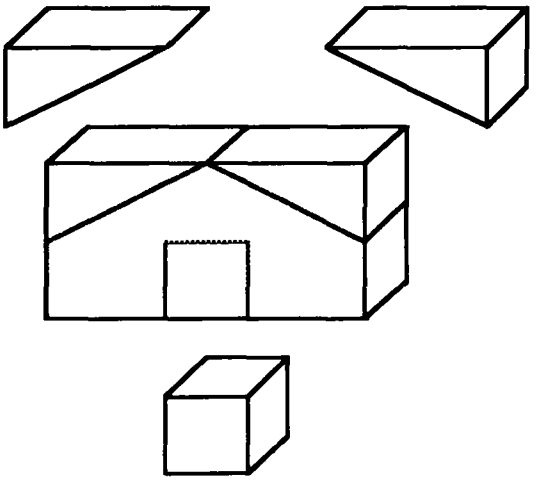


*fig. 4.8 Scène vue par découpage*

si j'impose une façon de représenter les objets, une limite rigide va vite s'établir, à moins qu'une redondance dans la description puisse prendre en compte toutes les représentations possibles, cela au prix de l'augmentation de la taille de la base de données et au prix d'un temps de saisie multiplié par le nombre de représentations.

Ne serait il pas important d'avoir une base de données simple et chacun pourra la lire et l'interpréter par ses propres moyens à l'aide de méthodes qui décrivent comment rechercher ce dont il a besoin?

### 4.2.2 Exemple de données géométriques simples en 2D

Imaginons qu'un modeleur 2D ne connait qu'une seule sorte d'entité géométrique 2D: le segment (défini par ses extémités).

Un utilisateur ne peut saisir que des segments et la base de données graphique du modeleur ne stocke que la topologie de segment.

Supposons que l'utilisateur saisit les segments dans la configuration suivante:

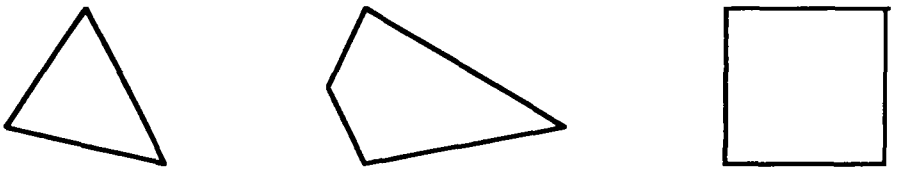


*fig. 4.9 Des figures simples*

Dans le premier cas on a des segments qui forment un triangle, dans le deuxième cas ils forment un quadrilatère, et dans le troisième cas un carré.

On peut saisir à l'aide d'un modeleur pareil autant de formes complexes que possible, mais le problème reste que le modeleur ne stocke que des segments et les informations de haut niveau tel que carré ou triangle ne sont pas stockées.

On peut par contre créer des fonctions d'analyses qui permettent d'extraire à partir des entités géométriques simples des objets plus complexes.
On peut par exemple décrire un triangle comme une suite de trois segments dont les extrémités coincident deux à deux et qui forment un pourtour fermé, ou bien un quadrilatère comme une suite de quatre segments ..., et un carré comme un quadrilatère dont les quatres segments sont deux à deux parallèles, tous de même longueur et formant des angles droits.

L'abstraction des objets géométriques est obtenue par les définitions de ces objets sous forme procédurale dynamique et non sous forme d'attributs statiques. Cette représentation permet à chaque utilisateur de définir à sa façon les objets dont il a besoin d'extraire.

Equipé de ces fonctions d'analyses tel que décrites plus haut ce même modeleur est maintenant capable de restituer des informations complexes autres que celles qui sont stockées dans sa base simple.

### 4.2.3 Concept d'analyseurs

On peut étendre cette approche exposé plus haut à des modeleurs 3D, qui traitent plutôt de la volumétrie, et plus particulièrement celle du bâtiment.

Bien que le modèle 3D dans la machine soit virtuel, sa puissance de représentation n'est guère inférieure à celle d'une vraie maquette. Une personne sait facilement interpréter une maquette, peut-on espérer doter la machine de fonctions qui lui permettent d'interpréter ce modèle?

On peut imaginer que le modeleur 3D ne stocke dans sa base de données que des entités de volumes simples, leurs topologies (prisme, cube,...), leurs textures (plein, vide, transparent, ...), et leurs positions dans le plan.

En partant du principe que vide et plein sont des notions de bases connues par le système, on peut définir des entités plus complexes tel que paroi, local, ou cloison.

Une paroi peut être définie comme un assemblage de volumes pleins contigus séparant deux espaces vides;
Un espace de local peut être défini comme un ensemble de vides contigus et délimités par des éléments pleins formant l'enveloppe.

### 4.2.4 Analyseurs et fonctions

A travers les deux exemples avant on peut noter les deux aspects suivants:

-la capacité de tirer des informations sémantiques à partir de données géométriques simples permet de simplifier la représentation d'un modèle dans une base de données.

-des fonctions dynamiques qui peuvent ramener des objets géométriques complexes (paroi, espace d'un local,...) ou des informations (au dessus, en dessous, ...) permettent à la base de données d'être dynamique et facilite sa mise à jour.

Nous allons essayer de combiner ces deux concepts d'analyseurs et de fonctions pour les exploiter dans le domaine du bâtiment.

Des fonctions d'analyseurs permettent de scanner une base de données géométrique simple contenant des volumes décrits simplement pour en tirer des informations complexes riches en sémantiques tel que paroi, local, communication,...

## 4.3 Présentation de notre approche

### 4.3.1 Choix du système

On a vu que les données dynamiques qui évoluent tout au long de la phase de conception sont surtout des informations spatiales et géométriques décrivant le projet (espace local, communication, ouvertures,...).

C'est dans cette partie des données que le maintien de cohérence et de la consistance des données est la plus critique. Elle représente aussi la majeure partie du projet qui peut être saisie à l'aide d'un système de CAO.

Dans notre approche, on a essayé d'expérimenter les concepts de fonctions analyseurs. Pour cela notre système de CAO est constitué d'une base de données simples, géométriques et sémantiquement pauvres; ce système est par contre équipé d'un ensemble de fonctions d'analyse spatial et géométrique pour la reconnaissance d'entités sémantiques.

On verra dans ce qui suit comment sont traités les informations géométriques et les informations non géométriques.

### 4.3.2 Les informations géométriques

#### 4.3.2.1 Représentation

Les informations géométriques qui peuvent être saisies à l'aide du modeleur de CAO sont des volumes élémentaires; la texture (i.e. la nature des volumes) est la principale distinction entre ces éléments.

Les textures utilisées sont:

-vide, pour représenter les espaces vides

-plein, pour représenter une partie d'une enveloppe (mur, plafond ...)
-transparent, pour représenter des ouvertures (portes ou fenêtres,...)

Aucune autre information de haut niveau ou d'ordre sémantique n'est ajoutée à la description des données.

Un projet est alors un ensemble de volumes de textures vides, pleins ou transparents et qui sont positionnés dans un système d'axes; l'ensemble est appelé "univers".

La notion mur, cloison ou linteau n'est ni mentionnée ni saisie ni désignée dans la base de données.

Bien sûr le modeleur est équipé d'outils pour la manipulation graphique des volumes. On a des fonctions de création, de modification, d'extraction, de déplacement,...; un objet particulier appelé curseur permet de réaliser ces fonctions interactivement d'une façon graphique sur l'écran; il sert aussi de dispositif de pointage à l'intérieur du modèle géométrique.

En parallèle, et pour faciliter la saisie, certaines fonctions de saisie complexes ont été implémentés; par exemple une fonction *creer-local* qui permet de créer les volumes nécessaires pour la représentation de la volumétrie d'un local en forme de parallélépipède en lui fournissant ses dimensions; il faut insister sur le fait que cette fonction ne crée aucune information pour stocker la notion local, mais se contente de créer la volumétrie, i.e. les volumes élémentaires de la géométrie du local.

Avec les fonctions de haut niveau on peut piloter le logiciel de CAO à l'aide de commandes simples tel que creer-local ou creer-paroi.

Les fonctions de pilotage et les fonctions d'analyse assurent la communication dans les deux sens avec le modeleur. On peut y stocker puis restituer de l'information à l'aide d'un langage évolué manipulant des entités sémantiques et ne stockant que des informations géométriques simples.

L'outil est aussi muni des fonctions classiques de visualisation en perspective et en plan de l'univers ou d'une partie de celui-ci afin de faciliter la saisie et la compréhension du projet.

#### 4.3.2.2 Les fonctions d'analyse

Au dessus du noyau physique géométrique, un ensemble de fonctions d'analyseurs est construit formant une surcouche.

Ces fonctions sont capables d'identifier différentes notions de haut niveau riches en sémantiques tel que plafond, murs, cloisons, linteaux,...

##### 4.3.2.2.1 Choix des fonctions

On stocke les informations pour pouvoir les restituer et les utiliser dans des processus de calcul ou de vérification.

Les modules des programmes de calcul ont besoin de paramètres en entrée (input) pour produire des résultats ou des sorties (output).

De même, les règles d'un système expert ont besoin de paramètres en entrée qui sont les déclencheurs de ces règles (les objets figurant dans les prémisses ou les conditions dans un raisonnement en chainage avant par exemple). On peut comparer une règle à un sous programme qui n'est déclenché que si ses paramètres existent et vérifient certaines conditions [PENNY NII H. et AL 79].

On peut traiter les bases de règles et les programmes de calcul en boites noires recevant des paramètres en entrée et fournissant des résultats en sortie. Il suffit de pouvoir extraire de notre base de données les informations nécessaires pour les faire tourner. On a alors choisi nos fonctions à implémenter en conséquence: on a des fonctions d'identification de locaux, de cloisons, de calcul de surfaces,...

Le choix de nos fonctions implémentées était basé aussi sur le principe de l'identification des informations implicites qui sont introduites au moment de la saisie; on a donc essayé de faire en sorte que les informations qui sont présentes dans un modèle ne soient pas perdues.

Par exemple, on a la fonction qui permet de retrouver un local qui a été créé par la fonction creer-local; on rappelle que seule la géométrie de ce local avait été stockée.

On s'est occupé surtout de notions qui ont rapport avec l'architecture intérieure du projet car la maquette réalisé est orienté vers une démonstration dans ce domaine.

En résumé, toute entité d'ordre sémantique, pouvant être extraite à partir de données géométriques, peut être implémenté; un moyen efficace de les reconnaitre est d'essayer de les identifier dans des règles utilisées dans le domaine étudié ou écrites pour un système expert dans ce domaine [VERLUISE F. 86].

#### 4.3.2.2.2 Principe des fonctions

Le principe de ces fonctions est simple; il suffit de décrire l'information conceptuelle de haut niveau en termes d'informations de bas niveau i.e. volumes pleins, vides ou transparent. Par la suite, la fonction scanne la base de données géométique du système de CAO à la recherche des entités décrites.

Par exemple, une fonction qui peut identifier l'espace intérieur d'un local, va chercher dans le modèle géométrique, à partir d'un point désigné graphiquement à l'intérieur du local à chercher, les éléments de volume de texture vides et qui sont contigus.

L'ensemble de ces éléments vide sera ensuite testé pour voir si chaque facette touche dans son intégralité à des volumes voisins vides , pleins ou transparents , c'est à dire si l'enveloppe du local est bien définie.

Si l'espace designé est un espace fermé, où tous les éléments vides sont enveloppés par des éléments pleins ou transparents, la fonction retourne cet ensemble de vides.

Si un des éléments vides de l'ensemble a une facette qui n'a pas de voisin, l'espace désigné n'est pas fermé et par suite ne peut pas représenter un espace de local, et la fonction retourne une erreur.

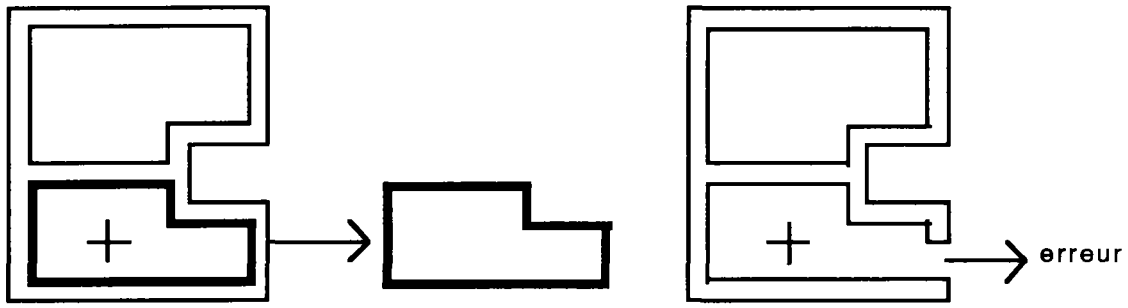



*fig. 4.10 Le principe de la fonction qui cherche l'espace intérieur d'un local*

Les algorithmes utilisés sont des versions simplifiées pour démontrer la faisabilité de fonctions assez "intelligentes" pour identifier des notions sémantiques du monde architectural.

Des algorithmes plus complexes mais plus efficaces peuvent être utilisés pour améliorer la puissance et l'efficacité de tels fonctions.

Le tracé de rayon est un exemple de méthodes mathématiques et géométriques qui peut être utilisé pour scanner efficacement la base de données géométrique à la recherche d'entités sémantiques utiles pour le raisonnement.

#### 4.3.2.2.3 Définitions multiples pour une même fonction

La même fonction de reconnaissance d'un local peut être définie de plusieurs façon afin de reconnaitre un local dans des contextes différents, à différentes étapes de sa conception, en fonction du degré d'avancement.

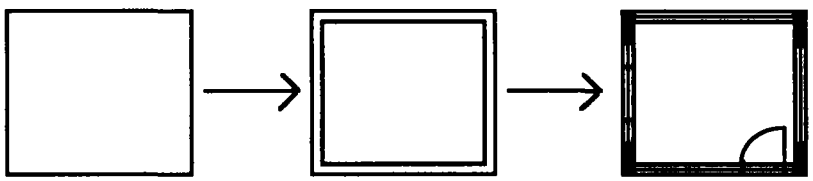


*fig. 4.11 Un local vu dans différents degrés d'avancements*

On peut représenter alors des objets incomplets et les stocker avec des entités géométriques. On verra aussi plus loin l'utilisation de contraintes pour le stockage d'informations sur des objets incomplets.

Que le local soit saisi par assemblage des parois, ou par subdivisions d'un grand local ou par extraction d'un volume vide à partir d'un volume plein, la fonction sait toujours interpréter un local et le retrouver. Ceci permet de simplifier la saisie car on ne s'occupera que du résultat graphique sans se soucier de la démarche (descendante ou ascendante).

#### 4.3.2.2.4 Appels de fonctions emboités

Les fonctions d'analyse implémentées ramènent un objet géométrique (un ensemble de volumes vides dans l'exemple précédent); cet objet a la même structure de description géométrique que tout l'univers (l'univers étant la description géométrique

de tout le projet); ainsi, une fonction d'analyse peut analyser le résultat d'une autre fonction d'analyse pour chercher des entités sémantiques. Ceci permet d'écrire des appels de fonctions emboités.

#### 4.3.2.2.5 Fonctions utilitaires

Toutes les fonctions implémentés ne sont pas des analyseurs pour trouver des informations sémantiques, mais aussi des fonctions qui cherchent de simples attributs géométriques tel que hauteur minimale dans un local ou volume d'un local.

Ces fonctions suivent le même principe que les fonctions d'analyse; elles cherchent l'information à extraire dans la base de données géométrique.

### 4.3.2.3 Langage de requête pour une base de données géométrique

Dans un système de gestion de base de données, un langage de requête permet à partir de fonctions simples de base tel que select, from ou where de construire des requêtes plus complexes qui permettent par exemple de tirer d'un ensemble d'hotels ceux qui n'ont que deux étoiles.

Par analogie, on peut imaginer un langage de requête qui contient des fonctions d'analyseurs qui permettent d'extraire des informations complexes dans une base de données géométrique simple où les informations sont codées en termes d'entités géométriques.

Tel un langage de requête, ces fonctions seront fournies à l'utilisateur; toutes les fonctions imaginables ne seront pas écrites. Un ensemble de fonctions élémentaires de base peut être défini à partir duquel l'utilisateur peut construire des requêtes plus complexes. Il utilisera les fonctions fournies avec des appels séquentiels ou emboités pour construire des requêtes plus complexes .

On peut par exemple combiner la fonction *trouver-local* (qui ramène un local) et le prédicat *communique?* (qui répond vrai si deux locaux communiquent) pour trouver tous les locaux qui communiquent avec un local donné.

De cette façon, le système peut être personnalisé par chaque utilisateur; celui-ci peut en effet écrire ces propres fonctions d'analyse, et profiter de bibliothèques de fonctions devéloppées par d'autres personnes pour enrichir son système.

Les fonctions à définir devraient être facilement implémentées; on peut prévoir pour cela un préprocesseur ou un compilateur qui permet de traduire des fonctions écrites avec une synthaxe prédéfinie en des fonctions utilisables par le système.

Les fonctions du langage de requête facilitent l'écriture des règles pour les systèmes experts; ces règles deviennent déclaratives et ne raisonnent pas sur le modèle géométrique en termes mathématiques mais plutôt avec des termes de plus haut niveau tel que surface-local ou chercher-cloison.

Les règles sont implémentées en une couche au dessus de ces fonctions du langage de requêtes. Si on change le modèle géométrique, les règles resteront les mêmes, il suffit de réécrire les fonctions d'analyse pour le nouveau modèle géométrique.

#### 4.3.2.4 Le maintien de cohérence

Cette approche d'intégration des données géométriques est simple et efficace. La base de données du modeleur de CAO n'est ni enrichie par des classes d'objets riches en sémantiques ni couplée avec une base de données contenant ces classes.

Des ensembles modulaires de fonctions d'analyse "intelligentes" sont construites autour du noyau géométrique; ces fonctions seront appliquées dynamiquement sur la base de données géométriques pour interpréter et trouver l'information recherchée (paroi, local,...). Par suite, si la mise à jour de la base de données géométriques est réalisée, les données extraites seront aussi à jour automatiquement.

Ceci garantit au moins que la partie principale des informations (la description géométrique tel que paroi ou local) est mise à jour automatiquement et la cohérence est maintenue assez facilement.

L'approche proposée, ou celle des fonctions d'analyseurs, permet de gérer facilement la cohérence des données; la base de données n'étant que géométrique et simple, la saisie et la mise à jour est facile et peut être réalisé par l'outil de CAO. L'appel des fonctions d'analyseurs permet à tout instant de calculer la réponse dynamiquement et cette réponse sera donc à l'image de la base géométrique déjà à jour.

Nous croyons aussi que cette approche simplifie l'utilisation des systèmes de CAO car la description du projet se limite à décrire les volumes et les espaces, donc les entités géométriques et spatiales du projet.
Le concepteur n'a pas à s'occuper de la saisie d'entités sémantiques tel que paroi extérieure ou cloison, ou même des des notions plus complexes tel que coins, liaison de mur, ou des notions spécialisées tel que murs porteurs ou linteaux.

Chaque utilisateur aura ses propres fonctions d'analyses qui permettent d'interpréter le modèle géométrique pour en extraire ce dont il a besoin.

#### 4.3.2.5 Extensibilité et modularité

Cette approche garantit aussi la facilité d'extension et la modularité du système.

Pour l'extensibilité, étendre la base de données revient à étendre son langage de requête en définissant des fonctions qui savent analyser et tirer une nouvelle forme d'information à partir de la base de données géométrique qui elle reste la même.

Les fonctions d'analyseurs peuvent être regroupées en petits modules; les modules appartiendront à différents utilisateurs et peuvent avoir un accès interdit pour certains utilisateurs; ceci permet d'assurer la modularité du modèle.

### 4.3.3 Les informations non géométriques

Toutes les informations nécessaires pour la représentation d'un projet de bâtiment ne sont pas d'ordre géométrique. On peut trouver:

-des objets abstraits indépendants de la géométrie du projet tel que les variables d'environnement (le site, le climat,...)

-des objets abstraits mais qui sont intimement liés à la géométrie du projet notament la fonction d'un local (séjour, cuisine, ...) ou la répartition des zones thermiques (zone froide, zone chaude,...)

-des objets complétement indépendants du projet tel qu'un composant de fenêtre (FE01: double vitrage, ouverture à la française, chassis aluminium,...); ces objets ne dépendent du projet que par leur position.

Pour chacun de ces types mentionnés, il faut choisir un moyen de représentation qui soit compatible avec l'approche décrite pour les objets géométriques.

#### 4.3.3.1 Les informations indépendantes de la géométrie

*Ce type d'objet dépend du projet mais ne dépend pas de la géométrie de celui-ci;* ce sont des objets stables qui varient très peu au cours de la phase de conception, et leur maintenance est simple.

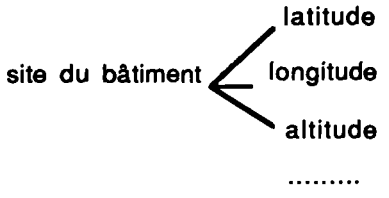



*fig. 4.12 Les champs de l'objet site*

Pour ce type d'objets, nous avons choisi de les implémenter sous forme d'objets dynamiques, instances de classes déjà définies; ces objets sont petits et dénombrables.

Les objets seront instanciés une fois par projet et leurs champs remplis par l'utilisateur; parce qu'ils sont surtout des données imposées plutôt que des données choisies, les valeurs stockées dans les champs seront consultés en lecture et très rarement en écriture.

#### 4.3.3.2 Les informations dépendant du projet et de la géométrie

Comme exemple de ce type d'informations on a choisi la fonction attribué à un local (séjour, cuisine,...). Ces informations sont définies géométriquement (liés à l'espace vide du local), mais par contre ne sont pas des objets géométriques.

On peut représenter ces objets de la même façon que les objets géométriques: on ajoute une texture cuisine ou une texture séjour, et le volume de l'espace intérieur aura cette texture.

Cependant on a choisi une autre façon de représentation, plus dynamique et qui s'apparente aux méthodes traditionnelles de dessin de plan.

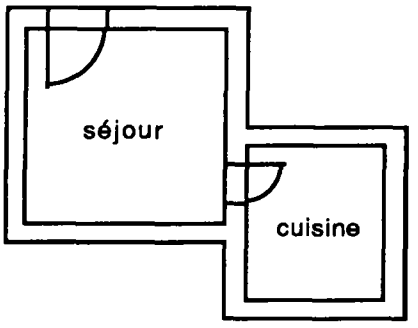



*fig. 4.13 L'interprétation du texte dans un plan*

Avec les méthodes traditionnelles, il suffit d'écrire cuisine dans le vide du local désigné pour que ce local soit interprété comme étant la cuisine.

Notre méthode consiste à avoir des objets fonctions, et qui contiennent un pointeur de l'espace qui permet de designer un emplacement géométrique; ainsi l'objet cuisine pointerait dans l'espace vide du local supposé être la cuisine.

Bien sûr le pointeur doit pointer dans un espace fermé, sinon une erreur est signalée.

#### 4.3.3.3 Les informations indépendantes du projet

Comme exemple de ce type d'objet on a choisi la structure couche des murs et les composants fenêtres.

Ces objets sont valides pour plusieurs projets. Ils peuvent être gérés par un SGBD externe au noyau du système et utilisés par celui-ci en consultation. Ces objets seront stockés dans des bibliothèques sous forme de catalogues et seront chargés au moment de leur utilisation.

Une représentation par classes et objets instances de ces classes a été choisie; Ils seront liés à la base de données géométrique de la même façon décrite avant , par l'intermédiaire de pointeurs spatials.

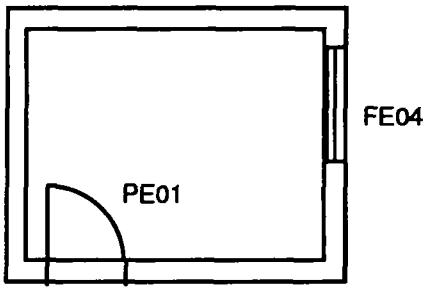



*fig. 4.14 Une utre interprétation du texte dans un plan*

Comme pour les méthodes traditionnelles, un pointeur désignant un emplacement géométrique permet de lier les objets tel que FE04 à la géométrie du projet.

### 4.3.4 Le problème de compatibilité

Différents types d'objets sont utilisés:

-des objets géométriques
-des objets abstraits (cuisine, site,...)
-des objets de catalogue

Malgré la différence, l'utilisation reste transparente, et la compatibilité assurée par le concept même de la programmation objet.

Les objets sont de classes différentes, mais la communication avec eux se fait par l'intermédiaire de messages.

Il faut tout simplement faire attention au type du résultat retourné par une méthode (booléen, géométrique, pointeur, ...).

Par exemple on envoie le message trouver-parois-local à l'objet univers et on récupère un objet géométrique contenant les parois du local désigné.
De même, on envoie le message vitesse du vent à l'objet site et on récupère la valeur de cette vitesse.

## 4.4 Extension du modèle

### 4.4.1 Les objets incomplets

Dans notre approche, on a adopté les contraintes comme moyen de représentation des objets incomplets. Les objets incomplets ne peuvent pas être représentés facilement par des entités géométriques, ce qui fait que notre base de données géométrique simple dans laquelle on peut stocker de l'information ne suffit pas.

Un local dont on ne connait pas les dimensions ni la position ne peut pas être saisi graphiquement; par contre on peut posséder des informations concernant ce local comme par exemple: ce local doit communiquer avec un autre local, ou bien sa surface doit être supérieure à une certaine valeur,...

Comme l'objet incomplet n'existe pas dans la base géométrique, on stockera les informations sous forme de contraintes. Une contrainte tel que surface séjour = 20 permet de stocker cette information pour l'objet qui n'existe pas. La contrainte sera utilisée plus tard à des fins de conception ou de vérification pour valider l'objet après sa création.

### 4.4.2 Les contraintes

#### 4.4.2.1 Représentation des contraintes

Une contrainte peut être utilisée de deux façons différentes [ASSEMAT C. et AL 88], [FORNARINO M. 90], [KAHKONEN et AL 87]:

-d'une façon active: comme une donnée à partir de laquelle on peut concevoir quelquechose; par exemple la contrainte:

surface du séjour > 30 $m^2$

c'est une contrainte qui peut aider dans le choix des dimensions du local séjour.

-d'une façon passive: comme un moyen de contrôle et d'évaluation; le projet solution doit être vérifié et doit satisfaire les contraintes suivant leur degré d'importance.

Ces contraintes peuvent être appliquées à différents types d'objets:

-des contraintes appliquées à des objets géométriques:
le local cuisine est adjacent au local séjour

-des contraintes appliquées à des processus:
concevoir espace nuit avant de concevoir espace jour

-des contraintes sur les différents acteurs
déplacer un mur est le travail d'un architecte

#### 4.4.2.2 Problème de satisfaction automatique de contraintes

Un système qui permet la satisfaction automatique des contraintes n'est pas réalisable d'une façon générale. Il existe des systèmes de CAO 2D qui permettent de satisfaire un certain nombre limité de contraintes simples tel que parallèlisme ou tangence, mais en 3D le problème est encore plus complexe et en bâtiment les contraintes ne sont pas très simples [FLEMMING U. et AL 89].

En effet, un problème de satisfaction de contraintes représenté en termes d'équations à résoudre peut présenter dans certains cas (en fonction du degré de liberté des variables) des difficultés à trouver la solution. On peut utiliser des méthodes de l'algèbre symbolique [HOLTZ N. 82], [CRAIG HOWARD. H. et AL 83], des méthodes d'optimisation à l'aide de graphes de critères [LE GAUFFRE P. 88], des techniques de propagation de contraintes [LUCAS J.Y. 90], [LAURIERE J.L. 76], des techniques basés sur la consistance d'un réseau [FORNARINO M. 90] ...
L'étude de ces approches sort du cadre de notre thèse, le problème traité étant plutôt le couplage d'outils de CAO avec un environnement d'intelligence artificielle.

Certaines contraintes peuvent être satisfaites par une infinité de solutions. L'allocation spatiale par exemple a toujours été un problème fastidieux à résoudre; certains programmes abordent le problème par la conception suivant un raisonnement essai-erreur jusqu'à satisfaction des contraintes imposées; ils procèdent par la génération de solutions à l'aide d'heuristiques prédéfinies; mais il faut surtout pouvoir proposer diférentes solutions.

Dans tous les cas, ces systèmes peuvent apporter une assistance au concepteur dans certaines phases du projet, sans toutefois prétendre l'automatisation de l'acte de conception.

#### 4.4.2.3 Maintien de cohérence: la vérification des contraintes

Le problème de vérification de contraintes est quand à lui plus simple à implémenter; ce qui explique d'ailleurs l'orientation vers les raisonnements du type essai-erreur qui génèrent des solutions pour les tester plus tard.

Une contrainte peut être munie de sa méthode de vérification écrite par le concepteur et qui permet de tester une contrainte et signaler une erreur éventuelle, et dans certains cas simples apporter des solutions ou plutôt des suggestions pour des solutions possibles [GERO J.S. 85], [HOLTZ N. 82].

Dans un système où on impose des contraintes, on ne peut pas s'attendre à maintenir le système dans un état cohérent (toutes les contraintes satisfaites). Encore moins dans la phase de conception où le concepteur intervient souvent et apporte des modifications très importantes [TROUSSE B. 89].

Il faut pouvoir gérer la vérification des contraintes; plusieurs façons sont possibles et sont plus ou moins difficiles à implémenter [BERLANDIER P. 88], [FORNARINO M. et AL 90]:

-manuellement: dans ce cas on déclenche manuellement, à un moment donné, les routines de vérifications; on a alors tout le contrôle et l'on peut choisir les instants critiques.

-automatiquement: dans ce cas aussi on a plusieurs façon de faire:

-prévoir des déclenchements du processus de vérification cycliques qui peuvent intervenir après un certain nombre d'opérations ou après l'écoulement d'un certain temps.
Ce type de vérification est peu intelligent, mais peut être suffisant dans certains cas.

-prévoir des vérifications automatiques plus sophistiquées, où une contrainte peut savoir à tout instant si elle est violée ou si une opération peut la violer et déclencher d'elle même automatiquement le processus de vérification.
On peut penser dans ce cas à des démons liés aux contraintes ou à certains objets de référence qui déclenchent les méthodes de vérification. Soit on peut imaginer un fonctionnement à la manière d'un agenda de blackboard dans lequel on stocke les actions réalisées pour en déduire les vérificateurs à déclencher sachant que certains types de modifications dans certains endroits peuvent violer un certain type de contrainte.

Il faut que le système accepte que la cohérence soit violée durant une certaine phase, dans ce cas toutes les contraintes ne sont pas vérifiées. Certaines contraintes dites faibles pourront être violées momentanément, par contre des contraintes dites fortes ne peuvent être violées à aucun moment.

Le maintien de cohérence au niveau des contraintes entre elles [AYEL M. et AL 86] sort du cadre de la thèse, mais on traite plutôt comment maintenir le projet cohérent avec les contraintes.

#### 4.4.2.4 Contraintes imposées et contraintes inférées

Les contraintes sont généralement imposées par le concepteur:
(surface séjour > 40)
Il existe aussi des contraintes qui peuvent être déduites d'autres:

-soit par la présence de propriétés de relation (symètrie, transitivité,...), par exemple:
cuisine adjacent-à séjour <=> séjour adjacent-à cuisine

-soit par la présence de méta-connaissance sur les contraintes dues à des connaissances implicites, par exemple:

cuisine communique-avec séjour engendre les contraintes suivantes:

cuisine existe
séjour existe
cuisine adjacent-à séjour <=> séjour adjacent-à cuisine
cuisine communique-avec séjour <=> séjour communique avec cuisine

Il faut savoir l'origine de la contrainte (concepteur, sytème, autre contrainte,...); ceci permet de maintenir un arbre ou un graphe de liaison facilitant le maintien de cohérence des contraintes: en effet, supprimer une contrainte devrait supprimer toutes les contraintes qui sont une conséquence d'elle.

#### 4.4.2.5 Les contraintes considérés dans notre essai

Dans notre système on a implémenté des contraintes qui ont surtout rapport à l'espace intérieur du projet (vu l'orientation de notre prototype). on a alors:

-*local* existe: pour dire que *local* existe
-*local1* adjacent *local2:* pour dire que *local1* est adjacent à *local2*
-*local1* communique avec *local2*: pour dire que *local1* communique avec *local2*
-*expression1* > *expression2:* qui peut lier deux expressions (surface local,...) ou une expression à une valeur. L'expression peut être une fonction à évaluer.

Les trois premiers types de contraintes s'appliquent à des locaux, la dernière s'applique à des attributs d'un local.

Une contrainte est une classe d'objet spéciale. Elle contient un champ stockant son type: (existe, adjacent, ...), deux champs qui stockent la partie gauche et la partie droite de l'expression de la contrainte.
Elle contient aussi un champ qui la lie à l'objet auquel elle est appliquée. Si cet objet existe, le lien est un pointeur spatial qui pointe dans le modèle géométrique sur l'objet; par contre, si l'objet n'existe pas le lien est fait par l'intermédiaire d'un symbole permettant de reconnaitre l'objet (pattern-matching).

On s'est limité dans notre application à des contraintes qui expriment des relations unaires ou binaires; les contraintes complexes liant plusieurs variables ne sont pas prises en compte.

### 4.4.3 Exemples d'utilisation

#### 4.4.3.1 Utilisation des contraintes en vérification

Un premier moyen de vérification manuel est implémenté. Chaque type de contrainte ayant une méthode qui permet de la tester, on peut déclencher manuellement la vérification d'une ou de plusieurs des contraintes imposées.

On peut, par exemple, lancer à tout moment la vérification de la contrainte cuisine communique avec séjour. Cette méthode commence par vérifier si les deux locaux existent, puis vérifie si les deux locaux sont adjacents, et finalement elle vérifie s'il existe une porte de communication entre les deux.

D'autre part, on a essayé d'implémenter un autre mécanisme de vérification automatique des contraintes. Celui-ci est basé sur le test de la nature des opérations de manipulations effectuées dans la base géométrique (placer des volumes pleins, placer des volumes vides,...), et de l'endroit où elles sont effectuées (dans l'espace d'un local, dans l'enveloppe dans local, à l'extérieur,...). A partir de cela on peut prévoir quel type de contraintes est susceptible d'être violé.

Certaines contraintes ont été munies de leur propres méthodes de satisfaction de contraintes, comme par exemple la contrainte communique peut être satisfaite si les deux locaux sont adjacents par la création d'une porte ce qui est une tâche simple et peu créative.

Par contre satisfaire la contrainte adjacent n'est pas du tout évident surtout qu'il peut y avoir une infinité de solutions valables. On verra dans le second paragraphe comment on a essayé de résoudre le problème de satisfaction automatique par un moyen de proposition et de génération de solutions suivants certaines heuristiques.

#### 4.4.3.2 Utilisation des contraintes en conception

En conception on peut rencontrer des problèmes de différents types:

-génération d'objets géométriques (mur, local...)
-positionnement (placer un local adjacent à un autre,...)
-automatisation de tâches non créatives (ouvrir une porte,...)

Pour illustrer l'utilisation des contraintes en conception on a essayé d'aborder un problème de conception qui fait appel aux contraintes qu'on a définies (existe, adjacent, communique,...) pour la création et le placement d'un local.

Le problème est résolu par l'implémentation de règles d'un système expert qui permet de proposer plusieurs solutions éventuelles, les stocker et continuer la conception plus tard à partir de n'importe quelle solution.

##### 4.4.3.2.1 Décomposition du problème en sous problèmes

On peut voir la conception comme un processus de satisfaction de buts. On spécifie explicitement et à priori l'enchainement des tâches. Le moteur du système expert devient un interprète d'une structure de contrôle qui parcourt les buts et déclenche pour chacun d'eux des résolveurs de buts qui peuvent être des sous buts, des règles ou des actions complexes. Le processus prend alors un caractère récursif.

La génération d'une solution ne pouvant pas être un parcours linéaire, l'originalité d'un système expert est alors sa capacité de retour arrière au cas de blocage pour la réctification de la trajectoire.

Le système expert peut travailler en chainage avant ou arrière [OXMAN R. et AL 87] :

-en chainage avant, par exemple, à partir de données ou de contraintes sur les dimensions et sur la position du local la solution est construite par les règles.

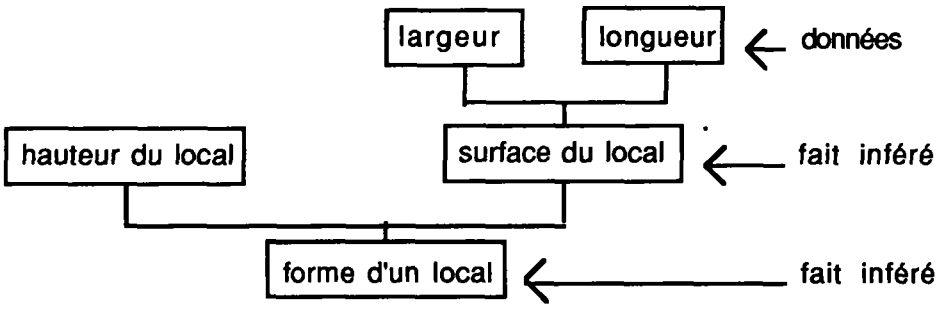



*fig. 4.15 Exemple d'un raisonnement par chainage avant*

-en chainage arrière, le but est de concevoir un local, pour cela on commence par concevoir la volumétrie puis la position, et les sous buts ainsi générés seront décomposés pour arriver à des entités simples à concevoir.

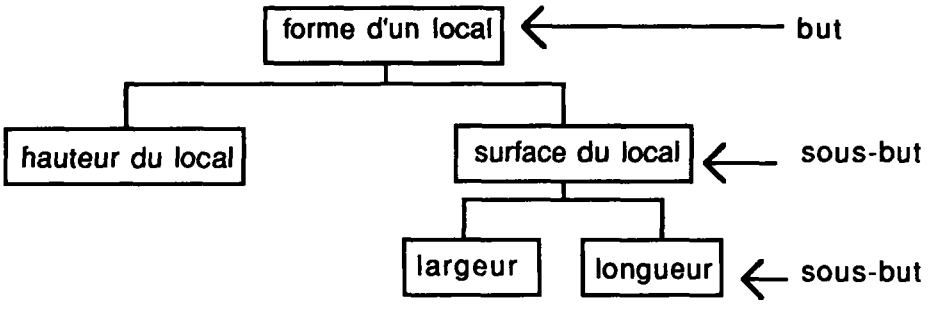



*fig. 4.16 Exemple d'un raisonnement par chainage arrière*

#### 4.4.3.2.2 Scénario de conception

Avec notre problème on a décomposé le problème de création et de placement d'un local en plusieurs sous buts structurés dans un arbre de tâches.

On commence par générer une volumétrie (forme, dimensions,...); Plusieurs solutions sont générées par des fonctions prédéfinies.

Puis on génère des positions potentielles pour le placement en fonction des dimensions et des volumétries déjà présentes. Plusieurs solutions sont aussi calculées.

On combine après les volumétries proposées avec les positions pour générer plusieurs solutions qui seront testées plus tard pour une compatibilité complète avec toutes les contraintes.

A la manière d'un blackboard, on a différents niveaux d'abstraction qui sont les objets créés avant de générer la solution (des objets volumes et des objets

positions,...). Cependant le contrôle n'est pas réalisé par des métaconnaissances mais plutôt par un arbre de tâches.

#### 4.4.3.2.3 Génération et test de solution

La génération des volumétries est faite en essayant de trouver des contraintes imposées aux dimensions du local à concevoir. A partir de ces dimensions, on essaie de générer une volumétrie convenable.

Le raisonnement à ce stade est hypothétique: résoudre un problème même si certaines valeurs ne sont pas connues, le système émet des hypothèses et explore les alternatives.

Les solutions possibles pouvant être infinies (surface du local > 20 peut engendrer une infinité de plans possibles).On applique pour cela des heuristiques pour ne générer que des volumes suivant des modes fixés d'avance (projection en plan carré ou suivant les dimensions d'un rectangle d'or).

On peut évidemment changer à tout moment les heuristiques pour générer d'autres types de volumes.

On peut imposer une volumétrie soit en imposant des dimensions fixes, soit en saisissant graphiquement la volumétrie.

De même, pour les positions possibles on cherche des contraintes imposées à la posistion (orientation, adjacence, communication,...) et des méthodes génèrent suivant des heuristiques des positions possibles. On choisit par exemple des alignements sur les coins ou des insertions dans des coins rentrants; on peut imposer des positions en intervenant directement.

#### 4.4.3.2.4 Maintien d'arbre d'états et de versions

Comme on a plusieurs volumétries générées et plusieurs positions possibles, on essaie de faire une combinaison pour produire toutes les solutions possibles.

Le système expert travaille par génération d'arbre d'états. Chaque état noeud de l'arbre est une version possible, ou un état intermédiaire.

Un état peut générer plusieurs états fils, et chaque fils peut avoir plusieurs fils,... Un arbre d'état est généré à partir d'un état initial, les états fils étant générés suivant un certain ordre imposé par la stratégie qui contrôle le raisonnement. On peut avoir un parcours en profondeur d'abord, ou en largeur d'abord ou suivant une stratégie dite évoluée décrite par le concepteur.

Les solutions sont testées et seules celles qui satisfont toutes les contraintes applicables généreront des états fils.

L'arbre d'états étant stocké on peut revenir à un état en arrière pour repartir (backtrack).

#### 4.4.3.2.5 Interaction et automatisation

La génération automatique des solutions s'appuie sur des heuristiques qui permettent de limiter l'explosion combinatoire de la génération d'une infinité de solutions.

La personalisation du système peut être assuré par la modification des heuristiques appliquées.

On peut, par exemple, changer les règles de génération de volumétrie (volumes en L, ou de forme complexe,...), ou bien modifier les règles de choix des positions (se positionner au milieu d'une paroi, ou sur un niveau décalé,...).

On aura toujours une génération automatique de plusieurs solutions et le maitien d'un arbre de versions.

Le concepteur a toujours le contrôle pour imposer lui-même une solution.

## 4.5 Conclusion

### 4.5.1 Limites et extensions possibles

L'approche que nous proposons permet l'utilisation de modeleurs de CAO et de leurs bases de données géométriques pour le stockage et la restitution d'informations de haut niveau riches en sémantique.

Le principe de l'approche est indépendant de l'outil de CAO utilisé, ce qui laisse le choix entre l'utilisation de modeleurs existants ou l'écriture de nouveaux outils comme c'était notre cas. Il faut cependant noter que la couche de fonctions d'analyseurs est très dépendante de l'implémentation de l'outil et surtout de sa représentation géométrique interne , donc du modèle utilisé; il faut par conséquent réécrire cette couche si on change de représentation.

Notre approche, comme on l'a déjà vu, se caractérise par:

-dynamicité et facilité de maintien de cohérence au niveau des données géométriques.

-modularité et extensibilité faciles à réaliser.

-saisie de données simple vu que la saisie et les modifications peuvent être graphique ou avec des commandes de haut niveau.

-possibilité de représentation multiple, et de prise en compte de points de vue différents de la même donnée pour différents acteurs par l'intermédiaire de fonctions spécialisées.

-des requêtes complexes construites à partir de requêtes simples du langage de requête défini pour l'extraction d'informations non implémentées, ce qui assure la personalisation du système et son extension.

-le concept objet adopté en programmation apporte aussi ses avantages de modularité, extensibilité et compatibilité entre objets différents.

Les points délicats de notre approche sont:

-le choix des fonctions et de l'information retournée.

-l'implémentation des fonctions qui forment la couche de bas niveau et qui dépend de la représentation géométrique adoptée, surtout qu'elles manipulent des notions géométriques et mathématiques assez complexes.

-les informations non géométriques ne sont pas traitées avec le même principe sans pour autant que la compatibilité soit atteinte.

La conception reste un problème difficile à aborder et à automatiser surtout dans la subjectivité de cet acte en fonction du concepteur. Nous croyons néanmoins, que notre système peut apporter certains outils surtout pour le stockage et l'échange de données ainsi que pour le raisonnement.

Pour étendre le modèle il faut l'orienter plus vers des architectures muli-acteurs; le principe de notre approche, avec une base géométrique simple commune à tous les acteurs et des modules de fonctions d'analyse spécialisés pour chaque acteur, parait adapté à ce genre d'architecture; dans notre étude on s'est occupé surtout des espaces intérieurs, il faut essayer d'exploiter le modèle dans d'autres domaines pour pouvoir identifier les fonctions à implémenter et les difficultés qu'elles présentent.

De même, il faut essayer d'enrichir les bases de connaissances pour aborder des sujets autres que celui des espaces intérieurs comme la conception et le positionnement de locaux traité dans notre étude.

# CHAPITRE 5

# Réalisation logicielle

## 5.1 Introduction

La thèse concerne la définition de l'architecture d'un environnement d'aide à la conception; un logiciel de CAO permet de saisir, visualiser et raisonner sur la géométrie d'un projet en le couplant avec des bases de connaissances en bâtiment.

Dans ce chapitre, on va exposer les détails d'implémentation du prototype qu'on a réalisé; ce prototype sert à démontrer la faisabilité des principes d'un "modeleur intelligent" énoncés dans le chapitre précédent. Ce prototype explore aussi les moyens de communication avec l'outil de CAO en vue de son exploitation par couplage avec des modules de systèmes experts.

Ce chapitre concerne surtout les détails d'implémentation et de programmation de l'outil. On parlera dans le chapitre suivant des applications réalisées avec ; ces applications sont des petits exemples d'utilisation des informations tirées du modeleur pour la résolution de certains problèmes nécessitant de l'expertise ou ce qu'on appelle plus généralement de l'intelligence artificielle.

On comence par l'exploration de notre environnement de travail et des outils utilisés; ensuite, on parlera de l'outil de CAO intégré qu'on a devéloppé pour l'application. On expliquera la façon avec laquelle notre implémentation traite:

-les informations sur les objets géométriques
-les informations sur les objets non géométriques
-les informations sur les objets incomplets.

## 5.2 Outils et environnement

L'utilisation d'un environnement de programmation orienté objet présente beaucoup d'avantages au niveau génie logiciel pour le développement d'outils complexes.

L'environnement graphique de multi-fenêtrage est aussi nécessaire pour permettre de visualiser et de travailler avec des vues multiples sur le projet tout en facilitant la construction d'une interface.

X, LE-LISP, AIDA et SMECI sont les outils de développement utilisés et qui remplissent les conditions déjà mentionnées (multi-fenêtrage, programmation orientée objet, boite à outils graphiques, et générateur de systèmes experts permettant de coupler le modèle géométrique avec des bases de connaissances pour le raisonnement.

X est un gestionnaire de fenêtres sous UNIX.

LE-LISP est le langage de programmation qu'on a utilisé pour développer notre système, car c'est le langage natif de SMECI, et le langage de développement dans SMECI est une sur-couche du LE-LISP.

AIDA est une boite à outils qui facilitent la construction d'interfaces dirigés par les événements (boutons, défileurs,...); c'est aussi une sur-couche de LE-LISP qui contient des fonctions pour l'utilisation de composants prédéfinis dans la construction d'applications. L'interface de SMECI est elle-même développée en AIDA.

## 5.2.1 Présentation de SMECI

### 5.2.1.1 Représentation des connaissances en SMECI

En SMECI, les connaissances peuvent être exprimées sous des formes multiples: objets et modèles d'objets, messages entre ces objets, règles, relations et procédures externes. Chaque relation est adaptée à un type particulier de connaissance.

Les objets permettent de décrire à la fois l'état du système, et celui de l'application en cours de traitement. Chaque objet peut être vu comme une structure de données autonome qui porte certaines valeurs qui lui sont propres, et hérite de nombreux comportements génériques définis dans la base de connaissances au niveau de sa catégorie ou de son prototype.

Le langage à objet de SMECI est dérivé de la couche objet microceyx de LE-LISP.

Les règles décrivent la connaissance dynamique du système. Elles peuvent être perçues comme de petits programmes qui contiennent leurs propres conditions d'applicabilité. Elles permettent de modifier les objets de manière procédurale ou opportuniste, et peuvent si nécessaire être contrôlées par d'autres règles, qui, bien qu'écrites dans la même synthaxe sont de vraies méta-règles.

Pour permettre de gérer simultanément plusieurs solutions d'un même problème, SMECI regroupe les objets dans des états. Les raisonnements tenus par les règles engendrent systématiquement des états, qui sont aussi des points de reprise potentiels du raisonnement.

#### 5.2.1.1.1 Représentation des objets

La structuration des connaissances s'effectue en SMECI à l'aide d'un langage de description à trois niveaux: Catégories, Prototypes et Objets:

-Les catégories définissent la structure des objets manipulés;

-Les prototypes définissent et spécialisent les comportements d'objets à l'intérieur de chaque catégorie;

-Les objets décrivent un cas concret.

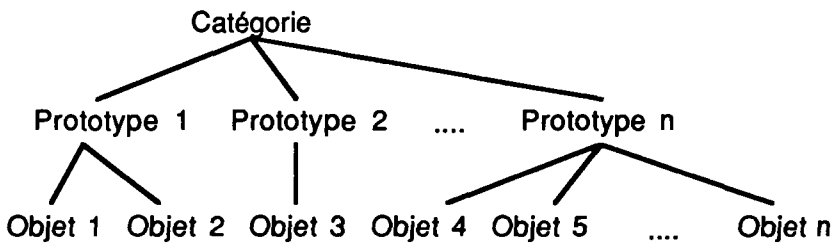



*fig. 5.1 Le langage de description à trois niveaux de SMECI*

#### 5.2.1.1.2 Les catégories

Une catégorie est décrite par un ensemble de champs. Ces champs définissent les propriétés caractéristiques de la catégorie et les liens que ces instances pourront avoir avec d'autres objets. Tous les objets sur lesquels le système raisonne appartiennent à une catégorie, dont ils héritent de tous les champs.

Pour chacun de ces champs, un certain nombre d'informations supplémentaires peuvent être fournies, dans des facettes système ou utilisateur: le type des valeurs autorisées, l'unité et le mode d'héritage dans les prototypes si le type est numérique, la catégorie utilisée, les liens inverses, les types du lien et les importations si le champ est appelé à contenir des objets, par exemple.

L'ensemble de ces descriptions des concepts du domaine de l'application définit les structures de données génériques des objets sur lesquels le raisonnement pourra s'effectuer.

#### 5.2.1.1.3 Les prototypes

Les prototypes servent à décrire un modèle prototypique du comportement des objets. Ils contiennent les valeurs numériques ou symboliques communes à tous les objets d'une catégorie, des informations sur les bornes admissibles des champs numériques, etc...

On peut associer à chaque catégorie une hiérarchie de sous-classes de même structure, qui définit une typologie à l'intérieur de la catégorie. Les propriétés de chaque sous-classe sont décrites à l'aide d'un prototype. Chaque prototype peut redéfinir les méthodes de calcul et les contraintes qui pèsent sur les objets qui lui sont rattachés.

#### 5.2.1.1.4 Les objets

Chaque objet appartient à une catégorie indiquée à sa création, qui définit sa structure. Chaque objet est également associé à un prototype qui définit les messages auxquels l'objet sait répondre ainsi que le domaine de variation de chacun de ses champs.

#### 5.2.1.1.5 Méthodes ou messages

Les catégories et prototypes décrivent les propriétés structurelles des objets. Ils servent également à leur associer des propriétés fonctionnelles. On peut attacher à chaque prototype des fonctions permettant d'effectuer des traitements particuliers

pour les objets de cette catégorie. Ces fonctions écrites en LE-LISP sont appelées des méthodes. Un mécanisme d'héritage permet de retrouver automatiquement la méthode applicable à un objet d'un certain prototype.

La description du monde sous la forme d'objets facilite considérablement le travail par rapport à d'autres techniques de représentations. En effet, il est possible de regrouper toutes les informations pertinentes pour une unique entité sémantique au sein d'une seule structure de données. Il devient aussi possible de créer et de manipuler indépendament un nombre quelconque d'objets différents, instances d'une même catégorie.

#### 5.2.1.1.6 Remarque

Le langage à objet de la version de SMECI sur laquelle on a travaillé est particulier ayant ses avantages et ses inconvénients, (prototypage limité, héritage simple, arbre de classe limité à trois niveaux, les sous classes sont des prototypes et ne peuvent pas être défini par extension d'une classe existante,...).

Les nouvelles versions apportent des remèdes à certains de ces défauts; cependant on a essayé dans notre implémentation de choisir un modèle qui soit indépendant des caractéristiques spéciales de SMECI, ce qui nous permet de garder une structure portable et facile à implémenter dans d'autres systèmes ou d'autres environnements à objets.

### 5.2.1.2.Le raisonnement dans SMECI

Ce chapitre est surtout consacré à l'implémentation du modèle et de l'outil de CAO. On n'utilisera dans ce chapitre que les caractéristiques du langage à objet de SMECI.

Cependant, dans le chapitre suivant consacré aux applications construites autour du modèle, on uilisera les potentiels de SMECI en tant que générateur de systèmes experts pour démontrer les possibilités de couplage du modèle qu'on propose avec un environnement avancé de bases de connaissances.

On se contentera dans ce chapitre d'expliquer le modèle de raisonnement de SMECI dans cette partie résérvée à la description de l'environnement, et on laissera pour le chapitre suivant la description des applications qui utilisent ce type de raisonnement.

#### 5.2.1.2.1 Règles de production

Les catégories et prototypes permettent de modéliser le domaine de l'application. Ils représentent l'outil de description de la connaissance locale propre à chaque type d'objet. Les règles de production contiennent les connaissances expertes sur la résolution du problème. Les règles réalisent les opérations suivantes:

1- filtrage des objets ou des groupes d'objets;
2- assertion des implications logiques de situations reconnues;
3- application d'actions aux objets filtrés.

#### 5.2.1.2.2 Le format des règles

Outre son nom, une règle de production comprend quatre parties:

-des prémisses décrivant les conditions que doivent satisfaire les ensembles d'objets retenus avant d'appliquer la règle;
-des conclusions décrivant les conséquences logiques de la règle sur les objets filtrés par les prémisses;
-des actions indiquant les actions à effectuer lors de l'activation de la règle;
-un commentaire qui associe à la règle un texte libre contenant éventuellement des variables et qui est utilisé par le mécanisme d'explication.

Les formules pouvant apparaitre dans la partie **prémisses** d'une règle constituent un sous-ensemble complet de la logique du premier ordre: on peut utiliser, à tout niveau d'imbrication, les quantificateurs existentiels et universels. On peut par ailleurs utiliser n'importe quel prédicat LISP portant sur les objets ou sur leur champs.

Dans la partie **conclusions** d'une règle, on peut:

-créer ou supprimer des objets;
-déterminer ou modifier les valeurs des champs des objets dans les limites de leurs plages de plausibilité;
-lier plusieurs champs par une formule arbitrairement complexe;
-invalider un état;
-effectuer une action externe (impression, question, appel à une routine externe, etc...).

Les règles SMECI ne permettent pas seulement de filtrer et modifier des objets instances de catégories définies par l'utilisateur, elles offrent de plus la caractéristique de permettre des raisonements sur le contrôle du système, voire sur le raisonnement lui-même. Ces possibilités uniques de méta-raisonnement permettent de gérer très finement le comportement du système.

#### 5.2.1.2.3 Structuration du raisonnement

L'utilisation de règles de production indépendantes pose rapidement le problème du contrôle de leur déclenchement; il faudra alors structurer ces règles en modules de plus haut niveau.

Cette structuration du raisonnement est réalisée en SMECI par des tâches qui, comme les sous-programmes en programmation algorithmique, définissent une vue macroscopique du comportement du système. A chaque tâche est attachée une base de règles qui décrit sa sémantique de façon déclarative.

Un module de connaissance indépendant est capable de résoudre individuellement une partie du problème complet.

Une base de règle est constituée d'une liste ordonnée de règles et d'un ensemble de champs contrôlant leur déclenchement. Ces champs permettent en SMECI de programmer explicitement le contrôle de l'exécution des règles. Ces paramètres

donnent un aspect relativement algorithmique au langage et facilitent l'écriture de structures de contrôle classiques, souvent bien utiles même en intelligence artificielle.

#### 5.2.1.2.4 Le moteur d'inférence

Le raisonnement de SMECI est fondé sur la notion d'état. Un état est défini par les valeurs des champs des objets qui existent, et correspond à un point de reprise potentielle du raisonnement. A tout moment plusieurs états peuvent exister simultanément. Un seul est considéré comme courant.

A chaque cycle le moteur d'inférences reprend le raisonnement à partir du "meilleur" état non encore examiné.

A partir de l'état de départ, le moteur d'inférences engendre dynamiquement un arbre d'états. Cet arbre est construit par application, sur chaque état à examiner, des règles activables associées à la tâche courante. L'ordre dans lequel les états sont examinés est déterminé par un objet stratégie modifiable. Il existe trois prototypes prédéfinis de stratégies, qui proposent une exploration de l'arbre des états en profondeur, largeur ou "meilleur" d'abord.

Le raisonnement de SMECI suit le cycle de base suivant: expansion de l'état courant, puis choix du meilleur état ouvert qui devient le nouvel état courant.

A partir de l'état courant, le moteur, dans la phase d'expansion, cherche à utiliser l'ensemble des règles de la tâche courante, tout en tenant compte des paramètres de contrôle attachés à cette tâche et à sa base de règles. Pour chaque règle applicable, le moteur engendre les états fils obtenus par application de la règle à l'état courant. Chaque nouvel état est ensuite noté par la fonction d'évaluation.

Si aucune des règles de la tâche courante n'est applicable dans l'état courant, le moteur active alors la tâche placée dans le champs "si-blocage" de la tâche courante.

Après l'expansion d'un état, SMECI choisit pour son cycle suivant l'état dont la note est minimale: ce sera le nouvel état courant. Les états non examinés (ou ouverts) restent en concurrence pour les choix d'états à venir. Le moteur d'inférences redcommence alors un nouveau cycle en activant la tâche courante du nouvel état courant.

### 5.2.1.3 Conclusion

L'utilisation de l'environnement SMECI, générateur de systèmes experts a permis de travailler avec le même langage de programmation et d'intégrer l'outil de CAO avec l'environnement de bases de connaissances implémentées en modules expertes.

Cette intégration peut paraitre une facilité idéale, cependant on a essayé dans notre implémentation de travailler tel que l'implémentation reste valable dans un environnement hétérogène; la seule condition est d'avoir un environnement à objet qui facilite le passage d'informations tel que des appels de fonctions externes,...

# 5.3 Les informations géométriques

## 5.3.1 L'outil de CAO intégré

### 5.3.1.1 Introduction

Indépendamment de l'implémentation informatique, l'idée de modeleurs intelligents est soutenue par ce rapport; l'implémentation n'est qu'un détail duquel ne dépend pas les principes énoncés dans le chapitre précédent. Les fonctions d'analyseurs peuvent identifier des entités complexes riches en sémantique à partir d'entités volumétriques géométriques simples.

Le modeleur à utiliser devrait être capable de modéliser des entités géométriques simples. Le modèle géométrique utilisé pour la représentation des données n'est pas imposé; cependant de ce modèle dépendront les fonctions d'analyses.

Le modeleur doit essentiellement être capable de représenter des entités tridimensionnelles où la notion de vide, de plein et de transparent est la seule distinction entre les différents éléments saisis.

Un projet est donc saisi comme un ensemble de vides et de pleins élémentaires ou complexes. On définit ensuite les notions riches en sémantiques tel que paroi ou espace de local à partir des éléments simples existants dans le modèle géométrique.

Exemple: Un local est-il un ensemble de parois? ou bien c'est un espace vide enveloppé? est-il défini par l'enveloppe extérieure ou par le vide intérieur?

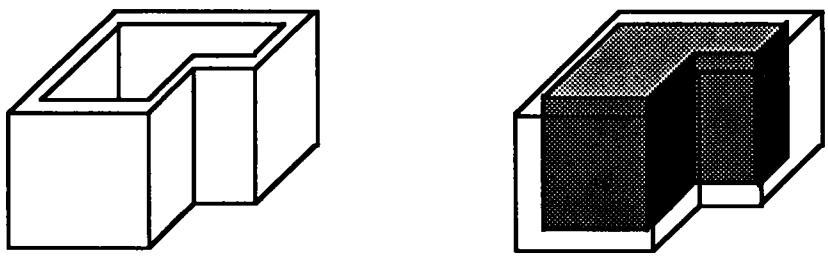


*fig. 5.2 Un local défini par son enveloppe extérieure ou par son vide intérieur*

On peut définir l'espace vide d'un local comme étant un ensemble de vides contigus non séparés par des éléments pleins. Pour détecter l'existence d'un ensemble de vides pareils dans le modèle géométrique plusieurs méthodes peuvent être uilisées dépendamment de la façon avec laquelle sont représentés les volumes élémentaires.

Par exemple, des méthodes de lancer de rayons permettent en partant d'un élément vide de vérifier tous les éléments vides qui lui sont adjacents et ainsi de suite pour chaque vide trouvé; l'ensemble des éléments vides juxtaposés et délimités par des éléments pleins formera le vide du local.

Pour retrouver l'enveloppe de ce local, on partira de l'ensemble des éléments vides définissant l'espace vide du local et on essaiera de retrouver les éléments pleins qui sont en contact direct avec ces vides.

On peut de la même façon, à l'aide d'utilitaires de calcul d'intersection, d'adjacence, d'inclusion ou autres passer des éléments volumétriques simples à la définition d'éléments plus complexes.

#### 5.3.1.2 Le modèle implémenté

Les principes n'imposent pas un modèle de représentation de données géométriques spécial, à part la possibilité d'avoir des entités solides tridimensionnelles. Une représentation en BREP, ou en CSG peut très bien convenir (voir chapitre 1).

N'ayant pas un logiciel de CAO à disposition, auquel on a accès au code source et au modèle de données, on a préféré de développer notre propre modeleur intégré dans l'environnement de SMECI. Cette intégration n'est pas du moins indispensable.

Pour l'exemple implémenté sur la machine, on a choisi une façon de modéliser assez simple en implémentation permettant de représenter facilement le modèle géométrique.

Cette méthode présente quelques inconvénients:

-la limitation à des volumétries à angles droits
-l'espace mémoire énorme occupé par le modèle, ce qui ne permet de modéliser que des projets petits et simples.
-les fonctions d'analyses implémentées sont lentes (lenteur croissante en fonction de la complexité du projet), et le temps de calcul est vite pénalisé pour un grand projet.

Ces inconvénients ne sont pas très apparents dans le cas de démonstrations simples ce qui est le but du prototype envisagé. La simplicité et la rapidité du devéloppement ont été favorisées sur le résultat final qui n'est que démonstratif.

En utilisant le modèle objet de SMECI, à trois niveaux (catégories, prototypes, objets), on a implémenté pour l'outil de CAO différents types de catégories.

la catégorie principale est nommée "univers" et contient la représentation du modèle géométrique du projet. D'autres types de catégories sont implémentés, certains types sont utiles pour le travail de saisie interactif avec le modeleur tel que la catégorie "curseur", ou le panneau de boutons; d'autres sont utiles pour la visualisation tel que la catégorie "fenêtre", "vue", ou "environnement". D'autres moins importants pour l'utilisateur, sont utilisés par le système pour effectuer des opérations tel que la projection; "point3d" et "pixel" en sont des exemples.

Dans notre modèle tout est objet, et le programme fonctionne alors par envoi de messages. On peut envoyer le message **dessiner** à l'objet univers pour visualiser le projet, de même on peut envoyer le message **dessiner** à l'objet curseur pour visualiser le curseur.

Certaines méthodes sont utlisées sans qu'elles ramènent des réponses, comme le message **dessiner**, d'autres ramènent des réponses comme le message **lire** envoyé à l'objet univers qui ramènent une information sur le contenu du projet à un endroit donné.

Dans la suite on essaiera de détailler les différentes catégories utilisées dans notre prototype pour l'outil de CAO et les méthodes qui leur sont associées ainsi que leurs modes de fonctionnement.

On détaillera ensuite les fonctions d'analyses implémentées, qui ne sont autres que des messages envoyés à l'objet univers pour en tirer les informations nécessaires. Un paragraphe spécial leur sera consacré.

Nous parlerons ensuite des autres catégories implémentées qui permettent d'ajouter des informations non géométriques au projet tel que décrit dans le chapitre précédent.

#### 5.3.1.2.1 La catégorie "voxel"

Le modèle simpliste utilisé représente le projet dans un univers cubique constitué d'une matrice tridimensionnelle de cubes élémentaires qui peuvent avoir une texture plein, vide ou transparent (voir le paragraphe "l'approche spatiale" dans le chapitre 1).

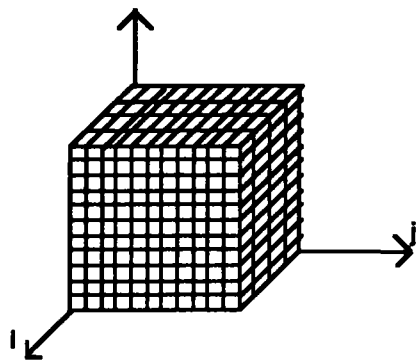



*fig. 5.3 L'univers cubique*

Un cube élémentaire est alors représenté par une catégorie appelée "voxel"; cette catégorie a un seul champs noté **texture**. Ce champs texture peut contenir, dans l'exemple réalisé, un symbole "vide", un symbole "blanc" ou un symbole "transparent".

On a par suite trois objets instances de la catégorie voxel:

-l'objet "blanc" est un voxel dont le champs texture est rempli par le symbole 'blanc.

-l'objet "vide" est un voxel dont le champs texture est rempli par le symbole 'vide.

-l'objet "transparent" est un voxel dont le champs texture est rempli par le symbole 'transparent.

Un objet blanc va représenter un cube élémentaire solide qui fait partie des parois du projet (parois horizontales ou verticales)

Un objet vide va représenter un cube élémentaire vide qui peut faire partie de l'espace intérieur d'un local ou de l'espace extérieur du projet.

Un objet transparent va représenter un cube élémentaire qui fait partie d'une ouverture quelconque (fenêtre ou porte).

Nous rappelons que la seule information qui distingue les différents éléments géométriques du projet, à part leur position dans un référentiel d'axes, est leur texture. Aucune autre information de type sémantique n'est modélisée.

La catégorie "voxel" n'a pas de prototype.

La catégorie "voxel" n'a pas de méthodes associées définies.

**Remarques**

-Seulement trois types de voxels représentent dans un univers toutes les entités; un voxel en fonction de sa texture et de son voisinage peut jouer un rôle différent dans le projet.

Un voxel "vide" peut représenter:

-l'espace intérieur d'un local
-l'espace extérieur du projet

Un voxel "blanc" peut représenter:

-une paroi (mur ou plafond)
-une paroi extérieure ou intérieure
-une paroi orientée (nord, sud,...)

Un voxel "transparent" peut représenter:

-une porte (intérieure ou extérieure)
-une fenêtre (intérieure ou extérieure)
-une porte commune à deux locaux
-une fenêtre orientée nord, ou sud ou ...
-.....

Les fonctions d'analyses permettent donc de chercher les voxels qui peuvent appartenir à une même entité (porte, mur,...) en fonction de la requête adressée. C'est cette caractéristique qui donne à notre modèle la simplicité de saisie et la dynamicité de maintien de cohérence au niveau de la base géométrique; chaque voxel bien qu'il garde la même texture change de fonction dépendament de ses voisins.

-Dans notre exemple, on a choisi trois types de textures de voxel: blanc, vide et transparent pour modéliser le projet. Ces textures ont paru suffisantes pour une étude de faisabilité concernant les espaces intérieurs, les communications et d'autres entités pour lesquels la démo est faite.

Cependant pour des besoins différents, on peut facilement étendre le modèle en ajoutant d'autres objets instances de la catégorie voxel de textures différentes tel qu'un

objet de texture radiateur ou de texture gaine pour représenter respestivement une partie d'un radiateur ou d'une gaine dans le projet.

Notre projet sera alors un ensemble de cubes élémentaires de textures blanc, vide, transparent, radiateur et gaine.

-Le champs texture est rempli par un symbole simple qui est le nom de la texture. On peut changer le type du contenu de ce champs en lui affectant des structures beaucoup plus complexes. Ces structures peuvent être elles-mêmes des instances de catégories décrivant d'une façon plus complète certaines informations concernant la texture tel que épaisseur, couleur, rugosité ou autres...

Dans notre exemple, où l'on traite des espaces intérieurs du projet, les informations minimales sur les textures étaient suffisantes.

#### 5.3.1.2.2 La catégorie "univers"

Le modèle géométrique du projet est donc une matrice tridimensionnelle de voxels qui peuvent avoir une texture blanc, vide et transparent.

Cette matrice est indicée i, j, k et chaque cube élémentaire est référencé par un triplet (i, j, k). La matrice est implémenté comme un vecteur à trois dimensions.

La catégorie "univers" est définie pour contenir la matrice du projet. Cette catégorie a les champs suivants:

-le champs **largeur**:
il contient un entier désignant la largeur du projet. cette largeur représente la dimension de la matrice de voxels en pixels sur l'écran. La largeur unitaire d'un voxel est imposé dès le départ.

-le champs **nbremax**:
ce champs est une méthode qui peut ramener le nombre total de voxels que peut contenir la matrice.

-les champs **imax, jmax, kmax**:
ces trois champs contiennent respectivement les dimensions maximales de la matrice dans les sens respectifs i, j, k.

-le champs **matrice3d**:
ce champs contient un pointeur sur le vecteur qui représente la matrice de voxels en mémoire.

La catégorie "univers" a présentement deux prototypes:

-univers_10
-univers_20

univers_10 correspond à un univers où l'on a forcé les valeurs imax, jmax, kmax respectivement à 10x10x10.

univers_20 correspond à un univers où l'on a forcé les valeurs imax, jmax, kmax respectivement à 20x20x10.

Pour pouvoir représenter d'autres types de matrices, il faut créer le prototype correspondant afin de pouvoir l'instancier.

**Les instances de la catégorie univers**

Plusieurs objets univers peuvent être créés, manipulés et stockés durant une session de travail.

Deux instances particulières sont créés et initialisés par le système au moment du lancement de l'outil: l'objet "ur", et l'objet "uf".

L'objet "ur" pour univers réel contiendra la matrice du projet à modéliser.

L'objet "uf" pour univers fictif sert à stocker la réponse que peut ramener une fonction d'analyse lorsqu'elle est appliquée à un autre univers comme "ur" par exemple.

On peut instancier autant d'objets univers nécessaires dans lesquels on peut stocker des matrices réponses à des requêtes d'analyse.

L'objet univers reçoit des messages pour s'afficher, se déplacer, s'initialiser, manipuler la matrice,...

**Les méthodes associées à la classe univers**

Nous exposons dans ce paragraphe les méthodes associées à la catégorie univers qui sont utilisées par l'outil de CAO. On traitera dans un autre paragraphe les méthodes qui sont des fonctions d'analyses.

Un objet univers reçoit des messages pour lire et écrire dans sa matrice, pour se dessiner, ou pour manipuler ses champs,...

-la méthode **init (u)**:
elle prend comme argument l'univers à qui s'adresse le message et permet de réinitialiser la matrice de ce dernier en affectant son champ matrice3d avec un nouveau pointeur sur une matrice à trois dimensions où tous les voxels ont pour valeur l'objet "vide".

-la méthode **lire (u i j k)**:
envoyé à un univers avec i, j, k comme paramètres, permet de ramener la valeur du voxel référencé par i, j, k dans la matrice de l'univers. La réponse est un objet de type "voxel".

-la méthode **écrire (u i j k voxel)**:
envoyé à un univers avec i, j, k, et un voxel comme paramètres permet de modifier la valeur du voxel référencé par i, j, k en lui donnant la valeur du voxel passé en paramètre.

Les méthodes lire et écrire sont les fonctions de base les plus élémentaires utilisées pour manipuler ou consulter la matrice d'un univers.

A l'aide de ces fonctions d'autres méthodes de plus haut niveau sont créées pour manipuler de façon plus complexe l'univers.

-la méthode **créer-paroi (u long larg haut)**:
elle permet de remplir un parallélépipède de longueur, largeur et hauteur passées en paramètre par des voxels blancs.

Les dimensions passées en paramètre sont des valeurs qui représentent un nombre de voxels élémentaires.

La méthode ne vérifie pas, mais au niveau de la saisie il faut s'assurer qu'au moins une des dimensions vaut 1, car les parois reconnues par notre modèle ont par convention une épaisseur uniforme de un voxel (voir paragraphe les hypothèses de départ).

-la méthode **créer-ouverture (u long larg haut)**:
Elle permet de remplir un parallélépipède de longueur, largeur et hauteur passés en paramètre par des voxels transparents.

Les dimensions passées en paramètre sont des valeurs qui représentent un nombre de voxels élémentaires.

La méthode ne vérifie pas, mais au niveau de la saisie il faut s'assurer qu'au moins une des dimensions vaut 1, car les ouvertures reconnues par notre modèle ont par convention une épaisseur uniforme de un voxel (voir paragraphe les hypothèses de départ).

-la méthode **créer-local (u long larg haut)**:
envoyé à un univers permet d'écrire plusieurs voxels blancs suivant les paramètres long, larg et haut; cette fonction remplit le parallélépipède de longueur long, de largeur larg, et de hauteur haut par des voxels blancs; le coin bas à gauche du parallélépipéde est la position du curseur. Puis elle remplit un parallélépipède à l'intérieur du premier par des voxels vides en ne laissant alors qu'une seule épaisseur de voxels comme enveloppe du local. On verra plus tard que l'une des hypothèses de départ est que les parois ont une épaisseur uniforme d'un voxel.

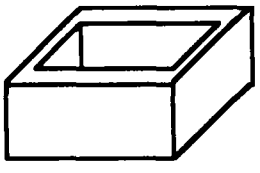


*fig. 5.4 La création d'un local*

Notre outil de CAO, quand il crée un local, une paroi ou une ouverture, il se contente d'écrire dans la matrice de l'univers réel les voxels correspondants sans stocker l'information local ou paroi ou fenêtre...

C'est aux fonctions d'analyse de restituer, comme on le verra plus loin, ce type d'information.

-la méthode **dessiner-persp (u imin jmin kmin imax jmax kmax)**: elle permet de dessiner tout ou une partie de l'objet univers en perspective cavalière dans une fenêtre à l'écran.

-la méthode **dessiner-plan (u imin jmin kmin imax jmax kmax)**: elle permet de dessiner tout ou une partie de l'objet univers en plan dans une fenêtre à l'écran.

Seulement deux types de projections ont été implémentées pour la démonstration, on peut ajouter facilement d'autres types de projection en fonction de nos besoins.

-la méthode **copier (usource udest imin jmin kmin imax jmax kmax)**: elle permet de copier tout ou une partie de la matrice d'un univers source dans celle d'un univers destination.

-la méthode **transformer (u quoi enquoi imin jmin kmin)**: elle permet de transformer tous les voxels d'une texture donnée (quoi) en d'autres voxels d'une texture différentes (en quoi).

#### 5.3.1.2.3 La catégorie "curseur"

Le curseur est un parallélépipède fictif qui a une dimension variable et qui peut se déplacer dans la matrice de l'univers. La taille minimale que peut avoir un curseur est la taille d'un voxel. La taille maximale est celle de l'univers; entre ces deux tailles il peut prendre toutes les tailles intermédiaires.

Le curseur ne peut pas sortir en dehors de l'univers.

La catégorie "curseur" a les champs suivants:

-les champs **coini, coinj, coink**:
Ce sont les indices i, j, k référençant l'emplacemet du coin le plus bas à gauche du parallélépipède du curseur; si le curseur a la taille minimale d'un voxel c'est alors la référence de ce voxel dans la matrice de l'univers.

-les champs **aretei, aretej, aretek**:
Ce sont, respectivement, les dimensions (en nombre de voxels) du parallélépipède représentant le curseur dans le sens des i, j, k. Ils contiennent la valeur 1, 1, 1, dans le cas d'un curseur de la taille d'un voxel.

La catégorie "curseur" n'a pas de prototype.

Une seule instance de la catégorie curseur existe par session. L'objet s'appelle aussi curseur.

**Les méthodes associées à la catégorie "curseur"**

-la méthode **dessiner (curseur u)**:
elle permet de visualiser en 2D ou en 3D sur l'écran l'objet curseur à un moment donné.

-les méthodes **déplacer-devant, déplacer-derriere, déplacer-droite, déplacer-gauche, déplacer-haut, déplacer-bas (curseur u):**
elles permettent de changer la valeur des champs coini, coinj, coink pour déplacer le curseur dans la matrice de l'univers. Ces fonctions incrémentent ou décrémentent de 1 les champs correspondants.

Le sens de déplacement devant augmente l'indice coini;
Le sens de déplacement derrière diminue l'indice coini;
Le sens de déplacement droite augmente l'indice coinj;
Le sens de déplacement gauche diminue l'indice coinj;
Le sens de déplacement haut augmente l'indice coink;
Le sens de déplacement bas diminue l'indice coink.

Dans la fenêtre de projection ces mouvements représentent le déplacement dans le sens réel visible à l'écran.

-la méthode **déplacer-curseur (curseur u):**
elle permet de déplacer interactivement à l'aide de la souris le curseur en faisant appel aux fonctions définies avant.

A tout moment le curseur peut modifier le projet en écrivant dans la matrice de voxels. on peut transformer le parallélépipède du curseur en voxels blancs, vides ou transparents par simple appui sur les touches "p", "v", ou "t" du clavier.

Le curseur est un moyen graphique de bas niveau qui permet de manipuler la matrice de voxels au voxel près.

On peut, bien sûr construire une paroi en remplissant des voxels contigus saisis à l'aide du curseur, et aussi construire un local en remplissant convenabement les voxels désirés.

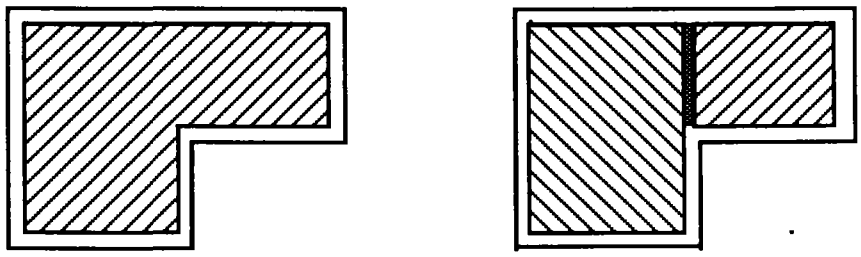


*fig. 5.5 L'ajout d'une paroi divise un local en deux*

En effet, il suffit d'ajouter avec le curseur une paroi pour créer un deuxième local dans l'univers.

-la méthode **accrocher-curseur (curseur u):**
elle permet de modifier la taille du curseur en augmentant interactivement sur l'écran la valeur des champs aretei, aretej, aretek.

#### 5.3.1.2.4 Les autres catégories et objets de l'outil de CAO

L'implémentation de l'outil de CAO est basée sur la manipulation d'objets SMECI.

La catégorie univers qui définit la matrice du projet est celle qui contient le modèle du projet.

D'autres catégories d'objets sont définies pour l'interface et pour la saisie, la manipulation et la visualisation de l'univers.

L'objet curseur se déplace dans la matrice de l'objet univers pour désigner des cubes élémentaires en les référençant à l'aide des indices i, j, k.

D'autres objets sont créés et ne sont pas utiles pour l'utilisateur du système, ils servent plutôt pour exécuter les fonctions et les messages:

**La catégorie "point"**

La catégorie "point" représente un point qui a trois coordonnées dans un repère orthonormé. Cette catégorie a trois champs:

-les champs **x, y, z:**
Ils représentent respectivement l'abscisse, l'ordonée et la cote d'un point.

La catégorie point a deux prototyes:

-point3d
-pixel

Le prototype point3d n'impose pas de restriction aux coordonnées du point; par contre le prototype pixel représente un point en 2D où le champs z est forcé à la valeur "nil".

Il existe huit objets instances du prototype point3d nommé: point_0, point_1, point_2, point_3, point_4, point_5, point_6, point_7. Ces points représentent les sommets d'un parallélépipède rectangle.

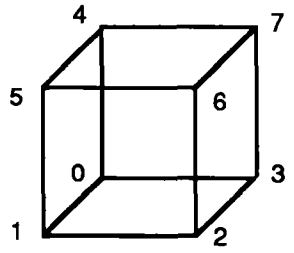



*fig. 5.6 L'ordre de numérotation des sommets d'un cube3d*

Le prototype pixel a aussi huit instances: pixel_0, pixel_1, pixel_2, pixel_3, pixel_4, pixel_5, pixel_6, pixel_7. Ces pixels représentent respectivement les sommets d'un polygone projection des point3d de même nom dans la fenêtre de projection.

La catégorie point a une méthode associée:

-la méthode **projeter (p3d p2d)**:
elle prend deux paramètres: le point3d (p3d) à projeter et le pixel (p2d) qui contiendra la réponse. la fonction remplit les champs du pixel par les valeurs des projections planes ou en perspective cavaliere déduites de la projection du point3d qui reçoit le message.

**La catégorie "cube3d"**

Cette catégorie représente un parallélépipède fictif en 3D; elle a un champ appelé **sommets** qui contient une liste des points 3D qui sont les sommets du parallélépipède.

Un seul objet cube3d instance de la catégorie cube3d existe; il contient comme liste de sommets les huits objets point3d instances de la catégorie point de prototype point3d.

**Les méthodes associées à cube3d**

-La méthode **projeter (cube3d)**:
elle est associée à la catégorie cube3d envoyée en message à un cube3d dont la liste de sommets est connues, ramène dans l'objet cube2d (voir paragraphe suivant) la liste des sommets du polygone projection de ce cube 3d (elle envoie bien sûr la méthode projeter à chaque point sommet du cube3d).

**La catégorie "cube2d"**

Cette catégorie représente le polygone projection d'un cube3d; elle a un champ appelé **sommets** qui contient une liste des pixels 2d qui sont les sommets du polygone projection du cube.

Un seul objet cube2d instance de la catégorie cube2d existe. il contient comme liste de sommets les huits objets pixel instances de la catégorie point de prototype pixel.

**Les méthodes associées à cube2d**

-La méthode **dessiner (c)**:
elle permet de dessiner l'objet cube2d qui reçoit le message.

**Les catégories "fenêtre", "vue" et "environnement"**

La catégorie "fenêtre" a les champs suivants:

-Les champs **x, y**:
ils contiennent respectivement la position du point en haut à gauche sur l'écran

-Les champs **w, h**:
ils contiennent respectivement la largeur et la hauteur de la fenêtre

-Le champ **pointeur**:
il contient le pointeur qui identifie dans AIDA la fenêtre pour pouvoir tracer dedans (fenêtre courante).

-Le champ **univers**:
il contient le nom de l'univers qui sera dessiné par défaut dans cette fenêtre.

-Le champ **projection**:
il contient le type de projection utilisé par défaut par la fenêtre. Deux types de projection sont implémentés: **plan** pour une vue de dessus en 2D, et **cavalière** pour une perspective en 3D.

On peut créer à chaque session autant d'objets instances de la classe "fenêtre" qu'on a besoin pour la visualisation.

La catégorie "vue" contient les champs suivants:

-Les champs **imin, jmin, kmin, imax, jmax, kmax**:
ils contiennent les valeurs minimales et maximales de la partie de la matrice de voxel qui peut être visualisée.

Un seul objet instance de la catégorie "vue" existe par session, on peut cependant étendre le modèle pour avoir une instance par fenêtre.

La catégorie "environnement" contient des informations d'ordre général comme la fenêtre courante, les panneaux de l'application ...

Un seul objet instance de la catégorie "environnement" existe par session.

Les objets fenêtre, vue, environnement, ... sont utilisés pour la visualisation.

## 5.3.2 Les fonctions d'analyse géométriques

### 5.3.2.1 Fonctions de saisie et fonctions de requêtes

Notre modèle de "modeleur intelligent" repose sur le principe de fonctions d'analyses. La base de données géométriques est simple et ne contient que des informations géométriques.

Les informations géométriques sont saisies avec des fonctions élémentaires (à l'aide du curseur), ou avec des fonctions complexes tel que "créer-local".

Les fonctions de saisies ne stockent aucune information d'ordre sémantique dans le modèle géométrique. Le rôle des fonctions d'analyses sera alors d'extraire les informations demandées par l'utilisateur à la manière de requêtes adressées à une base de données.

Toutes les fonctions implémentées sont des méthodes associées à la catégorie "univers". L'objet univers qui reçoit les messages sait répondre à la requête.

Généralement l'objet univers qui reçoit la plupart des requêtes est l'univers réel (i.e. l'objet "ur"), car c'est la matrice de ce dernier qui contient tout le projet.

Les réponses à ces messages peuvent être des valeurs booléennes vrai ou faux (le local cuisine est-il adjacent à séjour?), des valeurs numériques (surface d'un local), des objets géométriques (la facade nord du local séjour), ou d'autres types d'objets...

Dans le cas où le retour est un objet géométrique, celui-ci est ramené par défaut dans une matrice qui a la même dimension que la matrice de l'univers réel et qui contient la réponse à la requête; les voxels de cette matrice correspondent à la réponse.

Cette matrice se trouve par défaut dans l'objet "uf" (pour univers fictif); on peut à tout moment stocker cette matrice dans un autre objet univers créé spécialement en utilisant la fonction **"stocker-résultat-dans"**. On peut par la suite adresser une requête à ce nouvel objet univers pour simuler des appels de fonctions emboités.

### 5.3.2.2 Choix des fonctions

Le choix des fonctions d'analyses était dépendant du but final de la démonstration. On avait envisagé, dès le départ, de démontrer la faisabilité de nos principes sur les modeleurs intelligents dans le domaine de l'architecture et surtout celui des espaces intérieurs et des communications, ainsi que des entités tel que paroi, porte ou fenêtre...

Les fonctions d'analyses implémentées jusqu'à présent ont pour but de reconnaitre ce type de données dans un univers géométrique.

On démontre aussi dans notre modèle la possibilité de combiner les fonctions de bases implémentées. Les fonctions simples de base peuvent être combinées pour construire des fonctions plus complexes permettant de ramener des informations plus spécifiques.

Par un choix de départ, on a adopté une définition particulière pour chaque entité utilisée en lui donnant un sens au niveau sémantique et géométrique particulier.

Cette définition peut paraitre subjectif ou en contre-sens avec la vue d'un autre utilisateur. On a exposé dans le chapitre précédent l'extensibilité de notre approche, où le modèle peut être personnalisé. Chaque utilisateur peut implémenter la fonction d'analyse qui lui correspond.

Pour pouvoir utiliser correctement le système, il faut bien décrire la définition adoptée pour l'implémentation de la fonction afin de pouvoir utiliser ou modifier en fonction de la réponse qu'on attend du système.

### 5.3.2.3 Les fonctions de bas niveau et de haut niveau

Dans l'exemple qu'on a implémenté, tout est objet instance de classes (catégories en SMECI); les fonctions implémentées sont alors des méthodes associées à des classes.

Le programme fonctionne surtout par envoi de messages; comme on l'a déjà vu avec les fonctions de saisie associées à la catégorie "univers", ou la catégorie "curseur",...

Les requêtes sont aussi implémentées sous forme de méthodes associées à la catégorie "univers"; ainsi, un objet de type "univers" peut recevoir des messages auxquels il sait répondre en renvoyant des valeurs ou des objets; on parlera alors de fonctions de bas niveau.

La synthaxe d'envoi de messages ainsi que le passage de paramètres multiples rendait l'utilisation de l'interface assez complexe. Pour simplifier l'utilisation de l'outil on a alors créé des fonctions de plus haut niveau dont la syntaxe d'appel est simplifiée et assez significative et où les paramètres sont simplifiés ou réduits en prenant des valeurs par défaut, surtout pour ceux qui sont connus.

Par exemple, une fonction de requête est souvent envoyé à l'objet "ur" (i.e. univers réel qui stocke la modélisation du projet); les méthodes associées à la catégorie univers possèdent alors une autre définition où l'on considère implicitement qu'on adresse le message à "ur", et que la réponse, si elle est géométrique, sera stockée par défaut dans "uf" (i.e. l'univers fictif).

Dans ce paragraphe on passera en vue toutes les fonctions de bas niveau implémentées, et on mentionnera, quand elles existent, les fonctions d'interface de plus haut niveau à appel simplifié correspondantes.

On décrira pour chacune des fonctions la syntaxe d'appel complète; on expliquera les paramètres d'entrées et de sorties; on expliquera aussi brièvement l'algorithme utilisé, ainsi que ses possibilités d'extension ou de modification pour certains cas.

#### 5.3.2.4 Les entités définies

On commencera par exposer les entités définies par rapport à un local donné; puis nous parlerons de ces mêmes entités au niveau du projet: on parlera par exemple de parois nord du séjour et des parois nord du projet.

Les entités qu'on a essayé de définir dans notre prototype ne sont pas nécessairement ceux qui correspondent à un cas du monde réel; on rappelle que notre prototype n'a pas l'ambition d'être l'outil à utiliser dans des cas réels, ni il ne prétend traiter tous les cas de figure.

C'est en revanche une application qui a été écrite dans le seul but de démontrer la faisabilité et l'implémentation possible des principes de ce qu'on a appelé dans notre étude "les modeleurs intelligents".

En conséquent, certaines restrictions ou conventions de départ ont été adoptées pour faciliter la tâche d'implémentation mais qui ne sont pas un défaut qui remet en cause les principes fondamentaux.

#### 5.3.2.5 Les hypothèses de départ

Les restrictions de départ qu'on s'est imposé nous ont permis de:

-simplifier l'implémentation de l'outil de CAO
-simplifier les algorithmes de recherche
-réduire le temps de calcul

Ayant choisi d'implémenter un outil de CAO qui puisse nous servir à démontrer nos propositions, on a choisi une représentation particulière (l'énumération spatiale) qui ayant ses inconvénients, nous a permis, moyennant certaines restrictions d'appuyer notre thèse par un prototype opérationnel.

Le choix de représentation géométrique par énumération spatiale, bien qu'il a simplifié l'implémentation de l'outil de CAO, nous a imposé les restrictions suivantes:

-le projet doit être de taille réduite (la taille de la matrice étant critique pour les algorithmes de recherche et pour la taille de la mémoire).

Pour cette restriction on peut apporter des solutions soit en choisissant un autre type de représentation d'entités solides tel que BREP, ou CSG, soit en utilisant pour la matrice une autre représentation plus compactée qu'un vecteur à trois dimensions tel que par arbre octal ou OCTREE (voir chapitre 1).

-le projet est limité à des volumes à angles droits; la représentation par cubes élémentaires ne permet de saisir que des entités à angles droits.

De même on peut apporter un remède à cet inconvénient soit par l'utilisation d'un autre type de représentation tel que BREP ou CSG, soit par extension du modèle de la matrice vers des entités élémentaires non cubiques ou facettisées.

D'autres types de restrictions sont imposées pour simplifier les algorithmes des fonctions d'analyse en temps d'implémentation ainsi qu'en temps d'exécution:

-L'épaisseur d'une paroi est uniforme et égale à un voxel
-les façades sont toujours orientées suivant une direction principale (Nord, Sud, Est ou Ouest).

Ces restrictions peuvent être plus facilement dépassées par l'utilisation d'algorithmes plus généralisés, optimisés et ceci au prix d'un temps de développement et de calcul que l'on n'a pas jugé essentiel pour l'implémentation de notre prototype qui sert d'outil de démonstration des principes du modeleur intelligent et non pas d'un outil fini destiné à une utilisation concrète dans des cas réels.

#### 5.3.2.6 Les entités définies pour un local

##### 5.3.2.6.1 L'espace intérieur d'un local

Dans notre modèle, la notion d'espace intérieur d'un local est fondamentale. Elle permet de déduire d'autres types d'informations.

On a défini l'espace intérieur d'un local comme un ensemble de voxels vides contigus (i.e. ayant deux facettes en contact).

En partant d'un voxel de départ, n'importe quel voxel "vide", on cherche les voxels vides contigus en testant que la référence du voxel ne sort pas de la limite des dimensions de la matrice de l'univers.

Dans le cas où l'on touche aux limites de l'univers, l'espace est considéré infini, non fermé, et ne peut pas par la suite représenter l'espace intérieur d'un local.

Si l'espace trouvé est fini, donc ne touche pas aux limites de l'univers, l'ensemble des voxels vides ainsi trouvés représente l'espace intérieur d'un local. Cet ensemble sera nécessairement entouré par des voxels "blanc" ou "transparent" qui forment l'enveloppe.

La méthode **chercher-espace-local (u i j k)**:

C'est une méthode associée à la catégorie "univers" qui permet de trouver l'espce vide d'un local.

**-les paramètres:**

-u, est un objet de catégorie "univers" quelconque; souvent c'est l'objet "ur" (l'objet contenant la matrice qui représente le projet) qui reçoit le message.

-i, j, k sont les indices dans la matrice tridimensionnelle d'un voxel "vide" à partir duquel la recherche d'un espace doit démarrer.

**-Le retour de la méthode:**

La fonction retourne par défaut dans la matrice de l'objet "uf" les voxels qui représentent l'espace intérieur du local pointé.

On peut stocker la matrice de retour à l'aide de la fonction **stocker-résultat-dans** dans un autre objet de type "univers".

**-remarques:**

-la méthode teste seulement si l'espace ramené est fini ou infini; elle ne teste pas par contre si l'espace retourné est un espace valable ou pas.

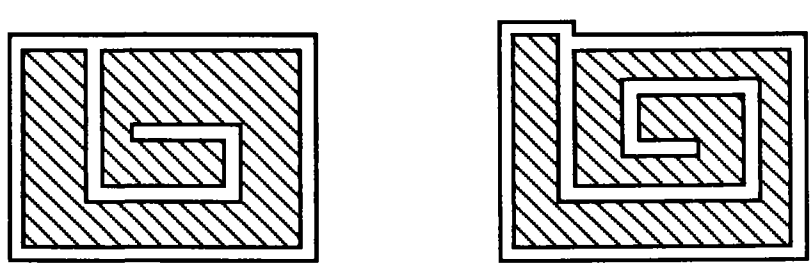


*fig. 5.7 Des locaux labyrinthes*

Dans l'exemple ci-dessus, la fonction ramène la partie hachurée qui représente un labyrinthe plutôt qu'un local.

-La méthode implémentée ne tient compte que des textures blanc, vide, et transparent.

Pour étendre le modèle afin d'inclure d'autres types de textures de voxels tel que radiateur ou gaine, il faut faire attention au retour de la fonction; car telle qu'elle est implémentée elle retourne le vide du local auquel elle soustrait les voxels radiateurs.

On peut réécrire une autre fonction qui ramène dans l'espace intérieur le volume occupé par un voxel radiateur car il sera considéré comme faisant partie de l'espace.

#### 5.3.2.6.2 L'enveloppe d'un local

Ayant défini l'espace intérieur d'un local (un ensemble de voxels "vide"), on peut facilement en déduire l'enveloppe du local: c'est en effet les voxels de textures "blanc" ou "transparent" qui sont en contact au moins avec un voxel "vide" de l'espace intérieur d'un local.

Ainsi la fonction d'analyse commence par chercher l'espace intérieur du local puis elle cherche à partir de l'ensemble de voxels vides trouvés ceux qui forment l'enveloppe du local.

-La méthode **chercher-enveloppe-local (u i j k)**:

C'est une méthode associée à la catégorie "univers" qui permet de chercher l'enveloppe d'un local:

**-les paramètres:**

-u, est un objet de catégorie "univers" quelconque; souvent c'est l'objet "ur" (l'objet contenant la matrice qui représente le projet) qui reçoit le message.

-i, j, k sont les indices dans la matrice tridimensionnelle d'un voxel "vide" à partir duquel la recherche de l'espace intérieur du local doit démarrer.

**-Le retour de la méthode:**

La fonction retourne par défaut dans la matrice de l'objet "uf" les voxels qui représentent l'enveloppe du local pointé.

On peut stocker la matrice de retour à l'aide de la fonction **stocker-résultat-dans** dans un autre objet de type "univers".

**-remarques:**

-Pour limiter le temps de calcul sur des machines non performantes, on a simplifié la définition d'une enveloppe en ne prenant en considération qu'une seule couche de voxels entourant l'espace intérieur, c.à.d. en se limitant à des enveloppes d'épaisseur un voxel.

L'extension de l'algorithme est simple mais pénalise le temps de réponse des fonctions, et en première phase on a adopté le modèle simplifié pour démontrer la faisabilité. En effet, il suffit, ayant trouvé la première couche de voxels de l'enveloppe, de chercher des voxels adjacents aux premiers tout en prenant en compte le sens de l'épaisseur d'une paroi.

#### 5.3.2.6.3 Les parois d'un local

Ayant l'enveloppe d'un local, on en déduit ce qu'on a appelé les parois d'un local. Ce sont les voxels de textures "blanc" qui forment l'enveloppe d'un local.

-La méthode **chercher-parois-local (u i j k)**:

C'est une méthode associée à la catégorie "univers" qui permet de chercher les parois d'un local:

**-remarques:**

même description que la méthode "chercher-enveloppe-local" précédente sauf que l'algorithme ne retient de l'enveloppe du local que les voxels de texture "blanc".

L'appelation paroi englobe alors les parois horizontales et verticales.

On utilise les types suivants pour marquer la différence entre les sous-types mentionnés ci-dessus:

On appele **mur** une paroi dont les voxels"blanc" sont supperposés verticalement; un mur devient alors l'ensemble des voxels contigus dont au moins une facette verticale est en contact avec un voxel "vide".

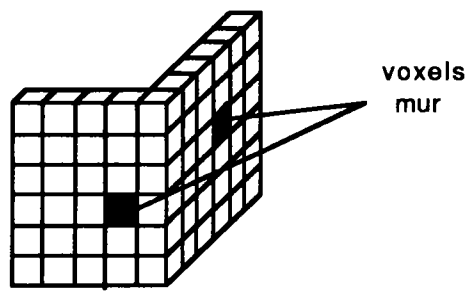



*fig. 5.8 Des voxels "mur"*

On appelle **plafond** ou **plancher** un ensemble de voxels "blanc" contigus dont au moins une facette horizontale est en contact avec un voxel "vide".

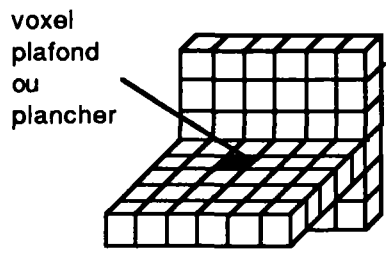



*fig. 5.9 Des voxels "plafond" ou "plancher"*

-La méthode **chercher-mur (u)**:

**-remarques:**

La méthode chercher-mur est un type spécial de méthode; elle ne cherche pas un mur dans la matrice du projet; elle est écrite pour distinguer dans une paroi les éléments verticaux ou mur (on rappelle que la méthode chercher-parois-local ramène les parois horizontales et verticales); Pour cela cette méthode suppose que la paroi dans laquelle il faut chercher est stockée dans "uf". Une fois lancée, la méthode chercher-mur ne garde des voxels de la paroi que ceux qui représentent un mur (donc ceux qui sont juxtaposés en position verticale).

-La méthode **chercher-plafond-et-plancher (u)**:

**-remarques:**

La méthode chercher-plafond-et-plancher est un type spécial de méthode; elle ne cherche pas un plafond ou un plancher dans la matrice du projet; elle est écrite pour distinguer dans une paroi les éléments horizontaux ou plafond et plancher (on rappelle que la méthode chercher-parois-local ramène les parois horizontales et verticales); Pour cela cette méthode suppose que la paroi dans laquelle il faut chercher est stockée dans "uf". Une fois lancée, la méthode chercher-plafond-et-plancher ne garde des voxels de la paroi que ceux qui représentent un plafond ou plancher (donc ceux qui sont juxtaposés en position horizontale).

-La méthode **chercher-plafond (u)**:

-La méthode **chercher-plancher (u)**:

**-remarques:**

Ces deux méthodes ressemblent à la précédente; la seule différence est que chacune ramène un type bien défini.

On appelle plafond un voxel qui a le vide du local en dessous.
On appelle plancher un voxel qui a le vide du local en dessus.

#### 5.3.2.6.4 Les ouvertures d'un local

Ayant l'enveloppe d'un local, on en déduit ce qu'on a appelé les ouvertures d'un local. Ce sont les voxels de textures "transparent" qui forment l'enveloppe d'un local.

-La méthode **chercher-ouvertures-local (u i j k)**:

C'est une méthode associée à la catégorie "univers" qui permet de chercher les ouvertures d'un local:

**-remarques:**

même description que la méthode "chercher-enveloppe-local" précédente sauf que l'algorithme ne retient de l'enveloppe du local que les voxels de texture "transparent".

L'appelation ouverture englobe alors les ouvertures horizontales et verticales. Dans notre modèle on a négligé le traitement des ouvertures horizontales.

D'autre part, une ouverture peut être une porte ou une fenêtre; là aussi on a adopté les conventions suivantes:

On appelle **porte** une ouverture dont le plus bas niveau est à la hauteur du plancher du local.

On ne teste pas à priori si les dimensions d'une ouverture appelée porte correspondent bien aux dimensions réglementaires d'une porte.

On appelle **fenêtre** une ouverture dont le plus bas niveau est plus élevé que le niveau intérieur du plancher du local.

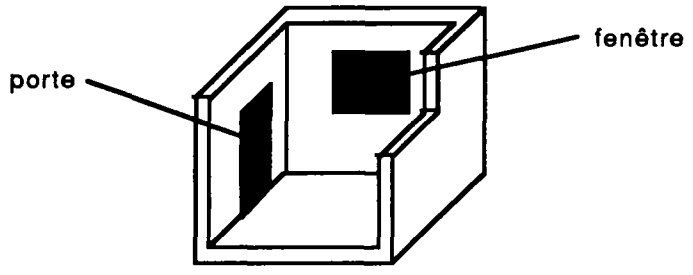



*fig. 5.10 La différence entre une porte rt une fenêtre*

-La méthode **chercher-fenetres-local (u i j k)**:

C'est une méthode associée à la catégorie "univers" qui permet de chercher les fenêtres d'un local:

**-remarques:**

même description que la méthode "chercher-ouvertures-local" précédente; cette méthode commence par chercher les ouvertures du local, puis pour chaque ouverture, elle essaie de trouver celle qui a le plus bas niveau plus élevé que le plancher du local.

-La méthode **chercher-portes-local (u i j k)**:

C'est une méthode associée à la catégorie "univers" qui permet de chercher les portes d'un local:

**-remarques:**

même description que la méthode "chercher-fenêtres-local" précédente; cette méthode commence par chercher les ouvertures du local, puis pour chaque ouverture, elle essaie de trouver celle qui a le plus bas niveau au même niveau que le plancher du local.

On rappelle que cette méthode ne teste pas les dimensions de l'ouverture.

#### 5.3.2.6.5 L'espace extérieur du projet

Après avoir défini l'espace intérieur et l'enveloppe d'un local, on a défini l'espace extérieur du projet comme étant l'ensemble de voxels "vide" qui n'appartiennent à aucun espace intérieur d'aucun local.

C'est donc l'ensemble de voxels "vide" qui entourent le projet et qui touchent aux limites de l'univers.

-la méthode **chercher-espace-extérieur (u)**:

C'est une méthode associée à la catégorie "univers" qui permet de chercher l'espace extérieur du projet:

**-remarques:**

Cette méthode utilise la méthode **chercher-tous-les-espaces-intérieurs**. cette dernière utilise la fonction **trouver-tous-les-pointeurs** décrite plus loin et qui permet d'identifier tous les pointeurs sur les espaces vides qui sont des espaces de locaux (i.e. finis).

Ayant trouvé tous les vides des locaux, la méthode **chercher-espace-extérieur** identifie les voxels "vide" qui n'apartiennent à aucun local.

#### 5.3.2.6.6 Parois extérieures et intérieures

Ayant défini l'espace extérieur d'un projet, on a essayé de distinguer entre les parois extérieures et intérieures d'un local.

On appelle **paroi extérieure** l'ensemble de voxels d'une paroi dont au moins une facette est en contact avec l'espace extérieur du projet.

On appelle **paroi intérieure** l'ensemble de voxels d'une paroi dont aucune des facettes n'est en contact avec l'espace extérieur du projet.

-La méthode **chercher-parois-externes-local (u i j k)**:
C'est une méthode associée à la catégorie "univers" qui permet de chercher les parois externes d'un local:

**-remarques:**

même description de paramètres que la méthode "chercher-parois-local" précédente; elle commence par chercher les parois du local, puis elle cherche l'espace extérieur du projet, et ne retient des voxels des parois du local que ceux qui sont en contact avec l'espace extérieur.

-La méthode **chercher-toutes-cloisons-local (u i j k)**:
C'est une méthode associée à la catégorie "univers" qui permet de chercher les parois internes d'un local:

**-remarques:**

même description de paramètres que la méthode "chercher-parois-local" précédente; elle commence par chercher les parois du local, puis elle cherche l'espace extérieur du projet, et ne retient des voxels des parois du local que ceux qui ne sont pas en contact avec l'espace extérieur.

#### 5.3.2.6.7 Ouvertures extérieures et intérieures

Ayant défini l'espace extérieur d'un projet, on a essayé de distinguer entre les ouvertures extérieures et intérieures d'un local.

On appelle **ouverture extérieure** (porte ou fenêtre) l'ensemble de voxels d'une ouverture dont au moins une facette est en contact avec l'espace extérieur du projet.

On appelle **ouverture intérieure** l'ensemble de voxels d'une ouverture dont aucune des facettes n'est en contact avec l'espace extérieur du projet.

-La méthode **chercher-ouvertures-externes-local (u i j k)**:
C'est une méthode associée à la catégorie "univers" qui permet de chercher les ouvertures externes d'un local:

**-remarques:**

même description de paramètres que la méthode "chercher-ouvertures-local" précédente; elle commence par chercher les ouvertures du local, puis elle cherche l'espace extérieur du projet,

et ne retient des voxels des ouvertures du local que ceux qui sont en contact avec l'espace extérieur.

-La méthode **chercher-ouvertures-internes-local (u i j k):**
C'est une méthode associée à la catégorie "univers" qui permet de chercher les ouvertures internes d'un local:

**-remarques:**

même description de paramètres que la méthode "chercher-ouvertures-local" précédente; elle commence par chercher les ouvertures du local, puis elle cherche l'espace extérieur du projet, et ne retient des voxels des ouvertures du local que ceux qui ne sont pas en contact avec l'espace extérieur.

#### 5.3.2.6.8 Cloison commune entre deux locaux

Ayant défini l'espace intérieur d'un local, on peut définir une cloison commune à deux locaux comme étant l'ensemble de voxels "blanc" qui sont en contact, en même temps, d'une part avec l'espace intérieur du premier local et d'autre part avec l'espace intérieur du deuxième local.

-La méthode **chercher-cloison (u i1 j1 k1 i2 j2 k2):**

C'est une méthode associée à la catégorie "univers" qui permet de trouver une cloison commune entre deux locaux.

**-les paramètres:**

-u, est un objet de catégorie "univers" quelconque; souvent c'est l'objet "ur" (l'objet contenant la matrice qui représente le projet) qui reçoit le message.

-i1, j1, k1 sont les indices dans la matrice tridimensionnelle d'un voxel "vide" à partir duquel la recherche de l'espace vide du premier local doit démarrer.

-i2, j2, k2 sont les indices dans la matrice tridimensionnelle d'un voxel "vide" à partir duquel la recherche de l'espace vide du deuxième local doit démarrer.

**-Le retour de la méthode:**

La fonction commence par chercher l'espace vide de chaque local, puis elle recherche les voxels "blanc" qui touchent en même temps au premier et au deuxième espace.

La fonction retourne par défaut dans la matrice de l'objet "uf" les voxels qui représentent la cloison commune des locaux pointés.

On peut stocker la matrice de retour à l'aide de la fonction **stocker-résultat-dans** dans un autre objet de type "univers".

**-remarques:**

-la méthode suppose aussi que la cloison a une épaisseur uniforme de 1 voxel.

#### 5.3.2.6.9 Communication entre deux locaux

Ayant défini l'espace intérieur d'un local, on peut définir une ouverture commune comme étant l'ensemble de voxels "transparent" qui sont en contact, en même temps, d'une part avec l'espace intérieur du premier local et d'autre part avec l'espace intérieur du deuxième local.

On appelle alors une porte commune ou une communication entre deux locaux, une ouverture commune à deux locaux, qui est une porte (voir définition d'une porte dans les paragraphes avant), et dont les dimensions minimales sont fixées dès le départ.

-La méthode **chercher-communication (u i1 j1 k1 i2 j2 k2 larg haut):**

C'est une méthode associée à la catégorie "univers" qui permet de trouver une communication entre deux locaux.

**-les paramètres:**

-u, est un objet de catégorie "univers" quelconque; souvent c'est l'objet "ur" (l'objet contenant la matrice qui représente le projet) qui reçoit le message.

-i1, j1, k1 sont les indices dans la matrice tridimensionnelle d'un voxel "vide" à partir duquel la recherche de l'espace vide du premier local doit démarrer.

-i2, j2, k2 sont les indices dans la matrice tridimensionnelle d'un voxel "vide" à partir duquel la recherche de l'espace vide du deuxième local doit démarrer.

-larg, haut sont les dimensions minimales (en nombre de voxels) de l'ouverture à trouver.

La largeur est la dimension dans le sens des i, ou dans le sens des j dans la matrice.

La hauteur est la dimension dans le sens des k de la matrice.

**-Le retour de la méthode:**

La fonction commence par chercher l'espace vide de chaque local, puis elle recherche les voxels "transparent" qui touchent en même temps au premier et au deuxième espace.

Elle vérifie ensuite si l'ouverture trouvée est une porte (c.à.d. ele commence au niveau du plancher d'un local); puis elle vérifie si les dimensions de l'ouverture trouvée sont supérieures aux dimensions minimales passées en paramètre.

La fonction retourne par défaut dans la matrice de l'objet "uf" les voxels qui représentent la communication entre les locaux pointés.

On peut stocker la matrice de retour à l'aide de la fonction **stocker-résultat-dans** dans un autre objet de type "univers".

**-remarques:**

-la méthode suppose aussi que l'ouverture a une épaisseur uniforme de 1 voxel.

-Dans notre prototype, on n'a pas défini les escaliers; donc deux locaux qui ne sont pas au même niveau peuvent avoir une porte de communication mais il faut avoir un escalier pour les lier.

On s'est contenté dans notre implémentation de chercher pour deux locaux dénivelés, une ouverture qui soit au niveau du plancher du local le plus élevé.

Plus tard, on peut définir l'entité escalier pour compléter la notion de communication entre deux locaux.

#### 5.3.2.6.10 Les orientations des parois et des ouvertures

Pour les parois et les ouvertures extérieures, on a défini quatre orientations. Notre prototype étant limité à des volumes à angles droits, on a aussi limité l'orientation du projet tel que les façades soient toujours parallèles à une direction principale: Nord, Sud, Est, Ouest.

Cette restriction n'a pour but que réduire la complexité des algorithmes de recherche.

On a alors défini le Nord comme étant la direction dans le sens des indices j décroissants dans la matrice de voxels (convention arbitraire); on en déduit alors les autres orientations:

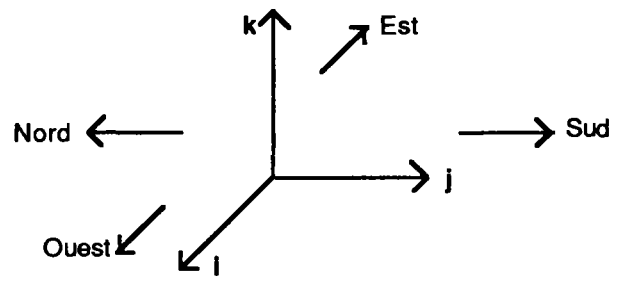



*fig. 5.11 Les orientations adoptées*

En adoptant ces conventions on a défini des sous-types des parois et ouvertures extérieures, et on a défini les méthodes d'analyses suivantes:

-chercher-parois-nord-local (u i j k)
-chercher-parois-sud-local (u i j k)
-chercher-parois-est-local (u i j k)
-chercher-parois-ouest-local (u i j k)

-chercher-fenetres-nord-local (u i j k)
-chercher-fenetres-sud-local (u i j k)
-chercher-fenetres-est-local (u i j k)
-chercher-fenetres-ouest-local (u i j k)

-chercher-portes-nord-local (u i j k)
-chercher-portes-sud-local (u i j k)
-chercher-portes-est-local (u i j k)
-chercher-portes-ouest-local (u i j k)

Ces fonctions commencent par chercher les entités parois, fenêtres ou portes du local, puis elles recherchent l'espace extérieure du projet; elles retiennent ensuite les voxels dont une facette orientée vers la direction recherchée touche l'espace extérieur.

**remarque**

Ces orientations ne correspondent pas nécessairement à un cas réel; mais on a implémenté un cas particulier pour montrer la possibilité d'inclure ce type d'information dans notre modèle.

On peut facilement étendre notre définition de l'orientation pour prendre en compte des cas plus réalistes mais cela au prix d'un surcoût en temps de développement d'algorithmes complexes, et en pénalisant le temps de réponse du prototype durant l'exécution.

En effet, une facette d'un voxel sera suivant une direction donnée si la normale sortante de cette facette forme un angle aigu avec la direction recherchée.

### 5.3.2.7 Les entités définies pour le projet

Toutes les définitions des entités déjà vues sont associées à un local donné.

On a défini aussi des fonctions d'analyses du même type mais qui sont associées à tout le projet. Elles sont définies avec les mêmes principes et désignation sauf qu'elles sont calculées pour tous les locaux en même temps.

Pour cela les méthodes suivantes sont définies:

-la méthode **trouver-tous-les-pointeurs (u)**:

elle permet d'identifier tous les locaux d'un projet. On a vu qu'un local est essentiellement défini par son espace vide intérieur qui est fini. Pour reconnaitre un espace intérieur il suffisait de lancer la méthode en lui donnant les références d'un voxel où il faut démarrer la recherche.

Ainsi, n'importe quel voxel vide de l'espace intérieur d'un local peut référencer le local. Pour une question d'uniformité et pour rendre unique la référence d'un local on a essayé de choisir un voxel particulier de l'ensemble des voxels vides pour désigner un local; le choix étant arbitraire on a choisi le voxel le plus bas et le plus à gauche comme référence unique d'un local; cette référence sera appelée "pointeur".

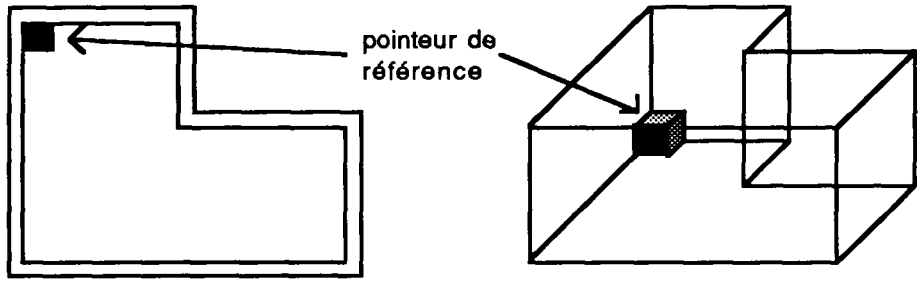



*fig. 5.12 Le pointeur de référence d'un local*

Cette méthode est alors écrite pour pouvoir retirer de la matrice des voxels, tous les voxels qui peuvent être considérés comme pointeur d'un vide de local; ce pointeur étant unique, cette fonction ramène alors une façon de reconnaitre tous les espaces vides finis et distincts d'un projet.

Cette méthode commence par chercher les voxels vides qui sont des coins en bas à gauche, puis vérifie pour chaque voxel s'il référence un espace d'un local fini et distinct des autres.

La méthode ramène une liste de pointeurs représentés en vecteurs à trois éléments.

-la méthode **trouver-nombre-locaux (u)**:

elle permet de trouver combien d'espaces vides finis et distincts il existe dans la matrice de voxels. Cette méthode utilise celle décrite avant et ramène le nombre de pointeurs de la liste de retour de la première méthode.

-Les autres méthodes sont:

-chercher-tous-les-espaces-intérieures (u)
-chercher-toutes-enveloppes-externes (u)
-chercher-toutes-parois-externes (u)
-chercher-toutes-ouvertures-externes (u)
-chercher-toutes-cloisons (u)
-chercher-toutes-ouvertures-internes (u)
-chercher-toutes-portes-externes (u)
-chercher-toutes-fenêtres-externes (u)
-chercher-toutes-portes-internes (u)
-chercher-toutes-parois-nord (u)
-chercher-toutes-parois-sud (u)
-chercher-toutes-parois-est (u)
-chercher-toutes-parois-ouest (u)

-chercher-toutes-portes-nord (u)
-chercher-toutes-portes-sud (u)
-chercher-toutes-portes-est (u)
-chercher-toutes-portes-ouest (u)
-chercher-toutes-fenêtres-nord (u)
-chercher-toutes-fenêtres-sud (u)
-chercher-toutes-fenêtres-est (u)
-chercher-toutes-fenêtres-ouest (u)

Comme on a défini pour un local, en partant de la définition de l'espace intérieur, toutes les entités (parois, murs, portes,...), on a défini la même chose pour tout le projet en remplaçant l'espace intérieur d'un local par tous les espaces intérieurs de tous les locaux identifiés par les fonctions précédentes.

Le nom des méthodes est assez explicatif et le retour de la fonction est analogue à celle définie pour un local.

#### 5.3.2.8 Requêtes complexes à partir de requêtes simples

Certaines des méthodes précédentes sont implémentées en combinant des requêtes simples initialement écrites pour un seul local afin d'identifier le même type d'entité pour tout le projet (i.e. tous les locaux ensembles).

On peut généraliser cette démarche pour qu'elle traite un, plusieurs ou tous les locaux d'un projet surtout que les définitions des entités sont basées sur la définition des entités d'un local.

Cette façon de composer par appels successifs ou emboités permet d'enrichir le système car l'utilisateur pourra assez simplement composer les requêtes dont il a besoin et qui ne sont pas prédéfinies.

On revient à l'idée décrite dans le chapitre précédent où l'on précise que notre système ne contient pas toutes les requêtes possibles et imaginables; seules les fonctions jugées de base seront implémentées pour aider à la construction de requêtes utilisateurs plus spécifiques qui satisfont ses besoins.

#### 5.3.2.9 Requêtes pour les attributs d'un local

Toutes les requêtes décrites jusqu'à présent permettent de déterminer un objet géométrique tel que paroi, ouverture, espace,...

On a implémenté dans notre prototype un autre genre de requêtes qui permettent de tirer des informations sur ces objets géométriques; on peut par exemple calculer la surface d'un vide intérieur d'un local, ou la hauteur minimale; de même pour une fenêtre, on peut calculer sa superficie.

Pour cela on a essayé d'implémenter deux méthodes possibles:

-la première méthode fait appel elle-même à la fonction qui ramène l'entité qu'il faut chercher, et fait le calcul nécessaire pour tirer la valeur d'un attribut:

un exemple de ce type d'implémentation est la méthode suivante:

**-chercher-surface-local(u i j k):**

*cette fonction commence par chercher l'espace intérieur du local puis elle* calcule la surface du vide au niveau du plancher (en comptant les voxels).

-la deuxième méthode qu'on a implémentée est plus générique; elle permet de lancer un calcul de surface sur n'importe quelle entité en supposant que l'entité est déjà déterminée par une requête:

Un exemple de ce type de méthodes est le calcul de surface d'une paroi ou d'une ouverture:

**-calcul-surface (u imin jmin kmin imx jmx kmx obj)**

cette fonction suppose que l'entité dont il faut calculer la surface se trouve déjà dans l'univers réponse, et suivant que le paramètre "obj" de la méthode vaut "blanc" ou "transparent" respectivement, cette fonction calcule la surface de paroi respectivement d'ouverture.

*Un autre exemple de ce type de méthodes est la méthode:*

**-calcul-hauteur-minimale (u i j k obj)**

cette fonction suppose que l'entité dont il faut calculer la hauteur minimale se trouve déjà dans l'univers réponse, et suivant que le paramètre "obj" de la méthode vaut "vide" ou "blanc" ou "transparent" respectivement, cette fonction calcule la hauteur minimale du vide d'un local, d'une paroi respectivement d'une ouverture.

### 5.3.3 Les *fonctions de haut niveau*

Les requêtes décrites jusqu'à présent sont des méthodes de bas niveau qui sont envoyées dans une syntaxe d'envoi de message aux objets concernés; les paramètres ne sont pas simples à rentrer.

Pour cela on a essayé de faciliter la tâche de l'utilisateur en construisant des fonctions de plus haut niveau qui permettent d'écrire des requêtes plus simplement et avec des syntaxes génériques:

Ainsi pour ces fonctions on a adopté des valeurs par défaut de certains paramètres:

-l'objet univers qui reçoit par défaut le message est l'objet "ur" (i.e. univers réel) qui contient la matrice du projet.

-l'objet univers qui contient la réponse par défaut est l'objet "uf" (i.e. univers fictif).

Les fonctions suivantes sont définies:

**-paroi (local orientation)**

**-mur (local orientation)**
**-plafond (local orientation)**
**-plancher (local orientation)**
**-porte (local orientation)**
**-fenêtre (local orientation)**

A part la prise en défaut de certains paramètres, ces fonctions sont écrites pour adresser un local par son nom et non plus par les trois indices i, j, k du pointeur de référence; (voir le paragraphe sur la catégorie "local" pour plus de détails sur l'association d'un nom à un pointeur de local).

Chaque fonction citée ci-dessus a deux paramètres:

-le premier peut être:

-soit le nom d'un local
-soit () pour représenter tout le projet

le deuxième paramètre peut être:

-soit un autre local (dans le cas par exemple d'une paroi commune à deux locaux)
-soit un des symboles suivants:
-ext ou int pour extérieur ou intérieur
-() pour extérieur et intérieur
-nord, sud, est, ouest pour une orientation

On peut par exemple écrire les requêtes suivantes:

```
-(paroi () ())              ; toutes les parois du projet
-(paroi () int)             ; toutes les cloisons intérieures du projet
-(paroi () nord)            ;toutes les façades nord du projet

-(paroi 'séjour ())         ; toutes les parois du séjour
-(paroi 'séjour int)        ; toutes les cloisons intérieures du séjour
-(paroi 'séjour nord)              ;toutes les façades nord du séjour
-(paroi 'séjour 'cuisine)   ;la cloison commune entre séjour et cuisine
-.....
```

Toutes les combinaisons possibles permettent de faire appel à toutes les fonctions de bas niveau correspondantes.

On a aussi les fonctions de haut niveau pour les attributs d'un local:

**-nombre-de-locaux**

Cette fonction ne prend pas de paramètres.

**-hauteur-min-local (local)**
**-surface-local (local)**
**-surfaces-ouvertures-local (local)**

Ces fonctions prennent un seul paramètre qui est le nom du local.

```
-surface-f          ;pour la surface de fenêtre
-surface-p          ;pour la surface de paroi
```

Ces deux fonctions supposent que l'entité fenêtre ou paroi dont il faut calculer la surface est déjà ramenée par une des requêtes décrites avant dans l'univers fictif "uf".

exemple: (surface-f (fenêtre ('séjour nord)))
cet exemple ramène la surface des fenetres du séjour orientées au nord.

## 5.4 Les informations non géométriques

### 5.4.1 Catégories et objets non géométriques

Comme on a vu dans le chapitre précédent, toutes les informations concernant un projet de bâtiment ne sont pas d'ordre géométrique. En effet, on a souligné l'existence de différents types d'informations qu'on a essayé de regrouper dans trois grandes familles:

-les informations indépendantes de la géométrie du projet: l'exemple le plus simple de ce type d'information est le site d'un projet;
site (longitude, latitude, altitude, ...)

-les informations non géométriques mais qui dépendent de la géométrie du projet: on traitera l'exemple de la fonction architecturale d'un local (cuisine, séjour, ...)

-les informations indépendantes du projet: ces objets qui invariants pour un grand nombre de projets et qui peuvent être regroupés dans un catalogue: comme la structure des couches des parois (10 cm béton +5 cm de polystyrène expansé du côté intérieur ...), ou le type des fenêtres et des portes.

Pour chacun des exemples cités ci-dessus, on a essayé de montrer un moyen de l'implémenter dans notre modèle. Ce modèle qui jusqu'à maintenant s'est contenté de traiter la géométrie du projet à l'aide de l'outil de CAO. On montrera aussi la manière d'assurer le lien, si nécessaire, de telles informations avec la géométrie du projet.

### 5.4.2 Les informations indépendantes de la géométrie du projet

#### 5.4.2.1 La catégorie site du bâtiment

On a choisi cette information comme représentative d'un type d'objets qui sont valables pour un projet, mais n'ont aucune dépendance de la géométrie du projet.

Des informations de ce type enrichissent le projet de données nécessaires à certains calculs ou évaluations.

Ce type d'informations, étant défini une fois au début de la phase de conception restera inchangé durant tout le processus; par suite la saisie et le maintien de

cohérence de ce genre d'informations ne présente pas de difficultés particulières. De plus un outil graphique n'est pas nécessaire pour la saisie.

Dans notre implémentation on a choisi de les regrouper dans des catégories pour les instancier une fois par projet et les utiliser au cours du processus de conception.

Dans un environnement où tout est objet, cette catégorie aura les champs nécessaires qui seront remplis dans l'objet instance par les valeurs convenables utilisées pour ce projet.

Dans notre maquette, on a implémenté la catégorie "site" d'un bâtiment qui possède trois champs à titre indicatif:

-le champ longitude
-le champ latitude
-le champ altitude

Ces champs contiendront, comme leur nom l'indique, la longitude, la latitude, et l'altitude correspondant au site du projet.

La valeur contenue dans chaque champ n'évoluera pas, car les caractéristiques du site ne dépendent pas des variantes géométriques du projet; elles sont imposées au concepteur et ne sont pas choisies ou modifiées par lui.

### 5.4.3 Les informations non géométriques dépendant de la géométrie du projet

#### 5.4.3.1 la catégorie local

Comme exemple de ce type d'informations, on a choisi d'implémenter la fonction architecturale d'un local. On s'est limité dans la maquette réalisée à une définition très restreinte de la fonction architecturale en la limitant à un nom affecté à un local. Le nom ainsi donné à un local lui donnera une signification particulière.

On trouvera un local qui s'appelle séjour et un autre qui s'appelle cuisine. On peut par la suite distinguer ces locaux par leur nom et non pas seulement par leurs positions ou leurs formes. Ce nom sera un moyen de manipuler des informations sur un local en le désignant par son nom.

On parlera par exemple de l'espace vide du local séjour, de l'enveloppe du local séjour, des ouvertures du local séjour, de la cloison entre le local séjour et le local cuisine, de la porte de communication entre le séjour et la cuisine, la façade nord du local séjour, ...

On remarque que le nom en lui-même affecté au local n'a pas de signification géométrique. Par contre, il est lié intimement à la géométrie du projet; ainsi la cloison entre séjour et cuisine peut changer plusieurs fois au cours de la phase de conception.

On propose pour manipuler ce genre d'information d'utiliser une implémentation qui ressemble à la méthode traditionnelle utilisée pour l'élaboration des plans par les architectes.

Le local séjour est désigné dans un plan en écrivant dans l'espace supposé jouer le rôle du séjour le mot ou suite de caractères "séjour". A partir de cet emplacement, le lecteur de plans pourra en déduire toutes les informations s'y rattachant: parois du séjour, surface du séjour, fenêtres du séjour,...

Dans notre modèle, le vide de l'espace d'un local est une suite de voxels vides contigus et entourés par une enveloppe de voxels blancs ou transparents. Par suite, n'importe quel voxel vide de cet ensemble de voxels vides peut désigner le vide du local.

On a choisi de désigner une référence unique par local; cette référence c'est les indices i, j, k dans la matrice de l'univers d'un voxel particulier situé le plus à gauche et le plus bas dans le vide d'un local donné. On associe par la suite à ce pointeur du local le nom choisi comme "séjour", par exemple.

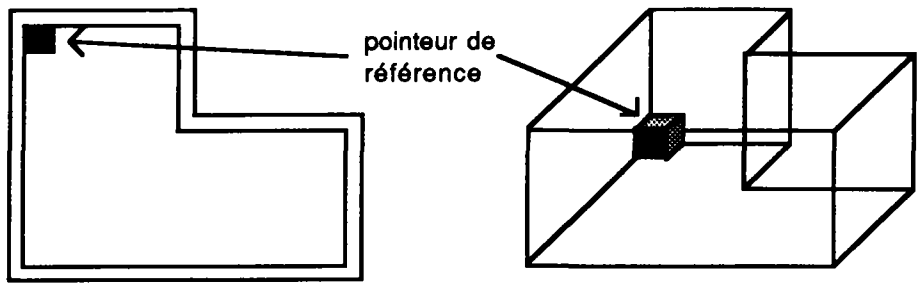



*fig. 5.12 Le pointeur de référence d'un local*

On a créé pour cela la catégorie "local"; cette catégorie a un seul champ nommé pointeur. Un pointeur est alors un vecteur à trois éléments #[i j k] qui représente la position du voxel vide référence du vide d'un local.

On peut instancier la catégorie "local" autant de fois que nécessaire pour désigner les différents locaux.

L'objet séjour, par exemple, instance de la catégorie local et dont le champ pointeur contient le vecteur #[2 3 1]; l'objet cuisine, instance de la catégorie "local", contient dans le champ pointeur le vecteur #[7 6 1].

### 5.4.3.2 Les méthodes associées à la catégorie univers

On peut affecter les noms des locaux interactivement sur l'écran en pointant avec la souris dans le local à nommer. Pour cela, on a associé à la catégorie univers (car c'est la matrice de l'univers qui fournit la valeur du pointeur) les méthodes suivantes:

-la méthode **trouver-pointeur-local-v-h-g (u i j k)**:
elle permet de trouver le pointeur ou la référence unique qui désigne le vide d'un local; c'est le voxel particulier de l'ensemble des voxels vides d'un local qui est le plus bas et le plus à gauche dans l'espace vide du local.

-la méthode **attribuer-fonction (u i j k)**:
elle associe un nom à un pointeur en créant un nouvel objet de catégorie "local" et en remplissant le champ pointeur par la valeur ramenée par la méthode précédente.

Bien sûr des tests sont effectués pour vérifier si le nom à attribuer existe déjà, ou si le vide pointé a déjà une fonction, ou bien si le voxel pointé ne peut pas représenter un local bien défini par son vide et son enveloppe (par exemple, la méthode signale une erreur si le local désigné n'est pas fermé et peut s'étendre à l'infini).

-la méthode **trouver-ptr (u local)**:
elle permet de trouver le pointeur associé à un local dont le nom est passé en paramètre.

## 5.4.4 Les informations indépendantes du projet ou catalogues:

### 5.4.4.1 La catégorie "composition"

Comme exemple de ce type d'information, on a choisi la structure couche des parois et les types des composants de fenêtres ou de portes.

Une structure de couche de mur composée de 20 cm d'épaisseur avec 15 cm de béton et 5cm de polystirène expansé appliqué à l'intérieur, ...
Ce type de couche reste valable quel que soit le projet ou sa forme; ces caractéristiques d'épaisseur, de résistance thermique, ...ne changeront pas.

De même un type de fenêtre avec un chassis en aluminium, ouverture à l'italienne, double vitrage, ... peut être spécifié par différents industriels, mais les caractéristiques de ce type ne dépendent pas du projet ni de sa géométrie.

Pour pouvoir coupler ce genre d'informations avec notre modèle géométrique on a choisi la façon traditionnelle de construire des fichiers regroupant des objets prédéfinis ou catalogues l'un regroupant des structures de couches de parois et l'autre des types de fenêtres et de portes.

On a la catégorie structure couche et ses instances ainsi que la catégorie type de fenêtres et de portes et leurs instances.

On a ensuite essayé de présenter un moyen de coupler ces informations avec le projet, vu qu'ils y sont liés par leurs positions.

Pour cela on a utilisée une nouvelle catégorie: la catégorie "composition".

De nouveau, on a essayé de s'inspirer de la méthode traditionnelle et on peut noter deux façons de faire:

-soit on peut utiliser une méthode qui ressemble à la désignation de la fonction d'un local; on écrit dans le plan à côté de chaque fenêtre à quel type elle appartient et on fait accompagner le plan par des détails qui sont la représentation d'après le catalogue de la fenêtre.

-soit d'une façon plus générique, et facile à utiliser et c'est celle-là qu'on a choisie d'implémenter:
Dans une légende accompagnant le plan, on peut trouver les inscriptions suivantes:

-toutes les façades sont en béton
-les façades du séjour sont en brique
-les cloisons sont en placoplâtre
- .....

Cette affectation est générique dans le sens où on passe du général qui peut englober beaucoup d'entités dans le projet (les façades), à des entités plus spécialisées (la façade nord du séjour).

On a alors essayer d'implémenter ce type de désignation dans notre modèle. Cela suppose bien sûr que les entités : façades, façades nord, ou cloison séjour cuisine, ... sont reconnues par notre modèle.

On a créé la catégorie "composition" avec un seul champ appelé matériaux. Ce champs matériaux contient un pointeur référençant le catalogue ou les fichiers des objets structures couches ou type des fenêtres ou de portes.

D'autre part, le nom de l'instance de la catégorie "composition" créée indiquera l'entité du projet à laquelle on affecte les matériaux.

Exemples d''objets de catégorie "composition":

-objet: façade
matériaux: béton

-objet: facade-nord
matériaux: brique

-objet: cloison-séjour-sdb
matériaux: parpaing

-objet: fenêtre-séjour-sud
matériaux: FE01

- .....

On reconnait les différentes entités par un moyen de "pattern-matching" pour des raisons d'efficacité. Cela n'impose pas une méthode, mais propose une solution valable.

Ainsi un nom valable serait une concaténation de plusieurs noms, dans un ordre précis:

-en premier on choisit entre: façade, cloison, fenêtre-ext, fenêtre-int, porte-ext, porte-int.

-en second on a:
-soit le nom d'un local (séjour, cuisine, ...)
-soit une orientation: nord, sud, est, ouest

-en troisième position on a :
-soit le nom d'un autre local si le premier élément était cloison, ou porte-int

-soit une orientation: nord, sud, est, ouest si le premier élément est façade, ou fenêtre-ext, ou porte-ext.

Des exemples de noms de composition valables:

-façade
-façade-nord
-façade-sejour
-façade-séjour-nord
-cloison-séjour-cuisine
-porte-ext-séjour-nord
-.....

Bien sûr, on peut affiner de plus en plus le nom de la composition pour désigner d'une façon plus précise les éléments du projet:

-la première porte à droite de la façade du séjour nord ...

Pour cela, il faut que le modèle puisse reconnaitre les termes utilisés (première à droite, ...). Dans notre maquette, on s'est limité à un ensemble réduit de possibilités à titre de démonstration.

#### 5.4.4.2 Les méthodes pour instancier "composition"

-La méthode **attribuer-composition:**
elle associe à un objet de type composition un symbole en affectant le champ matériaux avec ce symbole qui désigne un objet de catalogue.

### 5.4.5 La fonction information ou comment faire le lien entre une entité et une composition

Seulement trois types de voxels représentent dans un univers toutes les entités; un voxel en fonction de sa texture et de son voisinage peut jouer un rôle différent dans le projet.

Un voxel "vide" peut représenter:

-l'espace intérieur d'un local
-l'espace extérieur du projet

Un voxel "blanc" peut représenter:

-une paroi (mur ou plafond)
-une paroi extérieure ou intérieure
-une paroi orientée (nord, sud,...)

Un voxel "transparent" peut représenter:

-une porte (intérieure ou extérieure)
-une fenêtre (intérieure ou extérieure)
-une porte commune à deux locaux
-une fenêtre orientée nord, ou sud ou ...
-.....

Les fonctions d'analyses permettent donc de chercher les voxels qui peuvent appartenir à une même entité (porte, mur,...) en fonction de la requête adressée. C'est cette caractéristique qui donne à notre modèle la simplicité de saisie et la dynamicité de maintien de cohérence au niveau de la base géométrique.

Chaque voxel, bien qu'il garde la même texture, change de fonction dépendament de ces voisins.

Chaque voxel a par suite une fonction:

-façade
-façade-nord
-cloison séjour-cuisine
-.....

Dans notre modèle, c'est cette fonction qui est utilisée pour attribuer une composition à chaque entité: façade en brique, facade-nord-séjour en parpaing,...

La fonction **information (i j k)** permet de détecter quel rôle joue un voxel dans le projet (celui qui est pointé par i j k ); elle recherche pour le voxel en question son rôle en essayant de trouver s'il est entouré de vide, ou de voxels blanc, ...

Cette fonction utilise les mêmes algorithmes que ceux utilisés par les méthodes correspondantes à la définition des entités d'un local. Un voxel est une cloison s'il touche en même temps d'un côté à l'espace intérieur d'un local, et de l'autre côté à l'espace intérieur d'un autre local. Par contre c'est une façade (façade est synonyme de mur extérieur), s'il touche d'un côté à l'espace intérieur d'un local et de l'autre côté à l'espace extérieur du projet...

Pour plus de détails revoir la définition des entités d'un local.

La fonction information fait aussi la correspondance entre l'entité trouvée, ou le rôle du voxel et la composition qui lui est associée.

Elle commence par chercher par les objets compositions les plus spécialisés, et si elle ne trouve pas, elle cherche dans une composition plus générale, jusqu'à ce qu'elle trouve une correspondance.

Exemple: si le voxel joue le rôle de façade-séjour-nord, la fonction information recherche s'il existe un objet composition façade-séjour-nord; si oui, elle attribue le champ matériaux de cette composition au voxel; sinon elle recherche un objet composition moins spécifique tel que façade-séjour; sinon elle cherche encore plus général tel que façade.

## 5.5 Les objets incomplets et les contraintes

### 5.5.1 Introduction

Les contraintes représentent dans notre aproche un moyen de prendre en compte les informations connues sur des objets incomplets. Les objets incomplets ne contiennent pas encore les informations nécessaires pour construire les entités

géométriques correspondantes; par suite, c'est difficile de les modéliser à l'aide d'une base de données géométrique, celle de l'outil de CAO.

Un local dont on ne connait ni les dimensions ni la position ne peut pas être saisi par l'outil de CAO; par contre, on peut, à un moment donné, posséder des informations concernant ce local:

-La surface du local est > 30 $m^2$
-Ce local doit communiquer avec un autre local
- .....

Comme la base géométrique et les catégories qu'on a présenté déjà ne permettent pas de représenter tels types d'informations, on a créé une catégorie appelée "cntrnt" (pour contrainte, le mot contrainte n'a pas été utilisée car ce mot est réservé pour le système SMECI).

Ces contraintes, comme on le verra dans le chapitre suivant, peuvent être utilisées en vérification ou en conception.

### 5.5.2 La catégorie "cntrnt"

La catégorie "cntrnt" implémenté a quatre champs:

-le champ **appliquée-à**:
dans notre exemple, on a surtout choisi de travailler avec les espaces intérieurs et la communication des locaux; ainsi, les types de contraintes utilisés sont applicables à des locaux.

Chaque contrainte est relié au local auquel elle s'applique par l'intermédiaire de ce champ. Il contient un symbole qui représente le nom affecté à un local (le même nom qui désigne un local et qui est associé à un pointeur dans la catégorie "local" décrite avant).

C'est une façon de regrouper les contraintes par local, on distinguera par exemple, les contraintes appliquées au local séjour de celles appliquées au local cuisine,...

On facilite ainsi la mise en oeuvre de mécanismes de vérifications automatiques de contraintes tel que décrites dans le paragraphe 4.4.2.3 du chapitre précédent.

On pourra, plus tard, par extension de ce modèle, prendre en compte des spécifications liées à des entités plus petites qu'un local: contraintes appliquées à une paroi, ou à une fenêtre,...

On s'est limité dans notre démo, de regrouper les contraintes par local, ce qui a paru suffisant surtout que les types de contraintes considérés dans notre exemple se rapportent tous à des contraintes sur des locaux ou des attributs de locaux.

-le champ **type-de-contrainte**:
ce champs permet de distinguer les différentes contraintes entre elles; il peut contenir un symbole choisi parmi les types suivants:

-**existe**: ce type est utilisé pour imposer une contrainte d'existence à un local; au début de la phase de conception, je peux imposer qu'un local garage doit exister pour le projet; au moment de la vérification, une erreur est signalée s'il n'existe pas de local saisi ayant pour fonction garage.

-**adjacent**: ce symbole est utilisé pour imposer une contrainte d'adjacence entre deux locaux; on peut par exemple imposer que le local séjour soit adjacent au local cuisine.

-**communique**: ce symbole est utilisé pour imposer une contrainte de communication entre deux locaux; on peut par exemple imposer que le local séjour soit communicant avec le local cuisine.

-**>, >=, <, <= ou =** : ces symboles ont la même signification que le symbole mathématique correspondant (plus grand, plus grand ou égal, plus petit, plus petit ou égal, égal). on peut construire avec ces symboles des contraintes qui sont des relations qui relient des variables ou des expressions mathématiques à des valeurs numériques ou même des variables et des expressions mathématiques.

Par exemple, on peut avoir les contraintes suivantes:

-(surface-f (fenetre séjour nord)) = 0
-(surface-f (fenêtre séjour ()) >= 0.2 * (surface-local séjour)

Ce type de contraintes s'applique à des attributs d'un local.

-les champs **élément1** et **élément2**:
ces deux champs contiennent respectivement la partie gauche et la partie droite (si elle existe) de la contrainte ou relation imposée.

Exemples:

| | | |
|---|---|---|
| -séjour existe | -> | élément1 = séjour<br>élément2 = nil |
| -séjour adjacent cuisine | -> | élément1 = séjour<br>élément2 = cuisine |
| -(surface-local séjour) > 20 | -> | élément1 = (surface-local séjour)<br>élément2 = 20 |

Dans notre modèle, on s'est limité à l'étude de contraintes unaires (à un seul élément), et binaires (à deux éléments). On peut bien sûr étendre le modèle pour prendre en compte d'autres types de contraintes qui peuvent être des relations à plusieurs éléments.

#### 5.5.2.1 Les fonctions pour instancier des contraintes

Pour créer des contraintes, ou instancier la catégorie "cntrnt", on a écrit des fonctions en LISP qui facilitent le remplissage des champs d'une contrainte en passant les valeurs en pramètres:

-la fonction **cntrnt-existe** (local):
Elle permet de créer une instance de la catégorie "cntrnt" dont le champs type-de-contrainte contient le symbole "existe". Le champs appliquée-à contient le nom du local passé en paramètre. Le champs élément1 contient le nom du local passé en paramètre.

-la fonction **cntrnt-adjacent** (local1 local2):
Elle crée deux instances de la catégorie "cntrnt"; la première instance contient dans le champ appliquée-à le nom du local1, et dans les champs élément1 et élément2 respectivement le nom de local1 et local2; la deuxième instance contient dans le champ appliquée-à le nom du local2, et dans les champs élément1 et élément2 respectivement le nom de local2 et local1.

Dans les deux instances le champ type-de-contrainte contient le symbole "adjacent".

Cette fonction crée aussi, si elles n'existent pas, les contraintes existe local1 et existe local2, car les deux locaux doivent exister pour pouvoir être adjacents.

-la fonction **cntrnt-communique** (local1 local2):
Elle crée deux instances de la catégorie "cntrnt"; la première instance contient dans le champ appliquée-à le nom du local1, et dans les champs élément1 et élément2 respectivement le nom de local1 et local2; la deuxième instance contient dans le champ appliquée-à le nom du local2, et dans les champs élément1 et élément2 respectivement le nom de local2 et local1.

Dans les deux instances le champ type-de-contrainte contient le symbole "communique".

Cette fonction crée aussi, si elles n'existent pas, les contraintes existe local1 et existe local2, car les deux locaux doivent exister pour pouvoir être adjacents. De même elle crée, si elle n'existe pas, la contrainte (adjacent local1 local2), car les deux locaux doivent être adjacents pour pouvoir communiquer.
La communication signifie dans notre démo que deux locaux ont une cloison commune et une porte commune, non pas qu'il peut y avoir un moyen d'aller d'un local dans l'autre en passant par d'autres locaux.

-la fonction **cntrnt-relation** (expression1 symbole expression2 local):
Elle crée une contrainte de relation appliquée à local.
*symbole est le signe mathématique* (>, >=, <, <=, =) *qui est affecté au* champs type-de-contrainte.

expression1 et expression2 sont deux expressions en LISP (variable ou fonction ramenant une valeur numérique pour expression1, et valeur ou variable ou fonction ramenant une valeur numérique pour expression2); ces expressions peuvent ramener des valeurs d'un attribut de local (surface, hauteur minimum, ou surface ouvertures,...); pour plus de détails sur les fonctions pouvant ramener des valeurs d'attributs de local voir le paragraphe consacré aux fonctions d'analyses implémentées.

expression1, et expression2 doivent être écrites suivant une synthaxe d'appel de LISP, car au moment de la vérification, on évalue à l'aide de la fonction EVAL les contenus d'élément1 et élément2.

#### 5.5.2.2 Les fonctions de vérification de contraintes

On a essayé dans la démo implémentée de montrer comment on peut vérifier des *contraintes, une par une ou toutes celles appliquées à un local donné.*

On peut alors à l'aide de ce mécanisme construire des outils de vérification plus sophistiqués, à déclenchement automatique tel que décrit dans le paragraphe 4.4.2.3 du chapitre 4.

Attention, vérifier une contrainte signifie dans notre exemple tester si dans le projet saisi la contrainte n'est pas violée; à ne pas confondre avec une méthode automatique qui permet de résoudre le conflit si une contrainte est violée (référence chapitre 4 paragrahe ?).

Dans la version proposée seul un déclenchement manuel est implémenté et cela surtout dû à une limitation de performances du matériel utilisé.

Pour chaque type de contrainte on a créé la fonction qui permet de tester sa validité en fonction du projet saisi avec l'outil de CAO. On a alors les fonctions suivantes:

-la fonction **vérifier-contrainte-existence** (local):
Cette fonction commence par chercher s'il existe une instance de la classe "cntrnt" qui est du type "existe" et qui est appliquée au local passé en paramètre.

S'il existe une contrainte, elle essaie ensuite de trouver dans les objets instances de la catégorie "local" s'il existe un pointeur associé à un local portant le même nom que le local de l'élément1 de la contrainte. Elle signale une erreur dans le cas contraire.

-la fonction **vérifier-contrainte-adjacence** (local):
Cette fonction commence par chercher toutes les contraintes de type "adjacent" et qui sont appliquées au local passé en paramètre.

S'il existe des contraintes de ce type, elle essaie de vérifier pour chacune si elle n'est pas violée dans l'état actuel du projet saisi.

Cette fonction utilise la fonction prédicat de haut niveau:
(adjacent? local1 local2)

Ce prédicat retourne vrai si local1 et loca2 ont une partie de cloison en commun, et nil sinon.

La fonction "adjacent?" fait appel à une méthode de bas niveau envoyé à l'univers réel pour trouver une partie de cloison commune:

(synd 'chercher-cloison ur i1 j1 k1 i2 j2 k2)

Pour plus de détails sur cette méthode se référer à la définition des fonctions d'analyses implémentées.

Bien sûr la fonction de vérification commence par tester tout d'abord si les deux locaux existent.

-la fonction **vérifier-contrainte-comunication** (local):
Cette fonction commence par chercher toutes les contraintes de type "communique" et qui sont appliquées au local passé en paramètre.

S'il existe des contraintes de ce type, elle essaie de vérifier pour chacune si elle n'est pas violée dans l'état actuel du projet saisi.

Cette fonction utilise la fonction prédicat de haut niveau:

(y-a-une-porte? local1 local2 larg haut)

Ce prédicat retourne vrai si local1 et loca2 ont une partie d'ouverture en commun de largeur larg et de hauteur haut, et nil sinon.

La fonction "y-a-une-porte?" fait appel à une méthode de bas niveau envoyé à l'univers réel pour trouver une partie d'ouverture commune:

(synd 'chercher-communication ur i1 j1 k1 i2 j2 k2 larg haut)

Pour plus de détails sur cette méthode se référer à la définition des fonctions d'analyses implémentées.

Bien sûr la fonction de vérification commence par tester tout d'abord si les deux locaux existent.

-la fonction **vérifier-contrainte-relation** (local):
Elle permet de tester toutes les contraintes de type relation mathématique appliquée au local passé en paramètre.

Comme on l'a déjà décrit, les relations sont en fait des expressions LISP qui peuvent être évaluées et qui permettent de retirer des informations concernant les attributs d'un local.

Comme exemples de ces fonctions on peut citer:

-(surface-local local)
-(hauteur-min-local local)
-.....

Pour plus d'informations sur ce type de fonctions et les fonctions implémentées se réferer à la partie de description des fonctions d'analyses.

-la fonction **vérifier-toutes-les-contraintes (local)**:
Elle permet de vérifier toutes les contraintes appliquées au local passé en paramètre en faisant appel aux fonctions précédentes les unes après les autres.

## 5.6 Conclusion

L'implémentation réalisée ne prétend pas être exhaustive et complète; elle permet de résoudre beaucoup de problèmes en donnant un axe directeur ou une méthode applicable pour des problèmes délicats relevés, en montrant une façon de les implémenter; on propose alors des solutions possibles qui sont compatibles avec notre proposition de couplage des deux mondes de la CAO et de l'IA tel que présentée dans le chapitre précédent.

Les implémentations proposées sont fonction d'un choix de départ comme l'environnement de programmation, le langage ou même le choix du modèle géométrique; bien que l'implémentation dépend intimement de ces choix de départ, ceci ne représente qu'une démonstration de la faisabilité de notre approche; celle-ci reste valable pour d'autres outils, environnements ou modèles d'implémentation tout en conservant les principes fondamentaux exposés dans le chapitre précédent.

La méthode d'implémentation utilisée actuellement, bien qu'elle présente des inconvénients (volumes à angles droits, petits projets, lenteur de calcul,...), a beaucoup d'avantages surtout au niveau de la simplicité des concepts, de la rapidité de programmation et de l'implémentation des fonctions d'analyse.

Ce modeleur "intelligent" peut être implémenté à l'aide d'autres moyens informatiques et d'autres types de modélisations géométriques (parler de facettes et de volumes).

Des méthodes de tracé de rayon, de calcul d'adjacence et d'inclusions ou autres méthodes mathématiques peuvent être adaptées à un autre modèle de données géométriques d'un modeleur de CAO déjà existant et qui est basé sur la notion de volumes pleins et vides.

# Chapitre 6

# Applications

## 6.1 Introduction

A l'aide de notre outil de CAO, on a développé plusieurs applications qui mettent en valeur les concepts exposés dans les chapitres précédents.

Nous allons voir dans ce chapitre des applications sur les possibilités d'utilisation d'un tel outil tant au niveau de l'utilisation du système de CAO qu'au niveau du développement de systèmes experts couplés avec le système de saisie graphique, permettant d'avoir un outil d'aide à la conception.

Notre démo est surtout orientée vers une utilisation du modèle dans le domaine de la conception d'espaces architecturaux, des locaux, des adjacences et des communications.

L'extension du modèle vers d'autres types d'utilisation reste valide en appliquant les mêmes principes et concepts de ce qu'on a appelé "le modeleur intelligent".

## 6.2 Première application

### 6.2.1 La saisie d'un projet

On essaiera de montrer dans cet exemple la facilité apportée par notre outil au niveau de la saisie. Une saisie graphique interactive permet d'avoir accés aux différentes entités du projet d'ordre géométriques ou sémantiques en vue de son évaluation (visuelle ou logicielle) par un expert humain, ou par un système expert contenant les bases de règles nécessaires auquel on fait appel dans l'environnement de travail.

Cet exemple montre les capacités du système dans le stockage et la restauration des informations. Ce rôle que l'outil doit assurer pour le couplage avec les évaluateurs ou les systèmes experts de conception.

### 6.2.2 Le postage des contraintes

Une première approche de saisie du concepteur peut être d'imposer au projet toutes les contraintes de départ en fonction du programme du projet, de la demande du client, de son expertise personnelle ou d'après certaines règles d'art ou de bon sens...

Ces contraintes peuvent être décrites au niveau de la saisie, ou bien s'il s'agit de contraintes valables pour plusieurs projets, elles peuvent être stockées dans des fichiers utilisables suivant le cas.

Le concepteur peut commencer par imposer par exemple les contraintes suivantes:

-séjour existe
-cuisine existe
-sam existe
-sdb existe

Ces contraintes traduisent le fait que le projet contient quatres locaux dont la fonction est respectivement: séjour, cuisine, salle à manger et salle de bain.

D'autre part, il peut imposer des contraintes d'adjacence:

-séjour adjacent à cuisine
-séjour adjacent à sam
-cuisine adjacent à sam
-sdb adjacent à cuisine

des contraintes de communication aussi:

-sejour communique avec sam
-sejour communique avec cuisine
-cuisine communique avec sam
-cuisine comunique avec sdb

On peut aussi imposer des contraintes de relations:

-surface-local du sejour > 20
-surface-fenêtre sud du séjour = 0,5 * surface-local sejour
-surface-fenêtres-nord du séjour = 0
-surface-local cuisine > surface-local sam
-.....

Ces contraintes imposées sur le projet devrait être vérifiées à la fin de l'étape de conception. A tout moment de la phase de saisie graphique, le concepteur peut déclencher la vérification des contraintes pour l'évaluer par rapport aux critères de départ.

L'intérêt de l'approche est l'utilisation d'un langage de haut niveau pour la formulation des contraintes; le concepteur manipule alors les entités par leur nom ou par leur signification dans le projet (cloison, façade...), sans se soucier de la représentation en termes géométriques de ces entités.

On verra plus tard l'évaluation des contraintes de départ par rapport au projet.

### 6.2.3 La saisie de la volumétrie

L'interface graphique permet de saisir le projet interactivement à l'écran. Le concepteur commencera par saisir la volumétrie des locaux; ensuite, il leur affectera

des noms (une fonction architecturale tel que séjour, cuisine,...), pour faciliter la distinction des locaux.

Il faut préciser que l'analyseur géométrique reconnait un local sans que celui-ci ait un nom, mais pour désigner ce local il utilise un pointeur géométrique (voir chapitre précédent), difficilement reconnaissable par l'utilisateur.

La saisie sera limitée aux entités espace de local, ouvertures ou parois; le concepteur peut saisir un local en définissant son enveloppe extérieure, il peut ajouter des parois pour subdiviser les locaux et il peut ajouter des ouvertures dans les parois intérieures ou extérieures.

Pour cela il peut soit utiliser le curseur qui remplit là où il se trouve des voxels blanc, vide ou transparent, soit utiliser des fonctions de plus haut niveau qui sont écrites pour faciliter la saisie:

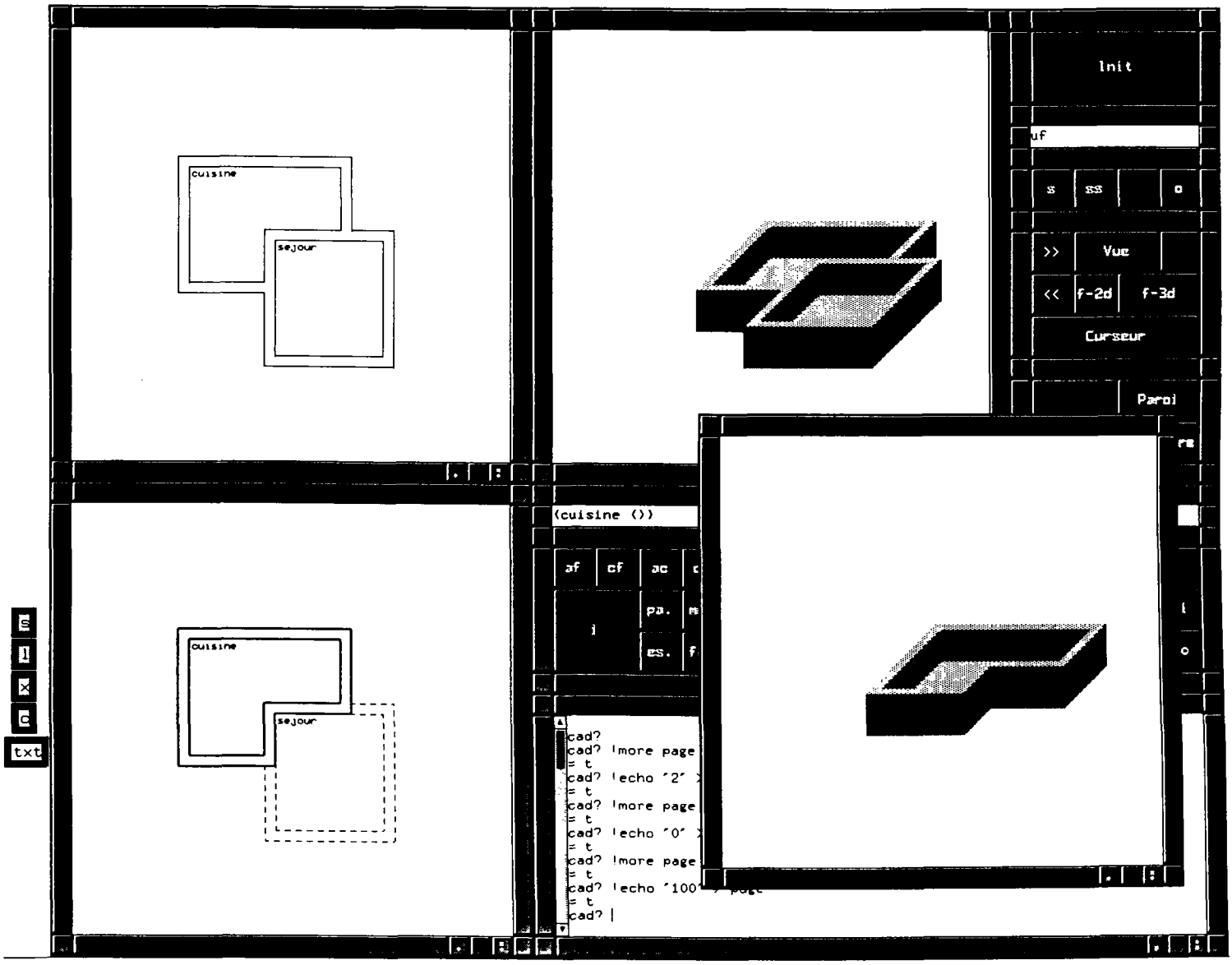
-créer-local
-créer-paroi
-créer-ouverture

On rappelle que ces fonctions se contentent de créer une volumétrie en ajoutant les voxels correspondants sans ajouter aucune autre information d'ordre sémantique. Le concepteur n'ajoutera pas des informations concernant les cloisons, les facades, les linteaux,...

Le rôle du modeleur intelligent étant de reconnaitre dans ce qui est saisi les entités dont il possède une description générique, par un simple appel de fonctions d'analyse.

Ainsi il pourra reconnaitre les entités suivantes:

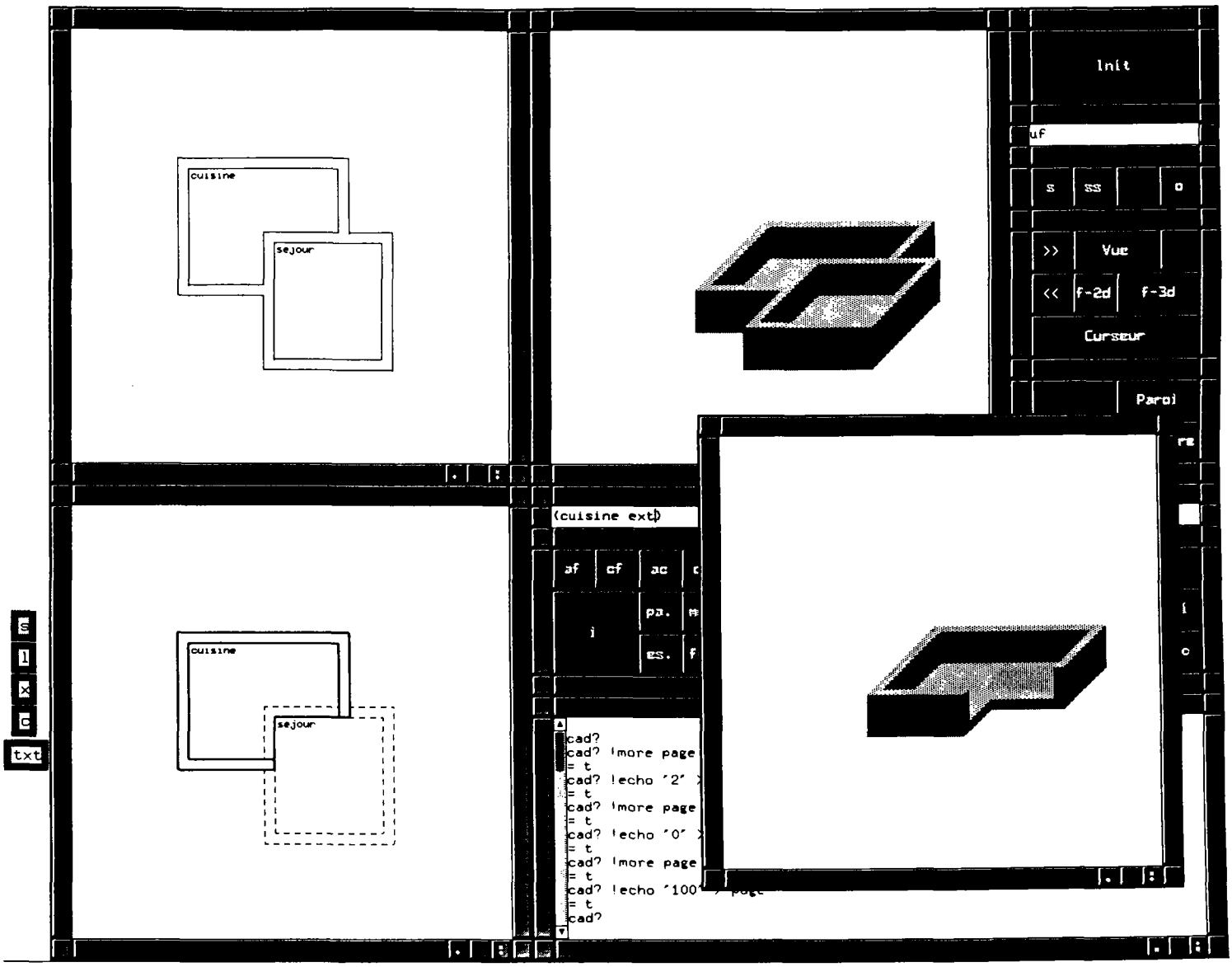
-les parois externes du projet
-les cloisons internes
-les parois du séjour
-les cloisons du séjour
-les ouvertures du séjour
-la cloison entre séjour et cuisine
-la façade du séjour
-la façade sud du séjour
-la façade nord du projet
-les ouvertures nord du projet
-.....



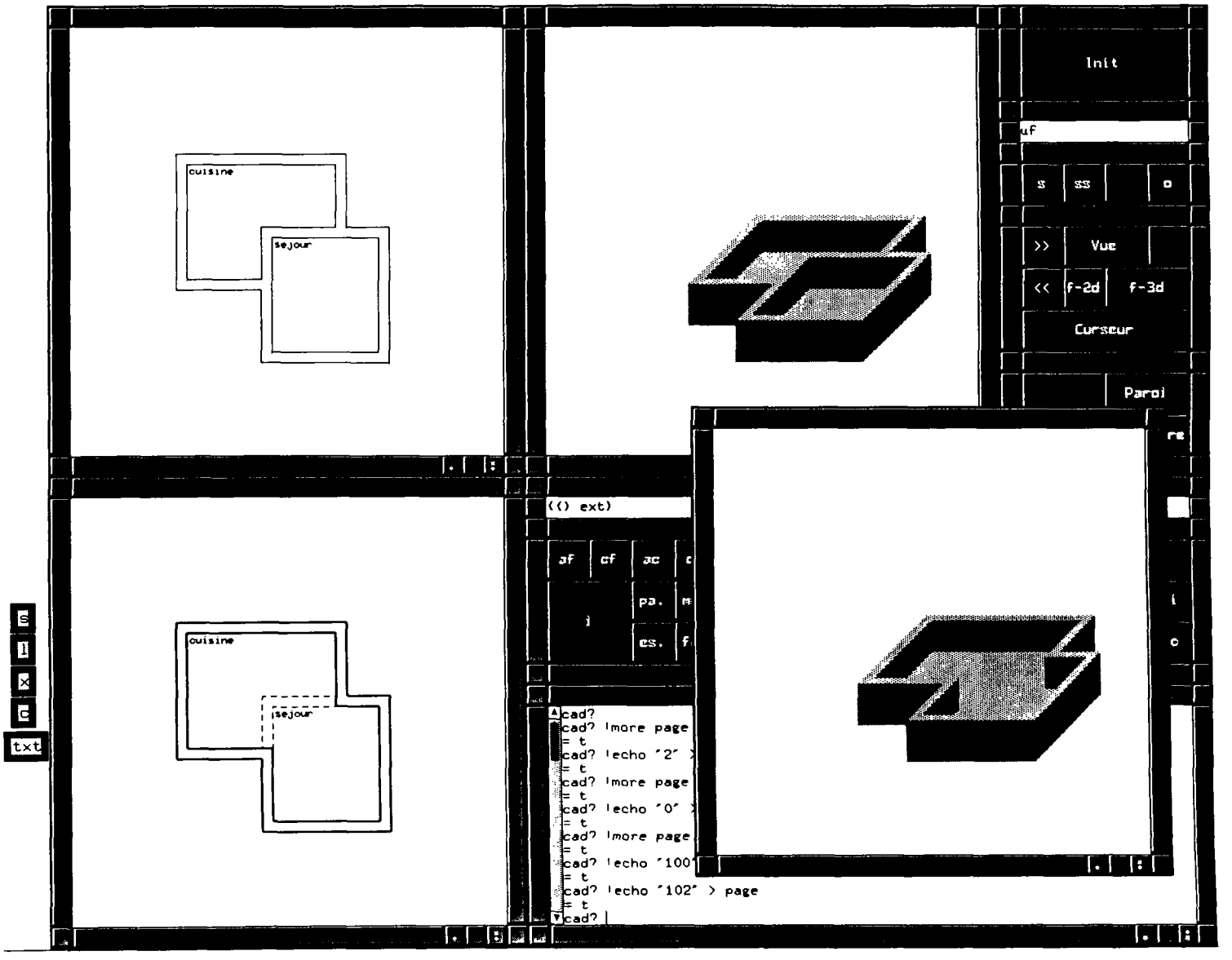

On peut voir ici comment la fonction d'analyse a pu identifier toutes les parois de la cuisine;

On rappelle que les objets identifie's sont en 3D comme on peut le ve'rifier dans la fene^tre a` droite.

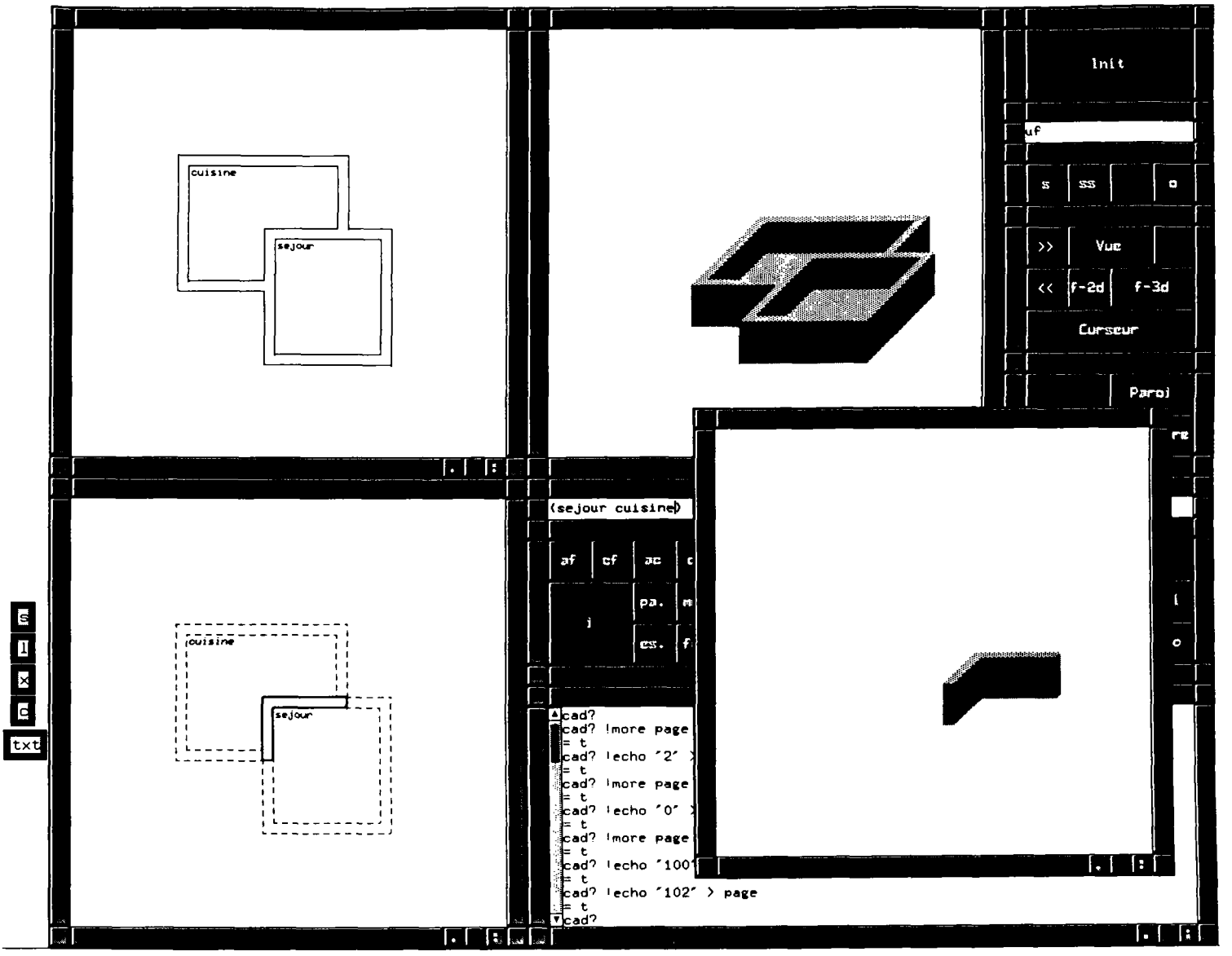
On ne visualise dans cette fene^tre que la partie des parois qui est situe'e en dessous du plan de coupe; bien su^r la fonction d'analyse elle rame`ne les parois comple`tes ce n'est qu'un effet de visualisation.




On peut voir ici comment la fonction d'analyse a pu identifier les parois exte'rieures de la cuisine;
De me^me ici, les parois identifie'es sont en 3D comme on peut le ve'rifier dans la fene^tre a` droite.

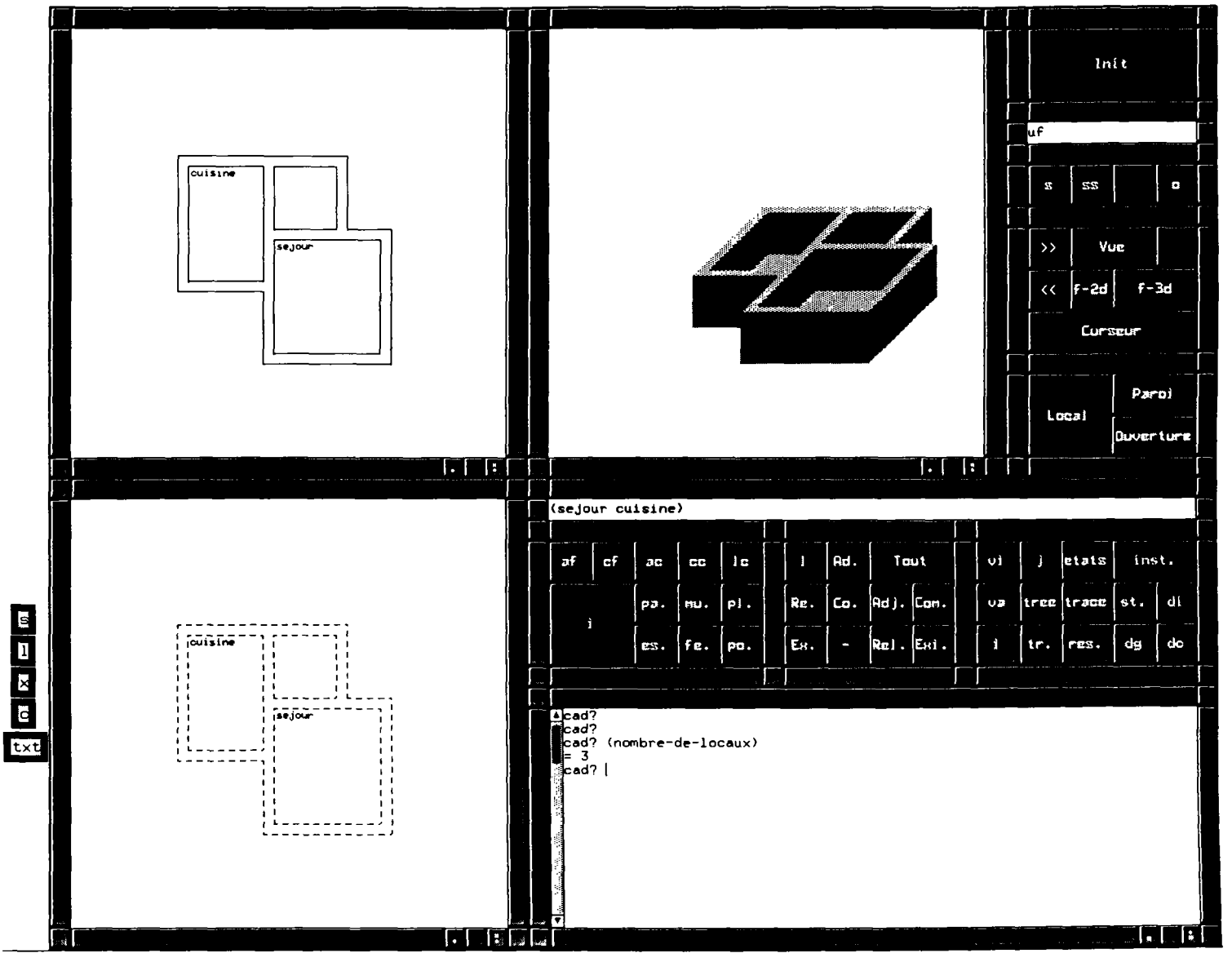



On peut voir ici les parois externes du projet

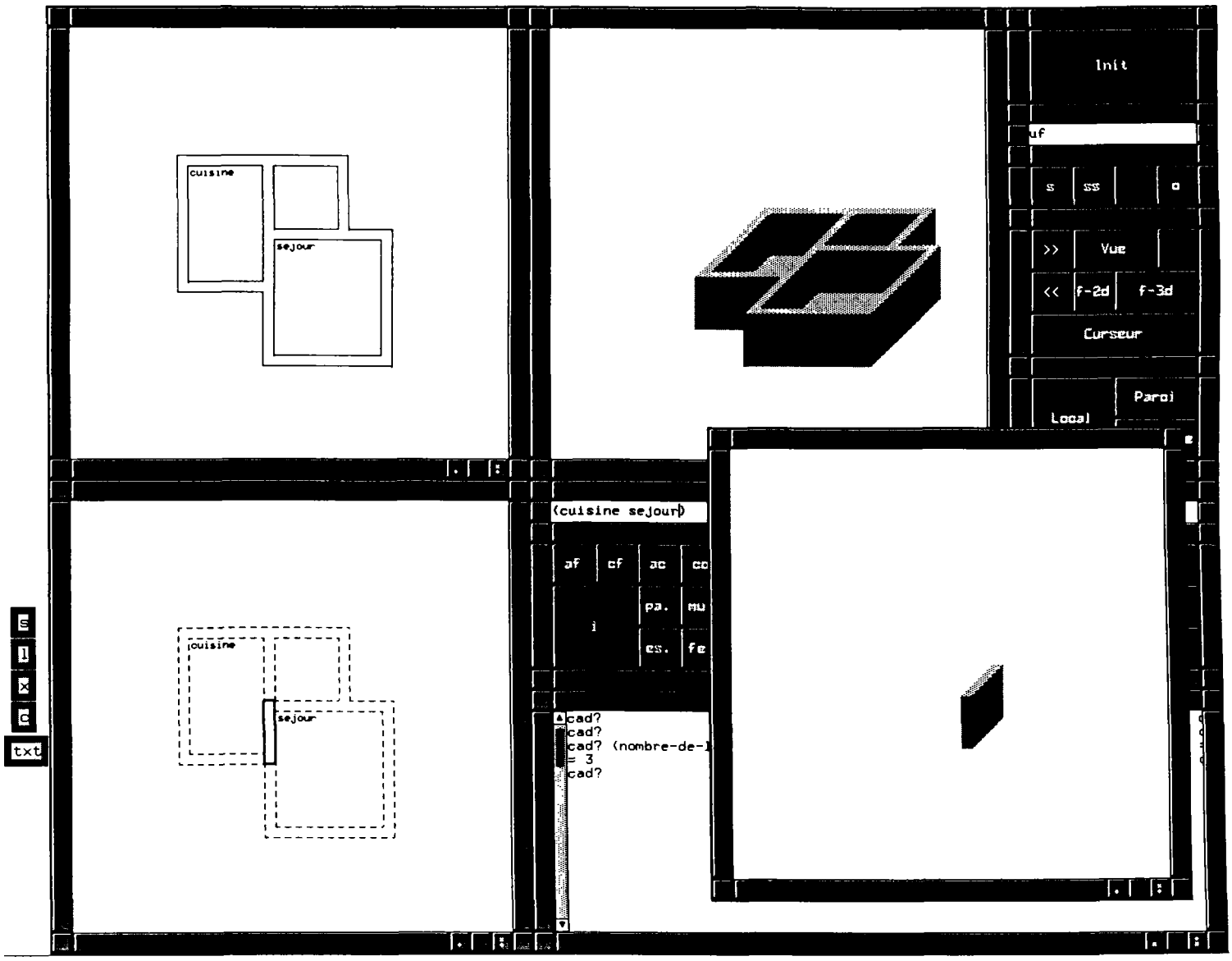



On peut voir ici la paroi commune entre le local se´jour et le local cuisine.

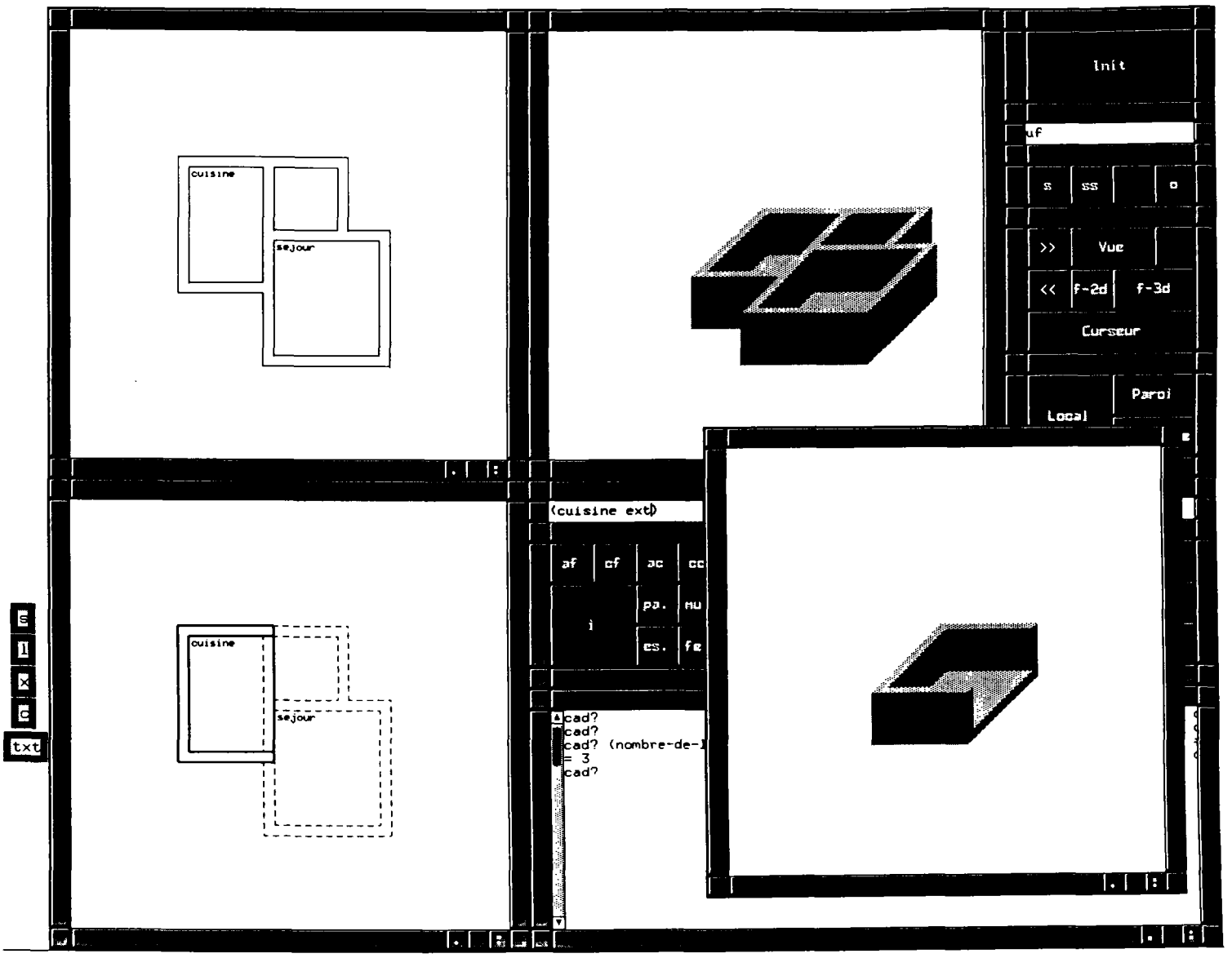
*On rappelle que toutes ces informations n´ont pas e´te´ introduites au moment de la saisie:* les fonctions d´analyses sont capables de les identifier a` partir de leur description ge´ne´rique.



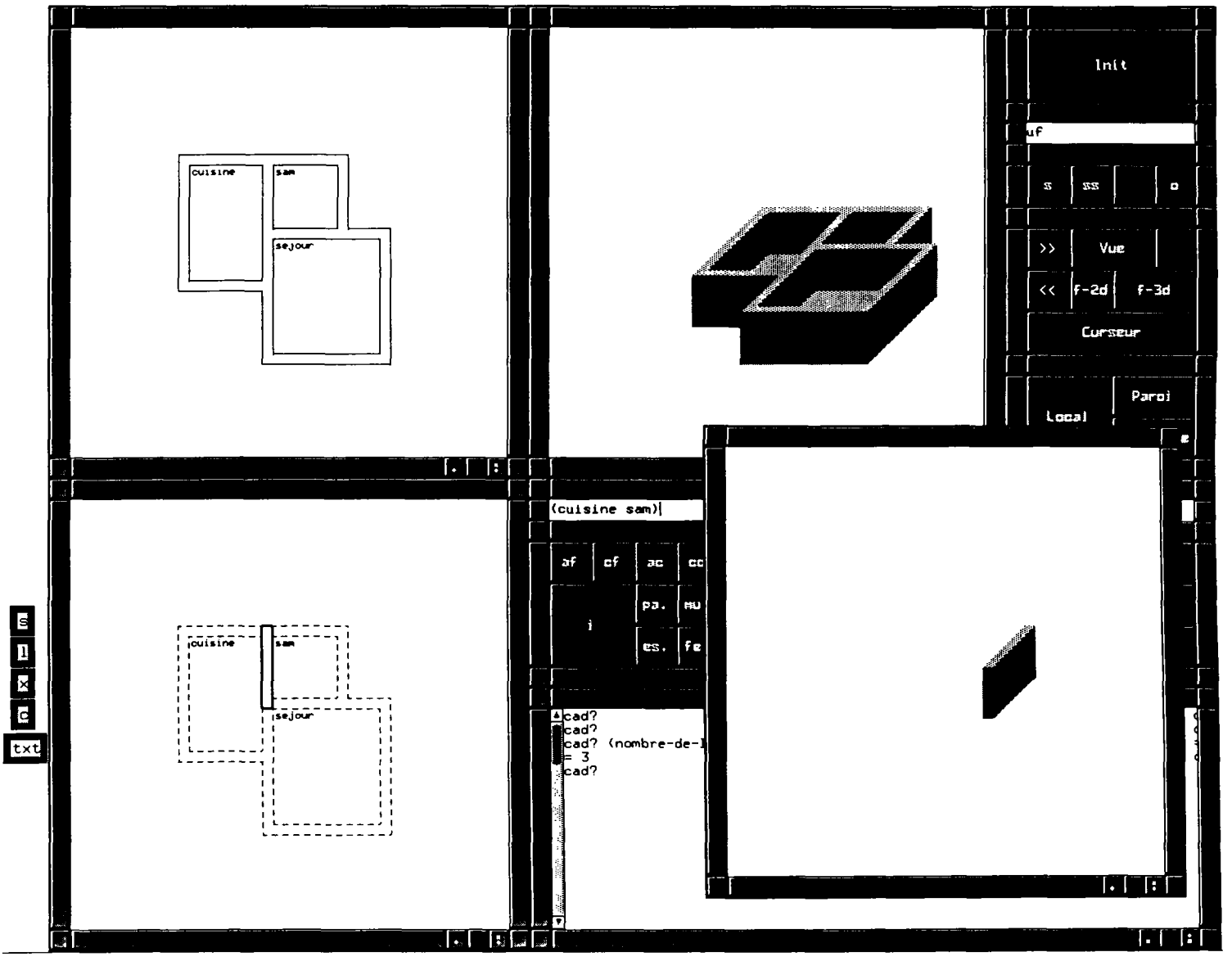

J'ajoute à mon projet une paroi qui va diviser le local cuisine; rien qu'avec cette seule modification les fonctions d'analyses son déjà capable de reconnaitre que le projet contient maintenant trois locaux (voir la réponse dans la fenêtre LISP à la requête "nombre-de-locaux").




De´ja` on peut ve´rifier que la cohe´rence ge´ome´trique est maintenue au niveau des donne´es; on peut facilement voir la re´ponse de la fonction d´analyse qui rame`ne la cloison entre le local se´jour et le local cuisine (a` comparer avec la re´ponse pre´ce´dente).



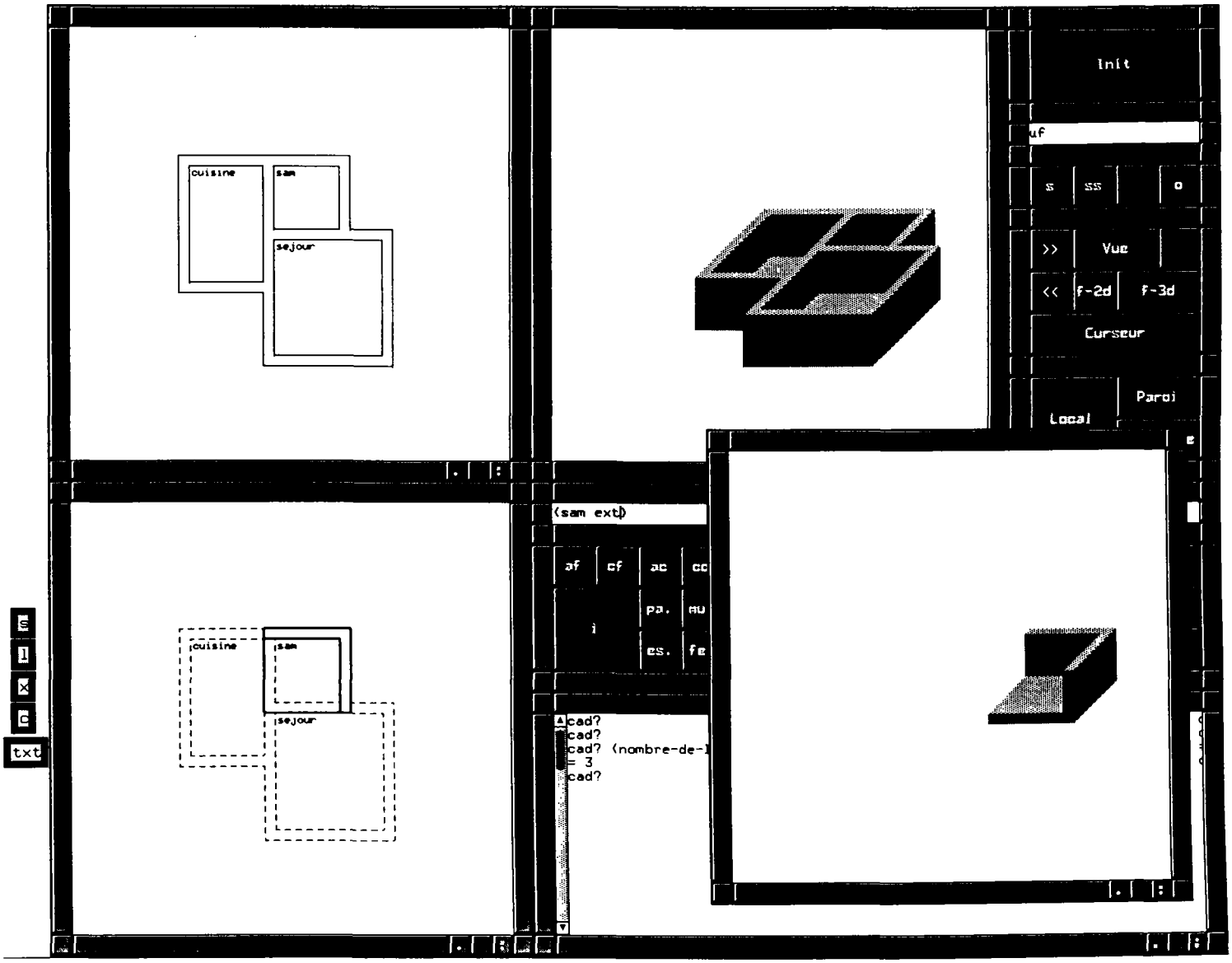

De me^me on peut ve´rifier la re´ponse de la fonction d´analyse qui rame`ne les parois exte´rieures du local cuisine (a` comparer avec la re´ponse pre´ce´dente).



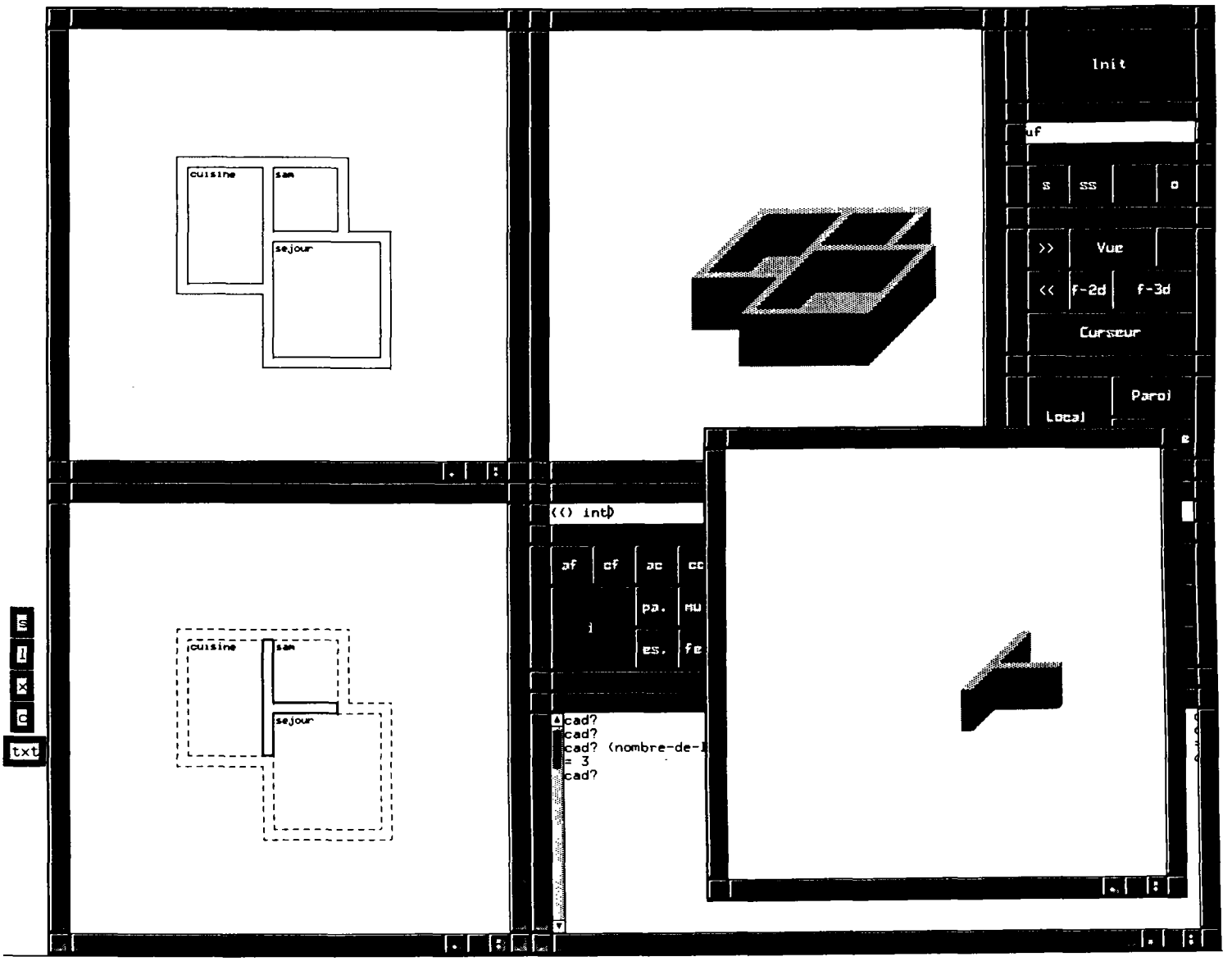

Le local qui a e'te' cre'e' implicitement par l'ajout d'une paroi est reconnu par le syste`me; on peut le manipuler de l'exte'rieur en s'adressant a` son pointeur ge'ome'trique (voir chapitre 5).

Cependant pour faciliter l'usage on lui a donne' un nom qui joue le ro^le de fonction architectural.

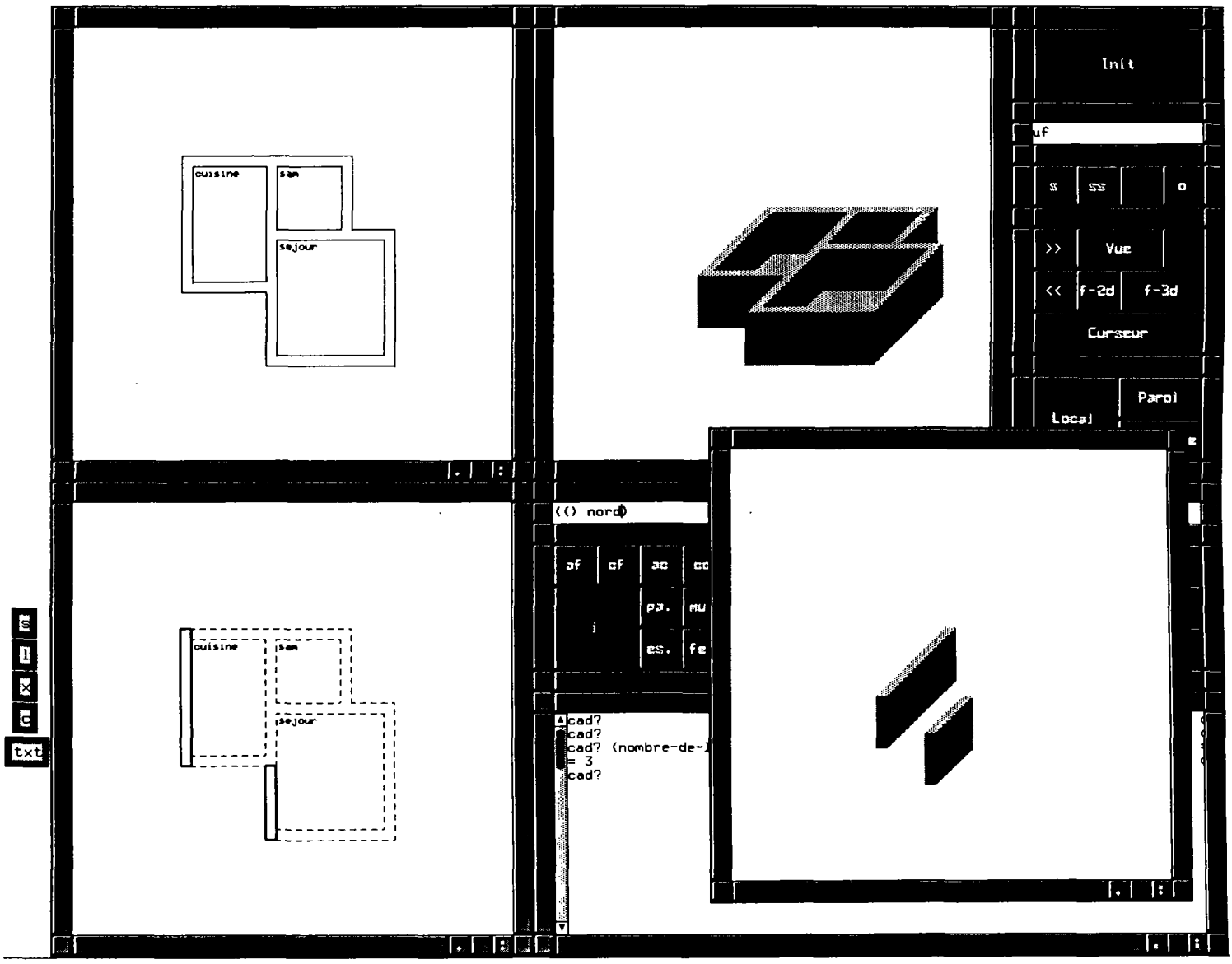
On peut par la suite demander la cloison qui se'pare la cuisine de la salle a` manger.



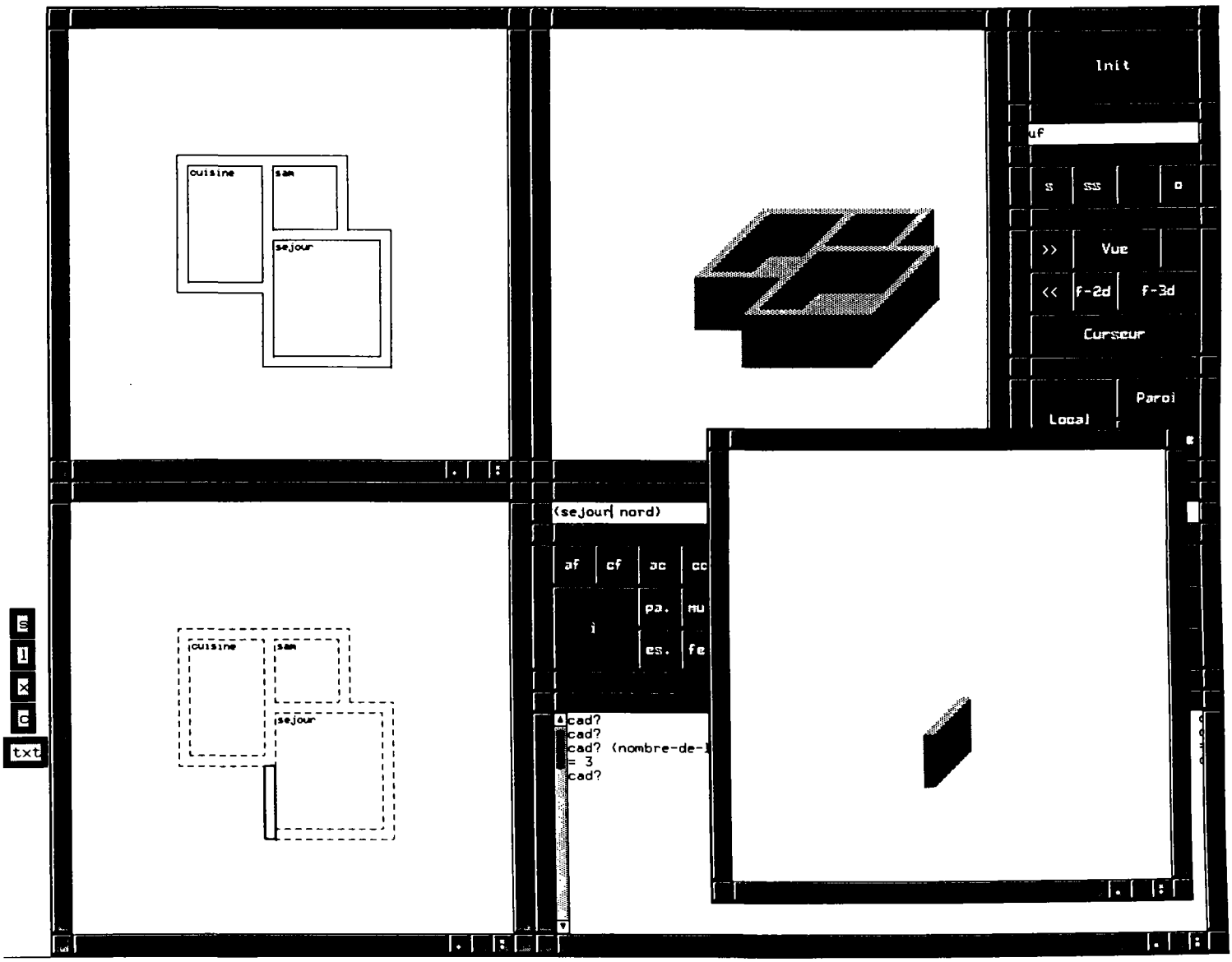

On peut de me^me demander les parois exte´rieures de la salle a`.manger.



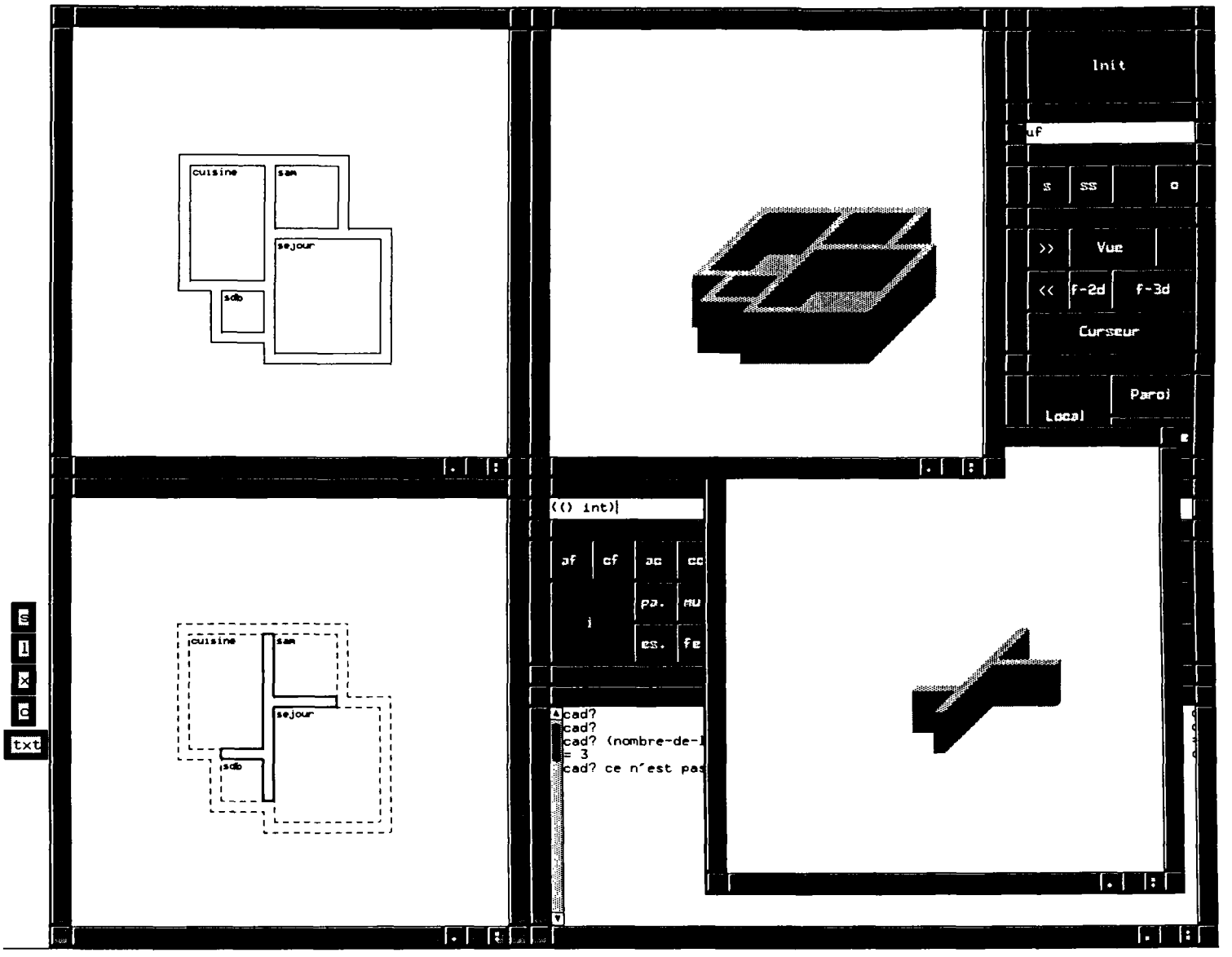

On ve´rifie aussi toutes les cloisons inte´rieures du projet.




On peut voir ici les facades nord du projet.

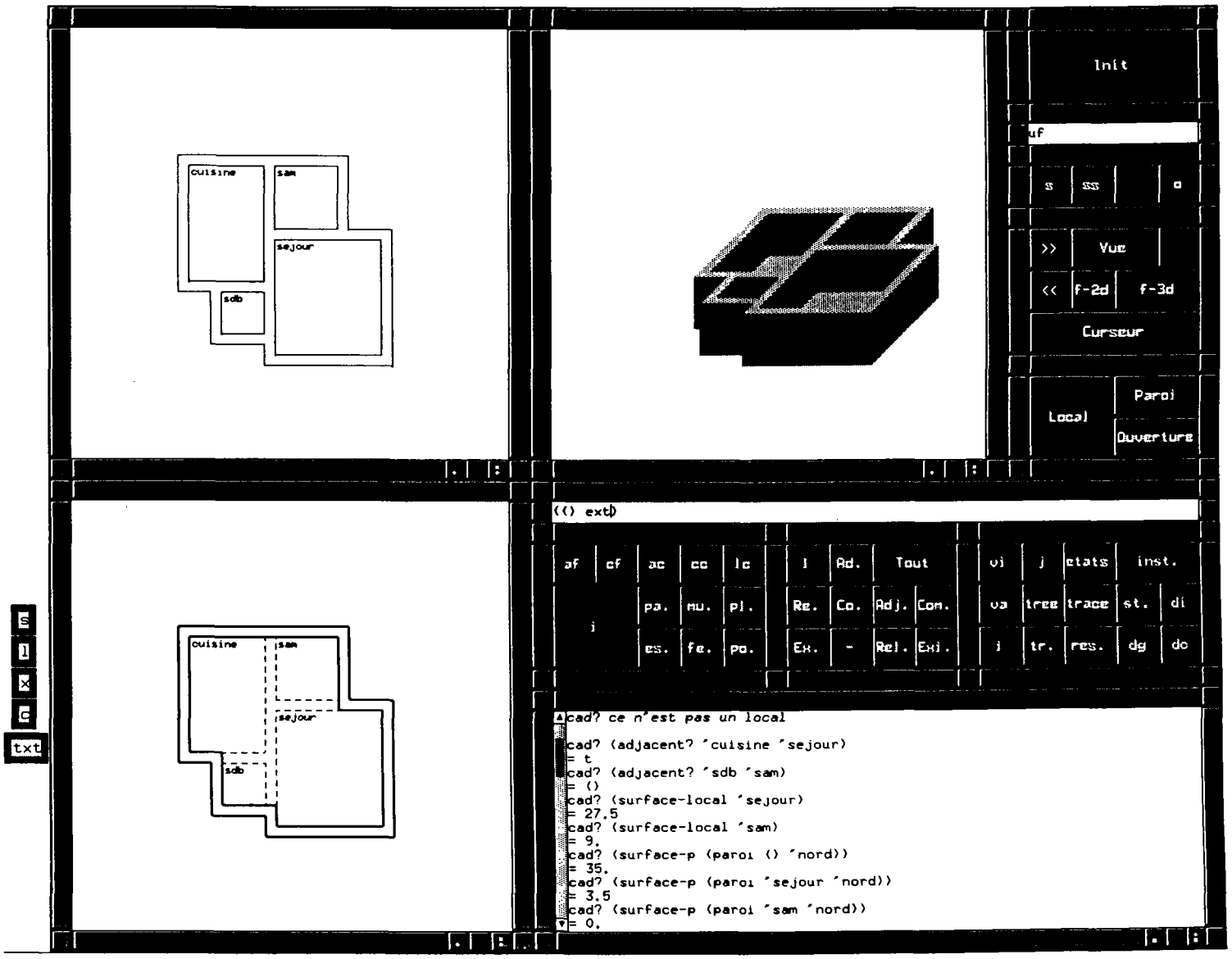



On peut voir ici les facades nord du local se´jour.




Pour rendre le projet plus complexe on ajoute´ le local sdb (salle de bain).

La` aussi je recalcule les cloisons inte´rieures du projet pour visualiser la modification.




Les fonctions d'analyses ne servent pas uniquement a` rechercher des entite's riches en se'mantiques mais aussi a` calculer certains attributs de ces objets:

-on peut voir la re'ponse "vrai" pour la reque^te "adjacent? cuisine sejour"
-ou "faux" pour la reque^te "adjacent? sdb sam"

-de me^me la surface du local se'jour et celle du loacal sam sont calcule'es
-aussi la surface des facades nord du projet, celle des facades nord du se'jour, et les facades nord de sam n'existant pas ont une surface nulle.

### 6.2.4 La vérification des contraintes

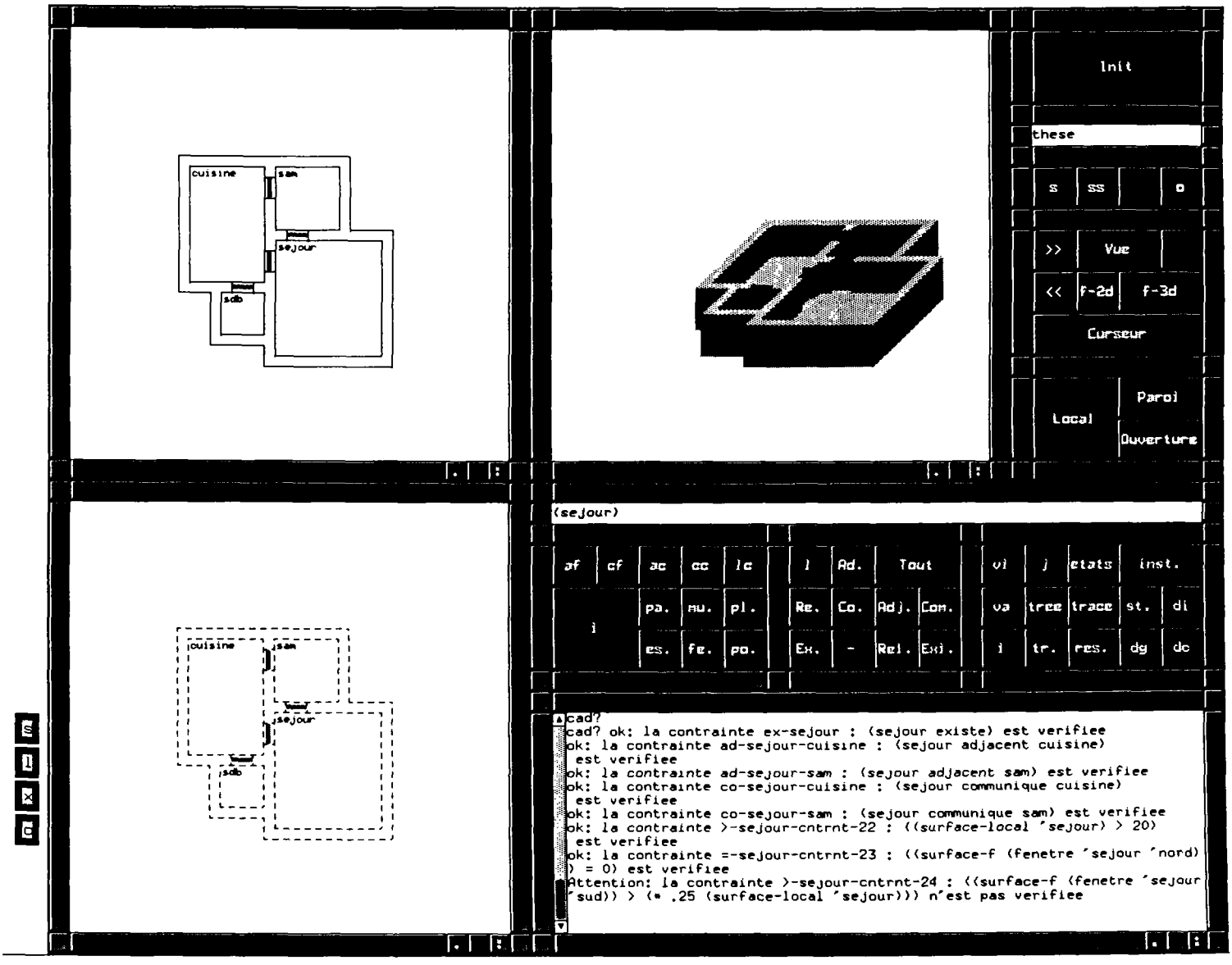
Le concepteur peut tester les contraintes si elles sont vérifiées ou pas.

En faisant une petite modification la cohérence des entités géométriques est maintenue automatiquement:

une façade devient une cloison, une paroi de séjour devient une paroi de cuisine,...

La vérification des contraintes est la première application qui montre le couplage de l'outil de CAO, donc des entités géométriques, avec un évaluateur qui manipule des entités de haut niveau riches en sémantiques.

Les données et les informations sont stockées à l'aide du modeleur géométrique et sont par la suite restaurées à l'aide des fonctions d'analyse tel que décrites dans le chapitre précédent.

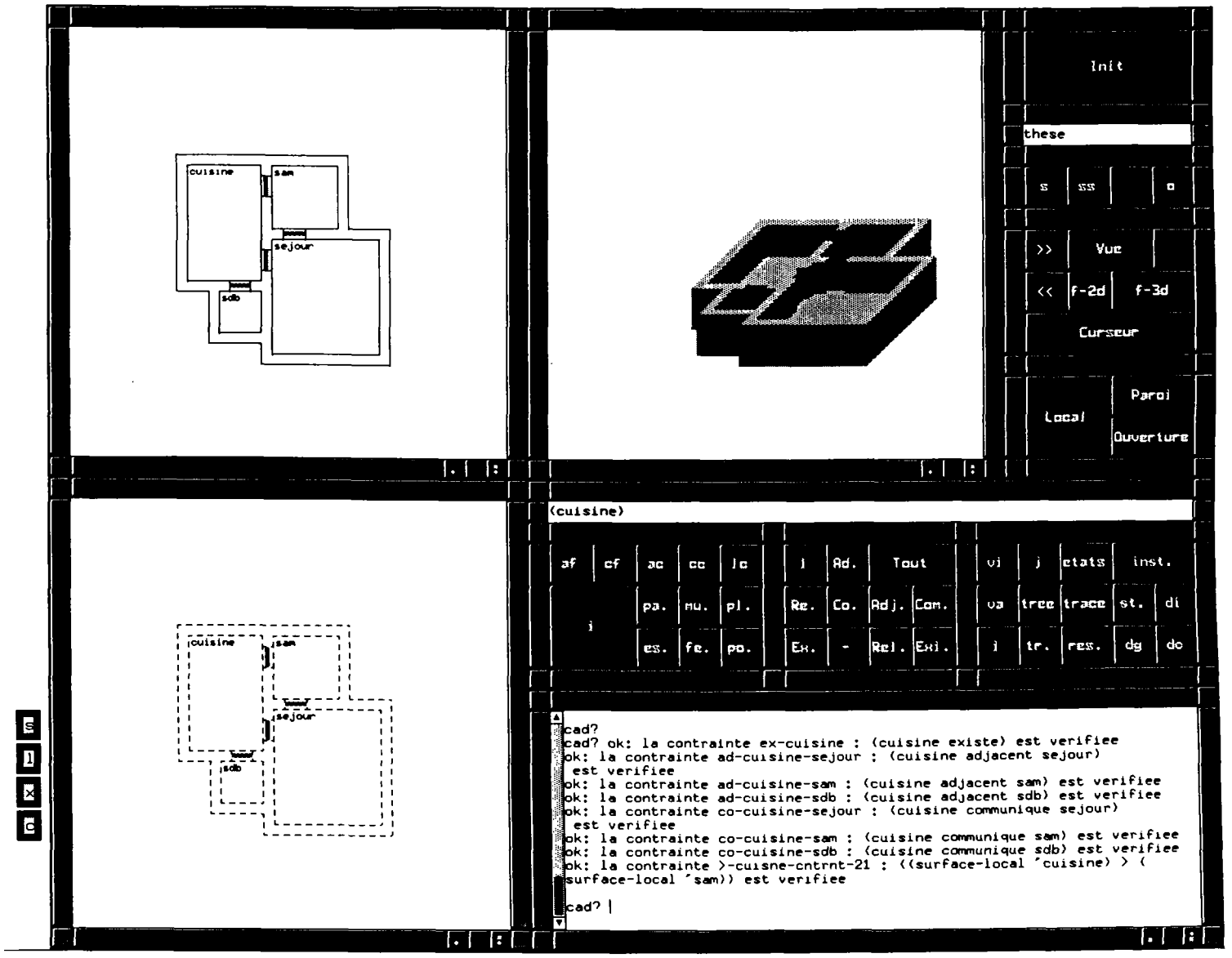



Ayant impose' les contraintes au de'part on peut lancer les me'thodes pour e'valuer le projet en fonction de ces contraintes.

On peut voir par exemple les contraintes applique'es au local se'jour; elles sont toutes ve'rifie'es:

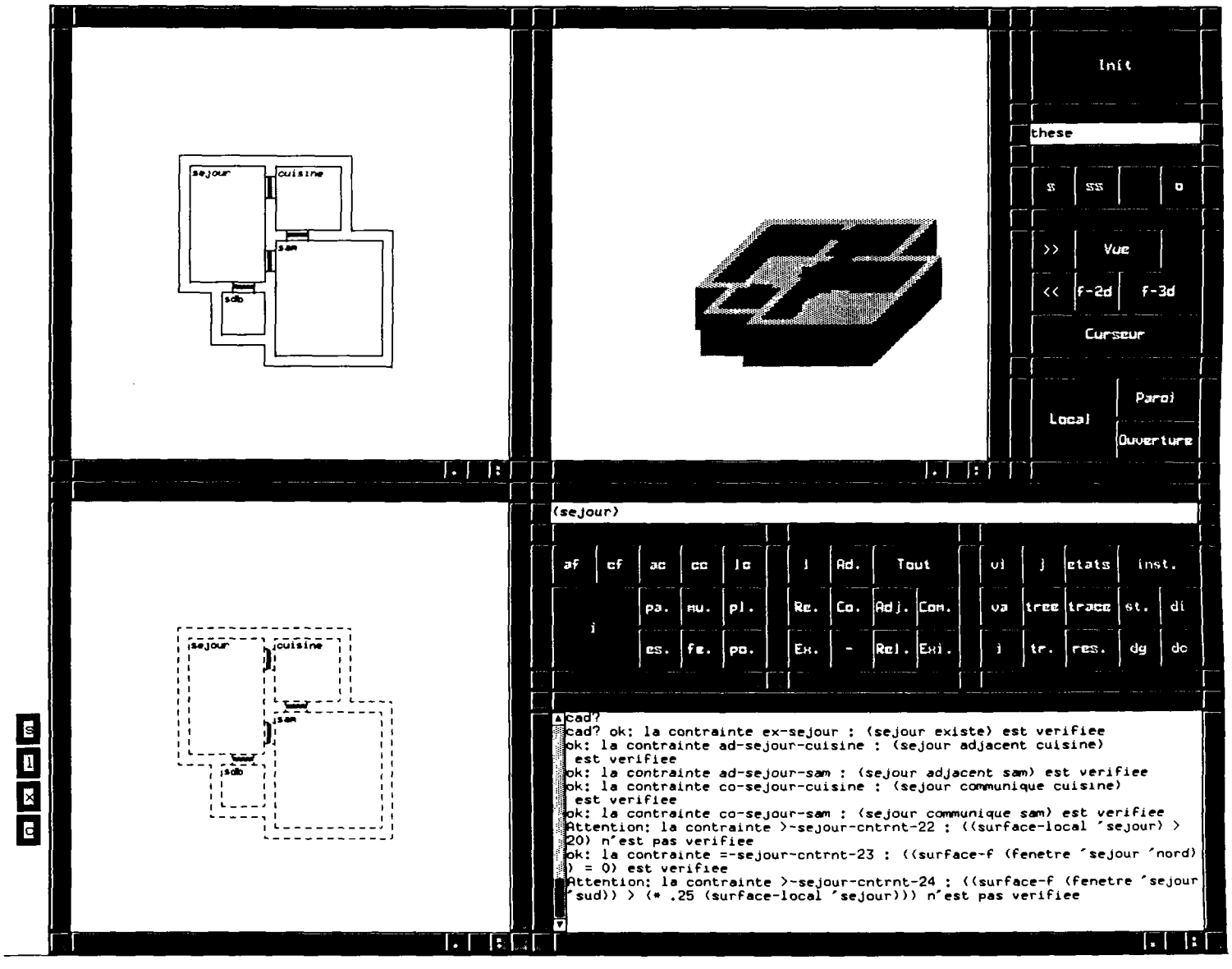
-se'jour existe
-se'jour adjacent a` cuisine et sam
-se'jour communique avec cuisine et sam
-la surface du se'jour est > 20
-la surface des fene^tres oriente'es nord est 0

*Une seule contrainte n'est pas ve'rifie'e dans l'e'tat actuel du projet: la surface des* fene^tres au sud ne vaut pas (0.25 * surface du local se'jour).




De me^me, on peut voir par exemple les contraintes applique'es au local cuisine; elles sont toutes ve'rifie'es:

-cuisine existe
-cuisine adjacent a` se'jour, sdb et sam
-cuisine communique avec se'jour, sdb et sam
-la surface de la cuisine est > la surface de la sam



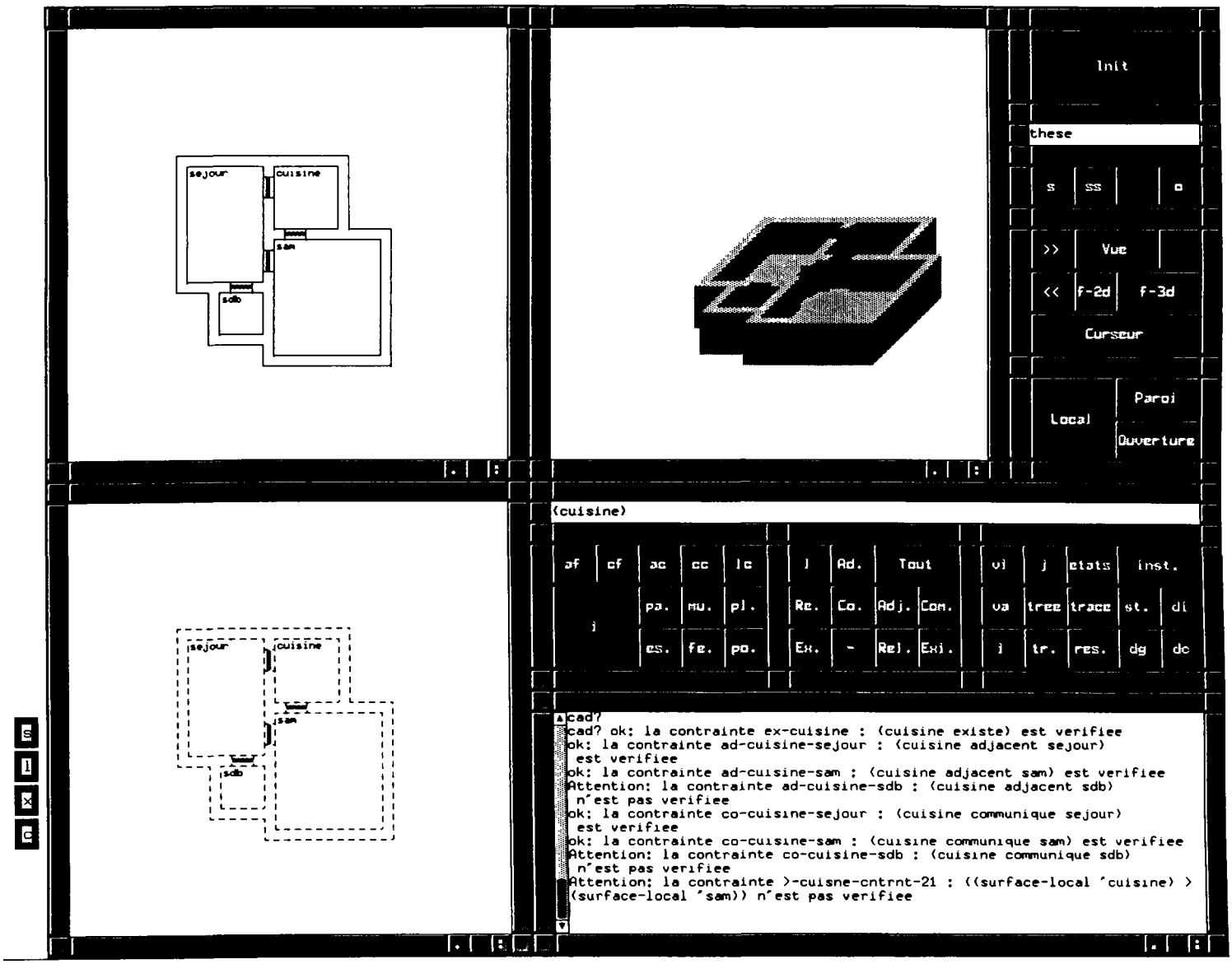

Je change la fonction des locaux comme dans la figurte ci-dessus et je relance les fonctions de ve´rification.

On peut voir par exemple les contraintes applique´es au local se´jour; elles sont toutes ve´rifie´es:

-se´jour existe
-se´jour adjacent a` cuisine et sam
-se´jour communique avec cuisine et sam
-la surface du se´jour est > 20
-la surface des fene^tres oriente´es nord est 0

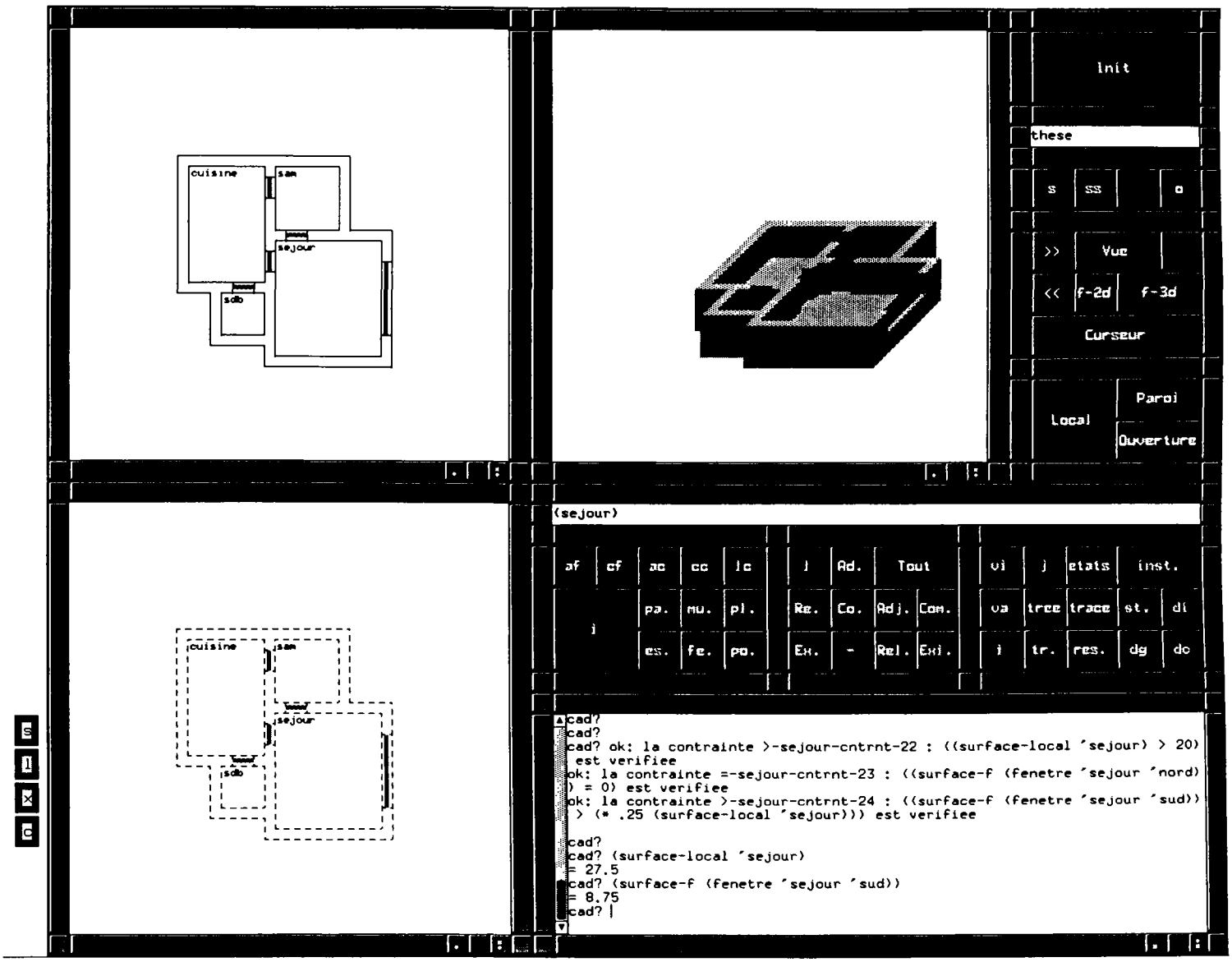
Les contraintes non ve´rifie´es sont dans l'e´tat actuel du projet:

-la surface des fene^tres au sud ne vaut pas (0.25 * surface du local se´jour)
-la surface du local se´jour est maintenant < 20




De me^me, on peut voir par exemple les contraintes applique'es au local cuisine: elles ne sont plus toutes ve'rifie'es:

-cuisine n'est plus adjacent a` sdb
-cuisine ne communique plus avec sdb
-la surface de la cuisine est maintenant < la surface de la sam

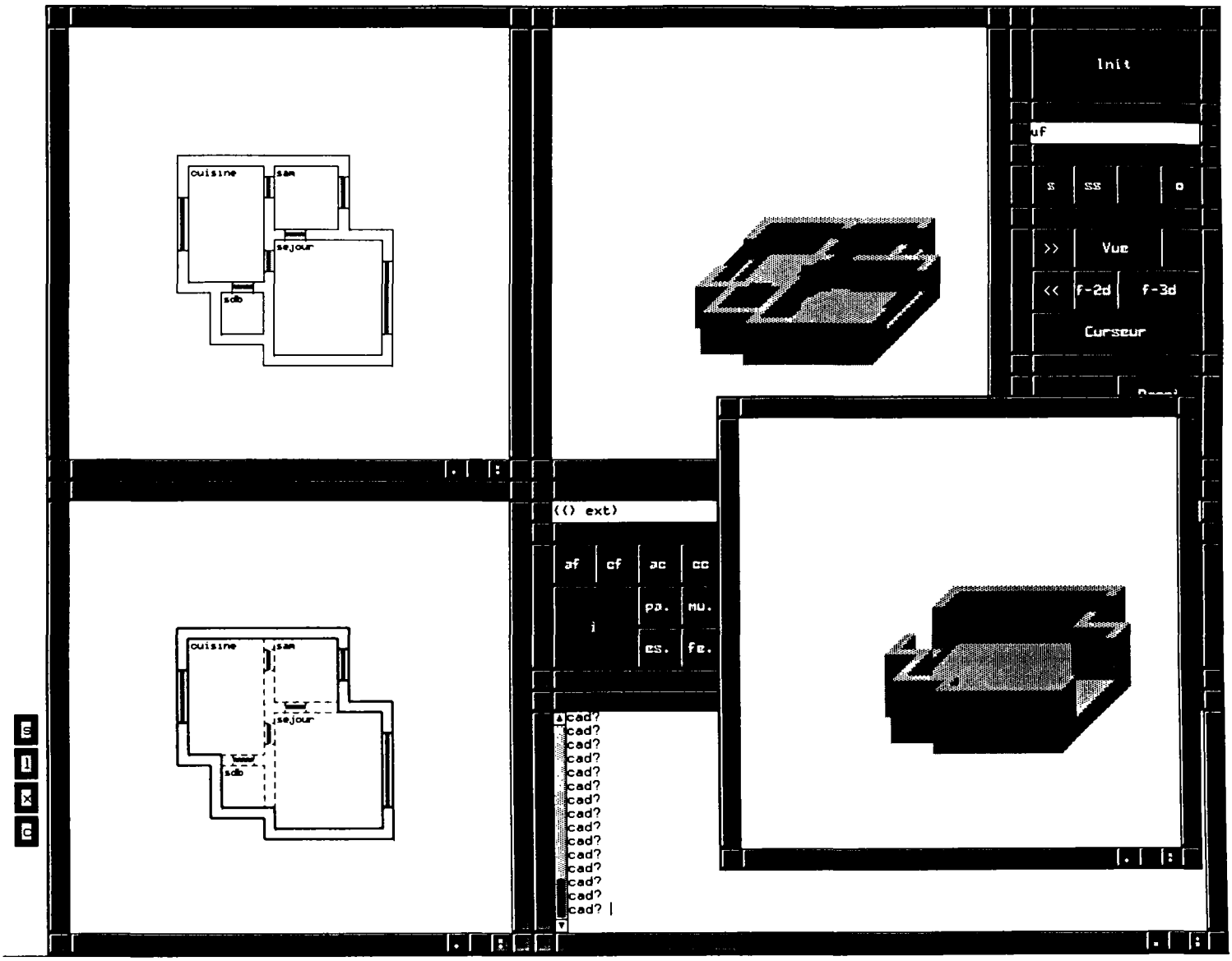



Je reprends la disposition initiale des locaux et puis j'ouvre une fene^tre dans la paroi sud du local se'jour.

Maintenant toutes les contraintes sont ve'rifie'es, et on a calcule' les valeurs suivantes:

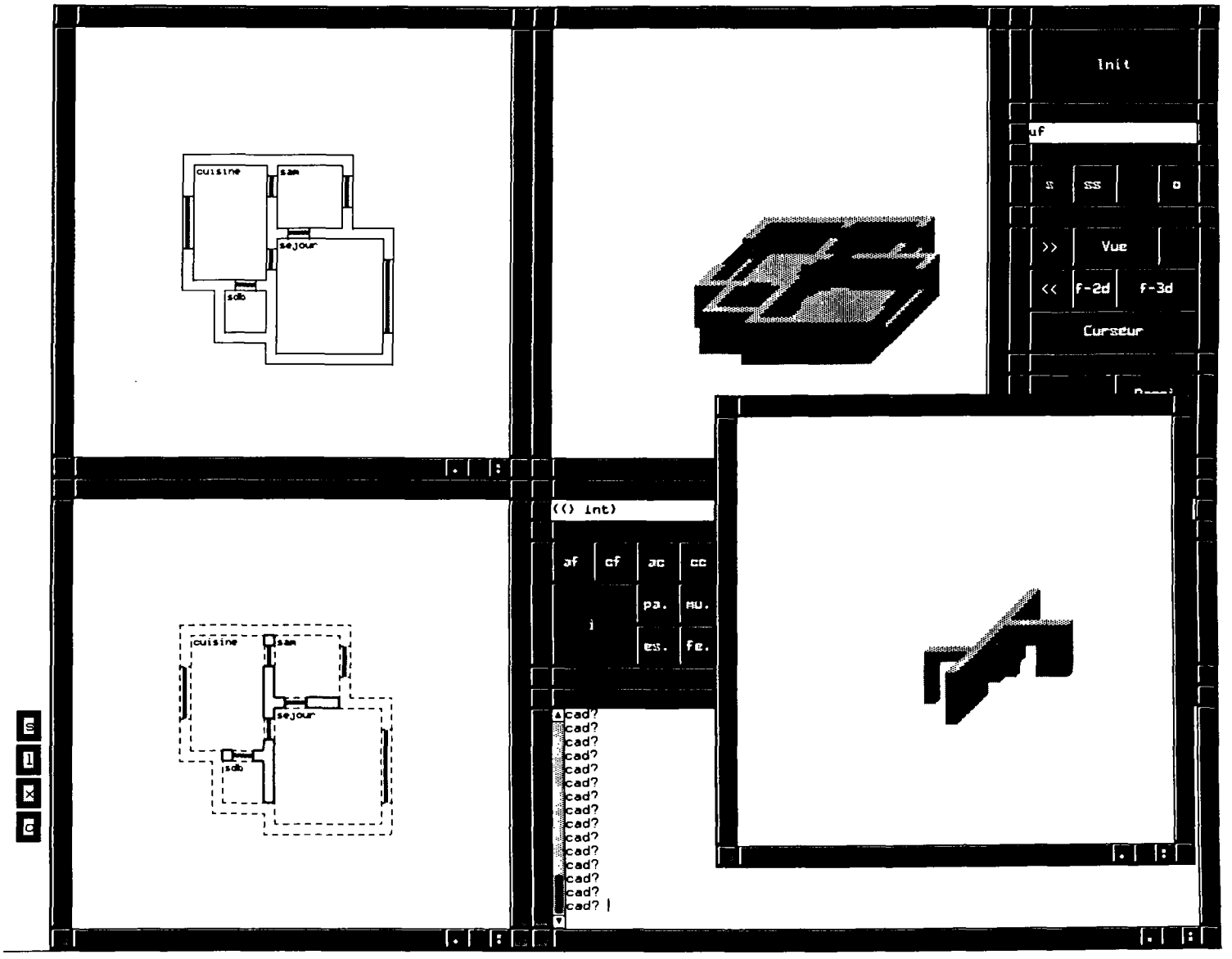
-surface se'jour = 27,5
-surface des fene^tres au sud du se'jour = 8,75

Ce qui respecte bien l'ine'quation impose'e.

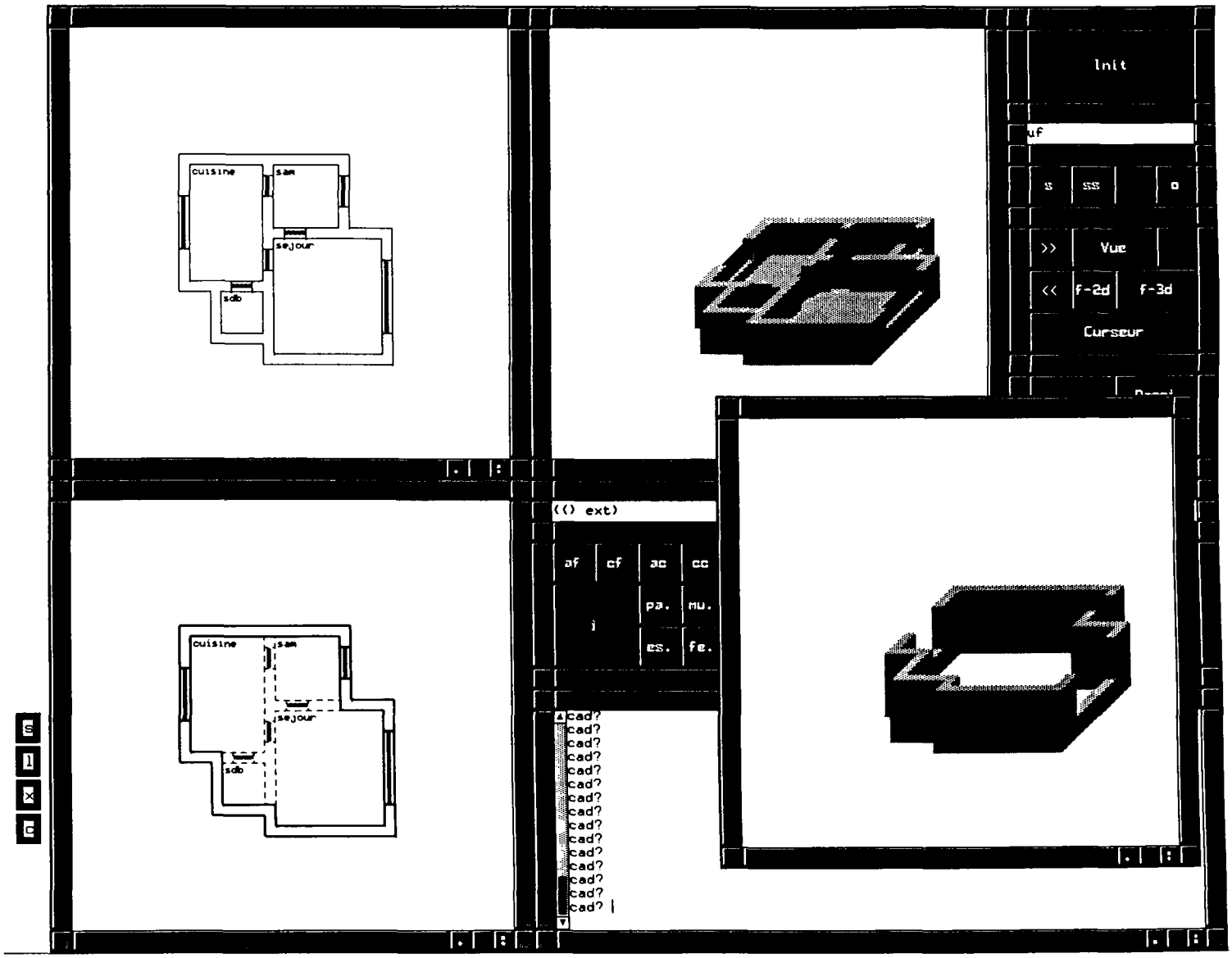



On cre´e d'autres ouvertures pour les locaux et on essaie de voir la modification apporte´e:

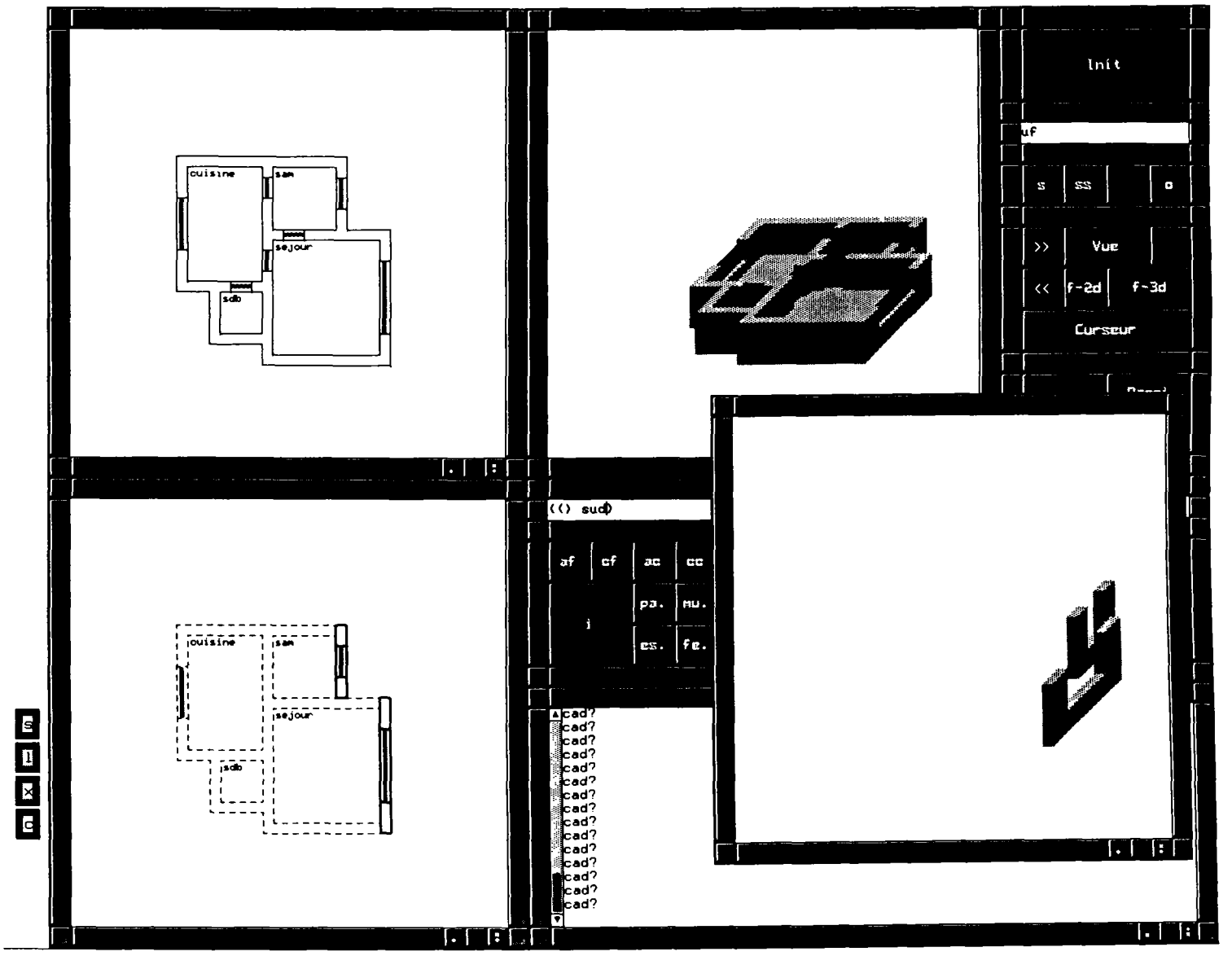
On peut voir ici la re´ponse a` la reque^te qui recherche toutes les parois exte´rieures du projet.



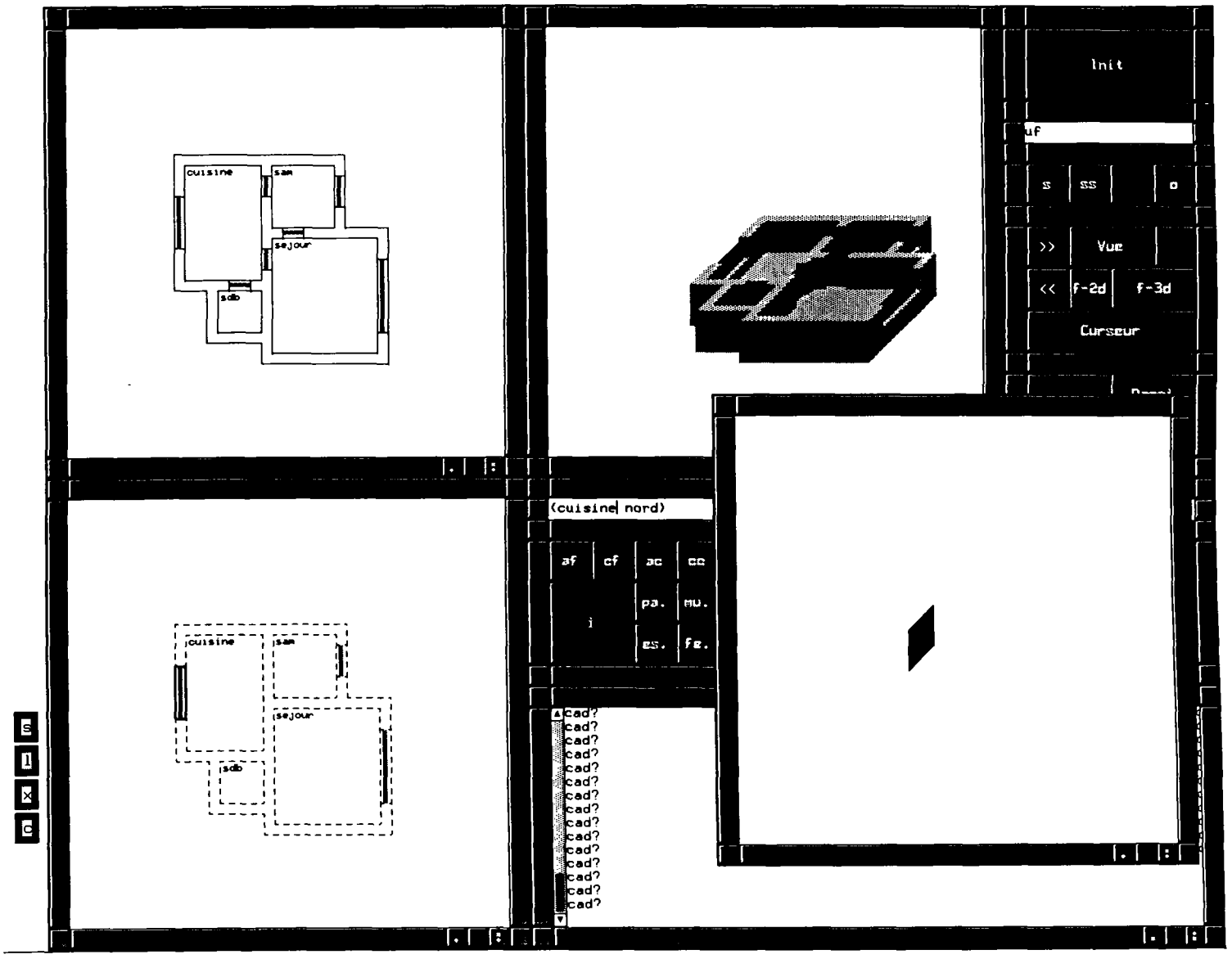

Ici on voit la re´ponse a` la requeˆte qui e´value les parois inte´rieures ou cloisons du projet.




Ici on voit la re´ponse a` la reque^te qui e´value les murs (i.e. les parois verticales) exte´rieures du projet.




Ici on voit la re´ponse a` la reque^te qui e´value les murs exte´rieures sud du projet.




Ici on voit la re´ponse a` la reque^te qui e´value les fene^tres exte´rieures nord de la cuisine.

## 6.3 Deuxième application

### la construction d'un réseau analogique à partir de la saisie géométrique

Cette application permet de montrer les possibilités de couplage de l'outil proposé avec des outils d'évaluations tel qu'un évaluateur thermique.

L'outil d'évaluation thermique qu'on vise est CSTBât un logiciel de simulation devéloppé au CSTB, qui permet de simuler l'évolution des températures dans un modèle de bâtiment [DUBOIS A.M. et AL 90]. Cet évaluateur utilise la représentation du bâtiment sous forme d'un réseau analogique où les parois et les zones sont représentés par des noeuds, qui sont reliés entre eux à l'aide de résistances qui représentent les échanges radiatifs, convectifs et conductifs entre les différents noeuds, calculées en fonction de la composition des matériaux.

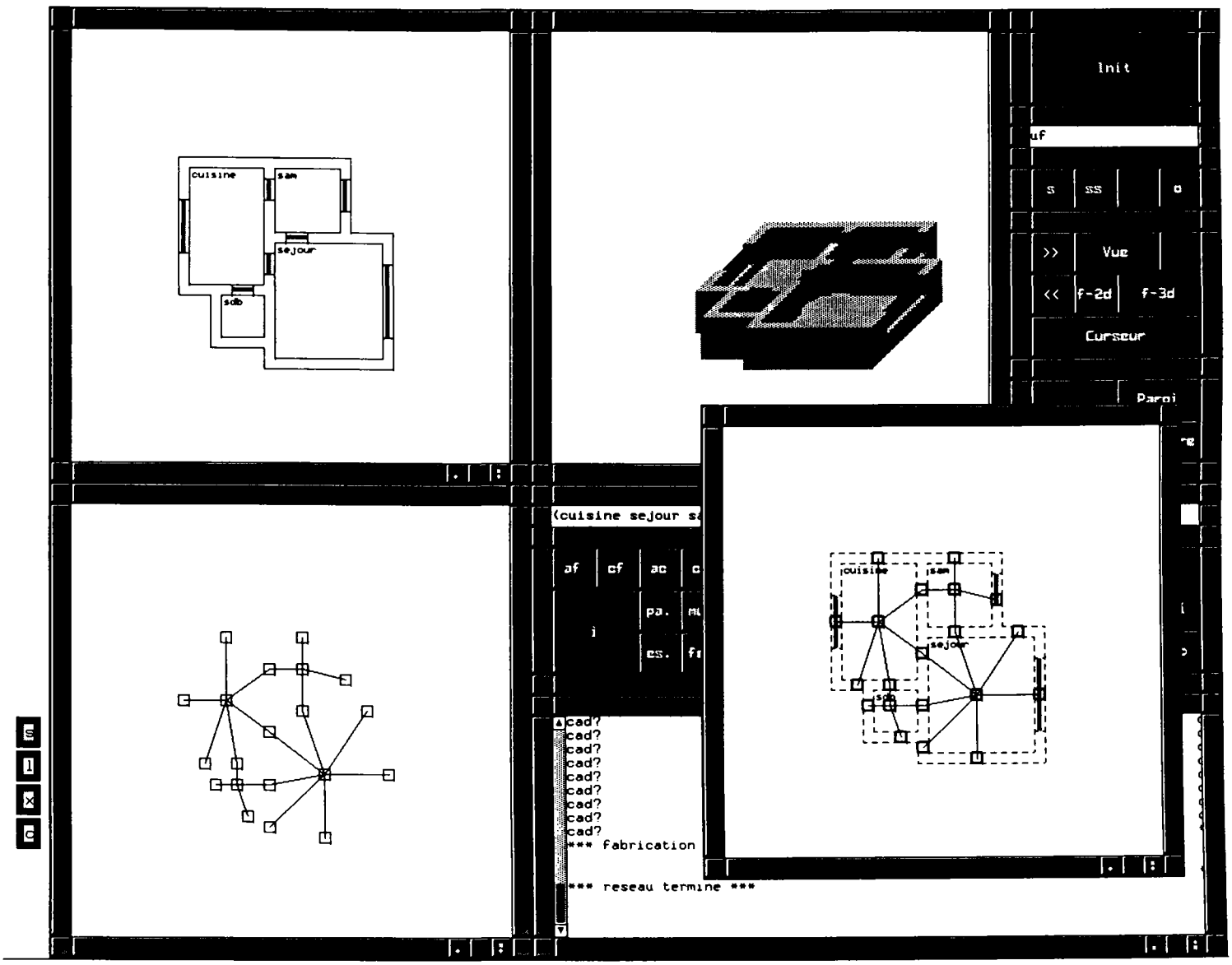
On a pris un exemple de décomposition simplifiée juste pour montrer la possibilité que peut offrir notre outil dans la reconnaissance des différentes entités.

Ainsi le programme recherche exhaustivement les espaces vides des locaux à étudier, les différentes parois extérieures verticales et horizontales qu'il découpe en facettes (plans distincts), puis les cloisons intérieures.

Les noeuds représenteront une entité dont la composition des matériaux est connue ce qui simplifie le calcul des liaisons entre ces noeuds.

#### Conclusion

L'intérêt de cet exemple est de montrer que l'écriture d'un programme est simplifiée par l'utilisation des appels à des fonctions d'analyses car le développeur écrira les fonctions en utilisant les fonctions de haut niveau en sémantique sans se soucier des détails des abscisses ou ordonnées graphiques.




On peut voir le re´seau analogique correspondant au projet ou chaque vide de local est repre´sente´ par un noeud et chaque entite´ ou facette d´une paroi (horizontale ou verticale) est repre´sente´e par un noeud. Les noeuds des parois sont relie´es aux noeuds du vide du local.

On rappelle que c´est une repre´sentation possible par un re´seau simplifie´ qui est donne´ a` titre de´monstratif.

## 6.4 Troisième application

### Le robot mobile: simulation de déplacement et calcul de trajectoire

Cette application illustre l'utilisation des entités de haut niveau par un système expert qui raisonne sur la géométrie du projet.

Un robot fictif, peut retrouver son chemin à l'intérieur du projet pour se déplacer entre deux locaux distincts (adjacents ou non) en trouvant toutes les solutions possibles ou le chemin le plus court (en terme de nombre de locaux parcourus).

Ayant trouvé le chemin à parcourir, il essaie de tracer une trajectoire qui lui permet de se déplacer et franchir les porters jusqu'à sa destination tout en évitant les obstacles et en optimisant sa trajectoire pour emprunter la plus courte.

La première partie est basé sur un système expert développé en langage de règles de SMECI. Un paquet de trois règles permettent au robot de retrouver son chemin:

-règle solution:
si le local où je suis maintenant est le local destination alors j'ai trouvé une solution

```
defregle   tester-solution

auteur georges

explication
   teste si le chemin deja parcouru est une solution
fin-explication

soit P un parcours

si
   $(eq (car (last chemin-a-parcourir^P)) local-arrivee^P)

alors

arret solution

action
     $(print "le chemin " chemin-a-parcourir^P " est une solution")

fin-regle
```

-règle impasse:
si le local où je suis a des portes qui ne communiquent qu'avec des locaux où je suis déjà passé alors le chemin déjà parcouru est une impasse
(le chemin solution que recherche le robot ne passe qu'une seule fois par un même local)

```
defregle   impasse

auteur georges

explication
   teste si le chemin parcouru ne va pas plus loin
fin-explication

soit P un parcours
   et R un robot

si $(eq (length (let ((l ())
                )
                 (mapcar (lambda (loc)
                           (if (y-a-une-porte? nom^loc
                                               (synd 'nom (car (last chemin-a-parcourir^P))) largeur^R
hauteur^R)
                               (setq l (append1 l loc))
                           )
                          )
                         (let ((li (tous-objets 'local))
                           )
                           (mapcar (lambda (loci)
                                     (setq li (remove loci li))
                                   )
                            chemin-a-parcourir^P)
                 li)
                )
           l)
     )
   0)
;si le local en question ne communique qu'avec un local deja parcouru
alors

arret contradiction

action
   $(print "le chemin " chemin-a-parcourir^P " est une impasse ou une boucle")

fin-regle
```

-règle trouver des possibilités (pour générer des hypothèses):
tout local qui communique avec le local où je suis maintenant et je n'y suis pas encore passé est un chemin possible pour trouver une solution.

```
defregle   trouver-solution

auteur georges

explication
   cherche la suite du chemin a parcourir
```

```
fin-explication


soit P un parcours
   et R un robot
    et L un local parmi $(let ((l (tous-objets 'local))
                        )
                         (mapcar (lambda (loc)
                                 (setq l (remove loc l))
                               )
                          chemin-a-parcourir^P)
                   l)


si
     $(y-a-une-porte? nom^L (synd 'nom (car (last chemin-a-parcourir^P))) largeur^R
hauteur^R)

alors

affecter chemin-a-parcourir^P avec $(append1 chemin-a-parcourir^P L)

fin-regle
```

Cette règle génére pour chaque possibilité une branche dans l'arbre des états maintenu par SMECI; les feuilles de cet arbre peuvent être devéloppées ultérieurement en appliquant le même paquet de règles. Certaines feuilles arrêtent le raisonnement soit par ce qu'on atteint une solution soit par ce qu'on est dans une impasse.

Pour la recherche d'une seule solution possible, les règles sont les mêmes à une différence que le moteur d'inférence suspend le raisonnement dès qu'il atteint une solution, et que l'exploration des règles est prévue pour retrouver le chemin le plus court en premier.

```
defregle   trouver-un-chemin-solution

auteur georges

explication
   teste si le chemin deja parcouru est une solution
fin-explication

soit P un parcours

si
   $(eq (car (last chemin-a-parcourir^P)) local-arrivee^P)

alors
```

```
arret solution

action
    $(print "le chemin " chemin-a-parcourir^P " est une solution")
    $(suspendre-raisonnement)

fin-regle
```

Dans cette première partie de l'application, le raisonnement d'un système expert était le moyen le plus simple et intuitif. Les règles sont écrites en faisant appel à des fonctions d'analyses pour chercher les locaux adjacents et les locaux communicants, ainsi que la position d'une porte et sa dimension.

Dans la deuxième partie de cette application, le robot essaie de trouver une trajectoire pour aller dans le local destination. Pour cette partie on n'a pas besoin d'un système expert et on a utilisé un style de programmation ordinaire.

Le robot cherche la porte de sortie et calcule la trajectoire pour y aller en ligne droite; s'il n'y a pas de paroi, il y va, sinon il arrive jusqu'à la paroi obstacle et il la longe jusqu'à ce qu'il arrive à côté de la porte pour la franchir et recalculer la nouvelle trajectoire.

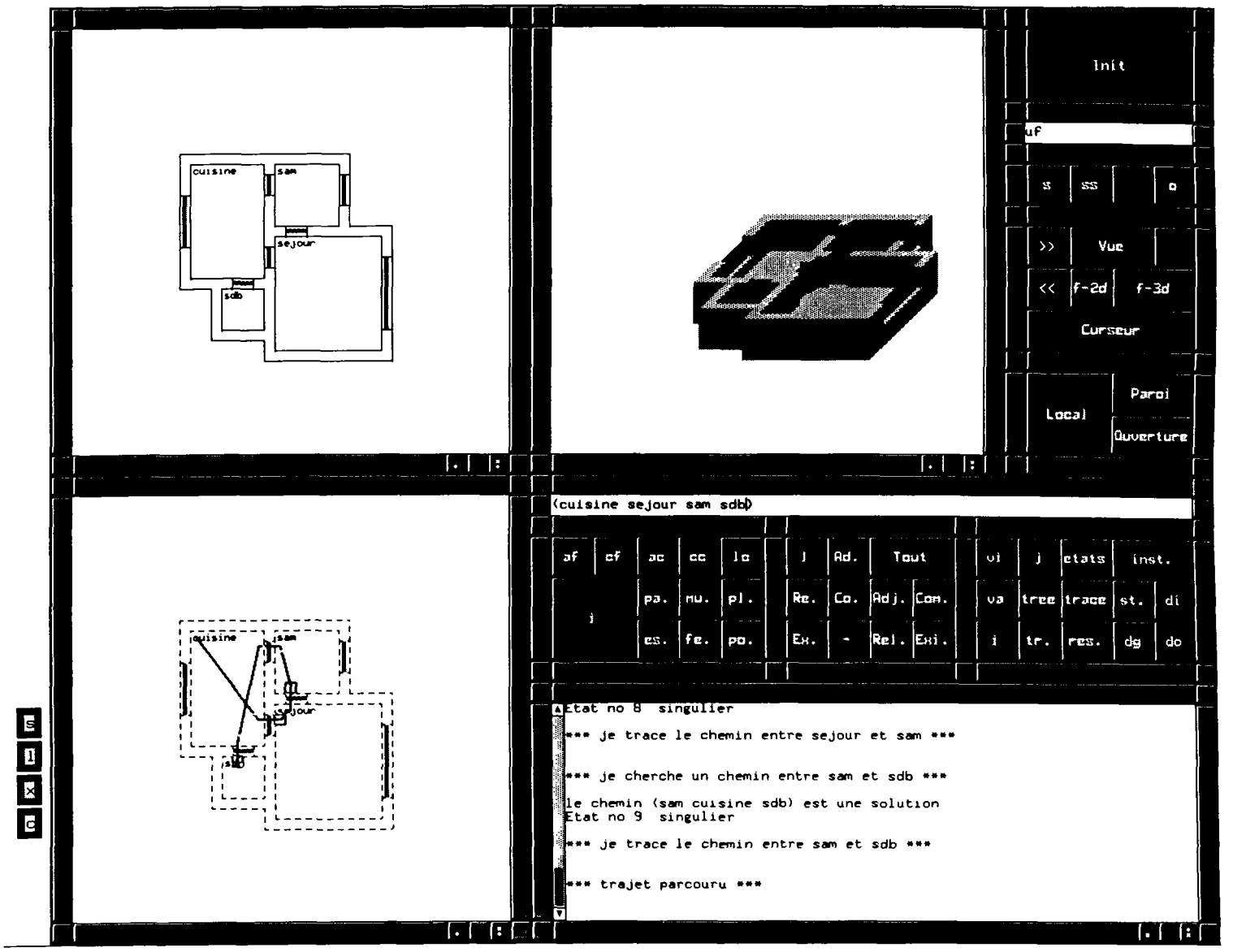
On a ajouté une partie de code qui peut optimiser un parcours en ligne brisées en essayant de lier deux points par une diagonale.

Pour cette partie donc on s'est limité à fournir une solution sans énumérer toutes les solutions possibles.

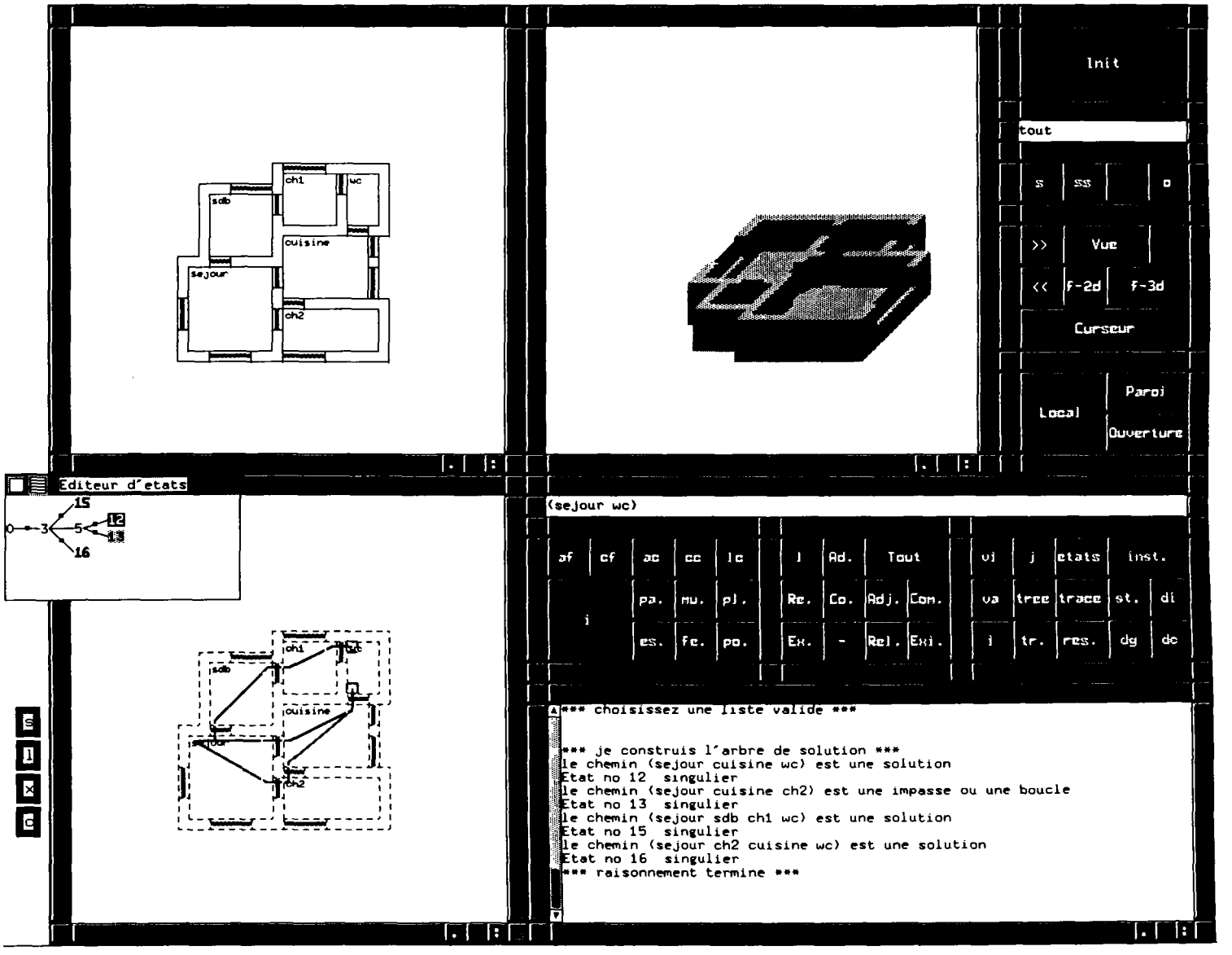
## Conclusion

Cette application montre surtout la combinaison de la programmation traditionnelle avec une programmation à l'aide de règles de production et de systèmes experts.

De même qu'elle montre la simplification de la résolution de certains problèmes par l'utilisation d'un système expert, et surtout la simplification qu'apporte notre modèle pour l'écriture de règles à l'aide d'un langage de haut niveau en utilisant les concepts du modeleur intelligent.



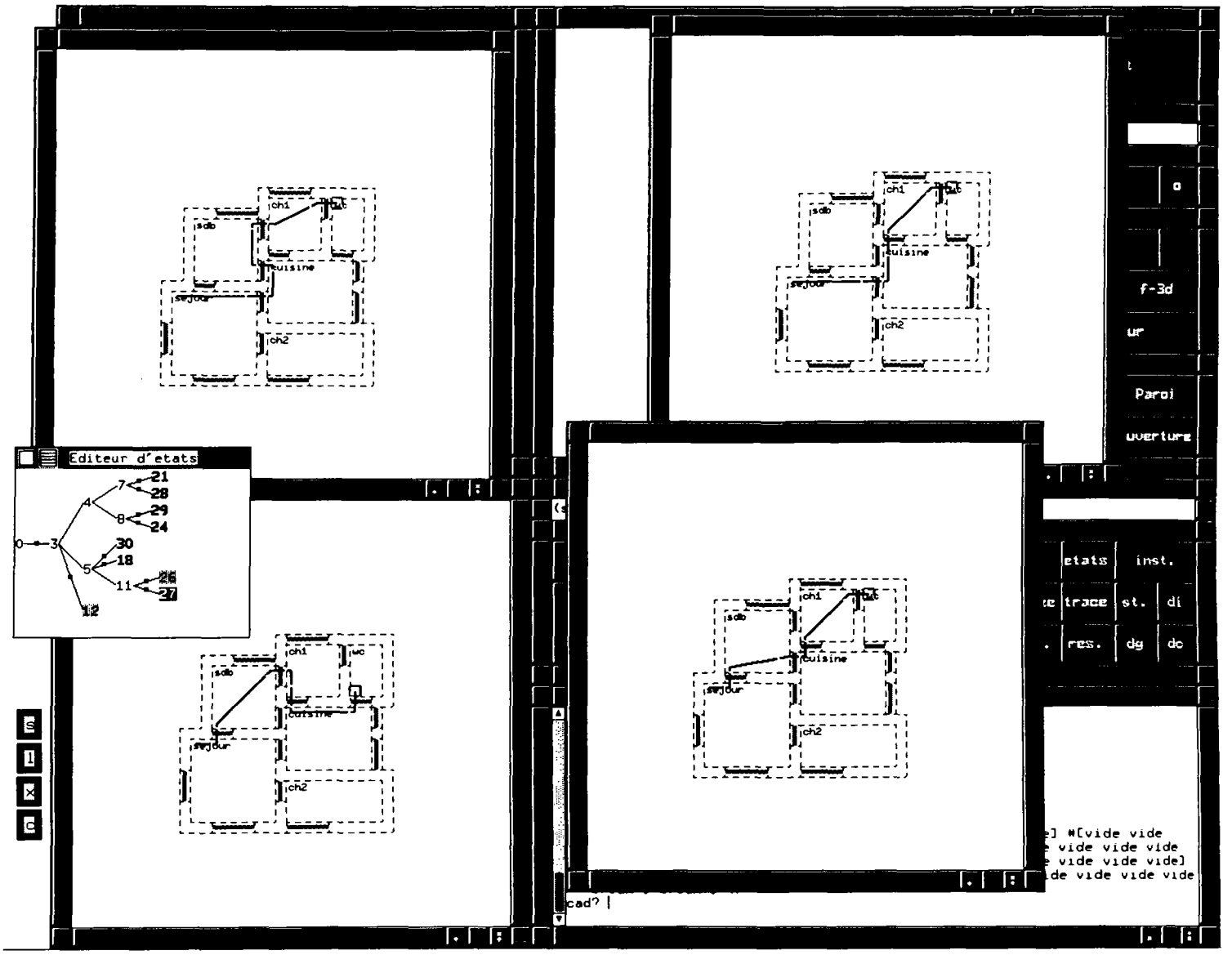

On peut voir ici un parcours du robot mobile qui a essaye´ de de´crire un trajet partant du local cuisine pour aller au local se´jour puis au local sam puis au local sdb.




Ici le robot a essayé de trouver tous les chemins possibles qui permettent d'aller du local séjour au local wc. On peut voir sur l'arbre des états générés par le raisonnement les solutions sont marquées en gras.

Il trouve trois solutions qu'on a essayé de visualiser:

-séjour -> cuisine -> wc
-séjour -> sdb -> ch1 -> wc
-séjour -> ch2 -> cuisine -> wc

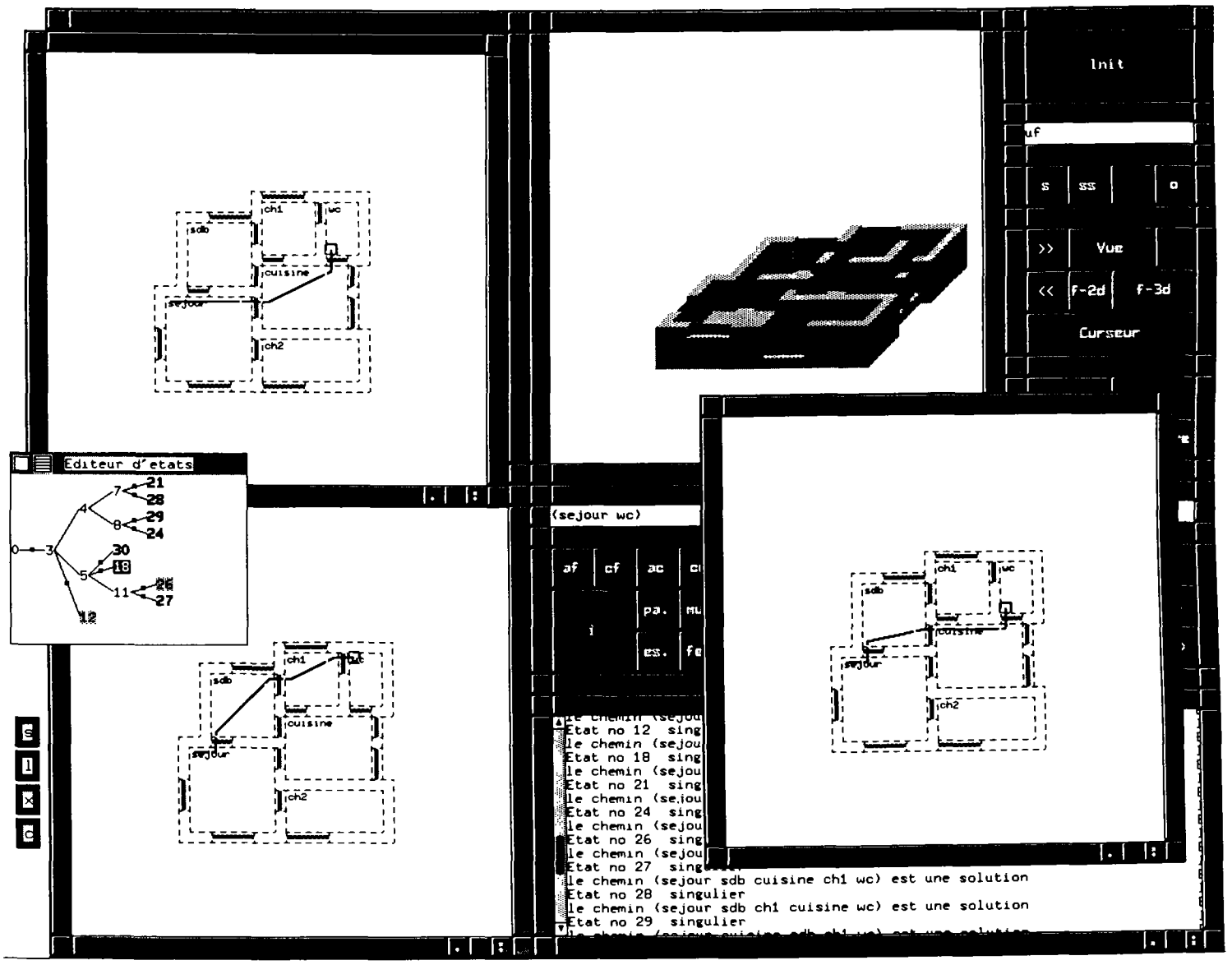



On apporte quelques modifications au plan:

-on ferme la porte entre ch2 et cuisine
-on ouvre une porte entre cuisine et ch1
-on ouvre une porte entre cuisine et sdb

Il trouve alors sept solutions (voir l'arbre des e'tats).
On a essaye' de visualiser 4 solutions ci dessus et 3 sur la page suivante:

-se'jour -> cuisine -> sdb -> ch1 -> wc
-se'jour -> sdb -> ch1 -> cuisine -> wc
-se'jour -> cuisine -> ch1 -> wc
-se'jour -> sdb -> cuisine -> ch1 -> wc




On visualise ici les 3 autres solutions:

-se´jour -> sdb -> ch1 -> wc
-se´jour -> sdb -> cuisine -> wc
-se´jour -> cuisine -> wc

## 6.5 Quatrième application

**Système expert pour la conception d'un circuit de chauffage hydraulique**

Cette application est créée pour montrer la possibilité d'utilisation de notre outil dans un problème de conception. La démarche adoptée étant du type générer puis tester.

Le problème à résoudre est la conception d'une installation de chauffage hydraulique à partir de la volumétrie du projet qui a été saisie à l'aide du modeleur de CAO. Plusieurs étapes sont alors envisagées:

-Le choix de la position des radiateurs
-le choix de l'endroit de passage de la tuyauterie

### 6.5.1 L'emplacement des radiateurs

Les radiateurs sont placés contre une paroi du local. On choisit de préférence une paroi externe pour compenser les déperditions avec l'extérieur; sinon le radiateur peut être placé contre une cloison s'il n'y a pas de parois externes ou bien par un choix du concepteur.

Dans notre application on a adopté l'hypothèse qu'il existe un seul radiateur par local. Pour choisir son emplacement on a écrit un paquet de règles qui permettent de choisir un emplacement contre une paroi.

```
defregle   placer-radiateur-contre-paroi-int

auteur georges

explication
   pour placer le radiateur dans paroi exterieure
fin-explication

soit   P un positions-radiateurs
   et L un local parmi $(tous-objets 'local)

si
  $(and (let ((flag t)
        )
         (mapcar (lambda (lis)
                  (if (eq (synd 'nom L) (car lis))
                    (setq flag ())
                  )
                 )
          liste-de-radiateurs^P)

       flag
      )
        (> (surface-p (paroi (synd 'nom L) 'int)) 0)
  )
```

```
alors

action
  $(let ((lpos (toutes-les-positions-radiateurs (synd 'nom L) 'int))
         (p ())
         (lp ())
        )
        (mapcar (lambda (pos)
                    (setq p (nouvel-objet 'pointeur-radiateur 'pointeur-radiateur))
                  (synd 'pointeur p pos)
                  (setq lp (append1 lp p))
                )
        lpos)
        (if lpos
            (synd 'liste-de-radiateurs P (append1
                                          (synd 'liste-de-radiateurs P)
                                         (list (synd 'nom L) lp)
                                        )
            )
        )
  )
fin-regle

defregle   placer-radiateur-contre-paroi-ext

auteur georges

explication
    pour placer le radiateur dans paroi exterieure
fin-explication

soit    P un positions-radiateurs
    et L un local parmi $(tous-objets 'local)

si
   $(and (let ((flag t)
               )
               (mapcar (lambda (lis)
                         (if (eq (synd 'nom L) (car lis))
                           (setq flag ())
                         )
                       )
                liste-de-radiateurs^P)

          flag
         )
           (> (surface-p (paroi (synd 'nom L) 'ext)) 0)
   )
alors

action
```

```
  $(let ((lpos (toutes-les-positions-radiateurs (synd 'nom L) 'ext))
        (p ())
        (lp ())
       )
       (mapcar (lambda (pos)
                    (setq p (nouvel-objet 'pointeur-radiateur 'pointeur-radiateur))
                (synd 'pointeur p pos)
                (setq lp (append1 lp p))
              )
       lpos)
       (if lpos
           (synd 'liste-de-radiateurs P (append1
                                          (synd 'liste-de-radiateurs P)
                                        (list (synd 'nom L) lp)
                                      )
           )
       )
  )
fin-regle

defregle   placer-radiateur-contre-paroi-nord

auteur georges

explication
   pour placer le radiateur dans paroi nord
fin-explication

soit    P un positions-radiateurs
    et L un local parmi $(tous-objets 'local)

si
  $(and (let ((flag t)
              )
              (mapcar (lambda (lis)
                          (if (eq (synd 'nom L) (car lis))
                              (setq flag ())
                          )
                      )
                liste-de-radiateurs^P)

         flag
        )
          (> (surface-p (paroi (synd 'nom L) 'nord)) 0)
  )
alors

action
  $(let ((lpos (toutes-les-positions-radiateurs (synd 'nom L) 'nord))
        (p ())
        (lp ())
```

```
      )
       (mapcar (lambda (pos)
                    (setq p (nouvel-objet 'pointeur-radiateur 'pointeur-radiateur))
                 (synd 'pointeur p pos)
                 (setq lp (append1 lp p))
               )
       lpos)
       (if lpos
           (synd 'liste-de-radiateurs P (append1
                                          (synd 'liste-de-radiateurs P)
                                        (list (synd 'nom L) lp)
                                      )
       )
      )
  )
fin-regle

defregle    placer-radiateur-contre-paroi-est

auteur georges

explication
   pour placer le radiateur dans paroi est
fin-explication

soit    P un positions-radiateurs
    et L un local parmi $(tous-objets 'local)

si
  $(and (let ((flag t)
          )
           (mapcar (lambda (lis)
                     (if (eq (synd 'nom L) (car lis))
                       (setq flag ())
                     )
                   )
            liste-de-radiateurs^P)
       flag
      )
        (> (surface-p (paroi (synd 'nom L) 'est)) 0)
  )
alors

action
  $(let ((lpos (toutes-les-positions-radiateurs (synd 'nom L) 'est))
       (p ())
       (lp ())
      )
      (mapcar (lambda (pos)
                    (setq p (nouvel-objet 'pointeur-radiateur 'pointeur-radiateur))
                 (synd 'pointeur p pos)
```

```
            (setq lp (append1 lp p))
          )
      lpos)
      (if lpos
          (synd 'liste-de-radiateurs P (append1
                                        (synd 'liste-de-radiateurs P)
                                       (list (synd 'nom L) lp)
                                  )
        )
      )
  )
fin-regle

defregle    placer-radiateur-contre-parol-ouest

auteur georges

explication
   pour placer le radiateur dans paroi ouest
fin-explication

soit    P un positions-radiateurs
    et L un local parmi $(tous-objets 'local)

si
  $(and (let ((flag t)
         )
          (mapcar (lambda (lis)
                    (if (eq (synd 'nom L) (car lis))
                      (setq flag ())
                    )
                  )
           liste-de-radiateurs^P)
       flag
      )
       (> (surface-p (paroi (synd 'nom L) 'ouest)) 0)
  )
alors

action
  $(let ((lpos (toutes-les-positions-radiateurs (synd 'nom L) 'ouest))
      (p ())
      (lp ())
     )
     (mapcar (lambda (pos)
                (setq p (nouvel-objet 'pointeur-radiateur 'pointeur-radiateur))
               (synd 'pointeur p pos)
               (setq lp (append1 lp p))
             )
     lpos)
     (if lpos
```

```
          (synd 'liste-de-radiateurs P (append1
                                        (synd 'liste-de-radiateurs P)
                                       (list (synd 'nom L) lp)
                                )
        )
      )
  )
fin-regle

defregle    placer-radiateur-contre-paroi-sud

auteur georges

explication
   pour placer le radiateur dans paroi sud
fin-explication

soit    P un positions-radiateurs
    et L un local parmi $(tous-objets 'local)

si
  $(and (let ((flag t)
         )
          (mapcar (lambda (lis)
                    (if (eq (synd 'nom L) (car lis))
                      (setq flag ())
                    )
                  )
           liste-de-radiateurs^P)
       flag
      )
       (> (surface-p (paroi (synd 'nom L) 'sud)) 0)
  )
alors

action
  $(let ((lpos (toutes-les-positions-radiateurs (synd 'nom L) 'sud))
       (p ())
       (lp ())
      )
      (mapcar (lambda (pos)
                (setq p (nouvel-objet 'pointeur-radiateur 'pointeur-radiateur))
              (synd 'pointeur p pos)
              (setq lp (append1 lp p))
            )
      lpos)
      (if lpos
          (synd 'liste-de-radiateurs P (append1
                                        (synd 'liste-de-radiateurs P)
                                       (list (synd 'nom L) lp)
                                )
```


```
        )
      )
    )
fin-regle
```

L'ordre dans lequel sont appliquées les règles permet de fixer la priorité de choix des emplacements.

Si la règle de choisir un emplacement de radiateur contre la façade nord du local est placée avant la règle de choisir un emplacement contre la façade sud, alors si le local a une façade au nord, le radiateur sera placé contre cette façade.

On peut contrôler le choix d'emplacement de radiateurs en contrôlant l'ordre des règles. C'est une façon très simple de choix et qui ressemble au raisonnement d'un expert qui peut dire:

-si le local a une façade au nord alors mettre le radiateur contre cette façade

D'autre part, ce contrôle du choix est réalisé par la modification du paquet de règles d'une tâche SMECI; Ce choix permet en plus de réduire l'explosion combinatoire pour la génération de toutes les solutions possibles. En effet ce choix joue le rôle d'heuristiques qui orientent le raisonnement dès le départ sans l'engager dans une recherche exhaustive des solutions.

Bien sûr, contre une seule façade il y a aussi un grand nombre d'emplacement possible de radiateur; dans notre application on a choisi de le placer contre le milieu de la paroi; ce choix est aussi arbitraire et peut être contôlé par la modification de la méthode **trouver-tous-les-radiateurs**. Cette méthode est implémentée pour trouver le milieu de la paroi, cependant on peut la modifier pour qu'elle cherche un coin ou une porte pour se mettre à côté.

```
defregle      construire-installations

auteur georges

explication
   permet de trouver toutes les combinaisons de
   positions possibles de radiateurs
fin-explication

soit    I une installation
     et P un positions-radiateurs
      et PO un pointeur-radiateur  parmi $(car (cdr (car (synd 'liste-de-radiateurs P)
                                                  )))

si
   $(synd 'liste-de-radiateurs P)

alors
```

```
affecter liste-de-radiateurs^I avec $(append1 liste-de-radiateurs^I
                                    (list (caar (synd 'liste-de-radiateurs P)) pointeur^PO))
affecter liste-de-radiateurs^P avec $(cdr liste-de-radiateurs^P)

fin-regle
```

### 6.5.2 Le choix du circuit hydraulique

Ayant choisi l'emplacement de nos radiateurs, on procède à la création de la tuyauterie qui va relier ces radiateurs. La tuyauterie va relier les radiateurs entre eux en passant contre les parois d'un local.

on a adopté dans notre application l'hypothèse suivante:

> -deux radiateurs ne peuvent être connectés que s'ils se trouvent dans deux locaux adjacents, et s'il y a la possibilité de faire passer un tuyau sans bloquer le passage d'une porte.

Pour cela on commence par construire un graphe d'adjacence pour les locaux du projet.

(mettre un dessin ici)

Un graphe permet de déduire les liaisons possibles entre les différents locaux.

```
defregle      construire-graphe

auteur georges

explication
    construit le graphe d'adjacence
fin-explication

soit G un graphe
    I une installation
     L1 un local parmi $(tous-objets 'local)
     L2 un local parmi $(tous-objets 'local)

si
   $(and (neq L1 L2)
         (adjacent? nom^L1 nom^L2)
   )
alors

action

  $(let ((I (nouvel-objet 'liaison 'liaison))
         (tr1 (nouvel-objet 'troncon 'troncon))
         (tr2 (nouvel-objet 'troncon 'troncon))
         (tr3 (nouvel-objet 'troncon 'troncon))
         (tr4 (nouvel-objet 'troncon 'troncon))
```

```
      (lt)
      (lrep)
     )
     (synd 'element1 l L1)
     (synd 'element2 l L2)
     (synd 'liaisons G (append1 liaisons^G l))
      (setq lt (synd 'trouver-troncons l l))
      (synd 'tube tr1 (car lt))
      (synd 'tube tr2 (cadr lt))
      (synd 'tube tr3 (caddr lt))
      (synd 'tube tr4 (cadddr lt))
       (synd 'troncon l (list tr1 tr2 tr3 tr4))
 )
```

***fin-regle***

En construisant le graphe d'adjacence on a essayé de trouver les tronçons qui peuvent relier deux radiateurs ensemble.Pour cela on a noté la remarque suivante:

il y a au maximum quatre façons pour relier deux radiateurs dans deux locaux adjacents:

De ces solutions, on trie celles qui ne passent pas devant une porte.

***defregle elimlner-troncons-non-valables***

*auteur georges*

*explication*
*retire les troncons non valables*
*fin-explication*

*soit LI une liaison*
*TR un troncon parmi $(synd 'troncon LI)*

*si*
*$(synd 'passe-devant-une-porte? TR)*
*alors*

*action*

*$(synd 'troncon LI (remove TR (synd 'troncon LI)))*

***fin-regle***

Cependant le circuit ne doit pas nécessairement passer par un radiateur plusieurs fois; il suffit d'alimenter le radiateur une seule fois. Pour cela, à partir du graphe d'adjacence, on essaie de déterminer plusieurs solutions de circuits linéaires (dans lequel on n'a pas de boucles).

***defregle trouver-clrcult-solutlon***

```
auteur georges

explication
    cherche les liaisons d'un circuit a parcourir
fin-explication

soit   C un circuit
   et LI une liaison
   et N un noeud parmi $(let ((l liste-de-noeuds^C)
                              )
                         l)

si
  $(and (eq element1^LI local^N)
        (not (eq (length liste-de-noeuds^C)
                 (length (tous-objets 'local))
             )
        )
        (let ((flag t)
              )
              (mapcar (lambda (loc)
                      (if (eq loc element2^LI)
                          (setq flag ())
                      ))
                liste-des-voisins^N)
        flag
        )
        (let ((flag t)
              )
              (mapcar (lambda (n)
                      (if (eq local^n element2^LI)
                          (setq flag ())
                      ))
                liste-de-noeuds^C)
        flag
        )
  )

alors

creer nn un noeud

puis

affecter liste-de-noeuds^C avec $(append1 liste-de-noeuds^C nn)
affecter local^nn avec $(synd 'element2 LI)
affecter liste-des-voisins^nn avec $(append1 liste-des-voisins^nn element1^LI)
affecter liste-des-voisins^N avec $(append1 liste-des-voisins^N element2^LI)

fin-regle
```

A partir des circuits linéaires ainsi adoptés, on pourra déduire les liaisons possibles entre les radiateurs. Pour cela on teste pour chaque liaison entre deux locaux toutes les possibilités de passage de tuyaux pour relier les radiateurs en évitant bien sûr les solutions où le tuyau passe devant une porte.

Ayant pour chaque type de liaisons toutes les possibilités on essaie de combiner pour en déduire toutes les solutions possibles.

```
defregle      trouver-toutes-les-Installations

auteur georges

explication
  permet de trouver toutes les combinaisons de troncons possibles
fin-explication

soit   I une installation
    et CL un circuit-lineaire
    et TR un troncon parmi $(car (synd 'liste-de-troncons CL))

si
  $(synd 'liste-de-troncons CL)

alors

affecter liste-de-troncons^I avec $(append1 liste-de-troncons^I TR)
affecter liste-de-troncons^CL avec $(cdr liste-de-troncons^CL)


fin-regle
```

### 6.5.3 Exemple

Dans l'exemple qui suit on essaie de montrer comment le système a conçu le système de chauffage pour un plan qui contient quatre locaux et deux portes (bien sûr l'exemple n'est que démonstratif).

On peut voir dans les différentes illustrations qui suivent différentes variantes ou solutions proposées par le système.

Dans la fenêtre en haut à gauche on voit le graphe d'adjacence du projet.

Dans la fenêtre en haut à gauche on voit les différents types de circuits linéaires que l'on peut déduire à partir du graphe d'adjacence.

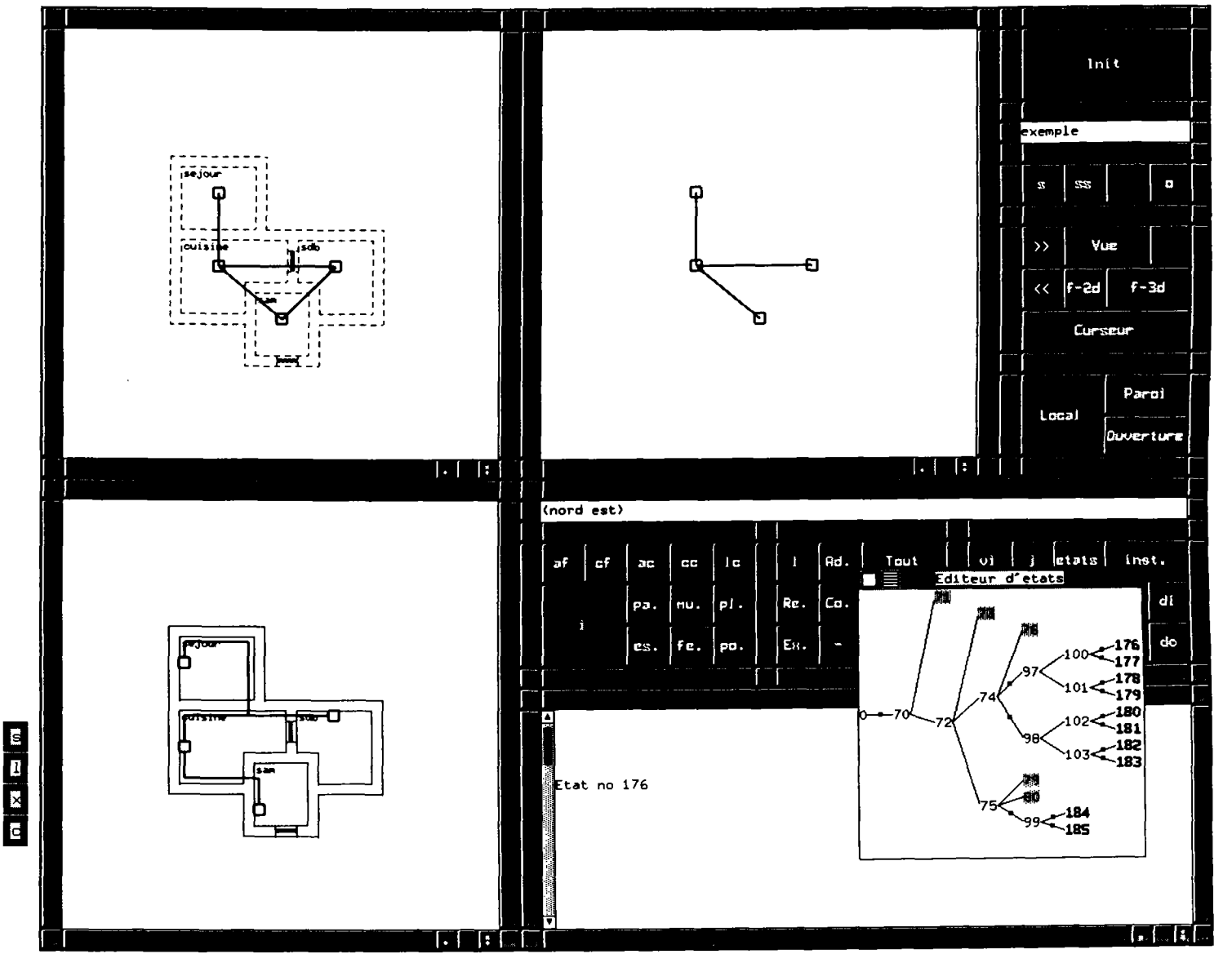
Dans la fenêtre en bas à gauche on voit les différents variantes d'installations proposées par le système: le carré représente un radiateur (un radiateur par local) et

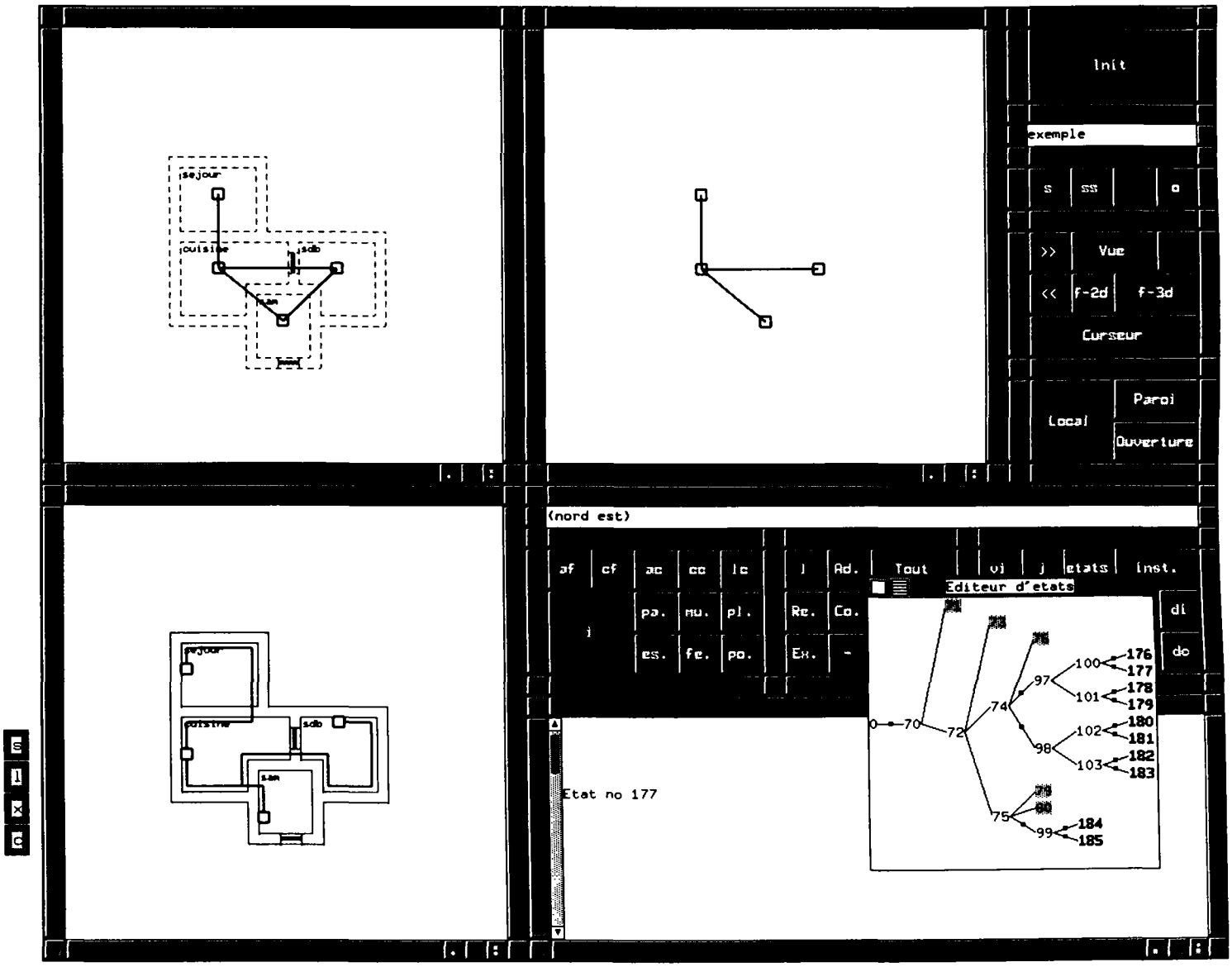
les lignes en gras représentent les tuyauteries ( elles ne passent pas devant une porte).

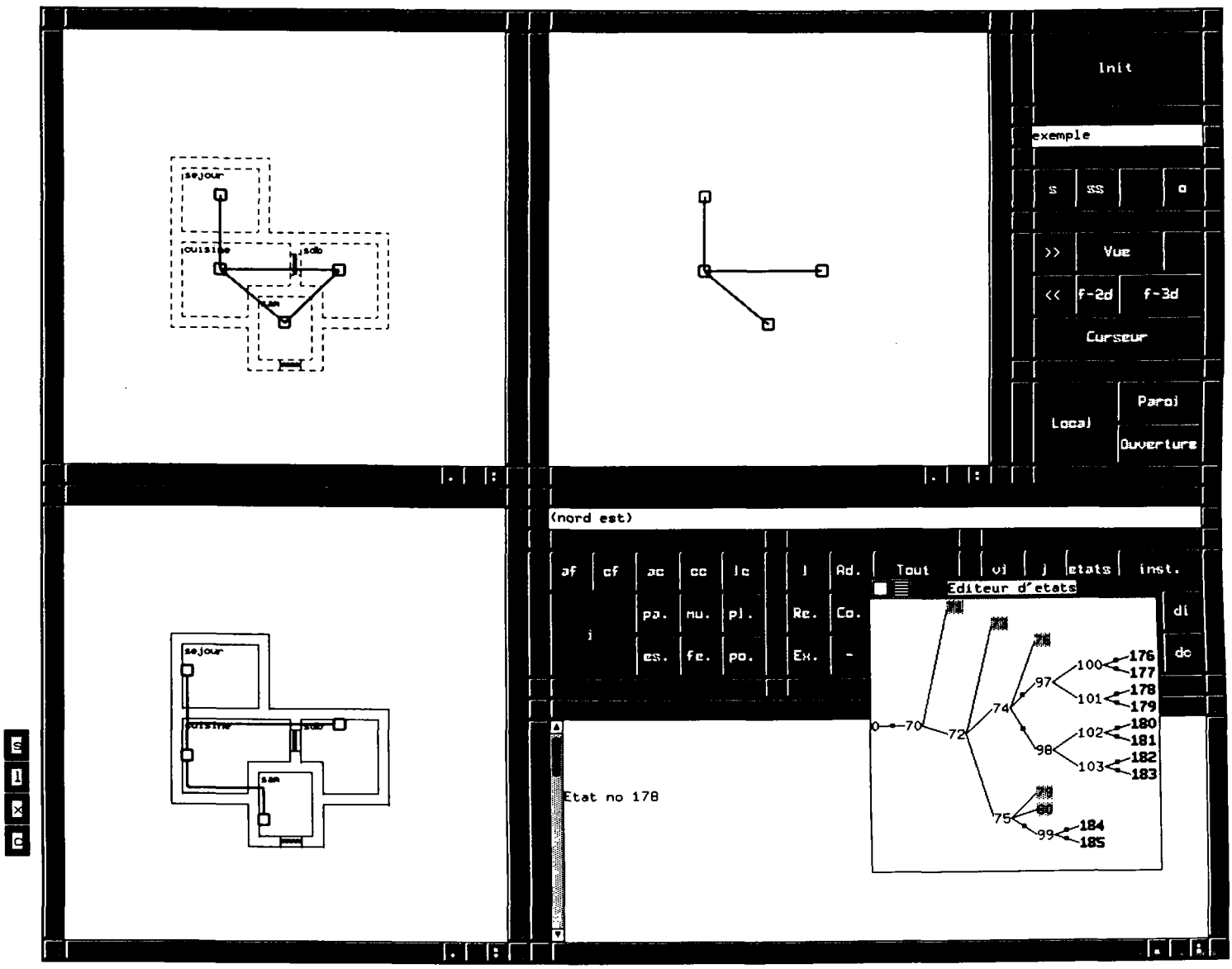
L'arbre des états affiché à droite représent le parcours du système pour aboutir aux solutions (les feuilles affichées en gras).

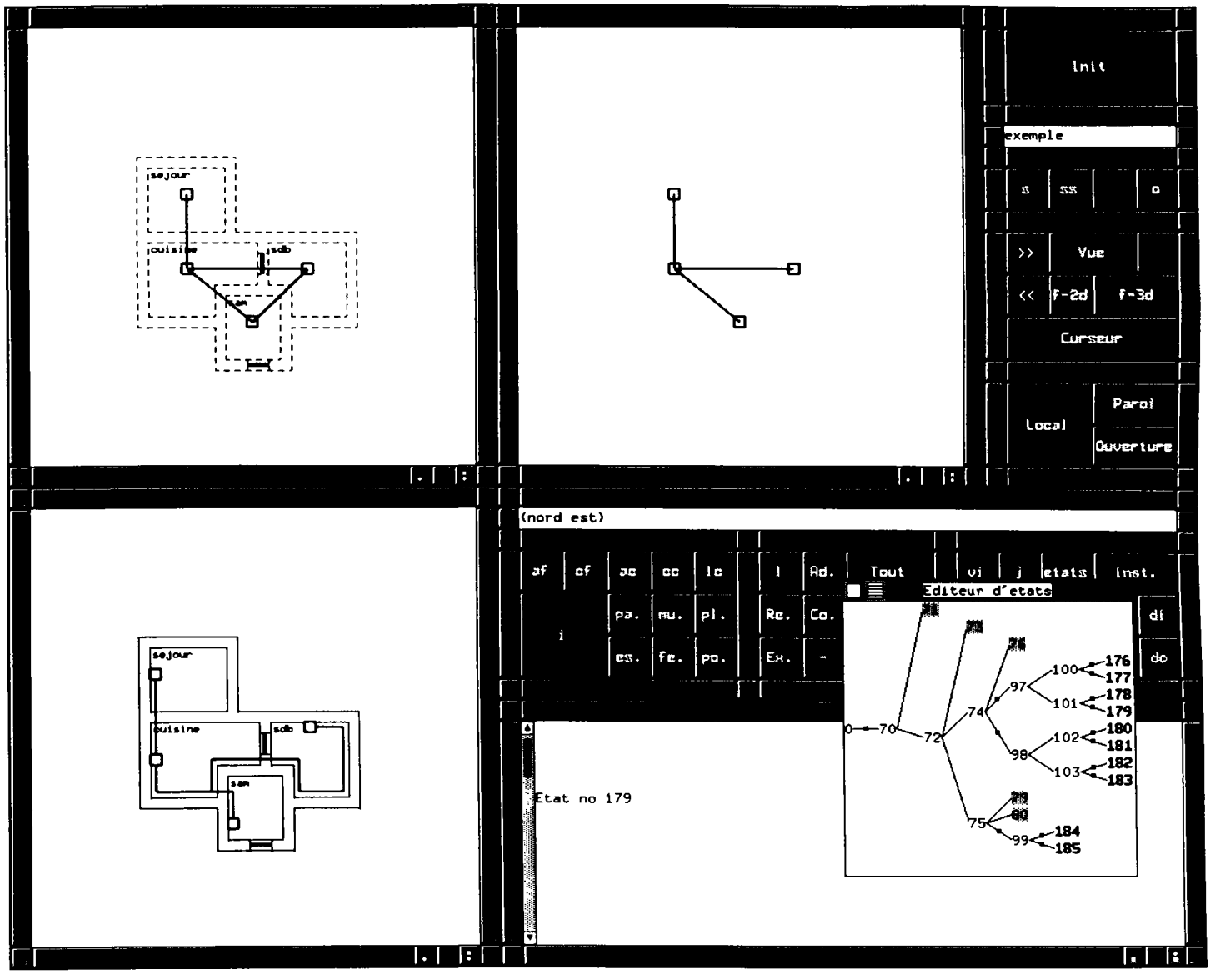
Ces dix solutions proposées sont calculées par rapport à un positionnement de radiateur. En effet on a demandé au système de placer les radiateurs contre les parois NORD d'un local si elles existent, sinon contre les parois EST.

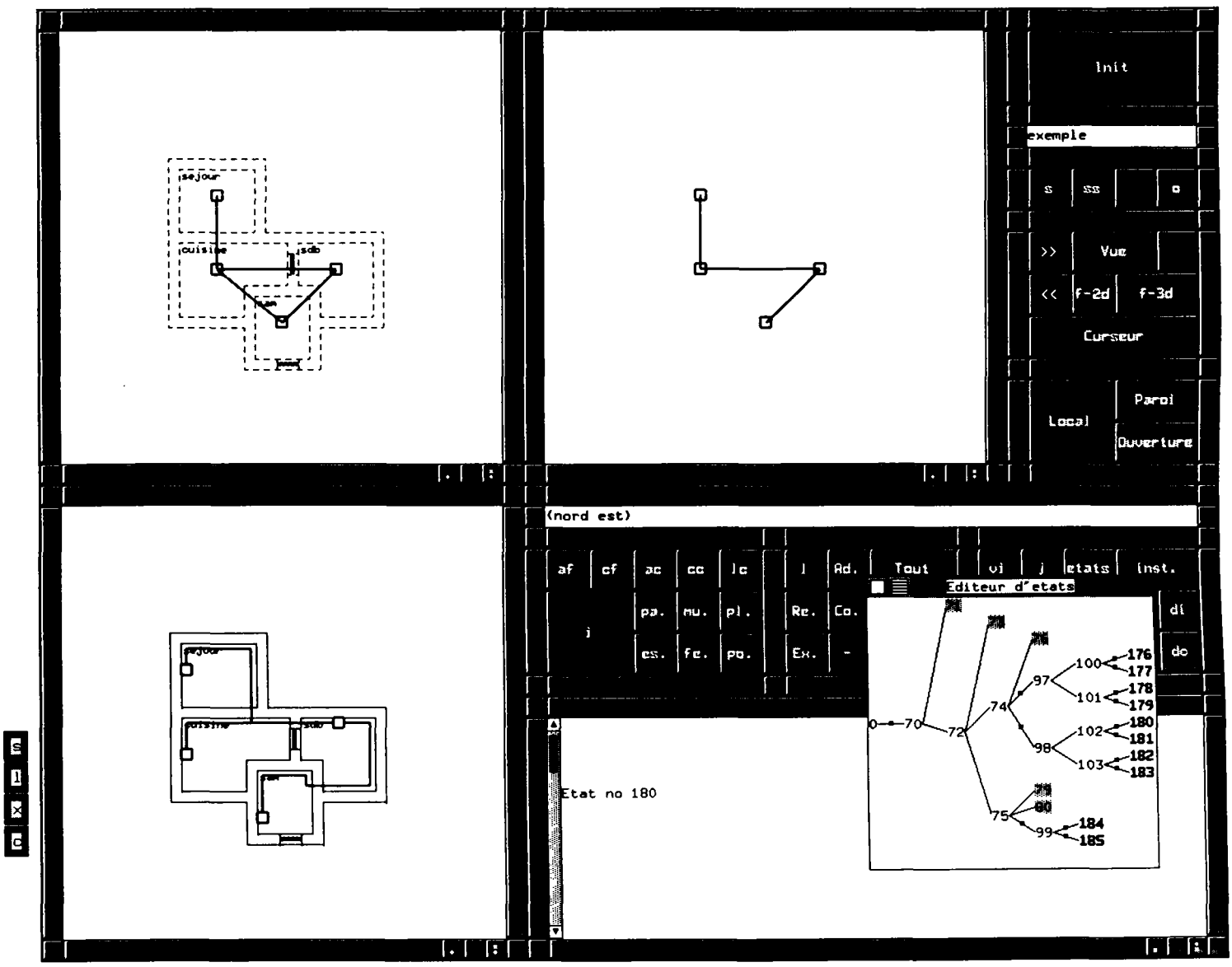
Dans d'autres configurations le système peut proposer plusieurs emplacements possibles de radiateurs, et pour chaque emplacement il essaie de proposer des circuits hydrauliques.

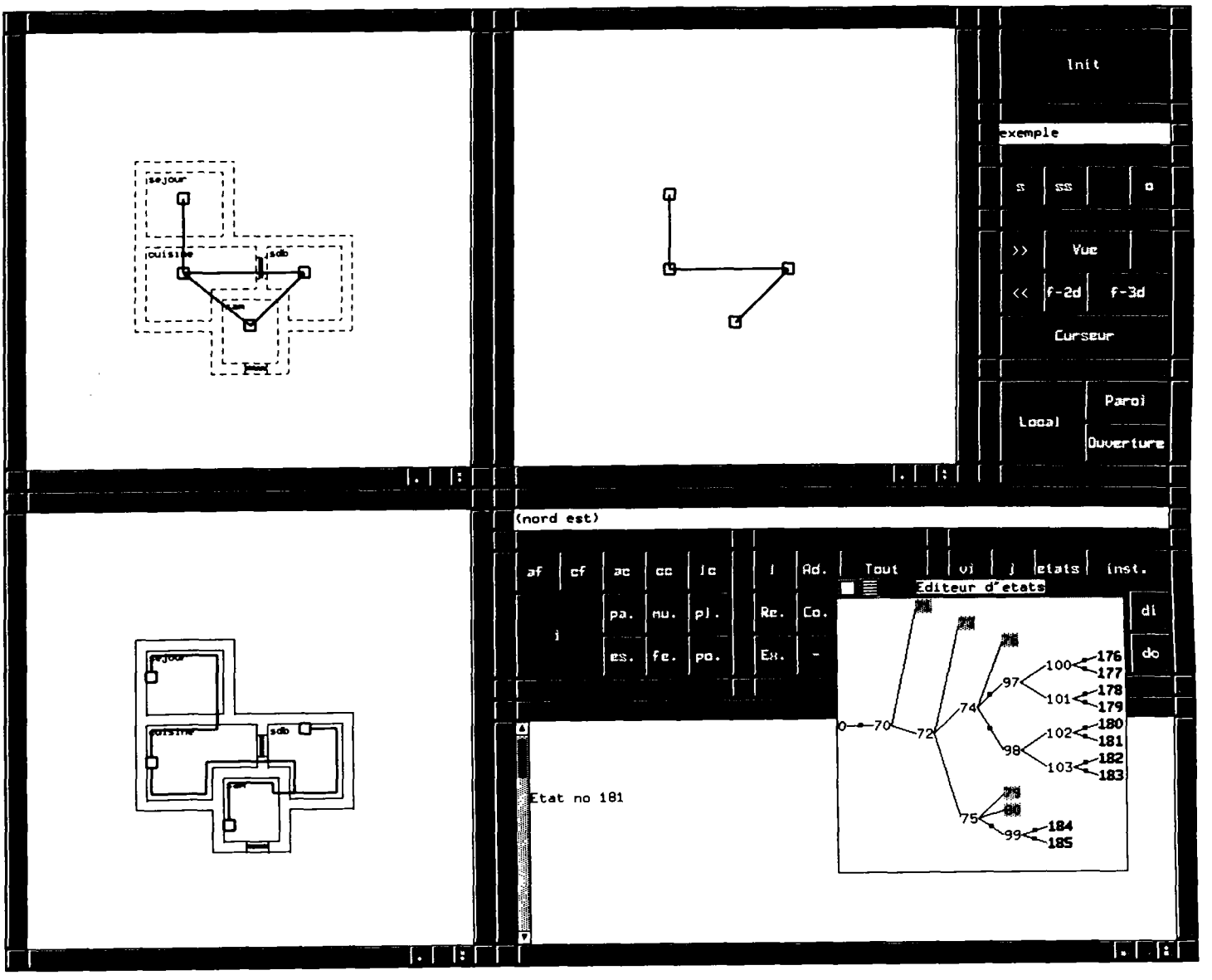

Init
exemple
s
ss
a
>>
Vue
<<
f-2d
f-3d
Curseur
Local
Paroi
Ouverture
sejour
cuisine
sdb
wc
(nord est)
af
cf
ac
cc
lc
l
Ad.
Tout
vi
j
etats
inst.
pa.
nu.
pl.
Re.
Co.
di
es.
fe.
po.
Ex.
do
Editeur d'etats
Etat no 176
0
70
72
74
75
97
98
99
100
101
102
103
176
177
178
179
180
181
182
183
184
185

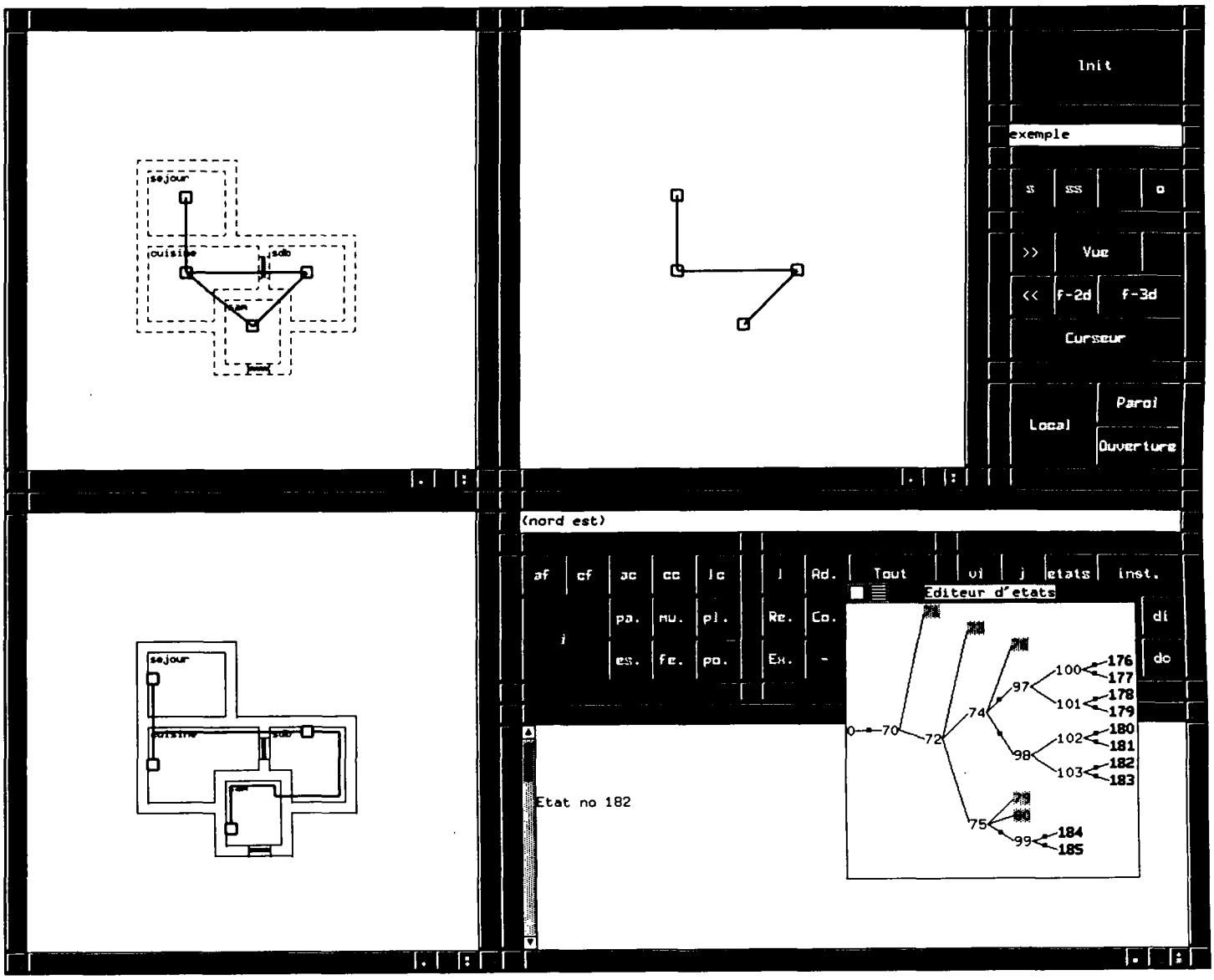

Init
exemple
s
ss
a
>>
Vue
<<
f-2d
f-3d
Curseur
Local
Paroi
Ouverture
(nord est)
af
cf
ac
cc
lc
l
Ad.
Tout
vi
j
etats
inst.
pa.
mu.
pl.
Re.
Co.
es.
fe.
po.
Ex.
Editeur d'etats
di
do
sejour
cuisine
sdb
Etat no 177
0
70
72
74
97
98
75
99
100
101
102
103
176
177
178
179
180
181
182
183
184
185

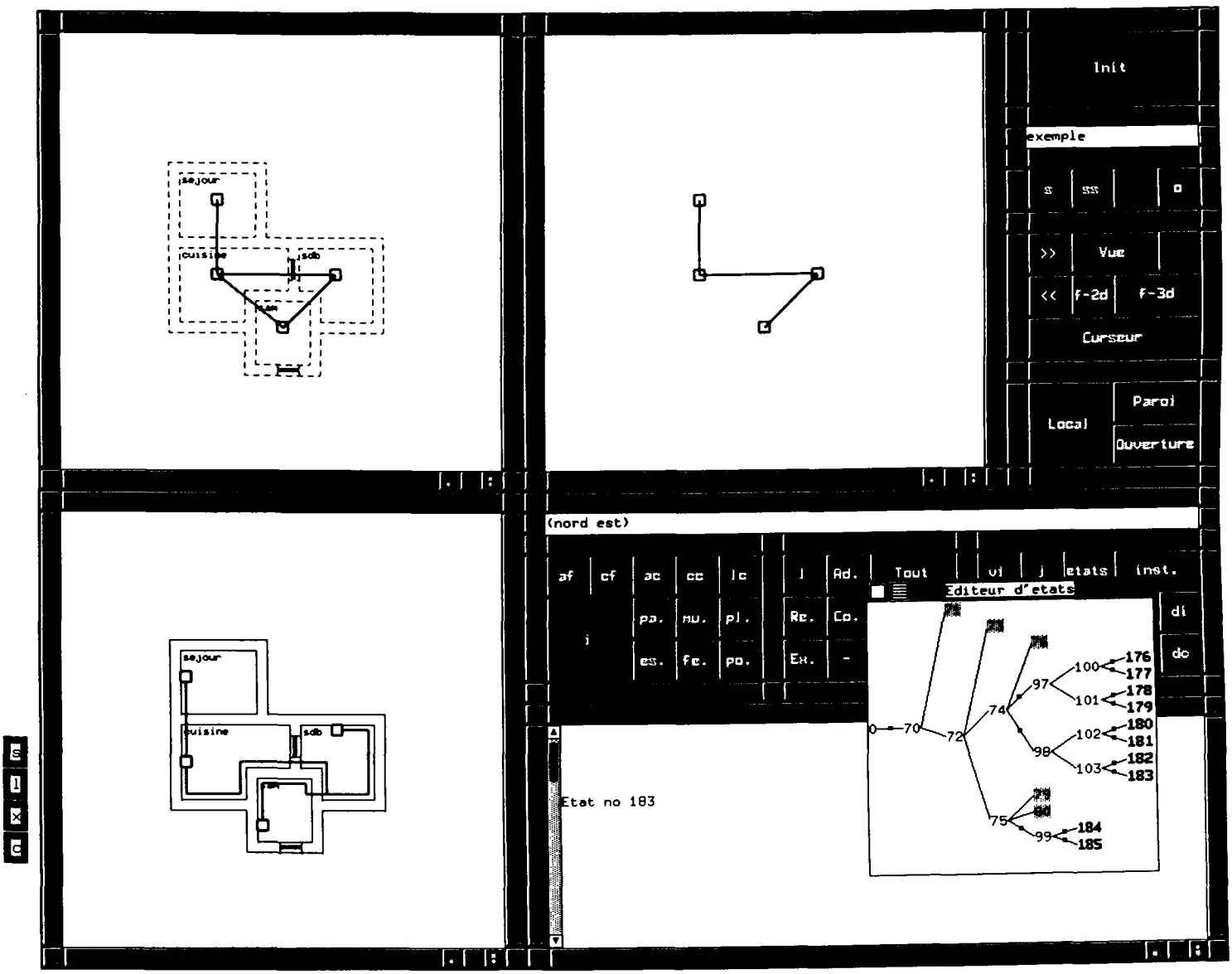

Init
exemple
s
ss
o
>>
Vue
<<
f-2d
f-3d
Curseur
Local
Paroi
Ouverture
(nord est)
af
cf
ac
cc
lc
l
Ad.
Tout
vi
j
etats
inst.
pa.
mu.
pl.
Re.
Co.
es.
fe.
po.
Ex.
di
dc
Editeur d'etats
sejour
cuisine
sdb
sam
Etat no 178
0
70
72
74
75
97
98
99
100
101
102
103
176
177
178
179
180
181
182
183
184
185

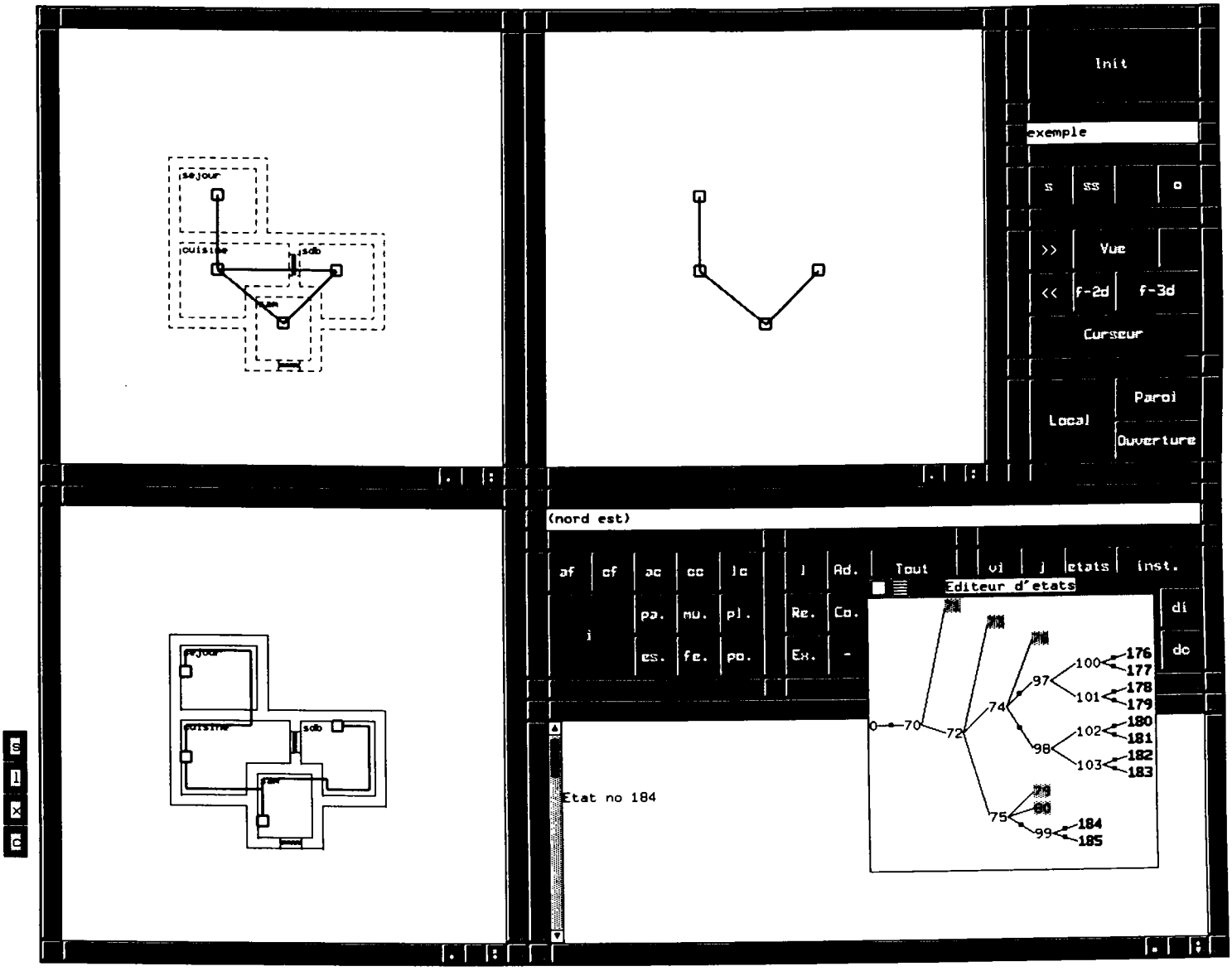

Init
exemple
s
ss
o
>>
Vue
<<
f-2d
f-3d
Curseur
Local
Paroi
Ouverture
(nord est)
af
cf
ac
cc
lc
l
Ad.
Tout
vi
j
etats
inst.
Editeur d'etats
pa.
mu.
pl.
Re.
Co.
di
es.
fe.
po.
Ex.
-
do
sejour
cuisine
sdb
sam
Etat no 179
0
70
72
74
97
98
75
99
100
101
102
103
176
177
178
179
180
181
182
183
184
185

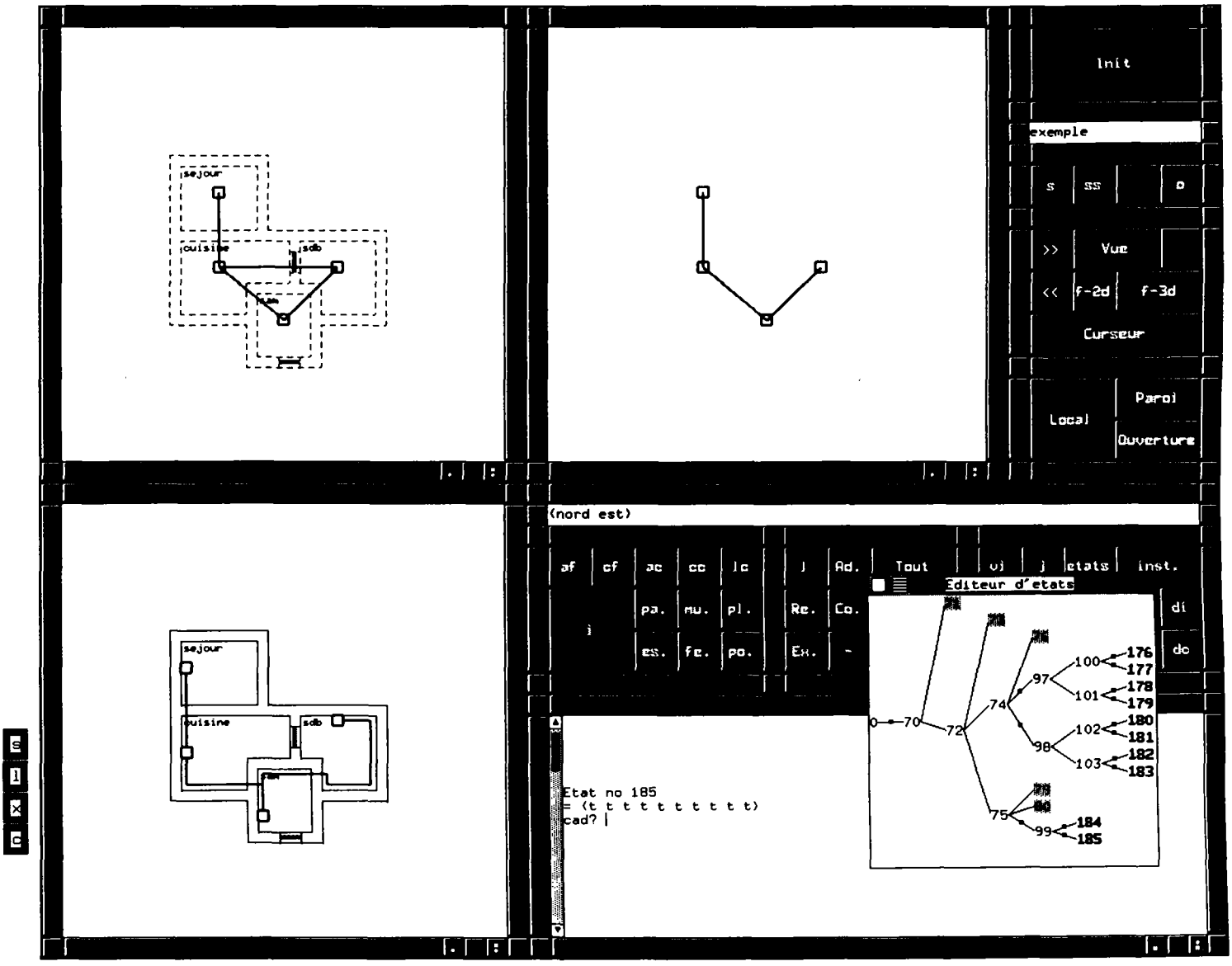

Init
exemple
s
ss
o
>>
Vue
<<
f-2d
f-3d
Curseur
Local
Paroi
Ouverture
(nord est)
af
cf
ac
cc
lc
l
Ad.
Tout
vi
j
etats
inst.
pa.
mu.
pl.
Re.
Co.
es.
fe.
po.
Ex.
di
do
Editeur d'etats
0
70
72
74
97
98
75
99
100
101
102
103
176
177
178
179
180
181
182
183
184
185
Etat no 180
sejour
cuisine
sdb
wc
S
I
X
C

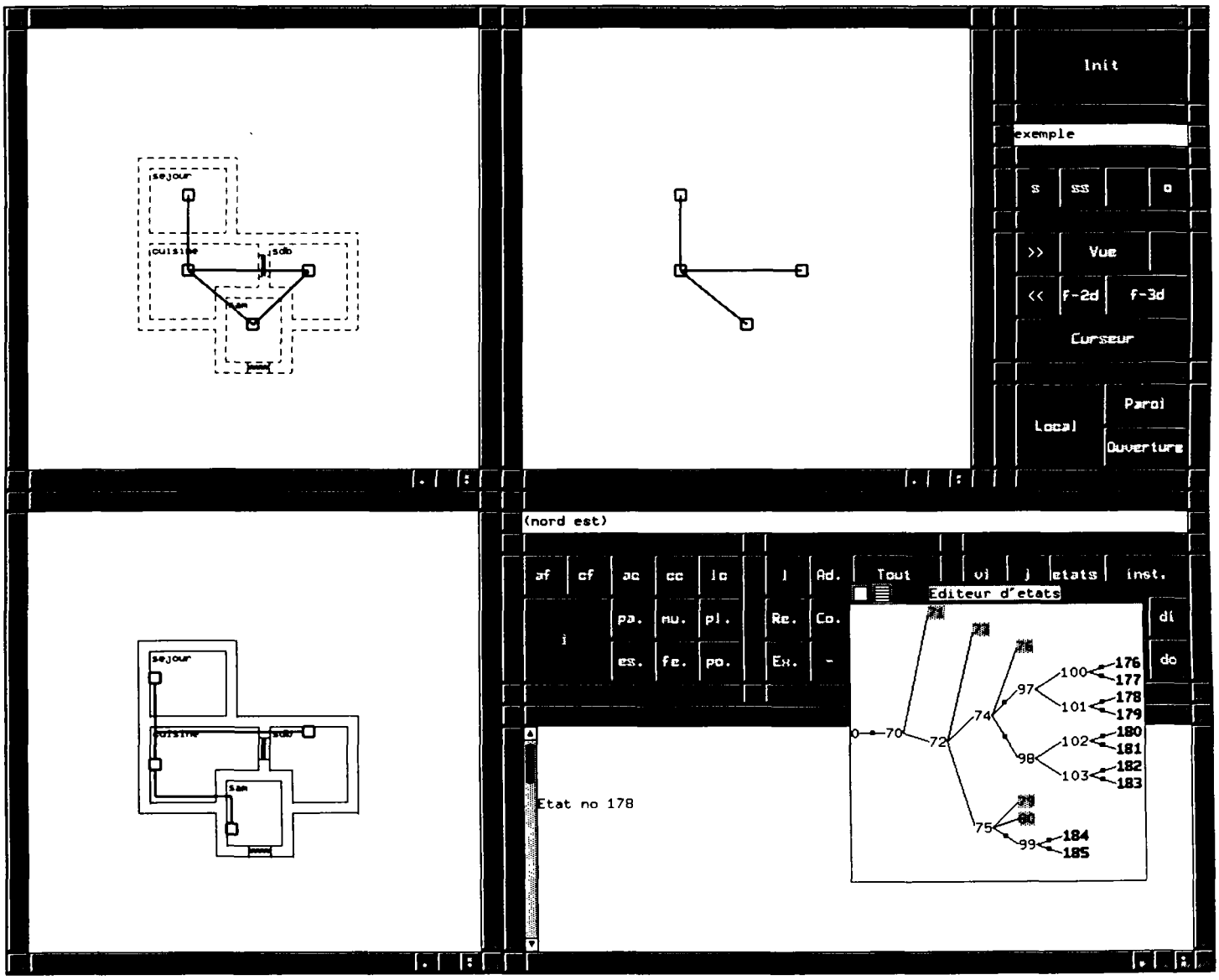

Init
exemple
s
sss
o
>>
Vue
<<
f-2d
f-3d
Curseur
Local
Paroi
Ouverture
(nord est)
af
cf
ac
cc
ic
i
Ad.
Tout
vi
j
etats
inst.
pa.
nu.
pl.
Re.
Co.
es.
fe.
po.
Ex.
di
do
Editeur d'etats
Etat no 181
sejour
cuisine
sdb
wc
0
70
72
74
75
97
98
99
100
101
102
103
176
177
178
179
180
181
182
183
184
185


Init
exemple
s
sss
o
>>
Vue
<<
f-2d
f-3d
Curseur
Local
Paroi
Ouverture
sejour
cuisine
sdb
(nord est)
af
cf
ac
cc
lc
l
Ad.
Tout
vi
j
etats
inst.
pa.
mu.
pl.
Re.
Co.
es.
fe.
po.
Ex.
di
do
Editeur d'etats
Etat no 182
0
70
72
74
75
97
98
99
100
101
102
103
176
177
178
179
180
181
182
183
184
185


Init
exemple
s
ss
o
>>
Vue
<<
f-2d
f-3d
Curseur
Local
Paroi
Ouverture
sejour
cuisine
sdb
(nord est)
af
cf
ac
cc
lc
l
Ad.
Tout
vi
j
etats
inst.
Editeur d'etats
pa.
mu.
pl.
Re.
Co.
es.
fe.
po.
Ex.
di
do
Etat no 183
0
70
72
74
75
97
98
99
100
101
102
103
176
177
178
179
180
181
182
183
184
185


Init
exemple
s
ss
o
>>
Vue
<<
f-2d
f-3d
Curseur
Local
Paroi
Ouverture
(nord est)
af
cf
ac
cc
lc
l
Ad.
Tout
vl
j
etats
inst.
pa.
mu.
pl.
Re.
Co.
es.
fe.
po.
Ex.
di
dc
Editeur d'etats
Etat no 184
sejour
cuisine
sdb
0
70
72
74
75
97
98
99
100
101
102
103
176
177
178
179
180
181
182
183
184
185


sejour
cuisine
sdb
Init
exemple
s
ss
p
>>
Vue
<<
f-2d
f-3d
Curseur
Local
Paroi
Ouverture
(nord est)
af
cf
ac
cc
lc
j
Ad.
Tout
vj
j
etats
inst.
pa.
mu.
pl.
Re.
Co.
es.
fe.
po.
Ex.
di
do
Editeur d'etats
0
70
72
74
75
97
98
99
100
101
102
103
176
177
178
179
180
181
182
183
184
185
Etat no 185
= (t t t t t t t t t)
cad?
S
l
x
c


Init
exemple
s
sss
o
>>
Vue
<<
f-2d
f-3d
Curseur
Local
Paroi
Ouverture
sejour
cuisine
sdb
san
(nord est)
af
cf
ac
cc
lc
l
Ad.
Tout
vi
j
etats
inst.
Editeur d'etats
pa.
mu.
pl.
Re.
Co.
di
es.
fe.
po.
Ex.
-
do
Etat no 178
0
70
72
74
75
97
98
99
100
101
102
103
176
177
178
179
180
181
182
183
184
185

# Chapitre 7

# Conclusion

## 7.1 Travail effectué

### 7.1.1 Modeleurs intelligents et données géométriques

Pour élaborer un outil informatique d'aide à la conception il fallait résoudre des problèmes liés à la représentation des données, de la connaissance et du raisonnement.

Une représentation appropriée devrait être utilisée pour la modélisation du projet.

On a développé une approche expérimentale basée sur un système de CAO intégré qui permet le stockage et la restauration d'informations; ce système a une base de données géométrique simple et pauvre en sémantique mais il est équipé d'un ensemble de fonctions d'analyse géométrique et spatiale.

Les analyseurs sont implémentés en une couche superposée au noyau géométrique. Ces fonctions sont capables de reconnaitre et d'interpréter différentes notions de haut niveau tel que murs, cloisons, communications,...

Le principe de ces fonctions est de pouvoir décrire comment une information de haut niveau peut être décrite par des informations de bas niveau (i.e. vide, plein ou transparent) et essayer par la suite d'identifier ces descriptions en scrutant la base de données géométrique du modèle.

Les fonctions d'analyses constituent un langage de requêtes géométriques simples qui permet de tirer des informations à partir du modèle géométrique du projet à la manière d'une base de données; en comparant à une base de données classique, notre modèle stocke les informations d'ordre géométrique simple, et permet de calculer les informations complexes non stockées par l'intermédiaire des fonctions d'analyses.

Les algorithmes utilisés sont une version simplifiée pour démontrer la faisabilité de fonctions assez "intelligentes" pour identifier des notions riches en sémantiques du monde architectural à partir de données géométriques simples.

Des méthodes plus complexes et plus efficaces peuvent être utilisées pour améliorer la puissance de résolution de telles fonctions. Le tracé de rayons est un exemple de méthodes mathématiques et géométriques qui peuvent scanner efficacement les données géométriques simples à la recherche d'informations de haut niveau utiles pour le raisonnement.

Dans son état actuel, à travers le prototype logiciel devéloppé, notre approche se caractérise par:

**-facilité de saisie:**
Pour pouvoir modéliser un projet de bâtiment, il suffit d'utiliser le modeleur géométrique et de saisir la volumétrie de ce projet par une approche très générale: une saisie d'éléments géométriques de la volumétrie globale, permet de déduire des informations sur des entités plus complexes qui sont de l'ordre du détail.

On a mis l'accent dans notre approche sur le rôle important que joue la saisie graphique et son apport en conception.

On croit que cette approche simplifie l'utilisation de systèmes de CAO, car la description du projet revient à décrire la volumétrie et les entités spatiales. Le concepteur n'a pas à s'occuper à saisir des entités sémantiques tel que murs, cloisons, ou des notions plus complexes tel que coins, intersections, ou même des notions spécialisées tel que linteau ou ponts thermiques,...

**-interactivité:**
Par l'utilisation d'une interface graphique, l'utilisation du système est très simplifiée, et permet une interactivité maximale surtout au niveau du travail visuel sur l'écran, tel que saisie géométrique, ou appel de fonction.

**-facilité de maintien de cohérence:**
Le modèle géométrique simple qui joue le rôle de base de données peut être facilement modifié et mis à jour; on rappelle qu'au moment des modifications on intervient surtout graphiquement et on modifie dans la volumétrie générale du projet sans préciser de modifications sur les détails; ces détails sont en fait inexistants dans la base de données elle-même mais sont accessibles par l'intermédiaire du langage de requête que représentent les fonctions d'analyses.

La mise à jour des informations concernant les détails devient alors une tâche simple, et facile à programmer, surtout que les fonctions d'analyses sont appelées dynamiquement, et reflète l'état actuel du modèle géométrique qui lui est mis à jour graphiquement.

**-facilité d'extension:**
On était concerné dans notre étude par une démonstration sur les espaces intérieurs d'un projet. On a donc développé une couche de fonctions au dessus d'un modeleur intégré qui sont capables d'identifier les espaces intérieurs et les enveloppes d'un local, les cloisons de séparation, les parois extérieures, les ouvertures, les communications, et quelques attributs géométriques (surface, hauteur, volume,..).

Le principe de fonctions d'analyse peut être généralisé sur d'autres domaines concernant d'autres acteurs que l'architecte comme l'ingénieur de structure ou le thermicien ou l'acousticien. Ils auront chacun leurs propres fonctions qui peuvent extraire du modèle géométrique les informations qui les intéressent.

Des requêtes complexes peuvent être construites à partir de requêtes simples.

**-facilité de représentations multiples:**
Une requête peut être définies de plusieurs façons pour prendre en compte les différences de points de vues, ou de l'état d'évolution d'un projet; par exemple la fonction pour reconnaitre une paroi peut être formulée *différemment pour un architecte que pour un thermicien (entre nu de* mur, ou entre axes,...).

**-modularité:**
On peut grouper les fonctions d'analyses dans des paquets indépendants, appartenant à un ou plusieurs acteurs. Cette caractéristique donne à notre approche un moyen efficace pour prendre en compte le caractère multi-acteur de l'approche de la conception, où le modèle sera facilement extensible et l'encapsulation des modules pour permettre de réglementer l'utilisation de certaines informations et les restreindre aux acteurs qui auront le droit de les utiliser.

On n'oublie pas aussi de mentionner l'apport de la programmation orientée objet à notre modèle qui a facilité la tâche de programation tout en tirant profit des qualités des objets en termes d'encapsulation, de modularité, de facilité de modélisation et d'extension...

L'intégration de données géométriques de cette façon est simple et efficace. On n'est amené ni à enrichir la base de données du modeleur par des classes d'objets riches en sémantique, ni à construire un autre base de données contenant ces classes pour qu'elle soit couplée plus tard avec l'outil de CAO.

### 7.1.2 Modèles de connaissance

Le deuxième volet de notre thèse consiste à traiter les possibilités de couplage de notre outil de CAO avec un environnement d'intelligence artificielle qui est capable de modéliser la connaissance nécessaire pour une aide à la conception.

Pour montrer les possibilités de notre approche, on a devéloppé quelques applications où l'on a montré l'utilisation possible:

On peut exploiter ce modèle comme un noyau pour beaucoup de types d'applications; il peut ainsi fournir les informations nécessaires à différents modules d'évaluateurs ou de modules de systèmes experts.

La modélisation de la connaissance a été mise en oeuvre par la construction de modules de systèmes experts; Ces systèmes experts sont écrits en utilisant un langage de haut niveau qui manipule des entités riches en sémantique et qui se rapproche du langage naturel.

Dans nos différents exemples, on a utilisé le langage de requêtes ou les fonctions d'analyse pour l'écriture de règles qui ont servi à modéliser les connaissances dans un domaine particulier.

La simplification de l'écriture de ces règles est rendu possible par l'utilisation des fonctions de requêtes dont l'appel a été défini afin de ressembler à du langage naturel; on a alors manipulé des entitées tel que cloison ou communication, et non pas des entités tel que cube, x, y, z, h ....

Bien sûr les informations de bas niveau sont codées dans les fonctions d'analyse, mais une fois écrites ces fonctions peuvent être utilisées dans la construction de requêtes complexes ainsi que pour l'écriture des règles d'un module de système expert.

Le langage de requête que fournit les fonctions d'analyse est un moyen très simple pour écrire des règles dont la synthaxe se rapproche du langage naturel, ce qui les rend plus accessibles à un expert humain, et surtout plus simple pour le maintien et la correction; il ne faut cependant pas croire que la réalisation de modules experts est simplifié au point qu'il suffit d'écrire quelques règles, car un moteur d'inférence reste assez compliqué à manier et une certaine expérience est nécessaire pour pouvoir le mettre en oeuvre, surtout en présence de moteurs performants qui sont capables de réaliser différents styles de raisonnement assez compliqués.

## 7.2 Points délicats du système

**-la rapidité de réponse du système:**
Le plus grand inconvénient d'un système tel que décrit avant, reste le temps de calcul des fonctions (calcul d'intersection, d'adjacence, dinclusion,...). On croit cependant que ce facteur peut être amélioré en optimisant les algorithmes et les codes sources, et en utilisant des machines performantes.

**-Le choix des fonctions:**
Les fonctions à implémenter sont choisies en fonction des applications à définir. Ce choix est un peu délicat à faire surtout pour la détermination du retour de la fonction ainsi que son algorithme de recherche.

Il faut pouvoir identifier les fonctions nécessaires, tout en gardant en l'esprit les possibilités offertes par le système pour l'utilisation des fonctions de base dans la construction de fonctions plus complexes.

**-L'implémentation des fonctions:**
Bien que les principes de notre approche sont indépendants de l'outil de CAO et de sa représentation interne, la couche de fonctions est très dépendante de cette représentation.

Par conséquent, il faut réécrire cette couche si on change de représentation. Cependant la couche des règles restera elle valable car elle ne dépend que du nom d'appel des fonctions et non pas de l'algorithme utilisé.

## 7.3 Utilisations et extensions possibles

Le but de notre thèse n'est pas de résoudre le problème de conception, ni le problème de coordination nécessaire pour un travail de conception multi-acteur, mais plutôt de donner les principes d'environnement qui peut supporter facilement la modélisation de ces problèmes.

La conception reste un problème difficile à aborder et à automatiser surtout dans la subjectivité de cet acte en fonction du concepteur. Nous croyons néanmoins, que notre système peut apporter certains outils surtout pour le stockage et l'échange de données ainsi que pour le raisonnement.

Comme le montre les exemples,notre approche nous a permis de mettre en oeuvre des applications de conception, telle que celle de la conception d'un réseau hydraulique de chauffage. On peut alors parler d'environnement de conception assistée par ordinateur.

Il ne faut pas cependant croire que passer à des applications pour une conception totalement automatisée est une tâche facile; mais avec ce que nous offrons dans notre environnement on a au moins les prémisses de base de tels systèmes.

Pour étendre le modèle il faut l'orienter plus vers des architectures muli-acteurs; le principe de notre approche, avec une base géométrique simple commune à tous les acteurs et des modules de fonctions d'analyse spécialisées pour chaque acteur, parait adapté à ce genre d'architecture; dans notre étude on s'est occupé surtout des espaces intérieurs, il faut essayer d'exploiter le modèle dans d'autres domaines pour pouvoir identifier les fonctions à implémenter et les difficultés qu'elles présentent.

De même, il faut essayer d'enrichir les bases de connaissances pour aborder des sujets différents que celui des espaces intérieurs comme la conception et le positionnement de locaux, la satisfaction automatique des contraintes (maintenant que la vérification est implémentée), en suivant le modèle de notre application exemple où l'on a abordé le problème de conception par la génération de solutions puis le test et l'évaluation.

D'autre part, on ne traite pas dans notre thèse d'une façon très détaillée le concept de catalogues; on peut s'intéresser à enrichir notre système par des catalogues qu'il faut alors saisir et étudier le maintien et la cohérence.

On n'oublie pas finalement de penser à étendre notre modèle d'implémentation afin de résoudre toutes les restrictions et les hypothèses simplificatrices adoptées au départ (se référer au chapitre 5).

# BIBLIOGRAPHIE

**AISH R.,90**
MASTER ARCHITECT: an object based architectural design and production system
CIB proceedings, conceptual modelling of buildings,SUEDE, 1990

**ANDRE J.M., 86**
vers un système d'aide intelligent pour l'aménagement spatial: CADOO
CIIAM86
S02210

**ANTOINE J.L - PUGELLI M., 90**
ERASME: un système multi-experts d'entretien routier
Actes de la convention IA 90, pp. 149-164

**ASSEMAT C. - MORIGNOT P. - VERNET P., 88**
COPLANER: un système expert en planification de bâtiment
Actes de EuropIA 88 - pp. 33-47
S03183

**ATKIN B., 87**
Prospects for linking CAD with construction management systems
Managing construction worldwide, fifth international symposium, LODON, 1987, vol. 1, pp 415-425
S02598A

**AUTRAN J. - FLORENZANO M. - LEMAITRE J. - PALISSIER C., 89**
SGBD avancées et CAO en architecture, une application à la description d'un projet de bâtiment
Actes de CAO et robotique en architecture et BTP, 1989, pp. 238-255

**AYEL M., PIPARD E., ROUSSET M.C., 86**
Le contrôle de cohérence dans les bases de connaissances
Journées nationales sur l'IA, Aix-les-Bains, NOVEMRE 1986, pp. 171-186
S02184

**BACHIMONT B., 90**
Représentation des connaissances de contrôle dans un tableau noir: l'architecture ABACAB
Avignon 90, pp. 253-267

**BAILLY C. - CHALLINE J.F. - GLOESS P.Y., 87**
Les langages orientés objets
Edition CEPADUES, 1987

**BEHESHTI R., 90**
The possibilities and limitations of building design in an information environment
Actes de EuropIA 90 - pp 331-337
S03788

**BENNIS K. - FREVILLE M.,**
Méthode de construction d'un index spatial R+TREE pour l'organisation des données géométriques
S04264

**BERLANDIER P., 88**
Intégration d'outils pour l'expression et la satisfaction de contraintes dans un générateur de systèmes experts
Rapport de recherche 924, INRIA-Sophia antipolis, Novembre 1988, programme 1

**BERTHOUMIEU P. - FERRIES B., 90**
Definition of a constraint manager. Implementation in the CAD system KREPIS and experimentation
VTT symposium 118, 2nd finish french colloquium for information technology in construction, ESPOO 1990, pp. 77-82
S03911

**BIJL A., 86**
Logic modelling in computer-aided design
Environment and planning B: planning and design, 1986, volume 13, pp. 233-241

**BILLET A. - CHELGHOUM K., 87**
Un modèle de représentation de connaissances pour la conception assistée par ordinateur
actes de MICAD 87 - pp. 433-447

**BJORK B.C., 90**
Issues in the developpement of a building product model standard
Actes de EuropIA 90 - pp 372-3391
S03788

**BOISSIER D. - AL HAJJAR J., 90**
Communication between CAD building systems
VTT symposium 118, 2nd finish french colloquium for information technology in construction, ESPOO 1990, pp. 122-130
S03911

**BOURDEAU M., 87**
Utilisation d'un environnement système-expert afin de résoudre des tâches de conception dans les projets de bâtiment
actes de MICAD 87 - pp. 393-405

**BRIDGES A., 91**
The teaching of computer aided architectural design
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991

**CAMMARATA S. - MELKANOFF M., 84**
An interacyive data dictionary facility for CAD/CAM data bases
Expert database systems international workshop proceedings, KIAWAH Island, 1984, pp. 423-440

**CAO X, PAN Y., GUAN H., 91**
Building integrated intelligent CAD systems
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991

**CARADANT D. - GOULETTE J.P., 88**
IA et conception en architecture et bâtiment: les objets du projet
Raport final de recherche, LI2A, école d'architecture de toulouse, Décembre 88

**CARADANT D., 87**
Application du concept d'objet à la représentation et à l'utilisation de connaissances dans un système de conception assistée par ordinateur en architecture
Thèse de doctorat, université Paul SABATIER de Toulouse, 1987

**CARADANT D., 86**
L'approche objet dans la CAO en architecture: état de l'art et propositions
CAO et robotique en architecture et BTP, MARSEILLE, 1986, pp. 121-130
S01989

**CARDENAS A., 87**
Heterogeneous distributed database management: the HD_DBMS
Proceedings of the IEEE, Vol. 75, No. 5, MAY 1987

**CARRARA G. - NOVEMBRI G., 90**
Knowledge assistants in the process of architectural design
Building and environment, Vol. 25, No 3, pp. 199-207, 1990

**CHEN C.C. - SCHMITT G.N., 90**
Design as learning
Actes de EuropIA 90 - pp 313-320
S03788

**CHOLVY L. - FOISSEAU J., 84**
Bases de données en CAO: modélisation d'objets complexes et des liaisons entre objets
Actes de MICAD 84, pp. 944-957

**CHOLVY L. - FOISSEAU J., 83**
ROSALIE: a CAD object-oriented and rule-based system
Information processing 83, R.E.A. Mason (ed.), 1983, pp. 501-505

**CLERC T. - SADGAL M., 87**
Prototype d'un système de placement intelligent dans le plan
Actes de MICAD 87, pp. 491-503

**COLETTE P., 87**
Un système à base de connaissances pour la synthèse automatique de macro-commandes en CAO
Actes de MICAD 87, pp. 465-477

**CORBY O., 88**
Un tableau réflexif pour la coopération de base de connaissances
Thèse de doctorat, université de Nice, 1988
S04017

**CRAIG HOWARD H., FENVES J., 83**
Representation and comparison of design specifications
Report No. R-83-141, department of civil engineering, CMU, 1983

**DANNER W., 88**
A global model for building project information: Analysis of conceptual structures
NBSIR 88_3754, US. dezpartment of commerce, 1988

**DANNER W., 87**
The use of artificial intelligence programming techniques for communication between incompatible building information system
NBSIR 87_3529, US. dezpartment of commerce, 1987

**DE GARRIDO L.A., 89**
The role of knowledge based systems in the core of architectural design
Revue internationale de CFAO et d'infographie, Vol. 4, No. 4, 1989, pp. 7-33

**DELCAMBRE B., 87**
Systèmes experts d'aide au choix techniques pour les projets de bâtiments
Intelligence artificielle et CAO en BTP, PARIS, NOVEMBRE 1987, pp. 163-183
S02807

**DUBOIS A.M. - LARET L., 90**
CIBAO: conception intégrée de bâtiments assistée par ordinateur
Rapport CSTB MGL/90, juin 1990

**DUFAU J. - MANGIN J.C. - MOMMESSIN M. - SAUCE G., 90**
A strategy for the development of a multitechnics CAD system integrating a knowledge basis
VTT symposium 118, 2nd finish french colloquium for information technology in construction, ESPOO 1990

**DUFAU J. - MANGIN J.C. - MOMMESSIN M. - SAUCE G., 90**
Modélisation et exploitation dynamique des connaissances et des données dans le projet CONCEPTOR
Actes de EuropIA 90 - pp 358-365
S03788

**DUPAGNE A. - HUNEAU J., 90**
Apports de l'intelligence artificielle à l'enseignement de l'architecture
Actes de EuropIA 90 - pp. 303-312
S0788

**EASTMAN C., 91**
The evolution of CAD: integrating multiple representations
Building and environment, Vol. 26, No 1, pp. 17-23, 1991

**EL DAHSAN K. - BARTHES J.P., 90**
Un système intelligent de CAO
Actes de la convention IA 90, pp. 667-682

**EVANS P.M., 90**
Rule-based applications for checking standards compliance of structural members
Building and environment, Vol. 25, No 3, pp. 235-240, 1990

**EXCOFFIER T., 90**
L'utilisation du langage "G" pour coder les arbres CSG
BIGRE 67, Janvier 1990, pp. 146-157

**FARRENY H. - GHALLAB M., 87**
Elements d'intelligence artificielle
Edition HERMES 1987

**FARRENY H., 85**
Les systèmes experts: principes et exemples
Edition CEPADUES, Toulouse 1985

**FAUVET M.C., 87**
Le partage des objets dans un SGBD pour la CAO
Revue internationale de CFAO et d'infographie, Vol. 2, No. 4, 1987, pp. 71-87

**FERRETY G., PEROCHE B., 90**
Une interface évoluée pour un modeleur 3D
BIGRE 67, Janvier 1990, pp. 22-28

**FERRIES B. - CARADANT D., 88**
KREPIS: rapport final de recherche
Convention MELATT 86-A6/04, convention AFME 6 04 0031, JUIN 1988

**FLEMMING U., 90**
Knowledge representation and acquisition in the LOOS system
Building and environment, Vol. 25, No 3, pp. 209-219, 1990

**FLEMMING U. - COYNE R. - GLAVIN T. - RYCHENER M., 89**
ROOSI - version one of a generative expert system for the design of building layouts
Actes de CAO et robotique en architecture et BTP, 1989, pp. 157-166

**FORNARINO M. - PINNA A.M., 90**
Un modèle objet logique et relationnel. Le langage OTHELO
Thèse de doctorat, université de Nice, 1990
S03818

**GARDAN Y. - SINGER D. - VIVIAN R. - ZAKARI A., 90**
Utilisation des langages à objets et des techniques d'apprentissage en CAO
Actes de la Convention IA 90 - pp. 641-665

**GARDAN Y. - SOHNOUNE M., 90**
L'intégration des surfaces dans les modèles de solides
Revue internationale de CFAO et d'infographie, Vol. 5, No. 2, 1990, pp. 83-98

**GARDARIN G., VALDURIEZ P., 90**
SGBD avancés
édition EYROLLES, 1990

**GARNESSON P. - GIRAUDON G. - MONTESINOS P.**
MESSIE: Un système multi-spécialistes en vision. Application à l'interprétation en imagerie aérienne
??? peut etre avignon

**GARRET J. H., 86**
SPEX: a knowledge based standards processor for structural component design
PHD's thesis, department of civil engineering, CMU, 1986

**GARRET J .H., 90**
Application of knowledge-based system techniques to standards representation and usage
Building and environment, Vol. 25, No 3, pp. 241-251, 1990

**GERO J.S., 85**
An overview of knowledge engineering and its relevance to CAAD
*computer aided architectural design futures, international conference on computer* aided architectural design, ed. ALAN PIPES, SEPT. 1985, pp 107-119
S02600

**GLEIZES M.P. - GLIZE P. - TROUILHET S., 90**
La résolution distribuée de problèmes dans un environnement multi-agent
Actes de la convention IA 90, pp. 121-135

**GOULETTE J.P., 89**
MENON: un système expert orienté objet pour l'analyse de scènes en architecture
Actes de CAO et robotique en architecture et BTP, 1989, pp. 100-109

**GOULETTE J.P. - PEREZ J.P., 87**
Spécification d'un poste de CAO en architecture utilisant des techniques de langages à objets
Raport final de recherche, LI2A, école d'architecture de toulouse, 1987

**GUENA F., 87**
Vers un langage pour développer des systèmes de CAO de nouvelle génération pour le bâtiment
Intelligence artificielle et CAO en BTP, PARIS, NOVEMBRE 1987, pp.25-38
S0280

**GUENA F. - ZREIK K., 90**
Des techniques d'apprentissage pour le transfert d'expertise de conception
Actes de EuropIA 90 - pp. 138-147
S03788

**GUENA F. - ZREIK K., 88**
Un système de conception architecturale assistée par architecte
Actes de EuropIA 88 - pp. 133-150
S03183

**HABRAKEN N.**
Control hierarchies in complex artifacts

**HANROT S., 89**
Une base de connaissances architecturales pour un outil de CAO intelligent
Thèse de doctorat, université d'Aix-Marseille III, 1989

**HANROT S., 88**
Transfert de connaissaces architecturales vers une base orientée objet
Actes de EuropIA 88 - pp. 315-329
S03183

**HANROT S. - QUINTRAND P., 88**
Un système de CAO intelligent en architecture
Actes de EuropIA 88 - pp. 387-395
S03183

**HATON J.P., 89**
Panorama des systèmes multi-agents
Architecture avançées pour l'intelligence artificielle, Actes des onzièmes journées francophone sur l'informatique, NANCY, JANVIER 1989 pp. 247-261
S03285

**HAUTIN F., VAILLY A., 86**
La coopération entre systèmes experts
Journées nationales sur l'IA, Aix-les-Bains, NOVEMRE 1986, pp. 187-204
S02184

**HOLTZ N., 82**
Symbolic manipulation of design constraints, an aid to consistency management
PHD's thesis, design research center, CMU, 1982

**ILOG, 88**
SMECI, manuel de référence, version 1.42, 1989

**ILOG, 89**
AIDA, environnement de développement d'applications, version 1.4, 1989

**ILOG, 89**
LE-LISP de l'INRIA,le manuel de référence, version 15.22, 1989

**ILOG 89**
MASAI, générateur d'interfaces, version 1.0, 1989

**KAHKONEN K., BJORK B.C., 87**
Computerization of building standards
Research report 484, Tecnical resarch centre of Finland (VTT), Espoo 1987

**KARAKATSANIS A.G., 85**
FLODER: A floor designer expert system
Master's thesis, department of civil engineering, CMU, 1985

**KENJI I., 91**
Using an object oriented project model for integrated construction engineering systems
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991

**KOHLER N. - MOREL N., 90**
Computer building representation report on an international workshop
Actes de EuroplA 90 - pp 365-371
S03788

**KOSKELA L., 90**
Computer integrated construction - problems and prospects
VTT symposium 118, 2nd finish french colloquium for information technology in construction, ESPOO 1990, pp. 237-241
S03911

**LAASRI H., MAITRE B., 89**
Coopération dans un univers multiagents basée sur le modèle du blackboard: études et réalisations
Thèse de doctorat, université de Nancy I, 1989
S03623

**LAURIERE J.L., 76**
Un langage et un programme pour énoncer et résoudre des poblèmes combinatoires
Thèse de doctorat d'état, université Pierre et Marie Curie, Paris 6, 1976

**LAWSON B., 80**
How designers think
The architectural press ltd, London, 1980

**LEBAHAR J.C., 85**
Etudes en CFAO, architecture et bâtiment
édition HERMES, 1985
S01719

**LEBAHAR J.C., 83**
Le dessin d'architecte
édition PARENTHESES, 1983
S00790

**LE GAUFRE P., 88**
Méthodologie de conception et intelligence artificielle en bâtiment. Etudes pour un système multi-expert d'aide à la décision
Thèse de doctorat, institut national des sciences appliquées de Lyon, 1988

**LEGAUFFRE P. - MIRAMOND M., 87**
Recherche sur l'optimisation des méthodes de conception dans l'habitat dans un contexte de CAO: DEDALE, un système multi-expert pour la construction et la résolution de problèmes multi-critères
Rapport final, institut national des sciences appliquées de Lyon, 1987

**LIGHT R.A. - GOSSARD D.C., 82**
Modification of geometric models through variational geometry
Computer aided design 14 (4) 1982

**LOYE D., 88**
Une approche base de données orientée objets pour l'aide au commandement
Avignon 88, conférences spécialisées, pp.167-185

**LUCAS J.Y., 90**
SIREN, un résolveur de problèmes qui synthétise des programmes
Actes de la convention IA 90, pp. 87-102

**LUCAS M., MARTIN D., MARTIN P., PLEMENOS D., 90**
Le projet exploformes, quelques pas vers la modélisation déclarative des formes
BIGRE 67, Janvier 1990, pp. 35-49

**LUITEN G.T., 91**
A conceptual modeling approach to the development of integrated building project models and systems
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991

**MACULET R., 90**
ARCHITRAME: une maquette de système expert pour la modélisation d'un bâtiment
Actes de EuropIA 90 - pp. 104-112
S03788

**MARSHALL C., 71**
Graph theory
Wiley interscience edition, 1971

**MARCUS S., STOUT J., McDERMOTT J., 87**
VT: an expert elevator designer that uses knowledge-based backtracking
The AI magazine, Winter 1987, pp. 41-57

**MARK E., 90**
A design automation paradox
CIB proceedings, conceptual modelling of buildings,SUEDE, 1990, pp. 267-274

**MARTINI K., POWELL G., 91**
Geometric modelling approaches for civil engineering and building design
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991

**MASINI G. - NAPOLI A. - COLNET D. - LEONARD D. - TOMBRE K., 89**
Les langages à objets
Edition InterEditions, 1989

**MICHIE D. - JOHNSTON R., 87**
L'ordinateur créatif
Edition EYROLLES, 1987

**MIRANDA S. - BUSTA J.M., 90**
L'art des bases de données 2: les bases de données relationnelles
Edition EYROLLES, 1990

**MONCEYRON E.,90**
EXPORT, un poste de travail d'ingénierie portuaire
Avignon 90, pp. 123-135

**MONTALBAN M., 87**
Prise en compte de spécifications en ingénierie, application aux systèmes experts de conception
Thèse de doctorat, université de Nice, 1987

**MONTALBAN M. - HAREN P. - DELCAMBRE B., 87**
Systèmes experts de conception basés sur les spécifications
Revue internationale de CFAO et d'infographie, Vol. 2, No. 4, 1987

**MOREL N. - HAGEN F., 90**
Aide à la conception thermique d'un bâtiment: un système expert pour architectes
Actes de EuropIA 90 - pp. 122-129
S03788

**NARAT V., 86**
Les différentes techniques de représentation de connaissances utilisées en intelligence artificielle
Note d'étude, I8203F, EDF, 1986

**NGUYEN C., 90**
Conception et réalisation d'un logiciel pour la construction interactive d'objets 3D par le modèle CSG
BIGRE 67, Janvier 1990, pp. 158-163

**OXMAN R. - GERO S., 87**
Using an expert system for design diagnosis and design synthesis
Expert systems, February 1987, Vol. 4, No.1, pp. 4-15

**OXMAN R. - OXMAN R., 91**
Formal knowledge in knowledge-based CAD
Building and environment, Vol. 26, No 1, pp. 35-40, 1991

**PAPAMICHAEL K. - SELKOWITZ S.,**
Modelling the building design process and the related expertise
Draft report

**PENNY NII H., 86**
Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures
The AI magazine summer, 1986, pp. 38-53

**PENNY NII H., 86**
Blackboard systems: Blackboard application systems, blackboard systems from a knowledge engineering perspective
The AI magazine August, 1986, pp. 82-106

**PENNY NII H. - NELLEKE A., 79**
AGE: a knowledge based programm for building knowledge based programms
Proceedings of IJCAI 79, TOKYO 1979

**PEREZ P., 86**
L'élément d'architecture: usage et définition dans les systèmes constructifs et langages objets
Actes de CAO et robotique en architecture et construction, MARSEILLE 1986, pp. 110-120
S01989

**PEROCHE B. - ARGENCE J. - GHAZANFARPOUR D., 90**
La synthèse d'images
Edition HERMES, Paris 1990

**PIN P. - CHEN S., 76**
The entity-relationship model - toward a unified view of data
ACM transactions on database systems, vol. 1, No. 1, MARCH 1976, pp. 9-36

**PINSON S., 81**
Représentation des connaissances dans les systèmes experts
R.A.I.R.O. informatique, Computer science, Vol. 15, No. 4, 1981, pp. 343-367

**POHL J. - CHAPMAN A.,**
An expert design generator
Architectural science review, Vol. 31, pp. 75-86

**QUAFAFOU M. - DUBANT O. - PREVOT P.,**
L'approche objet dans la conception des systèmes graphiques
Revue de CFAO et d'infographie - V. 5, No. 4, pp. 55-72

**QUINTRAND P., 85**
La CAO en architecture
édition HERMES, 1985
S01615

**RETIK A. - WARSZAWSKI A. - BANAI A., 90**
The use of computer graphics as a scheduling tool
Building and environment, Vol. 25, No 2, pp. 133-142, 1990

**RICHENS P., 90**
Commercial software developers viewpoint
CIB proceedings, conceptual modelling of buildings,SUEDE, 1990, pp. 225-233

**ROCHE C. - LAURENT J.P., 89**
Les approches objets et le langage LRO2 (KEOPS)
Technique et science informatique, Vol. 8, No. 1, 1989, pp. 21-39

**ROSEN J.P., 90**
Pour une définition méthologique de la notion d'orienté objet
AFCET/INTERFACES, No. 96, octobre 1990, pp, 18-22

**ROSENMAN M.A., 90**
Application of expert systems to building design analysis and evaluation
Building and environment, Vol. 25, No 3, pp. 221-223, 1990

**SHAVIV E., 85**
Layout design problems: systematic approaches
computer aided architectural design futures, international conference on computer aided architectural design, ed. ALAN PIPES, SEPT. 1985, pp. 28-52
S02600

**SHUHEI Y., SHINICHIRO K., OHKAWA M., MASAFUMI N., YOSIHISA U., 91**
Architectural design support system DELTA
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991

**SLAVA M.T., 85**
Structure and organisation of project specifications
Master's thesis, department of civil engineering, CMU, 1985

**SRIRAM D., LOGCHER R.D., GROLEAU N., CHERNEFF J., 89**
DICE: an object oriented programming environment for cooperative engineering design
Technical report No.:IESL-89-03, intelligent engineering systems laboratory, department of civil engineering, MIT, 1989

**SRIRAM D. - STEPHANOPOULOS G. - LOGCHER R. - GOSSARD D. - GROLEAU N. - SERRANO D. - NAVINCHANDRA D., 89**
Knowledge-based system applications in engineering design: researh at MIT
The AI magazine fall 1989, pp. 79-95

**STEINBERG L., 87**
Design as refinement plus constraint propagation: the VEXED experience
AAAI 87, Expert systems, pp. 830-835

**SWEDISH COUNCIL FOR BUILDING, 90**
Housing research and design in Sweden
édition Sven Thibery, 1990

**TAKAMATO T., MOROZUMI M., KIJIMA Y., SEGAWA Y., 91**
A study on the development of object oriented intelligent CAAd system
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991

**TAKEO K., MAKATO T., TAKEO Y., 91**
LORAN-T: Development and application of a newly integrated CAD system
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991

**TARKKO O., 91**
Building design experiments with CAD: some theory and practice
Building and environment, Vol. 26, No 1, pp. 41-48, 1991

**THORAVAL M. - MAZAS Ph. - CHATENET J.P., 90**
ARCHIX, système expert d'aide à la conception automobile
Actes de la convention IA 90, pp.625-639

**TREZFER F., 91**
A model for the integrated use of computer systems in building
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991

**TROUSSE B., 89**
Coopération entre systèmes à base de connaissances et outils de CAO: l'environnement multi-agent ANAXAGORE
Thèse de doctorat, université de Nice, 1989

**TROUSSE B., 90**
Architecture réflexive pour la coopération entre systèmes à base de connaissances et outils de CAO
Actes de MICAD 90, pp. 182-200
S03755

**VERLUISE F., 86**
Méthodologie de formulation des règles et expertises pour les systèmes constructifs. Expérimentation du système expert SNARK sur le système constructif S.E.S. et sur une enquête sémiotique
IN.PRO.BAT, édition PC, 1986

**VESCOVI M. - TOMASENA M., 90**
Générateur de systèmes experts d'aide à la conception caractérisés par une structuration hiérarchique des concepts
Actes de la convention IA 90, p.461-474

**VIGNARD P., 86**
Représentation de connaissances, mécanismes d'exploitation et d'apprentissages
Collection didactique, INRIA, 1986

**WALKER E. - HERMAN M. - KANADE T., 88**
A framework for representing and reasonning about three dimensional objects for vision
The AI magazine summer 1988, pp. 47-58

**WATANABE S., WATANABE H., 91**
Architectural concept modeling in the framework forrepresenting knowledge
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991

**WEEK D., 91**
The structure of CAD and the structure of form
*Building and environment*, Vol. 26, No 1, pp. 49-59, 1991

**WING J., ARBAB F., 85**
Geometric reasonning: a new paradigm for processing geometric information
*Department of computer science*, CMU, 1985

**WOODBURY R., 91**
Searching for designs: paradigm and practice
*Building and environment*, Vol. 26, No 1, pp. 61-73, 1991

**X2A, 88**
X2A: un système de conception assistée par ordinateur en avant projet sommaire de bâtiment
Collection RECHERCHES, 1988

**YOSHITSUGU A., 91**
Correlational analogy method for architectural CAD system
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991

**YUSUKE Y., HIROSHI T., 91**
Application of knowledge based systems to integrate design and consttruction planning
The fourth international conference on computing in civil and building enginering, July 29-31, Tokyo, JAPAN, 1991