Université de Montréal

Méthode proposant l'utilisation de la programmation fonctionnelle pour la génération de solutions architecturales partielles sous forme matricielle

par
Nathalie Charbonneau

Faculté de l'aménagement

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences appliquées (M.Sc.A.)
en **aménagement**

Octobre 2002

Université de Montréal

Faculté des Études supérieures

Ce mémoire intitulé :

Méthode proposant l'utilisation de la programmation fonctionnelle pour la génération de solutions architecturales partielles sous forme matricielle

présenté par :

Nathalie Charbonneau

A été évalué par un jury composé des personnes suivantes :

..... président du jury

Manon Guité..... directeur de recherche

Témy Tidafi..... co-directeur de recherche

..... membre du jury

TABLE DES MATIÈRES

TABLE DES FIGURES

REMERCIEMENTS

Merci à Manon Guité d'avoir encadré ce travail de recherche.

Merci à Temy Tidafi pour l'intérêt qu'il a démontré pour mon sujet de recherche, et ce, à chacune des étapes de mon travail.

Merci à Jean-François Rotgé qui, par son cours de programmation volumique, m'a en grande partie inspiré le sujet de ce mémoire.

Merci également à Arturo Corona pour son appui.

SOMMAIRE

Le but de ce travail de recherche est de proposer une méthode visant à mettre à contribution l'informatique dans la première phase du processus de design architectural. Nous développerons une approche dans laquelle l'outil informatique pourrait contribuer à stimuler le sens créatif du designer en dynamisant sa démarche de création. Nous modéliserons et programmerons un système informatique capable de proposer des solutions architecturales partielles, à partir de contraintes pré-établies par l'utilisateur.

Pour ce faire, dans un premier temps, nous proposerons une façon de délimiter un espace de conception, et ce, dans un langage interprétable par l'ordinateur. Nous illustrerons notre démarche par le biais d'une étude de cas portant sur un ensemble de bâtiments patrimoniaux, du type maison bourgeoise. Celles-ci seront analysées d'une part, dans le but d'identifier les différents espaces qui les composent et, d'autre part, dans le but d'identifier les relations de proximités obligatoires entre les espaces d'un même niveau et entre les espaces de niveaux différents.

Dans un deuxième temps, nous décrirons l'approche pour programmer un système qui proposerait au designer des solutions architecturales partielles du type étudié dans l'étude de cas. Ces solutions seront générées sous forme matricielle. Les matrices seront générées de façon aléatoire et testées afin de déterminer lesquelles sont en conformité avec les règles de proximité spécifiées dans l'espace de conception. Elles seront ensuite présentées à l'utilisateur sous forme d'assemblages de modules.

Dans un troisième temps, nous verrons comment nous pourrons fournir au système une plus grande quantité et une plus grande variété d'informations de façon à enrichir notre espace de conception et générer des solutions architecturales partielles plus variées et plus détaillées.

En dernier lieu, nous décrirons comment l'utilisateur pourra interagir avec le système et effectuer des choix parmi les différentes solutions partielles qui lui sont proposées. En structurant le système sous forme d'une arborescence, nous verrons de quelle façon le designer pourrait éventuellement arriver à une solution architecturale complète conforme à ses préférences et acceptable en regard des règles énoncées dans l'espace de conception.

Mots-clés : conception assistée par ordinateur, modélisation, programmation fonctionnelle.

ABSTRACT

The aim of this research project is to suggest a method that would involve the computer in the first phase of architectural design. We will develop a method in which the computer will stimulate the designer's creative sense. We will conceive a logical sequence of operation to be performed by the computer in propounding partial architectural solutions from the constraint established by the user.

In doing so, we will first suggest a way of circumscribing a "space concept", and this, in a language understandable by the computer. We will illustrate our procedure by means of a case study based on three heritage buildings. These will be analyzed first in order to identify the differents spaces of which they are made up, and secondly, in order to identify the relationship of necessary proximity between the spaces on the same level and between the spaces of the differents levels.

We will then describe a way of programming a system that will generate partial architectural solutions similar to those studied in the case study. Those solutions will be generated by means of matrixes. Those matrixes will be generate randomly and tested in order to find out which ones are in accordance with the proximity rules specified in the "space concept". They will then be presented to the user by means of modular assemblies.

We will then see how we can supply the system with more informations or a different kind of input in order to enlarge the "space concept" and generate more varied or more detailled partial architectural solutions.

Lastly, we will desbribe how the user could interact with the system and make different choices within the different partial solutions proposed. In

structuring the system as a tree, we will see in what way the designer could eventually find a complete architectural solution that would go along his wishes and the rules set out in the ''space concept''.

Key-words: computer assisted conception, modelisation, functional programming

À Émilie et Isabel

1. Introduction

Depuis maintenant plusieurs années, les architectes ont à leur disposition de nouveaux outils de travail : les logiciels dits de CAO. Il est cependant clair que l'utilisation de ceux-ci a été intégrée au processus de design architectural traditionnel sans repenser ce processus. Dans beaucoup de bureaux d'architectes, l'ordinateur est utilisé principalement pour les deux raisons suivantes :

D'une part, on utilise les logiciels de dessin pour tracer les plans préliminaires et d'exécution. Le fait de pouvoir utiliser l'ordinateur pour copier et éditer les dessins d'architecture allège le travail du technicien. Il y a donc augmentation de la vitesse, de la productivité. D'autre part, l'ordinateur est utilisé pour produire des images "photo-réalistes" du projet. On décrit la géométrie des formes et la texture des surfaces pour constituer un modèle dont on se sert pour produire des documents de présentation. Ces simulations visuelles remplacent les perspectives antérieurement tracées à la main et sont souvent utilisées comme outil de marketing pour faire la promotion du projet.

S'il est vrai que l'ordinateur aide le technicien à produire plus rapidement et plus efficacement les documents et qu'il aide le promoteur à "vendre son projet", on peut se demander quelle est l'aide qu'il fournit à l'architecte dans son travail.

Il faut bien reconnaître que la plupart des logiciels dits de CAO ne sont pas vraiment des logiciels d'aide à la conception mais plutôt de dessin assisté par ordinateur. Bien qu'il y ait progressivement émergence de nouveaux outils informatiques destinés aux designers, dans la plupart des cas, on remarque que l'architecte ne fait pas appel à l'ordinateur (ou alors très peu) pour l'épauler dans son travail de conception architecturale.

Pourtant, l'outil informatique pourrait être mis à contribution durant le processus de design. Par exemple, dans la première phase de ce processus, l'architecte doit traiter une grande quantité d'informations et produire une certaine quantité de solutions architecturales préliminaires acceptables en fonction des critères établis dans le programme architectural. Actuellement, l'exploration des alternatives de design est encore principalement faite par le biais d'esquisses à main levée qui sont progressivement modifiées par l'architecte, à mesure que le concept se précise. Il pourrait cependant être avantageux pour l'architecte de préciser progressivement le concept en procédant de façon différente.

Tout comme l'ont fait plusieurs chercheurs avant nous, tels que U. Flemming (1987), W.J. Mitchell (1990) et G. Stiny (1990), il nous semble qu'il serait intéressant de proposer une approche qui permettrait d'intégrer l'utilisation de l'outil informatique durant la première étape du processus de design. La "façon de travailler" de l'ordinateur a différentes caractéristiques qui pourraient être mises à contribution pour l'exploration d'un éventail de possibles solutions architecturales partielles. Pour cela, il faudrait d'une part identifier quelles sont ces caractéristiques et, d'autre part, déterminer de quelle façon elles pourraient aider l'architecte à préciser progressivement le concept.

1.1 Hypothèses

Nous chercherons à élaborer un outil d'aide à la conception qui, à partir de contraintes spécifiées par l'utilisateur, proposerait toute une gamme de solutions possibles. L'utilisation de cet outil pourrait enrichir la réflexion de l'architecte puisqu'il serait "confronté" à tout un éventail d'alternatives, ce qui l'aiderait à clarifier progressivement le concept.

La première des caractéristiques de l'outil informatique que nous pourrions exploiter est la mémoire de l'ordinateur. Comme on le sait, il est possible de fournir à un système informatique une grande quantité d'informations, pourvu que celles-ci soient traduites dans un langage interprétable par l'ordinateur. L'architecte pourrait donc avoir recours à cet outil pour l'aider à mémoriser les informations qu'il doit traiter, par exemple en terme d'interrelations entre les espaces, de dimensionnement, de géométrie, etc.

Deuxièmement, nous pourrions mettre à profit la capacité qu'a l'ordinateur de sélectionner des éléments et/ou de les assembler de façon aléatoire. Cette caractéristique lui permettrait de générer une multitude d'alternatives différentes, sans se restreindre à différents cas de figure déjà vus et déjà éprouvés. Notre outil d'aide à la conception pourrait donc proposer des solutions partielles inédites auxquelles l'utilisateur n'aurait pas nécessairement pensé.

Troisièmement, il y a ce que l'on pourrait appeler "l'efficacité" de l'ordinateur, c'est-à-dire sa capacité à effectuer une tâche répétitive de façon rapide et systématique. Cette caractéristique pourrait permettre à l'ordinateur de tester rapidement une grande quantité d'alternatives afin de déterminer lesquelles sont acceptables en regard des critères établis.

Nous posons l'hypothèse selon laquelle il serait possible, en exploitant ces trois caractéristiques, d'élaborer un outil d'aide à la conception architecturale qui contribuerait à stimuler le sens créatif de l'architecte durant la première phase du processus de design. Cet outil pourrait, dans un premier temps, assimiler certaines données du programme architectural. Dans un deuxième temps, il pourrait générer différentes solutions et les tester afin de déterminer lesquelles sont valides. Et finalement, dans un troisième temps, il pourrait présenter à l'utilisateur les solutions qui auraient été validées.

Ce type d'approche permettrait à l'architecte d'utiliser l'outil informatique non plus seulement comme un moyen d'augmenter la productivité mais également comme un véritable outil d'aide à la conception. Cette évolution pourrait possiblement enrichir le processus de conception et peut-être améliorer la qualité de l'architecture produite. C'est ce qui nous motive, dans le cadre de ce mémoire, à proposer une méthode qui, en tirant profit des technologies actuellement disponibles, nous permettrait d'aborder le processus de design sous un angle différent.

1.2 Objectifs de la recherche

Le but de ce travail de recherche est de proposer une méthode nous permettant d'élaborer un outil d'aide à la conception qui, à partir de contraintes spécifiées par l'utilisateur, proposerait toute une gamme de solutions architecturales valides.

Pour que cet outil soit efficace, il nous apparaît qu'il doit y avoir une bonne qualité de communication de l'utilisateur vers le système informatique. L'utilisateur, en l'occurrence l'architecte, devrait pouvoir communiquer avec le système pour lui transmettre différents types d'informations contenues dans le programme architectural. Notre premier objectif sera d'arriver à traduire certaines de ces informations dans un langage interprétable par l'ordinateur.

Inversement, il nous apparaît que le système devrait pouvoir communiquer efficacement avec l'utilisateur. Il devrait pouvoir lui soumettre les différentes solutions générées dans un "langage" interprétable par l'utilisateur. Notre deuxième objectif sera de traduire graphiquement les solutions générées de façon à ce que l'utilisateur puisse effectuer un choix éclairé parmi les différentes alternatives proposées.

Comme nous l'avons mentionné précédemment, nous chercherons à élaborer un outil qui contribuera à stimuler le sens créatif de l'utilisateur. Il ne s'agit pas de concevoir un générateur automatique de plans, mais bien de tenter de fournir au designer architectural un outil d'aide à la réflexion. Celui-ci pourrait l'aider à résoudre le problème complexe que constitue la conception d'un bâtiment en lui présentant tout un éventail de possibles solutions architecturales partielles. Il ne s'agit donc pas d'essayer de faire le travail du designer à sa place, mais plutôt de l'aider à vaincre ce que l'on appelle "l'angoisse de la page blanche".

1.3 Plan de travail

Nous commencerons par effectuer une revue de littérature afin d'identifier les différentes approches possibles à l'atteinte de notre objectif de recherche, c'est-à-dire l'utilisation de l'outil informatique pour la génération de formes et/ou de solutions architecturales. Nous verrons quels sont les points forts et les points faibles de ces différentes approches, ce qui nous amènera à mieux préciser le type de méthode que nous voulons proposer dans le cadre de ce travail de recherche.

Nous réaliserons ensuite une étude de cas qui nous servira à illustrer et à valider l'approche proposée. Puisque nous voulons générer des solutions architecturales, nous devrons, dans un premier temps, fournir au système des informations sur le type de bâtiment que nous voulons produire. Pour illustrer l'approche, nous avons choisi de nous inspirer d'un groupe de bâtiments patrimoniaux du même type, que nous appellerons l'échantillonnage. Nous

commenterons brièvement les caractéristiques de ces bâtiments ainsi que ce qui a motivé notre choix.

L'échantillonnage de bâtiments sera analysé afin d'extraire certaines des données du programme architectural. Nous identifierons les différents types d'espaces (que nous appellerons "modules") et une série de couples de modules symbolisant les proximités obligatoires entre ces espaces. Nous exposerons la méthode proposée pour transmettre au système ces données, et ce, dans un langage interprétable par l'ordinateur. L'ensemble de ces données, que nous appellerons "l'espace de conception", informera le système sur le type de solution que nous voulons générer.

Il sera question de l'étude de cas tout au long de notre exposé. Il ne s'agira bien sûr que d'un exemple d'application puisque dans la pratique, l'utilisateur du système serait libre de délimiter son propre espace de conception ou de s'inspirer, lui aussi, d'un groupe de bâtiments qu'il aurait sélectionné.

Nous exposerons de quelle façon nous avons procédé pour programmer un système capable de sélectionner et assembler les modules à l'aide de procédures de sélection aléatoire. Nous verrons de quelle façon le système pourra assembler les modules de façon à générer un éventail de solutions architecturales partielles sous forme matricielle.

Les données contenues dans le programme architectural, telles que nous les avons traduites lors de la délimitation de l'espace de conception, seront ensuite utilisées afin de tester les solutions partielles générées aléatoirement par le système. Ces tests nous permettront de déterminer quelles solutions sont valides en regard des critères établis. Celles-ci seront ensuite traduites graphiquement.

Nous avons pris le parti d'aborder progressivement la complexité. Dans le cadre de ce mémoire, nous exposerons la démarche suivie pour élaborer la première ébauche d'un système qui pourrait, dans une phase ultérieure de la recherche, devenir beaucoup plus complexe. Nous avons opté pour une approche progressive, et ce, tant au niveau du type de solution générée qu'au niveau de l'interaction entre le système et l'utilisateur.

D'une part, notre système ne proposera que des solutions partielles sous forme matricielle. Nous ne programmerons pas, dans le cadre de ce mémoire, un système proposant à l'utilisateur des solutions architecturales complètes. Nous verrons cependant comment il sera possible de complexifier progressivement le système de façon à produire des matrices représentant des solutions partielles de plus en plus variées et de plus en plus détaillées.

D'autre part, la génération de solutions architecturales se fera de façon automatique. Le système et l'utilisateur ne seront donc pas en intéraction. Nous proposerons cependant une approche visant à structurer notre système de façon à permettre à l'utilisateur d'effectuer des choix successifs sur des ensembles de solutions partielles, ce qui lui permettrait théoriquement d'arriver éventuellement à une solution architecturale complète.

Après avoir exposé la méthode que nous proposons pour générer des solutions architecturales, nous comparerons les résultats obtenus avec les objectifs fixés. Nous nous positionnerons ensuite par rapport aux différentes approches développées par les chercheurs dont nous avions commenté les travaux. Finalement, différentes pistes seront identifiées en vue de poursuivre cette recherche.

1.4 Définition des termes

Au cours de la présentation de l'approche proposée pour la modélisation et la programmation de notre système générateur de solutions architecturales, nous ferons référence à différents concepts et termes qui sont définis ci-dessous.

Couples de proximités obligatoires :

Liste regroupant deux modules devant obligatoirement occuper des positions mitoyennes dans la matrice de répartition des espaces.

Liste de couples de proximité partiellement obligatoire :

Liste regroupant deux couples de proximité dont au moins un devra se retrouver dans la solution partielle proposée.

"Dimensionless representations" :

Approche développée par William J. Mitchell (1987) et consistant à représenter schématiquement un bâtiment en ramenant toutes les dimensions ou largeurs de travées à l'unité.

Jonctions actives :

Sous-ensemble des jonctions possibles regroupant celles utilisées dans une solution partielle donnée.

Jonction effective :

Jonction occupée par un couple de proximité mettant en relation deux modules différents.

Liste des jonctions de la matrice de répartition :

Liste regroupant une série de couples composés des modules occupant des positions mitoyennes dans la matrice de répartition des espaces.

Listes de répartition :

Matrice de répartition des espaces traduite sous forme de liste.

Matrice 3D :

Matrice servant de "squelette" à la solution architecturale proposée. Liste regroupant des sous-listes représentant les différents niveaux du bâtiment.

Matrice inter-niveaux :

Matrice indiquant l'emplacement des circulations verticales et les types de jonctions.

Matrice espaces-jonctions :

Matrice de répartition des espaces incluant des informations sur les types de jonctions horizontales et verticales proposées par le système.

Matrices de répartition des espaces :

Matrice représentant, à l'aide des codes numériques des modules, la distribution des espaces pour un niveau donné.

Matrices de restriction :

Matrice mettant en relation une série de modules

- avec elle-même
- avec une autre série de modules
- avec des codes de jonctions
- avec des valeurs numériques

de façon à transmettre au système les données contenues dans le programme architectural.

Modules :

Code numérique représentant un type d'espace.

2. L'intégration de l'outil informatique au processus de design architectural

Depuis quelques années, le développement de l'informatique graphique et de la programmation ont entraîné une prise de conscience dans les écoles d'architecture. De plus en plus, on cherche à intégrer l'informatique au processus de design architectural. Différentes avenues sont explorées afin de proposer aux architectes de nouvelles façons de faire. Au cours des pages qui suivent, nous effectuerons un bref survol de la recherche dans ce domaine. Nous commenterons les différentes approches qui nous ont semblé les plus en rapport avec notre sujet de recherche, ce qui nous amènera à mieux cerner l'approche que nous souhaitons proposer.

2.1 Revue de littérature : approches développées

2.1.1 La génération automatique de solutions architecturales

Ulrich Flemming est un des chercheurs du département d'architecture de l'Université Carnegie-Mellon. Il a constaté que certains designers avaient peur que l'utilisation de l'informatique déshumanise le processus de design, mais il considère ces craintes comme étant sans fondement. Selon lui, l'ordinateur ne doit pas être considéré comme un substitut à l'être humain mais comme un outil dont l'utilité est, dans le processus de design, de présenter au designer un miroir qui lui retourne ses propres idées élaborées de différentes façons. Il s'agit donc d'une aide à la réflexion et la responsabilité de l'utiliser ou non, et de comment on l'utilise, revient à l'utilisateur.

Selon Flemming (1994), puisque l'ordinateur foncitonne très différemment de la façon dont nous le faisons, il peut remplir certaines tâches mieux que l'être

humain. Avant d'intégrer l'outil informatique au processus de design, il considère qu'il est primordial de décider des aspects du design pouvant être supportés par l'ordinateur et des bénéfices que l'on compte en tirer.

Comme nous l'avons mentionné, pour élaborer notre système générateur de solutions architecturales, nous utiliserons l'ordinateur pour des tâches dans lesquelles il excelle : la mémorisation, la sélection aléatoire et l'accomplissement de tâches répétitives. Le bénéfice que nous comptons en tirer est la stimulation du sens créatif de l'utilisateur puisque celui-ci se verra présenter des solutions inédites auxquelles il n'aurait pas nécessairement pensé.

Flemming (1987) a entrepris des travaux de recherche dans le domaine de la génération automatique de solutions architecturales et a travaillé, entre autres, à une étude de cas extrêmement intéressante. Il a cherché à utiliser l'ordinateur pour générer des maisons de style Queen Anne. Il a émis l'hypothèse qu'en choisissant, dans le cadre de son étude de cas, un style dont la géométrie était particulièrement complexe, il arriverait à programmer un système qui pourrait, par la suite, être utilisé pour générer des solutions architecturales de n'importe quel style.

L'étude de cas de Flemming portait sur l'ensemble patrimonial de Shadyside, à Pittsburgh. Il s'agissait de développer une meilleure compréhension du style architectural Queen Anne, dans le but de pouvoir insérer de nouveaux bâtiments sans briser la cohérence visuelle du secteur. Suite à l'étude des modèles existants sur le site, deux ensembles de règles de composition ont été établis dans le but de générer de nouveaux modèles de maisons.

Le premier ensemble de règles concernait l'organisation spatiale. Le système générait le plan du rez-de-chaussée en procédant par étapes. Le hall était d'abord placé et ensuite les pièces et l'escalier principal étaient disposées

autour, selon les règles pré-établies. On arrêtait à un certain niveau de détail. Par exemple, le système ne déterminait pas l'emplacement de l'escalier de service parce que les règles de proximité entre les espaces et d'interrelations avec l'étage étaient trop complexes à formuler. Après avoir généré le plan de base, le système procédait à une extrusion pour ajouter la troisième dimension et ce plan était recopié pour définir le deuxième étage.

Le deuxième ensemble de règles concernait l'articulation extérieure. Il s'agissait d'articuler le volume obtenu selon un style particulier. Premièrement, les espaces intérieurs étaient déformés de façon à génèrer des saillies. Deuxièmement, la toiture était ajoutée. (Flemming précise que, à cette étape, certains problèmes géométriques subsistaient.) Finalement, le système procédait aux rafinements géométriques et aux additions; cheminées, bow-windows, tourelles, porches etc.

La principale remarque que nous pourrions formuler face à cette approche est qu'elle implique une simplification non seulement au niveau des formes géométriques, mais également au niveau des règles d'assemblage entre les espaces. Bien que Flemming insiste sur l'importance de développer ses propres règles, il semble clair que certaines d'entre elles lui sont apparues trop complexes à formuler, comme par exemple les règles visant à assembler les pièces et à mettre en relation les étages.

Cependant, si la formulation de ces règles n'est pas adéquate, il sera impossible d'arriver à générer automatiquement, de la façon dont le propose Flemming, une solution architecturale acceptable. Dans ce contexte, on peut se demander si la formulation de règles d'assemblage est la bonne façon d'aborder le problème complexe qu'est la conception architecturale.

Ce questionnement rejoint les préoccupations de Scott C. Chase (1999). Ce chercheur s'est interrogé sur le fait que, depuis 1980, bien qu'il y ait eu

beaucoup de recherches entreprises sur l'utilisation de l'informatique pour la génération de formes et/ou de solutions architecturales, les progrès dans ce domaine ont été, quant à eux, plutôt lents.

Selon lui, c'est parce que les systèmes développés ont une approche "kit-of-part", c'est-à-dire qu'on cherche à assembler des éléments pour former un tout. Comme il est difficile d'anticiper tous les agencements possibles qu'un utilisateur potentiel pourrait souhaiter utiliser, beaucoup de restrictions et de simplifications sont imposées au processus de design pour arriver à programmer le système.

On peut se demander si les travaux de Flemming ne représentent pas un exemple de cette approche "kit-of-part", puisque le plan de rez-de-chaussée est généré en procédant par étapes. Le hall est d'abord placé et ensuite les pièces et l'escalier principal sont disposés, selon les règles pré-établies.

On constate clairement les limites de ce type d'approche. Le principal désavantage est que les règles visant à assembler les espaces et à mettre en relation les niveaux deviennent rapidement impossible à formuler. C'est pourquoi Flemming a eu recours à différents types de simplifications et de restrictions au niveau de la distribution des espaces.

D'une part, il prend le parti de ne pas tenir compte de certaines composantes comme, par exemple, l'escalier de service. Cependant, dans ce type de bâtiment, il s'agit d'une communication verticale importante. Comment justifier cette omission? D'autre part, il a recours à la duplication pour générer les plans des différents niveaux. Cependant, il est clair que la répartition des espaces d'un l'étage n'est pas nécessairement la copie de celle d'un rez-de-chaussée.

Le deuxième désavantage de l'approche "kit of part" est qu'il y a simplification du processus de design puisque procéder par étape revient en quelques sortes à appliquer une recette. Or, il apparaît évident que cette "recette" ne fonctionnera pas nécessairement à tous les coups.

En effet, Flemming (1987) fait remarquer qu'il est plus facile de développer des règles de composition qui font ce que l'on demande que de les empêcher de faire ce qu'elles ne doivent pas faire. On peut donc supposer que, parmi l'ensemble des solutions architecturales proposées par le système qu'il avait programmé, une certaine quantité était irrecevable.

En somme, l'intérêt d'une approche comme celle de Flemming (1987) réside principalement dans le fait qu'elle permettrait théoriquement d'arriver à définir un système qui pourrait, par la suite, être utilisé pour générer des solutions architecturales de n'importe quel style. C'est pourquoi il a opté, dans le cadre de son étude de cas, pour un type de bâtiments caractérisé par des interrelations entre les espaces présentant une certaine complexité et une géométrie offrant une certaine richesse.

Par contre, pour l'élaboration d'un système générateur de solutions architecturales, il semblerait souhaitable de s'éloigner de l'approche dite "kit-of-part", afin de ne pas imposer de simplifications au niveau de l'interrelation entre les espaces, que ce soit sur le plan horizontal ou sur le plan vertical.

L'aptitude à mettre les espaces en relation de façon adéquate nous semble être l'une des qualités essentielles que devrait avoir un système générateur de solutions architecturales. Dans un premier temps, c'est donc sur cet aspect de la question que nous devrions concentrer notre attention.

Afin de pouvoir traiter de façon adéquate le problème des relations entre les espaces et des interrelations entre les niveaux, il serait possible d'évacuer

toute la question de la géométrie. Les espaces seraient symbolisés par des modules et les solutions partielles par des assemblages de modules.

Plutôt que d'adopter l'approche de Flemming (1987) qui consiste à travailler avec des règles d'assemblage, il serait probablement souhaitable de formuler des règles de proximités obligatoires qui seraient traduites dans un langage interprétable par l'ordinateur pour être intégrées à l'espace de conception.

Un système basé sur ce type de règles produirait des assemblages de modules disposés de façon aléatoire et les testerait afin de vérifier si les règles de proximité obligatoires entre les espaces et d'interrelation entre les niveaux sont respectées. Si le résultat de ce test est positif, la solution serait proposée à l'utilisateur. S'il est négatif, la solution serait éliminée. De cette façon, l'utilisateur ne se verrait présenter que des solutions acceptables. Tant qu'il ne serait pas satisfait, c'est-à-dire tant qu'il n'aura pas été "séduit" par une solution, il demanderait au système de lui en proposer de nouvelles.

Un autre désavantage de l'approche de Flemming (1987) est qu'il nous propose un système générant automatiquement des solutions. Il n'y a pas d'interaction entre le système et l'utilisateur. Soit on présente à l'utilisateur une solution acceptable (qui lui plaît ou non), soit on lui présente une solution irrecevable. Pourtant, si l'on désire réellement contribuer à stimuler le sens créatif de l'utilisateur, il est primordial de lui permettre d'exprimer ses préférences et d'effectuer des choix.

2.1.2 L'interaction utilisateur-système

Selon Scott C. Chase (1999), il serait souhaitable que les designers passent de l'utilisation des outils "passifs" comme les logiciels de dessin assisté par ordinateur, à l'utilisation d'outils "actifs", comme les systèmes générateurs de solutions architecturales. Pour que ceux-ci soient effectifs, il recommande aux chercheurs de développer des interfaces performantes qui permettraient à l'utilisateur d'avoir recours à des outils de dessin pour construire le vocabulaire, énoncer les règles et définir un état initial.

Cette recommandation ne vise pas réellement à assurer une transition progressive vers des outils "actifs", mais illustre plutôt ce que l'on pourrait appeler "l'obsession de la souris". En effet, on oublie trop souvent qu'il y a à peine plus de vingt ans, les designers n'avaient jamais utilisé les fonctionnalités inhérentes aux logiciels de CAO. Pourquoi maintenant en faire un élément incontournable du processus de design ?

Il nous semble que l'intérêt d'un outil d'aide à la conception repose en grande partie sur la qualité des interactions possibles entre l'utilisateur et le système. Chase prend le parti de proposer à l'utilisateur des outils de dessin comme moyen de communication. Cependant, les travaux de plusieurs chercheurs (Krawczyk 1997, Darlymple & Gerzso 1998, Piazzalunga & Fitzhorn 1998) ont démontré qu'un langage de programmation est beaucoup plus puissant qu'un outil de dessin pour construire un vocabulaire ou formuler des règles.

De plus, le langage de programmation, tout comme l'outil de dessin, permet de travailler dans une séquence non-linéaire, c'est-à-dire d'effectuer un travail itératif entre le développement des règles et leur application.

Michael J. Dalyrmple et de J. Michael Gerzso (1998) sont deux chercheurs qui ont utilisé un langage de programmation pour la génération de solutions architecturales. Ils ont introduit la notion de "dessin exécutable", dans laquelle la forme architecturale est représentée par des entités graphiques (lignes ou formes) ainsi que par du code.

Contrairement à l'approche traditionnelle, où chaque dessin correspond à un design ou à certains aspects d'un design, le "dessin exécutable" correspond, quant à lui, à un ensemble de designs. Un design spécifique est le résultat de l'exécution du code à l'intérieur d'un dessin pour un ensemble particulier de valeurs de paramètres.

L'espace est subdivisé en sous-espaces 3D et le système permet de placer et d'exécuter du code dans un espace tri-dimensionnel. Les chercheurs font remarquer que la façon dont l'utilisateur interagit avec le système est une combinaison de comment il interagit avec un logiciel de CAO traditionnel et avec un tableur. Il semble donc, encore une fois, qu'on cherche à s'appuyer sur les logiciels actuellement disponibles sur le marché pour sécuriser l'utilisateur du système.

Dalyrmple et Gerzso (1998) font remarquer que la complexité des autres systèmes rend nécessaire l'intervention d'un spécialiste pour éditer le code ou pour produire les règles. Avec le "dessin exécutable", le processus de design serait, d'après eux, plus intuitif puisque l'utilisateur peut "jouer" avec les variables et vérifier les résultats. Mais, peut-on se demander, au-delà de la modification de variables, comment arriver à formuler nos propres règles si on ne connaît pas le langage de programmation utilisé?

L'approche de Dalyrmple et Gerzso (1998) nous semble entretenir la dualité qui existe entre designer et programmeur. Cependant, le designer

architectural qui maîtriserait bien un langage de programmation n'aurait-il pas entre les mains un formidable outil d'aide à la conception?

Les travaux Robert J. Krawczyk (1997) constituent un bon exemple dans ce sens. Ce chercheur a mis sur pied un cours de programmation destiné aux étudiants en architecture de l'Illinois Institute of Technology. AutoLISP a été utilisé pour développer des programmes servant à générer des formes ne pouvant pas être obtenues en assemblant les primitives habituellement disponibles dans les logiciels de CAO.

Krawczyk (1997) a constaté que jusqu'à maintenant on a surtout reproduit des formes à l'ordinateur mais que le potentiel de l'outil informatique pourrait aller bien au-delà de cet emploi. En utilisant un langage de programmation, il considère que l'ordinateur pourrait nous permettre d'en inventer de nouvelles. C'est dans cette optique qu'il a développé des exercices visant à enseigner aux étudiants à générer des formes basées sur des règles et sur l'aléatoire.

Selon Krawczyk, le concepteur peut concevoir non seulement en dessinant, mais aussi en programmant. Il se produit une fusion des deux activités. Il a donc voulu mettre l'accent sur l'importance du feedback que les règles peuvent apporter et comment elles peuvent être utilisées pour développer un concept ou le clarifier. Ses travaux nous semblent extrêmement significatifs puisqu'ils ont contribué à démontrer à des étudiants en architecture que le développement d'un programme peut être le moyen d'exprimer une idée.

Toutefois, il nous semble que son approche pourrait être qualifiée de limitative. En effet, il a choisi AutoLISP comme langage de programmation, ce qui l'a contraint à avoir obligatoirement recours au logiciel AutoCAD. Il est possible d'aborder différemment la question et de tenter de s'affranchir de ce type de contrainte afin de ne pas se limiter à l'utilisation d'un logiciel en particulier. Dans cette optique, deux avenues différentes peuvent être explorées.

Il est possible de prendre le parti de développer un logiciel indépendant, c'est-à-dire de tenter d'élaborer en même temps la structure des données, les algorithmes géométriques et l'interface, en plus de mettre au point le système générateur de solutions architecturales. Il est clair que développer ces différentes composantes et les intégrer en un tout constitue un problème extrêmement complexe.

D'autre part, il est possible de prendre un parti semblable à celui de Piazzalunga et Fitzhorn (1998), c'est-à-dire de proposer une approche "minimaliste" (comparativement à celles de leurs confrères). Ces chercheurs ont décidé de centrer leur attention sur la définition de règles plutôt que sur l'algorithmie géométrique et sur l'interface. Ils ont intégré leur système générateur de solutions architecturales à un moteur de modélisation déjà existant.

Le moteur de modélisation que ces deux chercheurs ont choisi est ACIS, celui-là même qui a servi à développer environ 70% des logiciels de CAO actuellement disponibles sur le marché. Ils ont utilisé le langage de programmation Scheme pour avoir accès aux fonctions du moteur de modélisation.

En somme, le principal avantage de ce type d'approche réside dans le fait qu'en évacuant toute la question de l'algorithme des formes et de l'interface, il est possible d'aborder progressivement la complexité. De plus, l'utilisation d'un langage de programmation permet de s'affranchir de toutes les contraintes liées à l'utilisation des fonctionnalités inhérentes aux logiciels dits de CAO, ce qui permettrait d'offir au designer un outil plus "souple" que ceux proposés par Scott (1999) et Dalyrmple & Gerzso (1998).

Bien que, dans le cadre de cette recherche, nous ne travaillions pas sur la génération de formes, le système que nous définirons pourrait, tout comme celui de Krawczyk (1997), générer des solutions basées sur des règles et sur l'aléatoire. Les règles seraient délimitées dans l'espace de conception et l'aléatoire serait introduit par le biais des procédures de sélection à l'intérieur d'ensembles pré-définis.

2.2 L'approche proposée

Pour notre étude de cas, nous prendrons le parti de concentrer notre attention sur la question des proximités obligatoires entre les espaces d'un même niveau et entre les espaces de niveaux différents. Afin de pouvoir nous concentrer sur cet aspect de la problématique, nous évacuerons toute la question de la géométrie.

Les espaces seront symbolisés par des modules et les solutions partielles par des assemblages de modules. Nous ne travaillerons pas avec des règles d'assemblage. Nous établirons plutôt des règles de proximités obligatoires qui seront traduites dans un langage interprétable par l'ordinateur pour être intégrées à l'espace de conception.

Pour délimiter celui-ci, nous travaillerons avec des ensembles formés d'éléments et des ensembles formés de sous-ensembles. Ceux-ci seront traduits sous forme de listes et de listes de listes qui seront fournies au système. Celui-ci nous retournera des solutions architecturales sous forme de listes de listes ou matrices pouvant être utilisées de façons diverses. Elles pourront être transmises à un moteur de modélisation volumique existant ou à un logiciel de dessin ou de rendu graphique.

Pour générer et traiter ces listes, nous utiliserons Scheme (le même langage de programmation que Piazzalunga et Fitzhorn), et ce, pour les trois raisons suivantes: sa simplicité, sa flexibilité et ses possibilités de récursion.

En effet, le peu de formalisme des fonctions permettant l'écriture des programmes informatiques (qu'il s'agisse de la syntaxe ou de la sémantique), fait de Scheme un langage relativement simple. Comme le fait remarquer Temy Tidafi (1996), "Les programmes écrits en langage Scheme sont faciles à comprendre et leur analyse permet de voir comment l'auteur s'y est pris pour arriver à ses fins". Il s'agit d'un avantage puisqu'idéalement l'utilisateur du système devrait pouvoir comprendre de quelle façon les procédures sont codées et être capable de les modifier à sa convenance.

De plus, il s'agit d'un langage flexible puisque, contrairement à d'autres langages de programmation, ceux dits fonctionnels (comme Scheme) ne nécessitent pas un typage des variables considérées. Il n'est pas nécessaire de déclarer explicitement si une variable désigne un nombre réel, une procédure, une chaîne de caractères, etc.

Le langage Scheme nous permettra également d'écrire des procédures récursives. La propriété récursive d'un modèle se traduit par le fait que le modèle peut faire appel à sa propre définition pendant qu'il produit un résultat. Il s'agit d'un avantage intéressant puisque la récursivité nous permettra de coder de la façon la plus concise possible des procédures nous permettant de mélanger des éléments, de les assembler aléatoirement et de tester les assemblages.

Précisons que ces tests seront programmés sous la forme de procédures prédicat, c'est-à-dire retournant le résultat "vrai" ou "faux". Il n'y aura pas d'évaluation qualitative de la solution générée aléatoirement. Nous n'aurons pas recours à des techniques d'optimisation qui sélectionneraient automatiquement

la solution optimale pour la présenter à l'utilisateur. Nous prendrons plutôt le parti de lui présenter "toutes" les solutions générées par le système et qui auraient été validées.

Il nous semble utopique de penser que, pour résoudre un problème de conception, il existe dans chaque cas une solution optimale. Le système ne dictera pas ses choix puisqu'il n'est pas souhaitable que l'utilisateur, c'est-à-dire le designer, devienne spectateur du processus de design. Étant donné que nous cherchons à élaborer un outil d'aide à la réflexion, il semble préférable de présenter à l'utilisateur un ensemble de solutions acceptables et de le laisser sélectionner celle(s) qui lui convient.

Les matrices générées par le système seront en fait des listes de listes de codes numériques. Elles pourront ultérieurement être exploitées de diverses façons. Pour la visualisation des solutions partielles, nous prendrons donc le parti de ne nous restreindre ni à l'utilisation d'un logiciel existant, ni à l'utilisation d'un moteur de modélisation en particulier.

2.2.1 Méthodologie

Comme nous l'avons mentionné précédemment, nous avons décidé d'aborder progressivement la complexité. Le type d'outil que nous cherchons à élaborer devrait idéalement permettre à l'utilisateur d'interagir avec le système afin d'élaborer une solution architecturale complète et tenant compte de ses préférences. Cependant, nous commencerons tout d'abord par programmer un système capable de générer de façon automatique une solution architecturale partielle sans qu'il y ait d'interaction possible entre le système et l'utilisateur.

Pour ce faire, la première étape consistera à délimiter notre espace de conception. Nous sélectionnerons un ensemble de bâtiments qui formera notre

échantillonnage. Nous analyserons ensuite les éléments de celui-ci. Il s'agira d'identifier les différents étages ou niveaux qui composent ces bâtiments et de dresser la liste de tous les espaces susceptibles de se retrouver à chacun de ces niveaux.

À chacun de ces espaces ou module, nous assignerons ensuite un code numérique. (Nous pourrions prendre le parti d'identifier les modules par des chaînes de caractère. Cependant, la notation numérique nous permettra éventuellement de reconnaître l'espace X comme appartenant à un niveau donné lorsque X est plus grand ou égal au nombre N et plus petit ou égal au nombre M.)

Puis, nous identifierons les différentes relations de proximité qui seront considérées comme étant obligatoires. Nous présenterons ces informations sous forme matricielle afin qu'elles puissent être transmises à des procédures et traitées par le système.

La deuxième étape consistera à programmer un système capable de générer une matrice 3D qui sera en quelque sorte le "squelette" d'une solution architecturale. La matrice 3D sera composée de sous-matrices représentant les différents niveaux du bâtiment. Chacune de ces sous-matrices sera composée des nombres symbolisant les différents espaces ou modules dont sera composée la solution architecturale.

La matrice 3D sera construite en assemblant les codes numériques de façon aléatoire. Elle sera ensuite testée afin de déterminer si cet assemblage tient compte des proximités obligatoires entre les espaces d'un même niveau, ceci de façon à respecter la circulation horizontale, et des proximités obligatoires entre les modules de différents niveaux, ceci de façon à respecter la circulation verticale. Ces tests représentent donc une étape cruciale pour assurer la

cohérence de la solution architecturale qui sera proposée par le système. Nous validerons ensuite ce système à l'aide d'un exemple d'application, l'étude de cas.

Comme nous l'avons mentionné, pour la programmation de notre système générateur de matrices 3D, nous utiliserons le langage de programmation Scheme. Toutes les procédures dont il sera question au cours des prochaines pages sont écrites dans ce langage et sont présentées en annexe.

3. L'étude de cas

3.1 L'échantillonnage

La méthode que nous proposons pour délimiter un espace de conception et pour générer des solutions architecturales partielles sera illustrée par le biais d'une étude de cas. Celle-ci nous permettra également, à la fin de notre exposé, de valider notre approche.

Notre étude de cas consistera à sélectionner un ensemble de bâtiments et à nous inspirer de ceux-ci pour délimiter un espace de conception de façon à pouvoir générer des solutions architecturales partielles du même type.

Dans un premier temps, nous déterminerons sur quels critères nous nous baserons pour sélectionner les bâtiments dont nous souhaitons nous inspirer. Une fois l'échantillonnage sélectionné, nous pourrons alors commencer à étudier la façon dont nous analyserons ces éléments afin de dégager une connaissance et de la traduire dans un langage interprétable par l'ordinateur.

L'analyse que nous ferons de ces éléments nous servira à identifier les espaces (ou modules) avec lesquels nous travaillerons et à déterminer de quelle façon ils sont mis en interrelation dans des bâtiments de ce type. Puisque nous avons pris le parti d'aborder progressivement la complexité, nous nous limiterons, dans une première étape, à n'étudier que cet aspect de la délimitation d'un espace de conception.

3.1.1 Critères de sélection

Nous avons choisi, pour délimiter notre espace de conception, de nous inspirer d'un type d'architecture comparable, au niveau de la complexité, aux maisons de style Queen Anne étudiées par Flemming (1987). Il s'agit de trois demeures bourgeoises conçues à la fin du XIXième siècle par l'architecte montréalais Edward Maxwell.

Nous tenons à souligner le fait que la méthode proposée ne se limite pas à délimiter un espace de conception uniquement à partir d'un échantillonnage exclusivement constitué de bâtiments conçus par un seul et même architecte. Pour notre étude de cas, nous avons cherché à former un échantillonnage relativement homogène. Bien que nous ayons sélectionné trois bâtiments conçus par Maxwell, il est clair que l'approche développée serait également valable pour un ensemble de bâtiments d'un même type, mais conçus par différents architectes.

Pour sélectionner les éléments de l'échantillonnage, nous nous sommes basés sur les critères suivants :

a) Le type de bâtiment
b) Le style architectural ou de la période de construction
c) La répartition des espaces
d) Les primitives géométriques utilisées

En ce qui a trait à la répartition des espaces et aux primitives géométriques utilisées, un échantillonnage valide devrait présenter un certain équilibre entre les constantes et les variantes des différents éléments. En effet, un trop grand nombre d'éléments constants nous conduirait à produire des solutions architecturales similaires aux membres de l'échantillonnage et semblables entre elles. Ceci rendrait inintéressant le système puisque nous

cherchons à produire un éventail varié de solutions architecturales possibles. Et si, au contraire, un trop grand nombre d'éléments variait, le système à modéliser s'en trouverait grandement complexifié sans pour autant nous assurer de la cohérence des solutions proposées.

3.1.1.1 Le type de bâtiment

Nous avons sélectionné des bâtiments conçus par l'architecte Edward Maxwell principalement pour deux raisons. Notre première préoccupation était la "qualité architecturale" des bâtiments de l'échantillonnage. Notre propos n'étant pas ici de définir ce vocable, nous avons donc porté notre choix sur des éléments considérés par les historiens de l'architecture comme étant des bâtiments de qualité.

La renommée de cet architecte montréalais n'est plus à faire. Par exemple, selon Robert G. Hill (1991), "la production des frères Maxwell est plus abondante et plus diversifiée que celle de toute autre agence canadienne à la même époque, et d'une qualité supérieure." Il ne s'agit évidemment pas ici d'une opinion isolée puisque plusieurs autres ouvrages analysent les qualités de l'œuvre de ces architectes de renom.

Bien que dans le cadre de ce mémoire, nous nous limitions à générer des solutions architecturales partielles sous forme d'assemblage de module, il est important de choisir un type d'architecture caractérisée par une certaine richesse au niveau géométrique et une certaine complexité au niveau des interrelations entre les espaces. Un échantillonnage ainsi formé est un des éléments essentiels qui nous permettront d'approfondir notre approche dans une phase ultérieure de la recherche.

Deuxièmement, et d'un point de vue plus pragmatique, notre décision a également été motivée par la possibilité de consulter les plans originaux des bâtiments sélectionnés. En effet, une quantité importante de plans des bâtiments conçus par l'agence Maxwell est regroupée dans la Collection d'Architecture Canadienne de l'Université McGill. Le but de cette recherche n'étant pas d'expérimenter différentes techniques d'acquisition de données spatiales, mais bien de modéliser un système pour la génération de solutions architecturales, l'accès aux plans des archives nous fait gagner un temps précieux puisque nous n'avons pas à effectuer de relevés.

L'œuvre de Edward Maxwell est d'une grande diversité puisqu'elle compte des hôtels, des gares, des banques, des musées, des bibliothèques, etc. La conception de tels bâtiments requièrent l'élaboration d'un programme architectural complexe, exigeant une connaissance approfondie des activités qui s'y déroulent. Nous pourrions prendre le parti d'étudier en profondeur l'un de ces types de bâtiment. Cependant, dans le but d'aborder progressivement la complexité, nous optons pour des résidences puisqu'il s'agit d'un type de bâtiment dont le nombre de pièces est relativement restreint et dont nous connaissons déjà (globalement) les fonctions.

Notons toutefois que, bien que les résidences abritent un nombre de fonctions inférieur aux bâtiments publics, le programme architectural de la maison bourgeoise du XIX^{ième} siècle est nettement plus complexe que celui de l'habitation uni-familiale telle que nous la connaissons actuellement. Afin de démontrer qu'il s'agit d'un type de bâtiment caractérisé par une certaine complexité au niveau de l'interrelation entre les espaces, nous expliquerons brièvement ce qui caractérise la demeure bourgeoise en Angleterre et en Amérique du Nord, à cette époque, et nous illustrerons notre propos à l'aide des plans d'une résidence typique de cette période.

Dans ce type d'habitation, on distingue soigneusement entre espaces publics et privés. La demeure bourgeoise est à l'exemple d'une forteresse. L'accès en est gardé. On dispose un sas à l'entrée pour que les visiteurs puissent être filtrés, dès le vestibule, par des serviteurs.

Le hall d'entrée et le salon font l'objet de la plus grande attention. Ils concentrent la richesse et les éléments de la représentation de l'habitation. Le salon doit communiquer avec le hall d'entrée et l'on ne doit pas avoir à traverser la salle à manger ni d'autres dégagements pour y accéder. Cette pièce, appelée le "Drawing Room" doit être assez vaste pour les réceptions mondaines et l'on doit pouvoir y circuler sans s'asseoir. Le salon est un lieu de séjour et/ou de passage, ce qui implique des portes doubles.

La salle à manger ("Dining Room") doit communiquer avec le salon et le hall d'entrée. Les dimensions de la pièce doivent assurer le bien-être de nombreux convives et la facilité du service. Les fenêtres de la salle à manger doivent être placées du côté d'un bout de la table, de façon à ce que la plupart des convives ne fassent pas ombre sur eux-même.

Il doit y avoir une pièce de service, appelée "Butler's Pantry", attenante à la salle à manger. Cette pièce doit être disposée de façon à ce qu'il n'y ait jamais de communication directe entre la salle-à-manger et la cuisine, en raison des odeurs et du bruit. La cuisine, quant à elle, est parfois installée au sous-sol. Lorsque c'est le cas, elle communique avec une pièce de service, appelée "Cook's Pantry" et est desservie par un escalier de service et un monte-plat.

Un autre espace public qui doit être liée au hall d'entrée est le cabinet ou la bibliothèque ("Library"). Cette pièce sert à recevoir des amis, des clients ou des fournisseurs. On retrouve également un fumoir et/ou une salle de billard, exclusivement à l'usage des hommes.

Jusqu'à la fin du XIX^{ième} siècle, les lieux d'aisances sont installés dans les endroits les plus biscornus et dans les petits espaces laissés libres par la volonté de géométrisation des pièces. Ils sont soigneusement éloignés des espaces de réception.

Les chambres, quant à elles, doivent être regroupées au premier étage car elles représentent l'habitation intime. Les couples doivent avoir des chambres séparées puisque partager la même chambre (ou pire, le même lit) est signe de pauvreté. La chambre principale est celle de Madame. Elle doit communiquer avec celle de Monsieur de façon à ce que les occupants ne soient pas obligés de traverser une partie publique de l'habitation pour passer d'une chambre à l'autre. Les annexes à chacune des chambres principales sont le cabinet de toilette, la salle de bain, la garde-robe et la lingerie.

Un exemple de la répartition de ces différents espaces est illustrée, ci-après, à l'aide du plan du rez-de-chaussée et de l'étage d'une demeure bourgeoise conçue par Maxwell (figures 3-1 et 3-2).

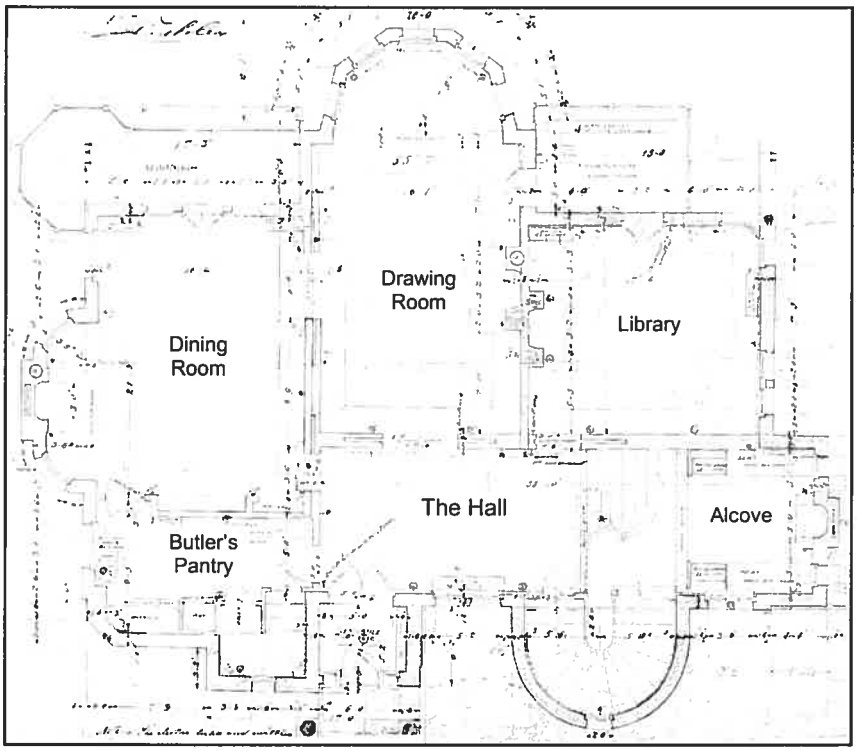

Figure 3-1 : Plan du rez-de-chaussée de la Maison Meredith

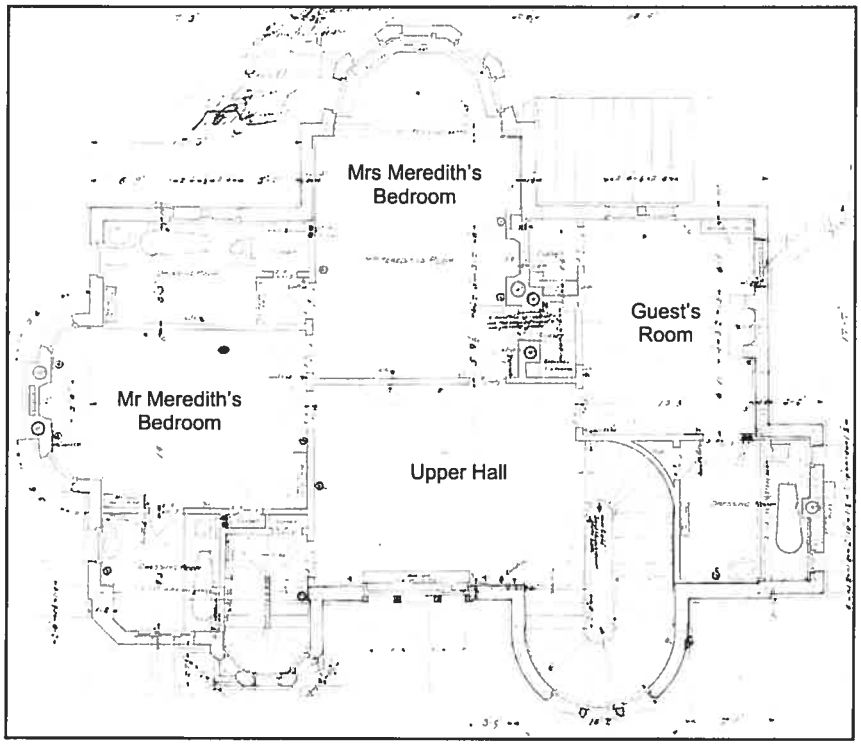

Figure 3-2: Plan du premier étage de la Maison Meredith

Au XIX^{ième} siècle, il est important de protèger la famille bourgeoise du regard des domestiques, considérés comme des intrus indispensables. Il faut reléguer ceux-ci le jour à la cuisine, le soir à l'étage supérieur, et pouvoir les appeler à distance. Couloirs et escaliers établissent donc une double circulation afin que maîtres et serviteurs ne s'entrecroisent jamais puisque les domestiques ne doivent pas apparaître publiquement lors des déplacements commandés par leurs fonctions.

Il faut également prévoir, au sous-sol ou au dernier étage, une pièce de séjour pour les domestiques (appelée "Servant's Hall") et des chambres, dont les dimensions varieront en fonction du rang de l'occupant dans la hiérarchie de la domesticité. (Voir figures 3-3 et 3-4, ci-après).

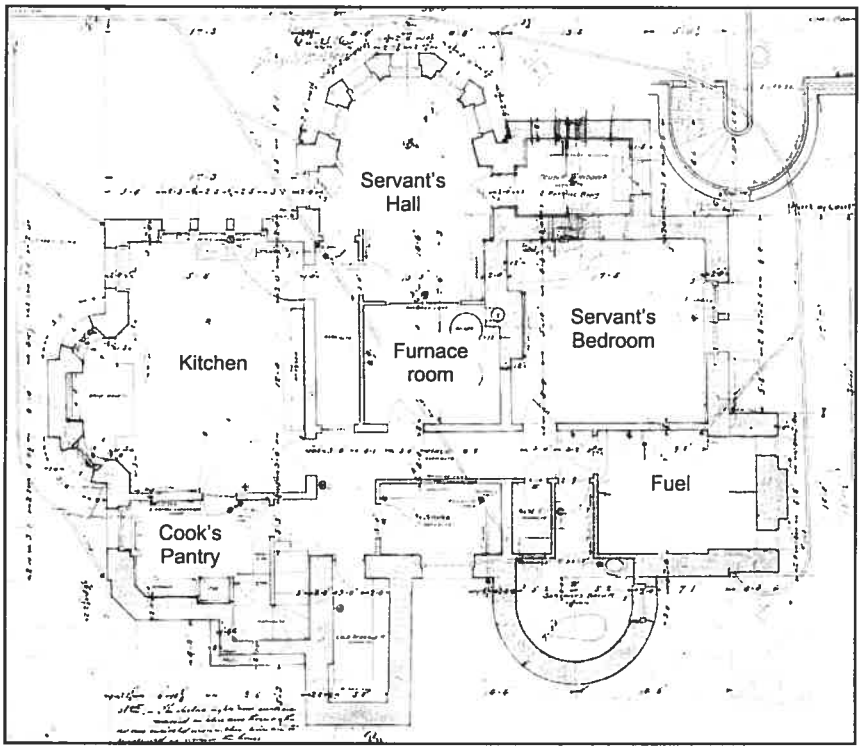

Figure 3-3 : Plan du sous-sol de la Maison Meredith

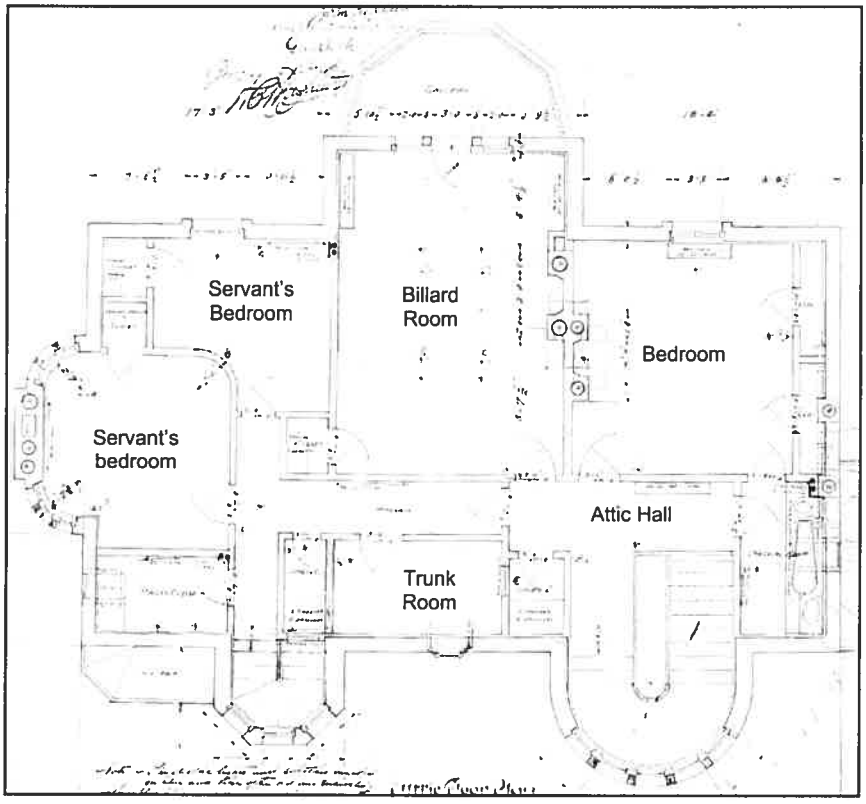

Figure 3-4 : Plan du deuxième étage (appelé "Attic Floor")

Bien évidemment, ces quelques précisions ne décrivent que de façon schématique le programme architectural d'une demeure bourgeoise du XIX^{ième} siècle. Étudier en profondeur le mode de vie de la bourgeoisie de l'époque dépasserait le cadre de cette étude. En fait, pour la modélisation de notre système générateur de solutions architecturales, nous nous contenterons, dans cette première étape, d'utiliser certaines des conventions en vigueur à l'époque pour informer le système des mitoyennetés obligatoires entre les différents espaces et des impératifs de la double circulation maîtres / serviteurs.

3.1.1.2 Le style architectural ou de la période de construction

Edward Maxwell a pratiqué sa profession durant trois époques marquées par de profondes transformations dans le domaine de l'architecture, soit la fin de l'époque victorienne, l'époque édouardienne et le début de l'époque moderne. Comme nous l'avons mentionné précédemment, nous désirons former un échantillonnage de solutions existantes d'une certaine homogénéité. Nous avons donc effectué une sélection au niveau de la date (ou période) de construction.

Nous nous sommes concentrés sur les œuvres de jeunesse d'Edward Maxwell, c'est-à-dire les années 1890. Durant cette décennie, après quelques années passées à Boston au service de la firme d'architectes Shepley, Rutan, & Coolidge, Maxwell rentre à Montréal où il ouvre son propre bureau. Son expérience bostonienne a alors contribué à en faire un adepte de l'architecture victorienne d'inspiration romane. Il démontre à cette époque un goût marqué pour l'architecture pittoresque.

Le mouvement pittoresque a pris naissance en Angleterre et a été importé au Canada par les immigrants anglais de la classe bourgeoise et de la haute

société. Ce groupe, quoiqu'il était relativement peu nombreux par rapport à la population canadienne de l'époque, a vite constitué une part importante de la clientèle de Maxwell. Concevoir les habitations de la grande bourgeoisie représentait souvent des commandes très intéressantes pour lui, car c'était cette classe sociale qui avait non seulement les ressources financières nécessaires pour exiger plus de confort, mais également les moyens culturels de souhaiter des changements et de permettre à l'architecte d'innover.

Le mouvement pittoresque, bien qu'il s'intéressa en premier lieu à l'art des jardins, n'en a pas moins exercé une profonde influence sur l'architecture au cours du XIX^{ième} siècle. Les normes abstraites, rigoureusement géométriques, de l'architecture classique ont fait place d'une part au respect de la nature et, d'autre part, à la création d'effets visuels intéressants.

Le "pittoresque" se définit comme ce qui est digne d'être peint, ce qui attire l'attention, charme ou amuse par un aspect original. En fait, à l'origine du mouvement pittoresque, on cherchait surtout à rapprocher l'aménagement paysager de la nature en recréant et en préservant les qualités visuelles qui lui sont propres : l'asymétrie, la diversité et la complexité des formes, des couleurs et des matériaux ainsi que leurs jeux de lumières et d'ombres.

C'est un mouvement qui permettait d'adopter une conception éclectique du style. Comme on le voit sur le croquis présenté à la figure 3-5, on pouvait choisir n'importe quel type d'élément ou de détail architectural pourvu qu'il charme par son aspect original.

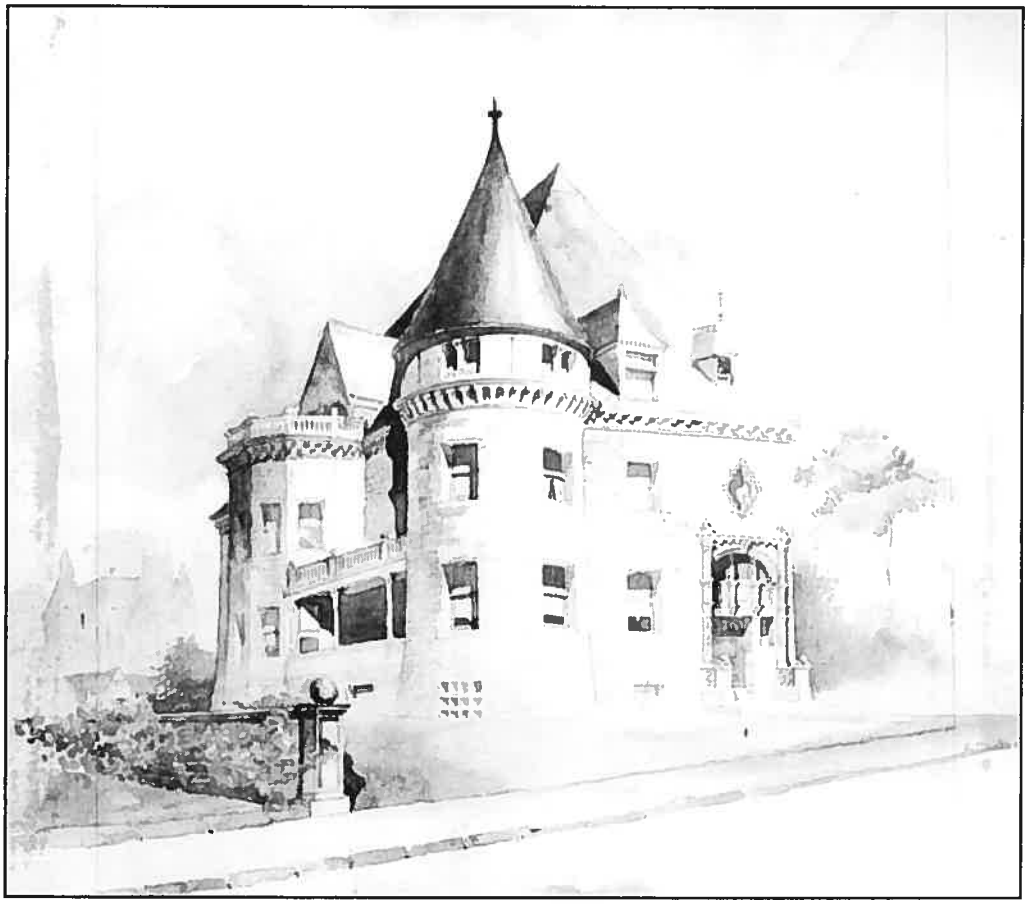

Figure 3-5 : Croquis de la maison Clouston

Parmi la production de Maxwell, nous avons choisi des bâtiments représentatifs du mouvement pittoresque parce qu'il s'agit d'un type d'architecture qui ne semble obéir à aucune règle pré-déterminée. Bien que, dans le cadre de ce travail, nous n'abordions pas l'aspect géométrique, nous gardons cependant en mémoire la possibilité de pousser plus loin notre démarche. Notre système générateur de solutions partielles produira des assemblages de modules à partir de procédures de sélection aléatoire mais il pourrait aussi, théoriquement, produire des assemblages de primitives géométriques.

Dans cette optique, il nous semble intéressant de pouvoir ultérieurement demander à l'outil informatique de nous présenter différents assemblages de formes géométriques auxquels nous n'aurions pas nécessairement pensé et qui pourraient éventuellement nous charmer par leur aspect original.

3.1.1.3 La répartition des espaces et les primitives géométriques utilisées

Edward Maxwell a construit une trentaine de maisons bourgeoises, à Montréal, à l'intérieur du Mille carré doré. Nous avons sélectionné trois maisons urbaines prestigieuses comptant parmi ses premières commandes. Il s'agit de la Maison Meredith, de la Maison Clouston et de la Maison Crathern.

La résidence Henry Vincent Meredith, dont nous avons présenté les plans à la section précédente, est située au 1110, avenue des Pins, à Montréal. Selon François Rémillard (1986), il s'agit de l'une des résidences d'Edward Maxwell les plus réussies. C'est un bâtiment de style victorien d'inspiration romane présentant de nombreux éléments pittoresques; tourelles semi-circulaires et polygonales, porche roman formé de colonnes coiffées d'un arc surbaissé, fenêtres en escaliers, cheminées néo-Tudor aux coffres multiples et aux mitrons crénelés, etc.

Cette résidence a été construite en 1894. Très représentative de la production de Maxwell durant les années 1890, elle est un bon exemple du style *château* qu'affectionnait l'architecte à son retour du séjour d'apprentissage à Boston. Ce style, qui alliait librement des éléments architecturaux du gothique tardif et de la Renaissance française (château de la Loire) était, à l'époque, très à la mode dans le nord-est des États-Unis.

Ce bâtiment a retenu notre attention entre autre parce que la palette des primitives géométriques utilisées inclut des formes aussi variées que le cylindre,

le cône, l'octogone, etc. Comme le démontrent les élévations présentées ci-après, il s'agit d'un bâtiment présentant une grande richesse au niveau géométrique.

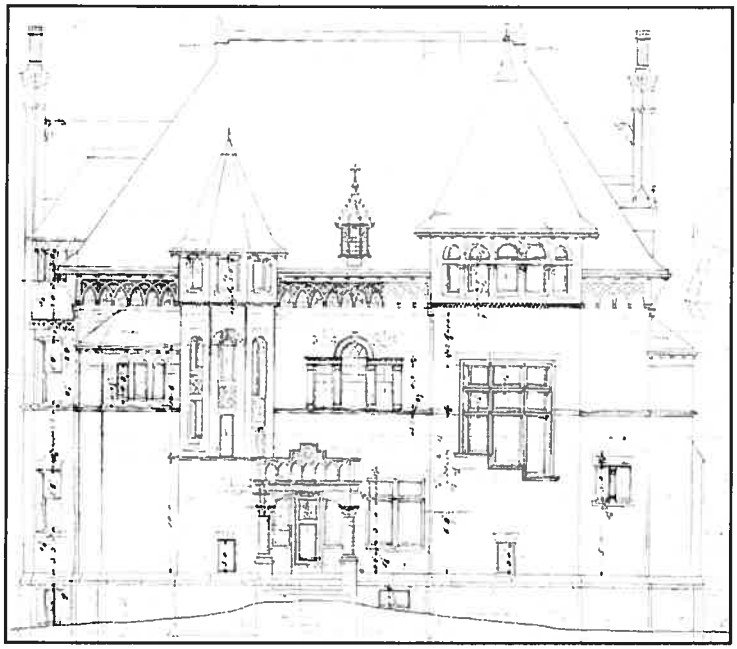

Figure 3-6 : La Maison Meredith. Élévation principale

Figure 3-7 : La Maison Meredith. Détail de l'élévation arrière

Le deuxième élément de l'échantillonnage est la Maison Edward S. Clouston. Cette résidence, malheureusement aujourd'hui démolie, était aussi appelée "La Maison du directeur général de la Banque de Montréal". Elle a été construite peu de temps après la Maison Meredith, entre 1893 et 1894. Tout comme celle-ci, il s'agit d'un exemple d'architecture pittoresque où l'on retrouve de nombreux éléments du style *château.*

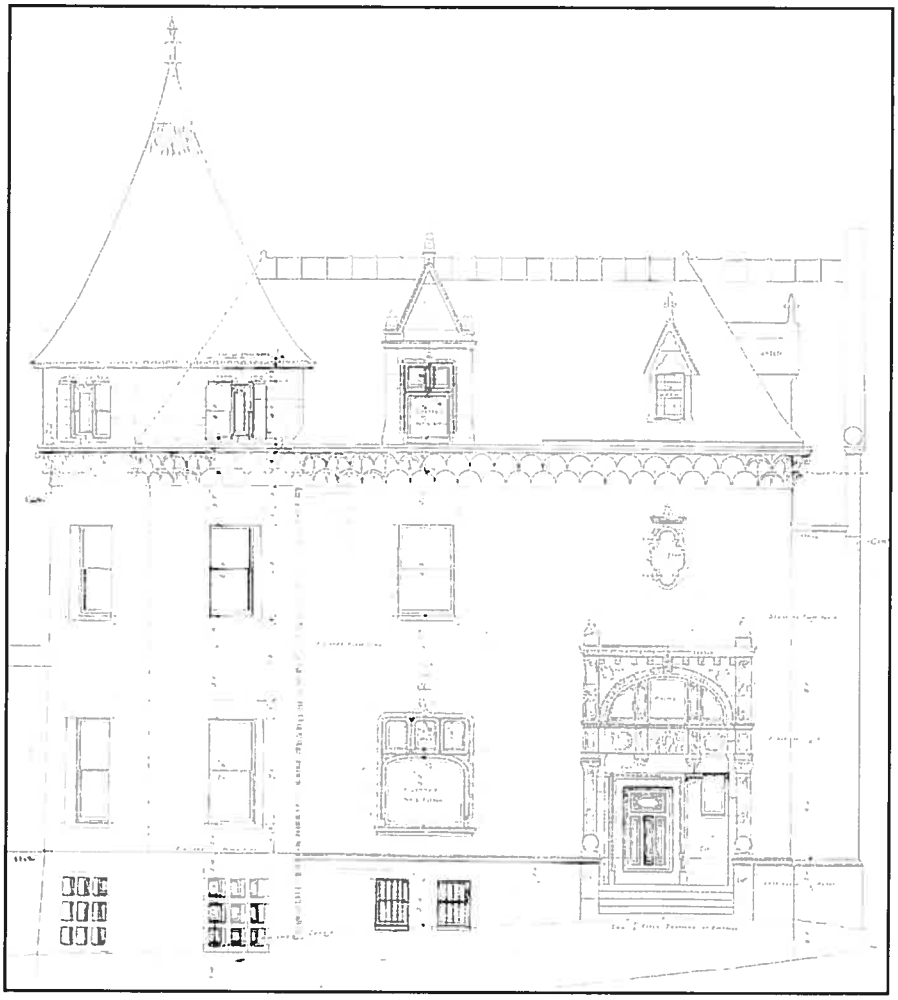

Figure 3-8 : La Maison Clouston. Élévation principale

Cette résidence à trois étages était divisée comme la plupart des grandes maisons urbaines victoriennes. Comme l'indiquent les plans présentés ci-

dessous, les logements des domestiques et les cuisines étaient placés au sous-sol. Les pièces d'apparat, le salon, la salle à manger et la salle de billard étaient disposées autour d'un grand hall central, au rez-de-chaussée. Les chambres à coucher principales et les pièces privées de la famille se trouvaient au premier étage. D'autres chambres à coucher ainsi que des pièces de rangement formaient le deuxième étage.

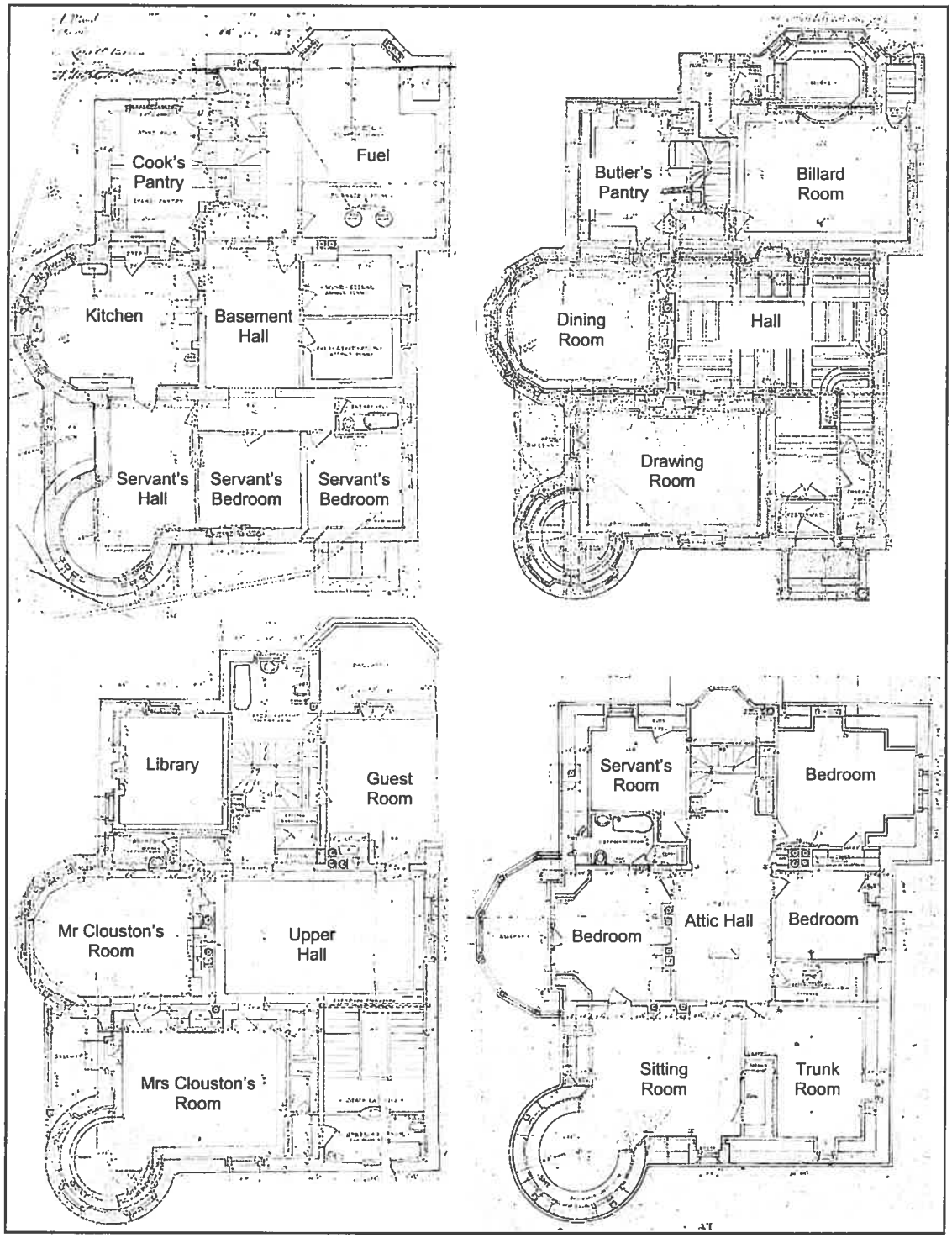

Figure 3-9 : La Maison Clouston. Plans des étages

Le troisième élément de l'échantillonnage est la Maison James Crathern. Elle est située au 1572, avenue du Docteur Penfield, Montréal. Il s'agit d'une demeure relativement plus "modeste" que les deux précédentes. Tout comme les deux autres, elle est compacte et massive, puisque construite en milieu urbain. Cependant, bien qu'il s'agisse d'une résidence uni-familiale du type maison bourgeoise, elle est analogue aux maisons en rangées dont la façade étroite donne sur la rue.

Construite entre 1892 et 1893, elle aussi représente un bon exemple de l'architecture victorienne d'inspiration romane. Elle est donc le fruit de la même source d'inspiration que la Maison Meredith. Comparativement aux deux autres résidences de l'échantillonnage, on retrouve sensiblement la même palette des primitives utilisées (le style *château*) et la même disposition des espaces de la maison bourgeoise; pièces principales autour du hall, chambres à l'étage et logements des domestiques au sous-sol.

Figure 3-10 : La Maison Crathern. Élévation principale

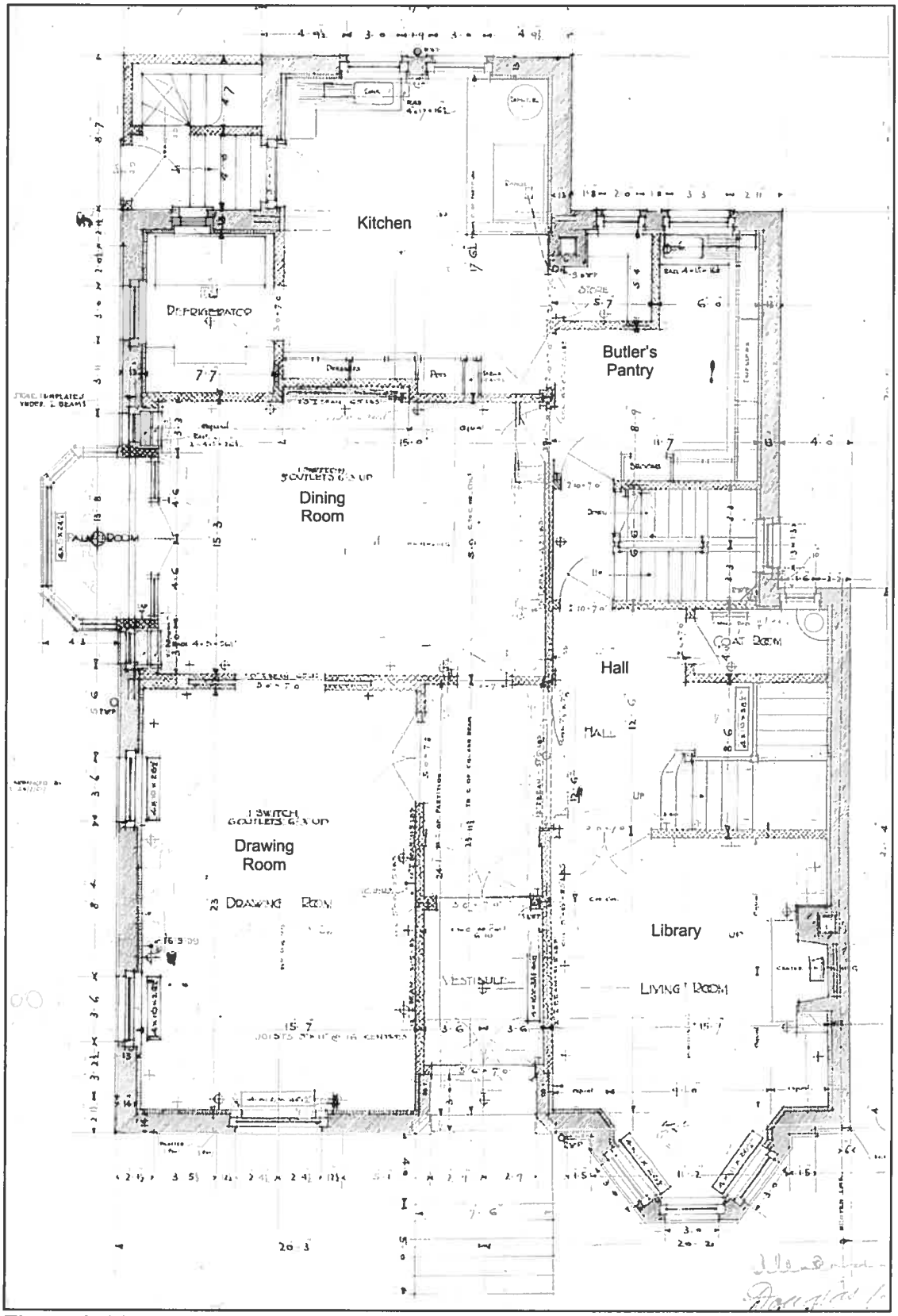

Figure 3-11 : La Maison Crathern. Plan du rez-de-chaussée

Les trois demeures bourgeoises que nous avons sélectionnées se prêtent particulièrement bien à une étude de cas comme celle que nous voulons entreprendre. C'est la complexité de ces bâtiments, au niveau de la géométrie et des interrelations entre les espaces, qui a retenu notre attention. Tout comme Flemming (1987), dans sa recherche portant sur la génération de maisons de style Queen Anne, nous considérons qu'en nous confrontant à un type d'architecture aussi complexe, nous pouvons assurer une validité externe à notre travail, c'est-à-dire élaborer des stratégies qui pourront fonctionner par la suite pour la génération de bâtiments moins complexes.

3.2 Délimitation de l'espace de conception

La délimitation d'un espace de conception peut regrouper une grande quantité d'informations en terme de types d'espaces, règles de proximité, types de jonctions entre les espaces, orientations des espaces, schémas de distribution, dimensionnements, géométries, etc.

Afin d'aborder progressivement la complexité, nous nous limiterons, dans cette première partie de l'exposé, à ne traiter que trois aspects. Premièrement, nous identifierons les différents types d'espaces (ou modules) avec lesquels nous travaillerons. Deuxièmement, nous déterminerons quels sont les couples de modules dont la proximité horizontale est obligatoire et quelles sont les listes de modules dont la proximité verticale est obligatoire. Troisièmement, nous déterminerons quels sont les différents schémas de distribution des espaces avec lesquels le système devra travailler pour proposer des solutions architecturales partielles.

3.2.1 Identification des espaces

Dans le cadre de notre étude de cas, les éléments de notre échantillonnage sont tous des bâtiments à quatre étages. On retrouve le "Basement" (sous-sol), le "Ground Floor" (rez-de-chaussée), le "First Floor" (premier étage) et le "Attic Floor" (étage des mansardes).

Suite à l'analyse des différents plans, nous pouvons dresser la liste de tous les espaces susceptibles de se retrouver à chacun de ces quatre étages. À chacun de ces espaces, nous assignons ensuite un code numérique. Certains des espaces apparaissent à différents étages, en fonction du bâtiment observé.

Ainsi, le même espace peut apparaître à deux reprises mais avec des codes numériques différents.

Pour la modélisation de notre système générateur de solutions architecturales, nous avons à travailler avec les niveaux et les listes d'espaces que nous avons codés numériquement de la façon suivante :

Niveau 0 - Basement

0 Hall-ss
1 Kitchen
2 Cook's pantry
3 Servant's hall
4 Servant's bedroom (avec Bathroom partagé)
5 Servant's bedroom (avec Bathroom partagé)
6 Store (+ Cold strorage + Wine cellar)
7 Furnace room (+ Fuel)

Niveau 1 – Ground Floor

10 Hall (incluant vestibule, escalier principal et alcôve selon le cas)
11 Drawing room
12 Dining room
13 Butler's pantry (incluant coat-room et lavatory)
14 Kitchen
15 Billard room
16 Library
17 Conservatory (ou loggia)

Niveau 2 – First Floor

20 Upper Hall
21 Mrs's room + dressing room
22 Mr's room + dressing room
23 Guest's room + dressing room
24 Bedroom1 (bathroom partagé)
25 Bedroom2 (bathroom partagé)
26 Library
27 Sitting room

Niveau 3 – Attic Floor

30 Hall attic

31 Trunk room
32 Bedroom + dressing room
33 Bedroom1 (bathroom partagé)
34 Bedroom2 (bathroom partagé)
35 Sitting room
36 Billard room
37 Servant's room
38 Servant's room

Autre

100 Extérieur

3.2.2 Proximités obligatoires

En analysant les éléments de l'échantillonnage, nous pouvons identifier, pour chacun des niveaux, les proximités obligatoires entre les espaces. Par exemple, au rez-de-chaussée, il est obligatoire que le "Drawing Room" et le "Hall" soient mitoyens. À l'étage, "Mr's Room" doit obligatoirement communiquer avec "Mrs's Room", et ainsi de suite.

Nous pouvons ainsi former une série de couples de mitoyennetés obligatoires du type "Drawing Room - Hall" (11 10), "Mr's Room – Mrs's Room" (21-22), etc. en utilisant les codes numériques déterminés précédemment. Ces informations vont pouvoir être transmises au système sous forme d'une série de matrices carrées mettant, pour chaque étage, les espaces en relation avec eux-mêmes et indiquant les mitoyennetés obligatoires sur le plan horizontal.

Dans une phase ultérieure de la recherche, d'autres matrices du même type pourraient fournir au système des informations supplémentaires. Par exemple, elle pourraient indiquer quelles sont les dimensions admissibles pour chacun des modules et quels sont les types de jonctions admissibles pour chacun des couples de modules. Quel que soit le type d'information fourni au système, nous appellerons ces matrices "*matrices de restriction*".

L'intérêt de présenter ces informations sous forme matricielle réside dans le fait que nous pouvons regrouper les données de la façon la plus compacte possible, ce qui permettrait éventuellement à l'utilisateur de modifier lui-même l'espace de conception. Par exemple, il serait intéressant qu'il puisse faire varier les proximités qu'il souhaite voir obligatoires, de façon à tester l'impact de ces modifications sur les solutions architecturales proposées par le système. C'est pourquoi nous cherchons à délimiter notre espace de conception de façon à ce que la saisie de données soit la plus simple possible.

Le code de lecture des matrices de restrictions des proximités pourrait être : -1= défendu, 0 = permis et 1 = obligatoire. Cependant, en analysant notre échantillonnage de bâtiments, nous réalisons le fait suivant : Bien qu'il existe des couples d'espaces ne devant jamais communiquer entre eux (comme la cuisine et la salle-à-manger), la mitoyenneté entre ces espaces n'est pas interdite. Nous n'utiliserons donc pas le code –1= défendu.

En ce qui a trait aux proximités obligatoires, notons qu' il peut arriver, dans certains cas, que l'on puisse avoir le choix entre deux couples d'espaces. Par exemple, à l'étage, si on retrouve parmi les pièces le "Sitting Room" (#27), celui-ci peut donner soit sur le "Upper Hall" (#20), soit sur "Mrs's Room" (#21), ou encore sur les deux. On dira alors pour les couples (27 20) et (27 21) qu'il s'agit de proximités partiellement obligatoires. Dans des cas semblables, nous complexifions le code de lecture de la matrice de proximités en ajoutant d'autres valeurs numériques (2, 3 ou 4).

Dans la matrice présentée ci-dessous, les couples ayant le code 1 représentent l'ensemble des proximités absolument obligatoires. Le chiffre 2 sert à regrouper les couples de proximité partiellement obligatoire "Mrs's Room – Sitting Room" et "Upper Hall – Sitting Room". Le chiffre 3 sert à regrouper les couples de proximité partiellement obligatoire "Upper Hall-Bedroom1" et "Upper-

Hall-Bedroom2" car ces deux chambres peuvent être desservies par un espace de service commun.

	20	21	22	23	24	25	26	27	100
20	0	**1**	**1**	**1**	**3**	**3**	**1**	**2**	0
21	**1**	0	**1**	0	0	0	0	**2**	**1**
22	**1**	**1**	0	0	0	0	0	0	**1**
23	**1**	0	0	0	0	0	0	0	**1**
24	**3**	0	0	0	0	0	0	0	**1**
25	**3**	0	0	0	0	0	0	0	**1**
26	**1**	0	0	0	0	0	0	0	**1**
27	**2**	**2**	0	0	0	0	0	0	**1**
100	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0

Figure 3-12 : Matrice de restriction des proximités horizontales pour le premier étage

Comme nous avons pris le parti d'utiliser le langage de programmation Scheme, nous pouvons alors transcrire la matrice de restriction des mitoyennetés horizontales en une liste de listes, comme suit :

```
((0 1 1 1 3 3 1 2 0)
(1 0 1 0 0 0 0 2 1)
(1 1 0 0 0 0 0 0 1)
(1 0 0 0 0 0 0 0 1)
(3 0 0 0 0 0 0 0 1)
(3 0 0 0 0 0 0 0 1)
(1 0 0 0 0 0 0 0 1)
(2 2 0 0 0 0 0 0 1)
(0 1 1 1 1 1 1 1 0))
```

En fournissant cette liste de listes à une procédure récursive, nous obtenons les listes de tous les couples de proximité ayant le même code. Nous avons ici :

Code #1 : (20 21) (20 22) (20 23) (20 26) (21 22) (21 100) (22 100) (23 100) (24 100) (25 100) (26 100) (27 100)

Code #2 : (20 27) (21 27)

Code #3 : (20 24) (20 25)

Lorsque le système proposera une solution pour un niveau, il dressera ensuite une liste de tous les couples de proximité présents dans cette solution. Afin de valider ou d'invalider cette solution partielle, il procédera en deux étapes. D'une part, il vérifiera si toutes les proximités obligatoires (ici codées #1) sont présentes. D'autre part, il vérifiera si au moins un des éléments de chacun des couples de couples de proximités partiellement obligatoire (ici codés #2 et #3) est présent dans la liste des couples de proximités de la solution proposée.

Les proximités obligatoires sur le plan vertical seront, quant à elles, indiquées au moyen de listes. Le système travaillera avec une liste de trois éléments pour positionner l'escalier principal afin de relier verticalement le rez-de-chaussée, l'étage et le "attic floor". La liste (10 20 30) (Hall Upper-Hall Attic-Hall) ne pourra pas être modifiée puisqu'il s'agit de trois espaces obligatoires.

Par contre, en ce qui a trait à l'emplacement de l'escalier de service, le système générera une liste en fonction de la composition des étages de la solution générée. Par exemple, entre le sous-sol et le rez-de-chaussée, nous aurons le couple Cook's pantry – Butler's pantry (2 13) ou le couple Servant's hall – Butler's pantry (3 13) dépendamment de l'emplacement de la cuisine, au sous-sol ou alors au rez-de-chaussée. À ce couple viendra s'ajouter le code 31 symbolisant le "Trunk Room" au dernier étage. Ces listes, (2 13 31) ou (3 13 31), indiqueront au système comment superposer les modules de façon adéquate lors de la génération de la matrice 3D.

Les listes de modules, les matrices de restriction des proximités horizontales et les listes de proximités verticales seront les éléments qui nous serviront, dans cette première version du système, à valider ou invalider une solutions architecturales partielles.

3.2.3 Matrices de répartition des espaces

Le troisième type d'information que nous fournirons au système pour délimiter l'espace de conception concerne les différents types d'assemblages 2D qui seront à la base de la génération de la matrice 3D. Nous fournirons au système un ensemble de schémas qui seraient autant de façons différentes de mettre des espaces en relation.

Lorsque viendra le temps de générer une matrice 2D, le système pourra choisir de façon aléatoire un schéma de distribution des espaces à l'intérieur de cet ensemble de schémas potentiels et lui appliquer la liste de modules. Il testera cet assemblage pour déterminer s'il est conforme à la matrice de restriction des proximités horizontales, dans lequel cas il servira à la génération de la matrice 3D.

Si, au contraire, il n'est pas conforme, un autre schéma de répartition des espaces sera alors testé. Et ainsi de suite, jusqu'à l'identification d'un schéma acceptable qui, lui, sera utilisé pour la génération de la solution. Pour chacun des niveaux, nous devrons donc tester plusieurs schémas de distribution avant d'arriver à une solution acceptable qui pourra être proposée à l'utilisateur.

Quels schémas de distribution fournirons-nous au système? Il existe bien évidemment une infinité de schémas pouvant représenter la façon de mettre des espaces en relation. Dans le but de simplifier notre démarche, plutôt que de

chercher à imaginer quels pourraient être ces schémas, nous avons choisi de nous inspirer des diagrammes présentés par W. J. Mitchell (1987) dans son ouvrage "Computer Aided Architectural Design". (Voir figure 3-13.)

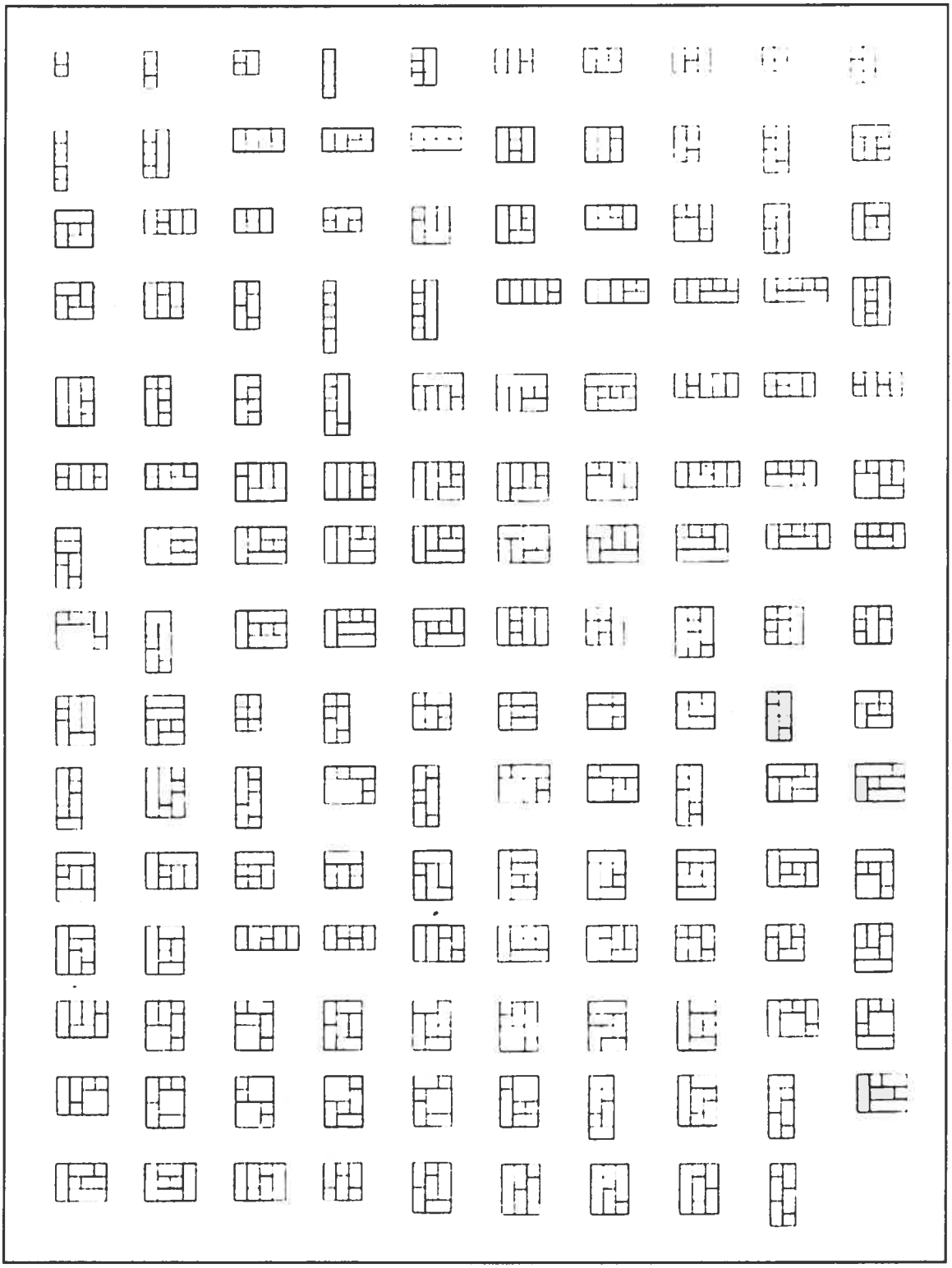

Figure 3-13 : Les plans schématiques en "dimensionless representation"

Ces schémas représentent des plans d'étages en "dimensionless representation". Cette approche consiste à représenter schématiquement le bâtiment en ramenant toute les dimensions, ou largeurs de travées, à l'unité. Elle prend, bien sûr, le parti réducteur de représenter tout espace par un carré ou un rectangle mais offre une certaine richesse, puisqu'elle nous permettra de traduire les schémas dans un langage interprétable par l'ordinateur.

Mitchell nous présente, de façon schématique, toute une série de plans d'étage en *dimensionless representations*. Il nous propose 22 façons de diviser un espace rectangulaire en 5 portions (excluant l'enfilade simple) et 115 façons de le diviser en 6. Étant donné que le nombre de pièces que l'on retrouve habituellement par étage dans une maison bourgeoise est de 5 ou 6, nous avons choisi de nous inspirer de ces plans schématiques pour fournir au système des informations sur la façon dont il pourra mettre les modules en relation pour un niveau donné.

Nous traduirons ces schémas sous forme matricielle afin qu'elles puissent être utilisées pour la génération de la matrice 3D. Nous appellerons ces matrices "matrice de répartition des espaces".

Pour traduire les schémas sous forme matricielle, nous prenons un schéma de subdivision et nous numérotons les espaces en allant de gauche à droite et de haut en bas. Nous construisons ainsi une matrice avec les modules 1, 2, 3, 4 et 5 ou avec 1, 2, 3, 4, 5 et 6, selon le cas. Chacune de ces matrices de répartition nous servira ensuite à générer plusieurs autres matrices. Illustrons notre propos à l'aide d'un exemple simple. Pour représenter un des schémas de distribution, les modules 1, 2, 3, 4, 5 et 6 pourraient, par exemple, être assemblés de façon à former la matrice suivante :

1 1 2 3
4 5 5 6.

Ces chiffres symbolisent le premier module de la liste, le deuxième module, le troisième, etc. Nous pouvons par la suite intervertir l'ordre des modules de toutes les façons possibles. Par exemple :

3 3 6 4 2 2 5 1 6 6 3 5
1 2 2 5 3 4 4 6 2 1 1 4.

Pour chacun des 22 schémas de division en 5 espaces, nous aurons (1 X 2 X 3 X 4 X 5) 120 possibilités, donc (22 X 120) 2 640 matrices de distribution. Pour chacun des 115 schémas de division en 6 espaces, nous aurons (1 X 2 X 3 X 4 X 5 X 6) 720 possibilités, donc (115 X 720) 82 800 matrices de distribution.

Pour générer ces ensembles de matrices de répartition des espaces, nous avons mis à contribution les avantages offerts par les fonctions récursives du langage de programmation Scheme. Bien entendu, à aucun moment l'utilisateur du système ne verra énumérées ces nombreuses matrices. Il s'agira plutôt d'un ensemble de solutions potentielles. Nous verrons, dans la prochaine section, comment le système pourra effectuer une sélection aléatoire à l'intérieur de cet ensemble, tester cette solution afin d'en déterminer la validité et, le cas échéant, s'en servir pour la génération de la matrice 3D.

3.3 Génération de la matrice 3D

3.3.1 Sélection des espaces

Pour générer une solution architecturale, la première étape est de choisir, parmi les modules identifiés dans l'espace de conception, ceux que nous désirerons combiner. Étant donné que, dans cette première version du système générateur de solutions architecturales, l'utilisateur ne peut indiquer ses préférences, c'est le système qui proposera une liste de modules.

Pour chacun des niveaux, la liste des pièces que nous avons identifiée précédemment est partagée en deux sous-listes. Nous regroupons d'une part les espaces obligatoires et, d'autre part, les espaces optionnels parmi lesquels le système effectuera ultérieurement un choix.

Ces listes d'espaces obligatoires et d'espaces optionnels sont fournies à une procédure Scheme (voir annexe, section 6) dont la fonction est de générer, pour chacun des niveaux, une liste provisoire de pièces regroupant l'ensemble des espaces obligatoires et une sélection aléatoire d'espaces optionnels. Par exemple, pour le rez-de-chaussée, la liste des espaces obligatoires est (10 11 12 13) et celle des espaces optionnels est (14 15 16 17). Ainsi, pour la génération d'une solution architecturale, le système pourrait proposer la liste (10 11 12 13 16) pour les pièces du rez-de-chaussée.

Nous avons défini le système de façon à ce que les listes d'espaces comprennent 5 ou 6 pièces pour chaque niveau. C'est la procédure génératrice de listes provisoires qui déterminera de façon aléatoire si un étage regroupe 5 ou alors 6 espaces. Notons que, dans le processus de génération d'une solution architecturale, le fait qu'un niveau compte un nombre "x" d'espaces n'implique pas que les autres niveaux auront obligatoirement ce même nombre d'espaces.

Ce type d'approche, bien que complexifiant sensiblement le système, cherche à tenir compte le mieux possible de l'analyse de l'échantillonnage et à ne pas imposer de restrictions superflues puisque, dans les bâtiments analysés, le nombre de pièces varie d'un étage à l'autre.

3.3.2 Validation des listes provisoires

Nous identifions un étage que nous considérerons comme niveau principal. Dans notre étude de cas portant sur la maison bourgeoise, l'étage le plus important est le rez-de-chaussée. La première liste que nous générerons sera donc celle de ce niveau.

Idéalement, le système que nous modélisons serait interactif. Il serait alors possible pour l'utilisateur d'effectuer des choix concernant la composition et l'organisation de l'étage principal. Les choix qu'il ferait, dès la première étape du processus, lui permettraient d'orienter selon ses préférences l'élaboration de la solution architecturale.

Cependant, dans cette première ébauche du système, la solution architecturale est élaborée de façon automatique, sans l'intervention de l'utilisateur. Le système générera une liste d'espaces pour le rez-de-chaussée qui sera considérée comme liste définitive et qui servira de repère pour la validation des listes d'espaces pour les autres étages. Celles-ci seront générées de façon aléatoire et transmises à des procédures dont la fonction sera de vérifier s'il n'y a pas de dédoublements d'espaces. Ces procédures sont présentées en annexe, section 7.

Nous générons une liste provisoire d'espaces pour le sous-sol. Une procédure Scheme effectue ensuite les vérifications suivantes. Puisque "Kitchen" doit obligatoirement faire partie des pièces de la maison mais ne doit évidemment pas se retrouver à plusieurs étages à la fois, s'il y a "Kitchen" (codé #14) dans la liste du rez-de-chaussée, il ne doit y avoir ni "Kitchen" (codé #1), ni "Cook's Pantry" (codé #2) dans la liste du sous-sol. Inversement, s'il n'y a pas "Kitchen" au rez-de-chaussée, il doit obligatoirement y avoir "Kitchen" et "Cook's Pantry" au sous-sol.

La récursivité nous permet de poser une action, d'effectuer un test sur le résultat de cette action et de reprendre ce processus tant que le test n'est pas positif. Nous avons donc défini une procédure récursive qui nous permet de générer une nouvelle liste provisoire d'espaces au sous-sol, tant et aussi longtemps que les conditions décrites ci-dessus ne sont pas remplies. Ce n'est que lorsque nous avons rempli ces conditions que la liste provisoire des espaces du sous-sol est considérée comme étant définitive.

Pour obtenir la liste des espaces à l'étage, nous générons une fois encore une liste provisoire. Nous comparons ensuite cette liste provisoire avec la liste définitive du rez-de-chaussée puisque "Library" (codé #16 au rez-de-chaussée et #26 à l'étage) est un espace optionnel qui ne doit apparaître qu'à un seul étage. Nous devons donc le retrouver dans une des deux listes ou dans aucune des deux mais jamais dans l'une et l'autre à la fois.

Là encore, une procédure récursive nous permet d'identifier pour l'étage une liste d'espaces conforme aux spécifications. Cette liste sera considérée comme définitive et sera utilisée pour la validation d'une liste d'espaces pour le "attic floor". À ces niveaux, nous avons des espaces optionnels ("Sitting Room" et "Billard Room") qui ne doivent pas apparaître aux deux étages en même temps.

Les listes d'espaces sont donc générées successivement pour chacun des étages. Lorsque nous avons terminé de générer ces quatre listes. Nous venons joindre à chacune la liste des couples de proximités obligatoires applicables. Pour ce faire, chacune de ces listes d'espaces définitives est comparée avec la liste de proximités obligatoires et partiellement obligatoires de l'étage correspondant. Le système peut ainsi générer une liste de proximités obligatoires applicables à cette liste de pièces. En effet, si dans notre liste définitive de pièces pour le rez-de-chaussée, nous ne retrouvons pas de "Library", il n'y a plus lieu de tenir compte de la proximité obligatoire "Library-Hall" pour la génération de cette solution architecturale.

Après lui avoir attribué une liste de proximités applicables, nous faisons subir un deuxième traitement à la liste d'espaces définitive. Ce traitement consiste à mélanger l'ordre des modules de façon aléatoire à l'aide d'une procédure récursive. Comme nous le verrons dans la section suivante, cette liste ainsi mélangée pourra ensuite être appliquée à l'une des listes de répartition.

3.3.3 Listes de répartition

Lorsque nous avons délimité notre espace de conception, nous avons fourni au système, traduites sous forme matricielle, les informations contenues dans les 137 diagrammes présentés par Mitchell.

Voyons maintenant comment nous procèdons pour traduire ces matrices en des listes qui pourront être utilisées dans une procédure Scheme. Prenons par exemple la matrice 3X3 suivante :

1	2	3
4	4	3
4	4	5.

Notons que, dans cette matrice, les chiffres 1, 2, 3, 4 et 5 servent respectivement à représenter le premier, le deuxième, le troisième, le quatrième et le cinquième espace de la liste des pièces dont l'ordre a été mélangé.

Nous transformons la matrice en une liste dont le premier élément est la liste des chiffres qui forment la matrice et le deuxième élément, la description de la dimension de la matrice (qui pourra être 3x3, 4x3, 3x4, 3x2, 2x3, 4x2, 2x4, 5x2 ou 2x5). Pour l'exemple illustré ci-dessus, nous fournirons donc au système la liste suivante (qui sera appelée "liste de répartition") :

((1 2 3 4 4 3 4 4 5) "3X3").

Dans la procédure présentée en annexe (section 9), nous avons ainsi décrit les 22 diagrammes de subdivisions en 5 espaces et les 115 diagrammes de subdivision en 6 espaces sous forme d'une liste de listes que nous appellerons "liste complète des listes de répartition".

Nous remarquons ici qu'en procédant de cette façon, il existera pratiquement cinq fois plus d'alternatives pour la subdivision en 6 portions que pour la subdivision en 5 portions. Étant donné que, dans la procédure de génération des listes de pièces, les probabilités de générer une liste de 5 espaces sont égales aux probabilités de générer une liste de 6 espaces, il nous apparaît souhaitable de fournir au système des quantités semblables de possibilités de subdivisions en 5 et en 6 sous-espaces.

En effet, prenons par exemple le cas où la liste des espaces du rez-de-chaussée compte 6 espaces et où le système a sélectionné pour cet étage une matrice de répartition 4x3, parmi un choix d'une vingtaine de matrice de même taille. Lorsque nous souhaitons générer la matrice de distribution de l'étage supérieur (voir annexe, section 16), nous devons obligatoirement choisir une matrice de répartition de même taille. Si la liste des pièces à l'étage compte 5 espaces et que pour ce nombre d'espaces le système n'a le choix qu'entre 2 ou

3 matrices 4x3, il est probable que la procédure ne parvienne pas à trouver une répartition des espaces qui soit conforme aux proximités horizontales obligatoires à l'étage. L'élaboration de cette solution architecturale serait alors stoppée et le système devrait reprendre le processus depuis le début.

La solution que nous proposons à ce problème est la suivante. À partir de chaque diagramme de subdivisions en 5 sous-espaces, nous dérivons cinq listes de répartition en dédoublant à tour de rôle la première rangée, la première colonne, la dernière rangée et la dernière colonne de façon à obtenir une variété de matrices 5 espaces approchant celle des matrices 6 espaces et d'obtenir une certaine uniformité dans les dimensions des matrices.

Reprenons l'exemple illustré ci-dessus. À partir de la matrice 3X3 suivante :

1 2 3
4 4 3
4 4 5,

nous dériverons les quatre matrices supplémentaires suivantes, de tailles 3x4 et 4x3 :

1 2 3	1 1 2 3	1 2 3	1 2 3 3
1 2 3	4 4 4 3	4 4 3	4 4 3 3
4 4 3	4 4 4 3	4 4 5	4 4 5 5
4 4 5		4 4 5	

Nous réduisons ainsi la disparité entre le nombre de matrices 5 espaces et le nombre de matrices 6 espaces puisque nous avons maintenant 5x22 matrices 5 espaces (110) par rapport à 115 matrices 6 espaces.

3.3.4 Tests de concordance

Au moment de générer la matrice 2D par exemple du rez-de-chaussée, le système sélectionne aléatoirement une liste de répartition et lui applique la liste des espaces dont l'ordre a été mélangé. Nous appellerons la matrice ainsi générée "matrice de répartition des espaces".

Le système effectue ensuite des tests à savoir si cette solution potentielle est valide (voir annexe, section 14). Il est clair qu'une matrice sélectionnée de façon aléatoire ne constituera pas à coup sûr une solution acceptable. En effet, puisque dans le type de solution architecturale que nous cherchons à générer, en l'occurrence la maison bourgeoise, la mitoyenneté du module 10 (le hall) et du module 11 (le "Drawing Room") est nécessaire, la solution partielle symbolisée par la matrice suivante ne serait alors pas valable.

13	⑩	15
12	12	15
12	12	⑪

Le système effectuera donc des tests en se référant à la matrice de restriction des proximités horizontales afin d'être en mesure d'identifier une matrice de distribution des espaces qui soit valide. La matrice de répartition des espaces sélectionnée sera gardée en mémoire afin d'être utilisée pour la génération de la matrice 3D.

Bien que dans le cadre de ce mémoire, nous nous limitions à n'effectuer que des tests à partir des matrices de restriction des proximités, rappelons que notre but serait d'arriver à modéliser ultérieurement un système qui se baserait sur un nombre toujours accru de matrices de restriction en vu de concevoir des solutions architecturales progressivement de plus en plus détaillées.

3.3.5 Génération de la matrice de répartition pour le rez-de-chaussée

Comme nous l'avons vu précédemment, le système commence par générer la liste des modules pour le rez-de-chaussée et la liste des proximités obligatoires pour cet ensemble de modules. Ensuite, le système sélectionne de façon aléatoire (mais tenant compte du nombre d'espaces) une liste de répartition dans la liste complète des listes de répartition.

À partir de cette liste de répartition et de cette liste de modules, une matrice de répartition est générée. Le système attribue un module à chacune des cases de la matrice selon l'ordre de la liste mélangée et indique la position de l'espace extérieur (codé #100). Par exemple, la liste de répartition

((1 2 3 4 5 3 4 5 3) "3x3"))

et la liste de pièces

(13 11 15 12 10)

donneront la matrice de répartition suivante pour le rez-de-chaussée :

100	100	100	100	100
100	13	11	15	100
100	12	10	15	100
100	12	10	15	100
100	100	100	100	100.

Cette matrice devra être testée pour en vérifier la validité. Une procédure (présentée en annexe, section 13) dressera la liste des jonctions des espaces entre eux et avec l'extérieur. Par exemple, dans le cas illustré ci-dessus, nous aurions la liste de couples de jonctions suivante :

((100 13) (100 11) (100 15) (100 13) (13 11) (11 15) (15 100) (13 12) (11 10) (15 15) (100 12) (12 10) (10 15) (15 100) (12 12) (10 10) (15 15) (100 12) (12 10) (10 15) (15 100) (12 100) (10 100) (15 100)).

Cette liste, que nous appellerons "liste des jonctions de la matrice de répartition", est ensuite comparée avec la *liste de proximités obligatoires* que nous avions générée précédemment pour la liste de pièces. Une procédure prédicat (c'est-à-dire retournant le résultat "vrai" ou "faux") vérifie alors si chacun des couples de proximité de la liste des proximités obligatoires se retrouve dans la liste de jonctions de la matrice de répartition.

Cette procédure prédicat (présentée en annexe, section 14), nous indique si la matrice de répartition proposée est acceptable pour le rez-de-chaussée, dans lequel cas elle sera conservée et utilisée ultérieurement. Si, au contraire, la procédure prédicat nous retournait le résultat "faux", l'ordre de la liste des pièces serait à nouveau mélangé, de façon à générer une nouvelle liste de proximités des espaces entre eux et avec l'extérieur pour une même liste de répartition.

Cette vérification se fait par le biais d'une procédure récursive (présentée en annexe, section 15). Dans le cas où il n'existerait pas de solution pour une certaine liste de modules avec une certaine liste de répartition, nous ne voulons évidemment pas que la procédure refasse la boucle à l'infini. Nous devons donc spécifier, à l'intérieur de la procédure quel est le nombre maximal de tests que nous souhaitons effectuer avec les mêmes listes. Prenons le parti de fixer ce nombre à 10.

Si après une dizaine de tentatives, la fonction prédicat retourne toujours "faux", une nouvelle liste de répartition est sélectionnée aléatoirement par le système, toujours en tenant compte du nombre d'espaces de la liste de pièces. Avec cette nouvelle liste de répartition, nous pouvons tester de nouvelles listes de jonctions de la matrice de répartition, et ce, en mélangeant de nouveau l'ordre de la liste de pièces de dix façons différentes.

Si après avoir sélectionné successivement dix listes de répartition (et donc avoir effectué une centaine de test), le résultat de la procédure prédicat est toujours "faux", le système génère une nouvelle liste de pièces pour le rez-de-chaussée et reprend le processus tant et aussi longtemps que la procédure prédicat ne retournera pas le résultat "vrai".

Lorsque nous obtenons le résultat "vrai", la matrice de répartition des espaces pour le rez-de-chaussée est identifiée. Nous gardons en mémoire cette matrice, ainsi que la liste de répartition, la liste des pièces mélangées et le code de grosseur de la matrice. Ces informations seront utilisées pour la génération / validation des matrices de répartition du sous-sol et de l'étage.

3.3.6 Génération de la matrice de répartition du sous-sol

Pour la génération de la matrice de répartition du sous-sol, nous devons tenir compte de la dimension de la matrice du rez-de-chaussée et des contraintes imposées par la circulation verticale.

Nous commençons par générer la liste des pièces du sous-sol et la liste des proximités obligatoires pour cet ensemble de pièces. Ensuite, une liste de répartition est sélectionnée par le système. Cependant, cette fois la sélection n'est pas effectuée de façon entièrement aléatoire dans l'ensemble de la liste complète des listes de répartition mais plutôt à l'intérieur de la sous-liste regroupant les listes pouvant générer les matrices de même dimension que la matrice du rez-de-chaussée.

La liste des proximités obligatoires pour la liste de pièces est ensuite comparée à la listes des jonctions de la matrice de répartition. Si la fonction prédicat retourne "faux", la liste des pièces du sous-sol est mélangée mais cette

fois pas de façon complètement aléatoire comme nous l'avions fait au rez-de-chaussée puisqu'il faut à présent tenir compte des proximités verticales.

Après avoir généré les quatre listes d'espaces définitives, le système vérifie si la cuisine se trouve au sous-sol ou au rez-de-chaussée. Dans le premier cas, la liste de mitoyennetés verticales pour l'escalier de service sera (2 13 31) (Cook's-Pantry Butler's-Pantry Trunk-Room). Dans le deuxième cas, la liste sera (3 13 31) (Servant's-Hall Butler's-Pantry Trunk-Room).

Prenons ce deuxième cas pour illustrer la méthode proposée. Le système doit donc mettre en relation verticale le module "Servant's Hall" (codé #3) au sous-sol avec le module "Butler's Pantry" (codé #13) au rez-de-chaussée, et ce, dans le but de déterminer l'emplacement de l'escalier de service. Supposons également que, pour la matrice de répartition du rez-de-chaussée, nous ayons retenu l'option suivante :
Ensemble des pièces mélangées : (12 10 13 16 11 14)
Liste de répartition : ((0 1 2 3 4 1 2 3 4 4 5 5 5) "4x3").

Nous remarquons que le "Butler's Pantry" (#13) se retrouve en troisième position dans la liste des pièces mélangées (list-ref 2). Le code 2, quant à lui, se retrouve dans la liste de répartition aux troisième et septième positions. Si nous désirons par exemple tester la liste de répartition ((0 1 1 1 0 2 3 4 0 5 5 5) "4x3")) pour le sous-sol, nous retrouvons respectivement en troisième et en septième positions les chiffres 1 et 3, ce qui revient à dire que, lorsque la liste des pièces du sous-sol sera mélangée, le "Servant's Hall" (#3) devra se retrouver soit en deuxième position (list-ref 1), soit en en quatrième position (list-ref 3).

La liste (1 2 3 5 6 7) sera mélangée de façon à obtenir par exemple (5 3 7 2 6 1) ou encore (7 2 1 5 3 6). La cohérence de la circulation verticale étant ainsi assurée, nous pourrons mélanger de façon aléatoire les autres éléments de

la liste des pièces et tester différentes matrices jusqu'à ce que les proximités horizontales au sous-sol soient elles aussi conformes.

Si après une dizaine de tentatives, la fonction prédicat (présentée en annexe, section 16) retourne toujours "faux", le système sélectionnera une nouvelle liste de répartition, encore une fois dans la même sous-section de la liste complète des listes de répartition.

Si après avoir successivement sélectionné une dizaine de listes de répartition, chacune testée une dizaine de fois avec la liste des pièces mélangée de façon différente, nous n'obtenons toujours pas un résultat positif, la procédure nous retournera une liste vide. Celle-ci indiquerait au système qu'aucune solution valide n'a été trouvé pour le sous-sol. Le processus de génération de la matrice 3D serait donc repris depuis le début.

3.3.7 Génération de la matrice de répartition de l'étage et du "attic floor"

Pour générer la matrice de répartition de l'étage, nous procédons de la même façon que pour la génération de la matrice de répartition du sous-sol, à deux exceptions près. D'une part, les tests de concordance diffèrent puisque, comme nous l'avons mentionné précédemment, nous retrouvons à l'étage des proximités partiellement obligatoires. D'autre part, les codes des modules à mettre en relation verticale sont différents. En effet, nous cherchons maintenant à superposer "Hall" (codé #10) et "Upper Hall" (codé #20) afin de déterminer l'emplacement de l'escalier principal.

Nous fournissons trois éléments à la procédure de génération de la matrice de l'étage: la liste des pièces de l'étage, la liste des proximités obligatoires et les listes de proximités partiellement obligatoires. Une procédure prédicat déterminera si chacune des proximités obligatoires et au moins une des

proximités partiellement obligatoires de chaque liste fait partie de la liste des jonctions de la matrice de répartition.

Encore une fois, si après avoir testé une dizaine de listes de répartition, chacune testée avec une dizaine de combinaisons différentes, le résultat est toujours "faux", la procédure (présentée en annexe, section 16) retournera une liste vide.

Les procédures de génération / validation des matrices de répartition pour le sous-sol et l'étage pourront retourner soit une liste de listes, c'est-à-dire une matrice, soit une liste vide. Une procédure déterminera si, à partir d'une matrice de répartition des espaces pour le rez-de-chaussée considérée comme définitive, l'une ou l'autre des matrices de répartition des espaces du sous-sol ou de l'étage est une liste vide, dans lequel cas la procédure générera une nouvelle matrice définitive pour le rez-de-chaussée qui servira de "input" pour la génération des matrices du sous-sol et de l'étage tant et aussi longtemps que le test ne sera pas positif.

Lorsque nous aurons obtenu des listes non vides pour chacun des trois premiers étage, nous aurons en main les informations nécessaires pour générer / valider une matrice de répartition pour le "attic foor". Là encore, une procédure récursive (présentée en annexe, section 17) reprendra le processus tant et aussi longtemps que nous n'aurons pas obtenu une liste non vide.

3.3.8 Exemple de matrice 3D proposée par le système

Voici un exemple de liste de listes générée par le système que nous avons modélisé.

(((3 4) (3 6) (0 6) (0 7) (0 7))

((14 10) (13 10) (12 10) (12 11) (12 11))
((27 27) (21 21) (22 20) (22 20) (23 23))
((31 37) (31 37) (32 30) (32 30) (36 36)))

Conformément à la démarche suivie, cette liste de listes de listes de codes numériques peut être interprétée graphiquement par les schémas présentés ci-après (figure 3-14), dans lesquels le pointillé court sert à indiquer l'emplacement de la circulation verticale pour les domestiques et le pointillé long sert à indiquer l'emplacement de la circulation verticale des maîtres.

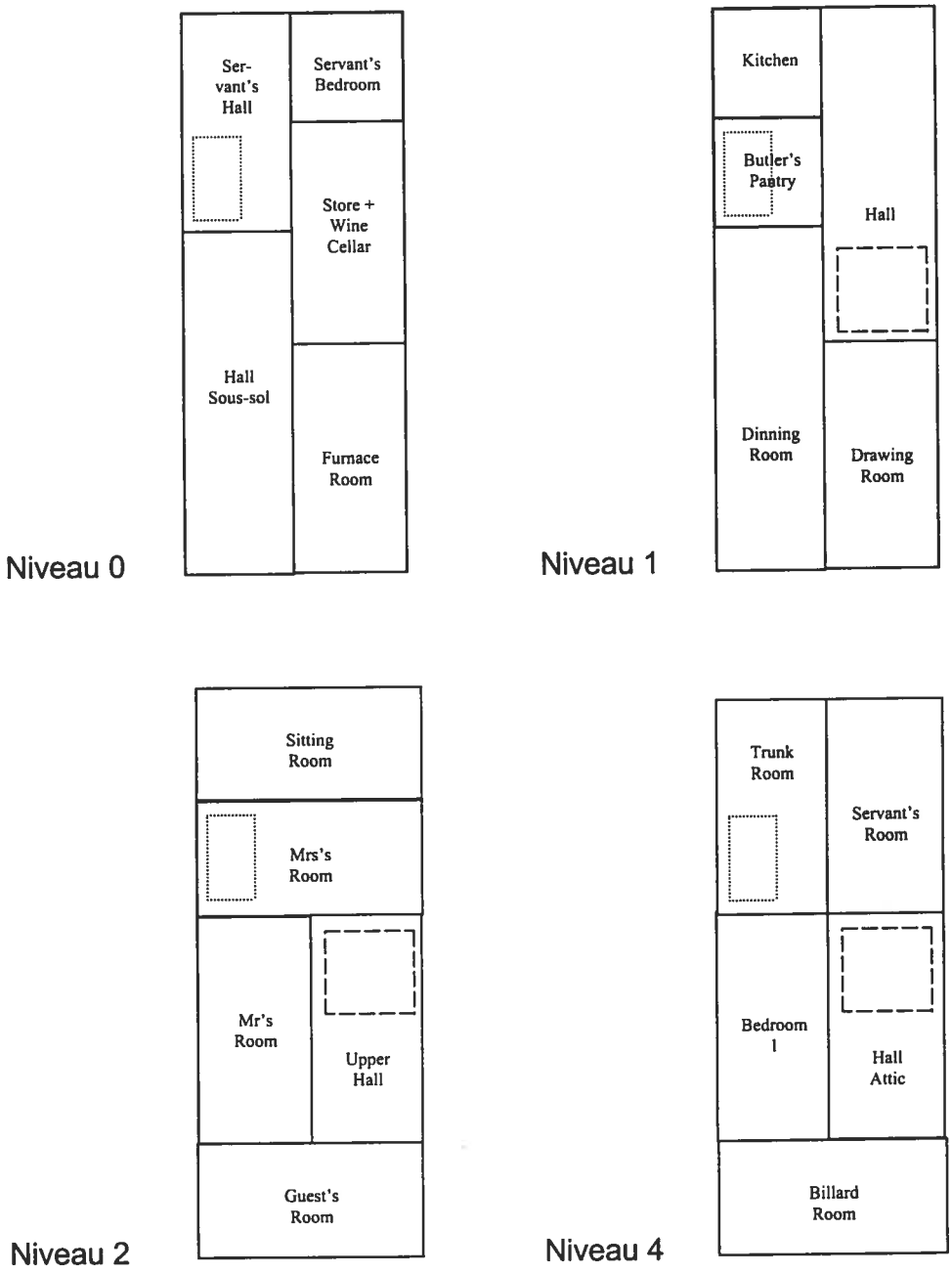

Figure 3-14: Représentation graphique de la matrice 3D sous forme de plans schématiques

Après analyse des résultats, nous constatons que, dans ce plan schématique d'étage, toutes les contraintes de proximités horizontales obligatoires et partiellement obligatoires, ainsi que toutes les contraintes de proximités verticales ont été respectées, ce qui était le but visé par le processus de génération de la matrice 3D.

4. Visualisation de la matrice

Les plans schématiques présentés à la fin du chapitre précédent n'avaient pour objet que de démontrer sommairement au lecteur la cohérence d'une matrice 3D proposée par le système. Voyons maintenant comment ces matrices 3D pourraient être présentées graphiquement afin que l'utilisateur du système puisse visualiser les solutions architecturales partielles et en faire une évaluation adéquate.

Dans cette première version du système, puisque la géométrie des différents espaces n'est pas définie, ce seront uniquement les relations entre les modules qui seront présentées à l'utilisateur. Les différents modules seront donc représentés sous forme de pastilles de couleurs tandis que les interrelations seront représentées sous forme de traits ou de flèches, construisant ainsi un assemblage du type "molécule".

Afin que l'utilisateur puisse distinguer les différents modules, nous utilisons un code de huit couleurs par étage. Par exemple, si le système propose la liste de listes suivante pour la répartition des espaces du rez-de-chaussée:

((13 13 13 16) (12 12 10 10) (11 11 11 11)),

nous avons la matrice de répartition des espaces suivante (incluant l'extérieur codé #100) :

100	100	100	100	100	100
100	**13**	**13**	**13**	**16**	100
100	**12**	**12**	**10**	**10**	100
100	**11**	**11**	**11**	**11**	100
100	100	100	100	100	100.

Avec le code de couleurs suivant :

Bleu	10	Hall
Rouge	11	Drawing room
Jaune	12	Dining room
Mauve	13	Butler's pantry
Orangé	14	Kitchen
Vert	15	Billard room
Gris foncé.....	16	Library
Gris pâle.....	17	Conservatory (ou loggia)
Blanc.....	100	Extérieur,

nous pourrions représenter graphiquement l'assemblage de modules de la façon suivante:

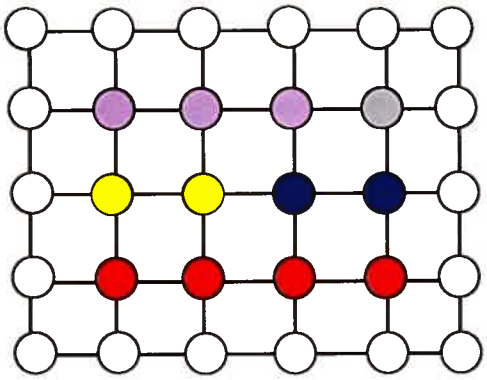

Figure 4-1: Représentation graphique de la matrice de répartition des espaces du rez-de-chaussée

Pour que l'utilisateur du système puisse faire une évaluation adéquate de la solution architecturale présentée, nous devons maintenant traduire graphiquement les interrelations entre les modules. Pour ce faire, nous devrons déterminer, pour chaque jonction, quelle position elle occupe et quels éléments elle joint. Nous obtiendrons ces informations en générant différentes listes à partir de la matrice de répartition des espaces de l'étage.

Commençons tout d'abord par codifier la position des jonctions. Rappelons que, dans cette version simplifiée du système, les dimensions des

matrices pouvant être générées sont 2x3, 2x4, 2x5, 3x2, 3x3, 3x4, 4x2, 4x3, 5x2. Le nombre maximum de travées horizontales et de travées verticales est donc de 5.

Pour codifier la position des jonctions, nous aurons recours au diagramme présenté en figure 4-2. Ce diagramme assigne un code numérique à chaque jonction, qu'il s'agisse d'une jonction entre une case de la matrice et une case mitoyenne ou d'une jonction entre une case et l'extérieur. Il pourra bien sûr être adapté selon nos besoins, si nous désirons élaborer un système pouvant générer des matrices de plus grandes dimensions.

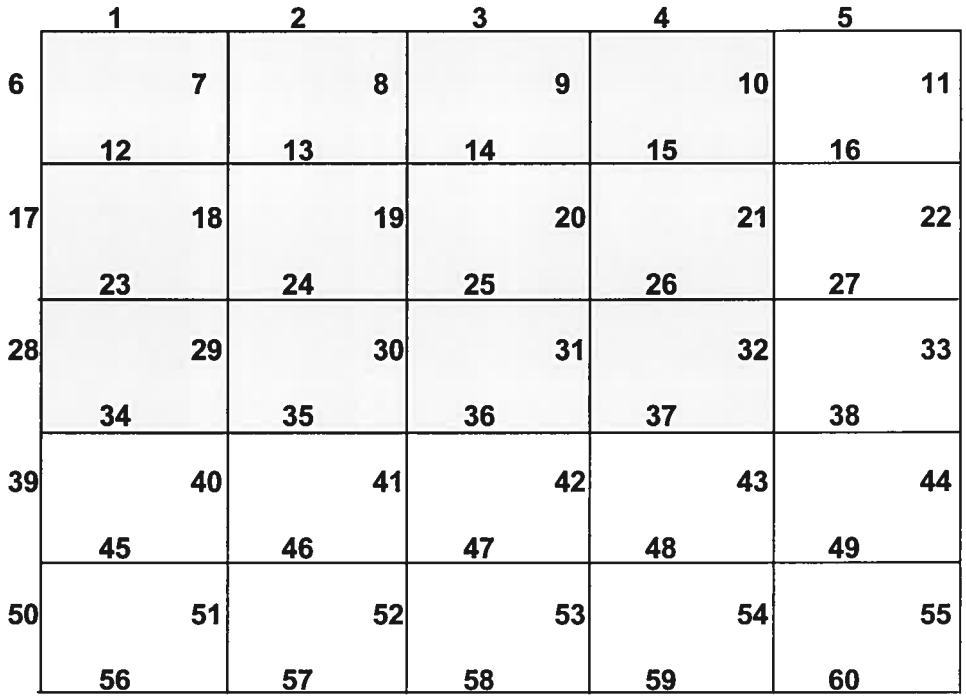

Figure 4-2. Diagramme de la position des codes d'emplacement des jonctions

Nous avons ici une soixantaine de jonctions possibles. Les zones présentées en gris sont les cases utilisées dans la matrice présentée précédemment. Comme nous pouvons le constater, les jonctions utilisées dans

cette solution architecturale partielle constituent un sous-ensemble de l'ensemble des jonctions possibles. Nous dirons qu'il s'agit des "jonctions actives".

Pour une matrice donnée, le système générera la liste des jonctions actives. Par exemple, dans ce cas-ci, il s'agirait de (1 2 3 4 6 7 8 9 10 12 13 14 15 17 18 19 20 21 23 24 25 26 28 29 30 31 32 34 35 36 37). Le système sera ensuite en mesure de générer une liste associant à chacun des codes de jonctions actives, le couple de modules mis en relation. Par exemple dans ce cas-ci, nous aurions :

((#1 (100 13))	Extérieur – Butler's Pantry
(#2 (100 13))	Extérieur – Butler's Pantry
(#3 (100 13))	Extérieur – Butler's Pantry
(#4 (100 16))	Extérieur - Library
:	
:	
(#18 (12 12))	Dinning Room – Dining Room
(#19 (12 10))	Dinning Room - Hall
:	
:	
etc.)	

À partir de cette liste, le système procédera à l'élimination de tous les couples de modules dont le premier et le deuxième élément sont identiques. Puisque ces couples ne mettent en relation que deux cases de la matrice et que celles-ci sont occupées par le même module, il ne s'agit pas vraiment d'une jonction à proprement parler. Nous dirons qu'il ne s'agit pas de "jonctions effectives".

Dans notre représentation du type "molécule", nous pourrions représenter les jonctions effectives par un trait noir et les autres jonctions par un trait de la

couleur des modules réunis. Les jonctions entre les modules seraient représentées à l'aide d'un segment. Les jonctions entre les modules et l'extérieur seraient, quant à elles, représentées à l'aide de flèches, ce qui nous permettrait d'éliminer les pastilles blanches représentant l'extérieur et donc de simplifier la représentation graphique.

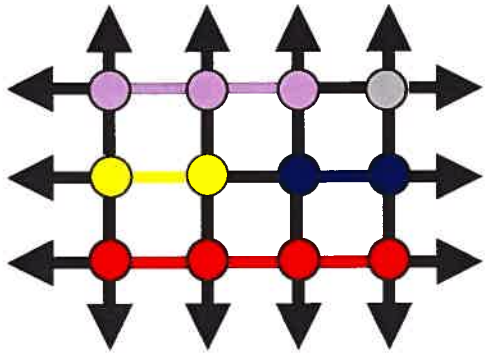

Figure 4-3. Les jonctions effectives

L'étape suivante consistera à générer une liste associant chaque couple de modules, à la liste des différentes positions où on le retrouve. En effet, si le couple de modules se répète, nous ne voulons pas que le système traite ces couples comme étant des jonctions différentes. Nous voulons plutôt que, lorsque les codes des jonctions sont des nombres consécutifs, le système traite ces couples comme étant une seule et même jonction occupant une série de positions mitoyennes.

Dans notre exemple, nous aurions la liste suivante :

(((100 13) #1 #2 #3)	Extérieur – Butler's Pantry
((100 16) #4)	Extérieur - Library
:	
:	
((12 10) #19)	Dining Room – Hall
((12 11) #23 #24)	Dining Room – Drawing Room

:

:

etc.)

Cette liste nous permettrait de représenter graphiquement la matrice de répartition des espaces de façon plus adéquate. Nous pourrions représenter les interrelations entre les modules et avec l'extérieur de la façon suivante :

Bleu 10	Hall
Rouge 11	Drawing room
Jaune 12	Dining room
Mauve 13	Butler's pantry
Orangé 14	Kitchen
Vert 15	Billard room
Gris foncé.....16	Library
Gris pâle.....17	Conservatory (ou loggia)
Blanc.....100	Extérieur

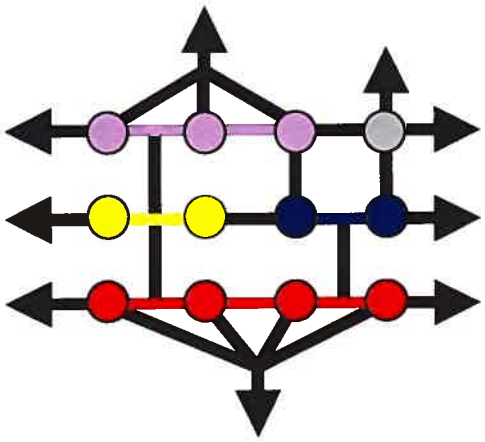

Figure 4-4. Optimisation de la représentation des jonctions effectives

Suite à la génération de la matrice 3D, l'utilisateur du système se verrait proposer quatre schémas de ce type (un pour chacun des étages). Il pourrait alors décider si ce type d'organisation lui convient ou non, dans lequel cas, il relancerait le système pour générer une nouvelle matrice 3D et donc une nouvelle série de diagrammes.

Il s'agit, bien évidemment, d'une solution architecturale partielle extrêmement schématique. Nous verrons, dans le prochain chapitre, comment il serait possible de générer une matrice 3D incluant davantage d'informations, de façon à proposer à l'utilisateur une solution architecturale partielle plus détaillée.

5. Complexification de la matrice 3D

Lors de la délimitation de notre espace de conception, nous nous sommes limités à fournir au système des informations de trois types :

- Des données concernant l'identification des différents espaces (par le biais des listes de modules),
- Des données concernant les relations de proximité obligatoires et partiellement obligatoires (par le biais des matrices de restrictions)
- Des données concernant différents schémas de répartition des espaces (par le biais des listes de répartition des espaces).

En fournissant au système une plus grande variété d'une plus grande quantité d'informations, il serait possible d'enrichir notre espace de conception, ce qui pourrait avoir deux effets. D'une part, si on augmentait le nombre de listes de répartition, il serait possible de générer un éventail de solutions plus varié. D'autre part, si on fournissait des matrices de restrictions additionnelles, il serait possible de générer des solutions partielles plus détaillées.

5.1 Matrices de formes irrégulières

Au chapitre précédent, lorsque nous avons étudié un mode de visualisation des solutions architecturales, nous avons constaté que la même jonction pouvait occuper plusieurs positions mitoyennes. Ceci est dû au fait qu'un même module occupe souvent différentes cases de la matrices, de façon à "remplir" la forme rectangulaire de la matrice.

Ce type d'approche nous contraint, pour représenter la solution architecturale proposée, à générer obligatoirement des diagrammes de forme

rectangulaire pour un niveau et en forme de parallélépipède rectangle pour l'ensemble des niveaux. Cette contrainte, en plus d'être superflue, pourrait biaiser la perception de l'utilisateur.

En effet, ce type de représentation pourrait inciter l'utilisateur à concevoir un bâtiment en forme de parallélépipède rectangle. Étant donné que l'objectif de la méthode que nous proposons est de stimuler le sens créatif de l'utilisateur, il serait intéressant de développer une approche qui l'inciterait à concevoir des bâtiments de formes diverses.

Lorsque nous générons les matrices, il est donc inutile de conserver tous les duplicata de jonctions. Pour éliminer certaines de ces redondances, il s'agirait de faire disparaître les modules occupant certaines cases de la matrice et de les remplacer par l'espace extérieur. Le fait de permettre au système de "déformer" ainsi la matrice pourrait stimuler encore davantage le sens créatif de l'utilisateur. Illustrons notre propos à l'aide d'un exemple.

Reprenons la matrice présentée précédemment :

100	100	100	100	100	100
100	13	13	16	16	100
100	12	12	10	10	100
100	11	11	11	11	100
100	100	100	100	100	100.

Jusqu'à maintenant, nous avons pris le parti d'entourer notre matrice d'une première et d'une dernière rangée et d'une première et d'une dernière colonne de code #100, symbolisant l'extérieur. Cependant, il serait également possible de "faire pénétrer l'extérieur" à l'intérieur de la matrice. Dans notre exemple, nous pourrions inclure l'extérieur (codé #100) de la façon suivante :

100	100	100	100	100	100
100	100	13	16	16	100
100	12	12	10	10	100
100	11	11	11	100	100
100	100	100	100	100	100

Cette matrice, que nous appellerons "matrice de forme irrégulière", pourrait être représentée graphiquement de la façon suivante :

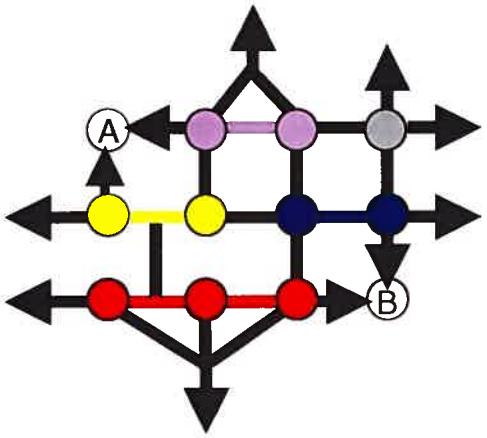

Figure 5-1 : Représentation graphique de la matrice de forme irrégulière

Le fait que des espaces intérieurs aient une jonction avec la même portion de l'espace extérieur, pourrait suggérer à l'utilisateur un nouveau type d'espace. Par exemple, le schéma illustré ci-dessus pourrait donner au designer l'idée d'aménager un porche (B) avec accès au hall et communication visuelle avec le Drawing Room ou encore d'aménager une terrasse (A) qui communiquerait avec le Dining room et le Butler's pantry.

L'introduction de matrices de formes irrégulières nous aiderait à stimuler encore davantage le sens créatif de l'utilisateur puisque nous pourrions produire un éventail de possibilités de solutions partielles encore plus varié. Pour ce faire, il faudrait ajouter des éléments à notre liste complète des listes de répartition. Voyons comment nous pourrions procéder.

Pour qu'une matrice soit de forme irrégulière, il faut qu'une ou plusieurs de ses cases soient occupées par le code représentant l'espace extérieur (#100). Bien entendu, il ne pourrait pas s'agir de cases choisies aléatoirement. Puisque ces suppressions ne devraient pas faire disparaître un ou des éléments de la liste des espaces. Les emplacements possibles de ces cases seraient déterminés au moment d'enrichir notre espace de conception.

Dans notre étude de cas, étant donné que notre échantillonnage de bâtiments ne comprend aucun élément ayant une cour intérieure, nous ne ferons pénétrer l'extérieur que dans le pourtour de la matrice. Nous pourrions par exemple chercher à supprimer différents coins du rectangle formé par la matrice, sans pour autant faire disparaître un module de l'étage.

À partir d'une liste de répartition, on peut en générer d'autres. Pour ce faire, nous pourrions définir une procédure récursive qui nous permettrait d'analyser chacune des matrices de forme régulière générée par chacun des éléments de la liste complète des listes de répartition. Nous pourrions effectuer des tests sur les quatre coins de ces matrices afin de déterminer lesquels pourraient être supprimés. Chaque coin serait représenté par le triplet suivant :

1- Le code de la case du coin
2- Le code de la case de gauche (ou de droite, selon le cas)
3- Le code de la case du bas (ou du haut, selon le cas).

Si la première valeur est égale à la deuxième et/ou à la troisième, ce coin pourra être supprimé sans faire disparaître un module de l'étage. Si, par contre, la première valeur n'est égale ni à la deuxième, ni à la troisième, c'est signe que la suppression de ce coin entraînerait la disparition d'un module et donc qu'il doit demeurer intact.

Les listes de répartition dont plusieurs coins auraient été identifiés comme étant "amovibles" seraient transmises à une procédure qui générerait d'autres listes de répartition en remplaçant la valeur des coins identifiés soit par un #6, soit par un #7. (Le chiffre 6 représente l'espace extérieur pour une matrice 5 espaces parce que le sixième élément d'une liste de cinq espaces est toujours 100, tandis que c'est le chiffre 7 qui représente l'extérieur pour une matrice 6 espaces.)

Ces nouvelles listes de répartition serviraient à générer des matrices de formes irrégulières. À partir d'une matrice, nous en formerions par exemple trois nouvelles, soit :

Afin d'aborder progressivement la complexité, nous pourrions dans un premier temps, nous limiter aux cas de figures suivants :

1- Suppression des deux coins d'un même côté
2- Suppression des deux coins opposés diagonalement
3- Suppression des quatre coins.

Nous aurions donc schématiquement les formes irrégulières suivantes pour une matrice 3 x 3.

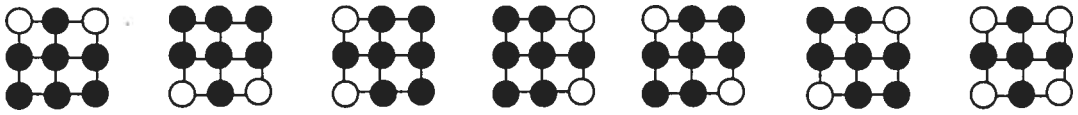

Figure 5-2 : Exemples de formes irrégulières

En somme, le nombre d'éléments de notre liste complète des listes de répartition serait grandement augmenté. Ces éléments devraient maintenant être classés selon trois critères. À un premier niveau, en fonction du nombre d'espaces et, à un deuxième niveau, en fonction de la dimension de la matrice et en fonction de la forme de la matrice générée. Une liste globale regrouperait tous ces sous-ensembles de listes et formerait notre nouvelle liste complète des listes de répartition.

5.2 Matrices espaces-jonctions

Jusqu'ici, nous n'avons travaillé que sur le positionnement des modules les uns par rapport aux autres. Afin que l'utilisateur soit en mesure d'évaluer de façon adéquate la solution partielle, il serait intéressant que le système génère également des propositions en ce qui a trait aux types de jonctions reliant les modules entre eux.

Pour ce faire, il faudrait que la matrice inclut des informations sur les types de jonctions reliant les espaces intérieurs entre eux et les espaces intérieurs avec l'extérieur. Nous pourrions complexifier la matrice 3D en y ajoutant des codes numériques faisant référence au type de jonction proposé entre les deux éléments de chacun des couples de modules. Voyons comment nous pourrions procéder.

En analysant notre échantillonnage de bâtiments, nous pourrions répertorier, les différents types de jonctions possibles et assigner à chacun un code numérique. Une version simplifiée des codes des jonctions pourrait par exemple être :

1-Porte principale
2-Porte de service
3-Portes françaises

4-Porte simple
5-Portes doubles
6-Arche
7-Portes doubles vitrées
8-Fenêtre simple
9-Fenêtre double
10-Fenêtre triple
11-Bow-window
12-Mur plein

A partir de cette liste, nous pourrions construire une matrice mettant en relation tous les couples de modules possibles pour un niveau donné avec les codes de toutes les jonctions répertoriées. Nous l'appellerons "matrice des jonctions horizontales". Le code de lecture de cette matrice pourrait être : 0 = interdit, 1 = permis. Par exemple, pour le rez-de-chaussée, une version simplifiée et abrégée de la matrice pourrait être la suivante :

	1	2	3	4	5	6	7	8	9	...
(10-11)	0	0	0	0	**1**	0	**1**	0	0	...
(10-12)	0	0	0	0	**1**	0	**1**	0	0	...
(10-13)	0	**1**	0	0	0	0	0	0	0	...
:										
:										
(11-12)	0	0	0	0	**1**	**1**	**1**	**1**	**1**	...
:										
:										
(11-100)	0	0	**1**	0	0	0	0	**1**	**1**	...
:	:	:	:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:	:	:	:

Figure 5-3: Matrice de restriction des jonctions horizontales

Cette matrice serait fournie à une procédure récursive qui nous permettrait de déterminer, pour une jonction donnée, quelles sont les possibilités, et de générer une liste des codes de jonctions acceptables. Le système effectuerait ensuite une sélection aléatoire à l'intérieur de cet ensemble. Pour chaque couple

de modules présent dans la solution partielle, il pourrait donc proposer une jonction adéquate en fonction des deux espaces mis en relation.

Dans le chapitre précédent, nous avons décrit la procédure (présentée en annexe, section 13) visant à générer la liste associant chaque couple de modules avec la liste des codes d'emplacements de jonctions. Nous viendrons maintenant y associer le code du type de jonction choisi aléatoirement par le système à l'intérieur de l'ensemble des jonctions acceptables. Nous aurions donc le triplet suivant : couple de module, code(s) d'emplacement(s), type de jonction.

Grâce à ce triplet, le système saurait si la jonction est appliquée sur un seul code d'emplacement ou sur plusieurs codes d'emplacement consécutifs. Lorsqu'il traduirait ces propositions de jonctions sous forme matricielle, il représenterait par son code numérique la jonction appliquée à un seul emplacement. Si par contre le triplet contient plusieurs codes d'emplacements, il pourrait représenter, par exemple, la jonction de code 8 appliquée sur deux jonctions identiques consécutives par les codes 8-1 et 8-2.

Pour chacun des niveaux, ces triplets nous permettraient ainsi de générer une nouvelle matrice incluant les codes de type de jonction qui auraient été sélectionnés aléatoirement par le système. Cette nouvelle matrice, que nous appellerons "matrice espaces-jonctions" serait de plus grande dimension que la matrice de répartition des espaces. Les codes numériques seraient distribués de la façon illustrée par la matrice présentée en figure 5-4 où "M" est utilisé pour symboliser un module, "J" une jonction et "X" un emplacement qui n'est occupé ni par un module, ni par une jonction.

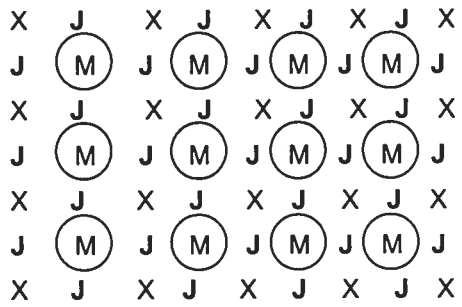

Figure 5-4 : Emplacement des différents types de code numérique de la matrice espaces-jonctions

Afin d'illustrer notre propos, reprenons la matrice de forme irrégulière de notre exemple précédent :

100	100	100	100	100	100
100	100	13	16	16	100
100	12	12	10	10	100
100	11	11	11	100	100
100	100	100	100	100	100

En appliquant cette méthode, le système se référant à la matrice des jonctions horizontales présentée précédemment, pourrait proposer, par exemple, la matrice espaces-jonctions suivante. (Pour une meilleure lisibilité, les codes de module sont présentés à l'intérieur de cercles et les codes de jonctions sont en caractères gras.)

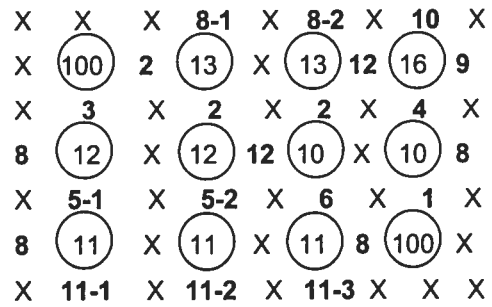

Figure 5-5: Exemple de matrice espaces-jonctions

Il s'agit des mêmes interrelations entre les espaces que dans l'exemple présenté précédemment mais en ajoutant des informations sur le type de jonction. Afin que l'utilisateur soit en mesure d'évaluer de façon adéquate la solution partielle, nous pouvons, encore une fois, représenter ces interrelations par un assemblage de type molécule ou par un organigramme, tels que présentés en figures 5-6 et 5-7.

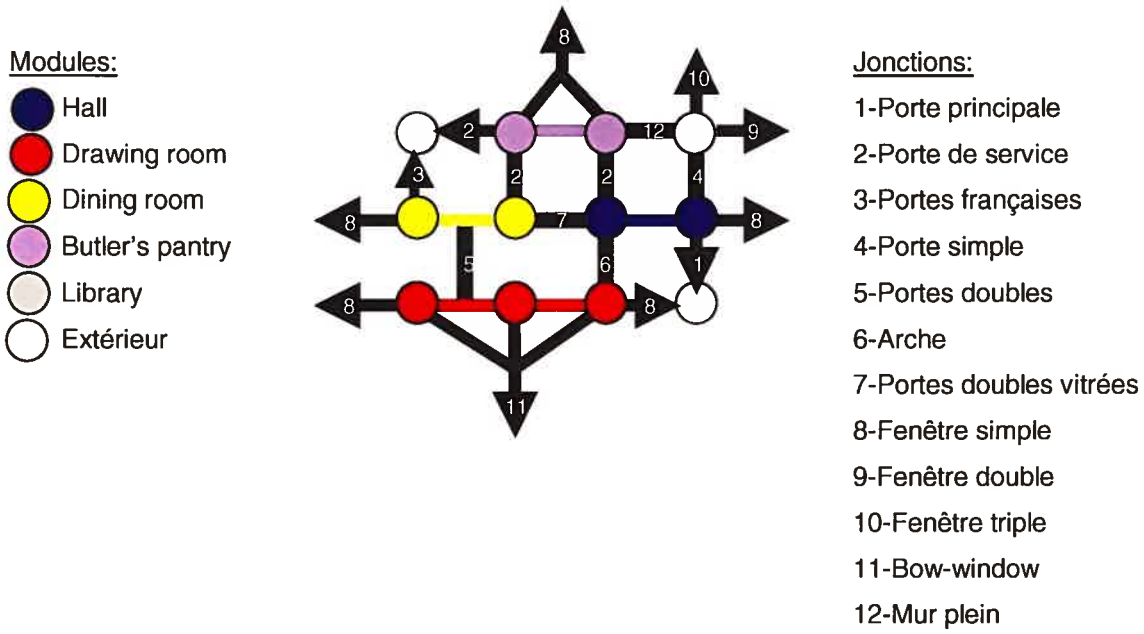

Figure 5-6. Représentation graphique de la matrice espaces-jonctions

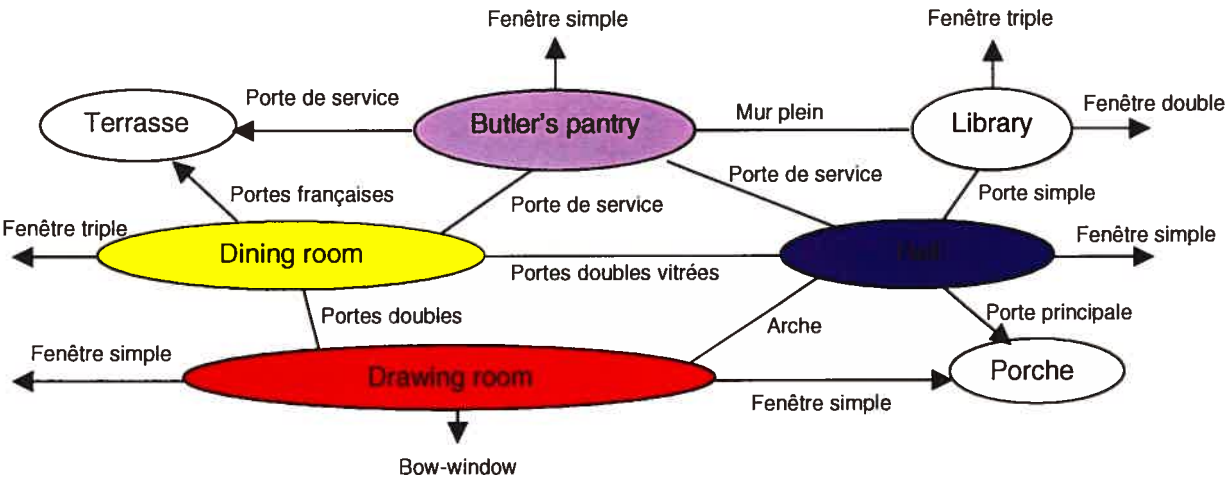

Figure 5-7. Organigramme de l'interrelation entre les modules.

Nous avons illustré la méthode proposée à l'aide de la matrice de répartition des espaces du niveau rez-de-chaussée. Il ne s'agit, bien sûr, que de l'un des niveaux de la matrice 3D. Le même processus de sélection aléatoire du type de jonction, à partir d'une matrice des jonctions horizontales, pourrait se répéter à chaque niveau afin de constituer une matrice 3D.

Jusqu'ici, nous n'avons traité que des jonctions au plan horizontal. Cependant, pour générer une matrice 3D complète, il faudrait également traiter des jonctions sur le plan vertical. Entre chaque matrice espaces-jonctions et celle du niveau mitoyen, nous viendrons insérer une matrice indiquant les types de jonctions verticales. Nous l'appellerons "matrice inter-niveaux".

Afin de générer une matrice 3D cohérente, les matrices inter-niveaux devraient être de mêmes dimensions que les matrices des niveaux. On utilisera donc le même gabarit mais l'emplacement des modules deviendra jonction verticale (codé J) et l'emplacement des jonctions horizontales deviendra espace nul (codé X), tel que présenté en figure 5-8.

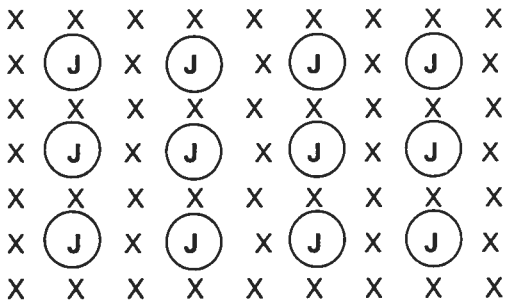

Figure 5-8 : Positionnement des codes de la matrice inter-niveaux

Au moment de définir notre espace de conception, nous avons déterminé quels étaient les deux types de communication verticale et quelles listes de modules elles devaient mettre en interrelation. Nous viendrons maintenant enrichir notre espace de conception en fournissant au système, pour chaque type de circulation verticale, une liste regroupant différents types de jonctions:

Communication verticale #1 (escalier principal)
1- Escalier large à deux volées
2- Escalier large à trois volées
3- Escalier large en demi-cercle

Communication verticale #2 (escalier de service)
4- Escalier étroit à deux volées
5- Escalier étroit en colimaçon

Lors de la génération de la matrice 3D, nous avons vu de quelle façon le système déterminait la position des communications verticales. Le système viendra maintenant y assigner un code d'emplacement selon le schéma présenté en figure 5-9.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Figure 5-9 : Codes d'emplacement des jonctions verticales

Après avoir effectué une sélection aléatoire à l'intérieur des listes de codes de jonctions, nous aurions le triplet (code d'emplacement vertical, type de communication verticale, type de jonction verticale). Par exemple, (emplacement #8, communication #1, type de jonction #3).

Entre les niveaux rez-de-chaussée et étage de l'exemple présenté précédemment, le système pourrait proposer la matrice inter-niveaux présentée en figure 5-10, où le chiffre 3 représente un escalier principal en demi-cercle, le chiffre 5 un escalier de service en colimaçon, et le chiffre 0 l'absence de circulation verticale.

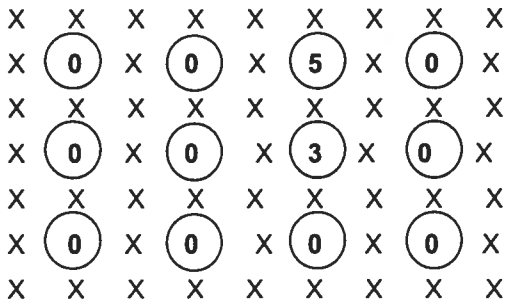

Figure 5-10 : Exemple de matrice inter-niveaux

En empilant alternativement les matrices de niveau et les matrices inter-niveaux, nous pourrions générer une matrice 3D incluant davantage d'information, et donc présentant une solution partielle plus détaillée.

Les matrices 3D espaces-jonctions pourraient être représentées graphiquement de la façon illustrée en figure 5-11. Ces organigrammes auraient pour fonction, rappelons-le, de stimuler le sens créatif de l'utilisateur. À chaque fois que celui-ci relancerait le programme, le système proposerait une nouvelle solution partielle formée d'une sélection différente de modules, assemblés de façon différente et reliés par des propositions de jonctions différentes. L'utilisateur serait donc "confronté" à une grande variété de solutions partielles possibles, à l'intérieur de l'espace de conception qu'il aurait délimité.

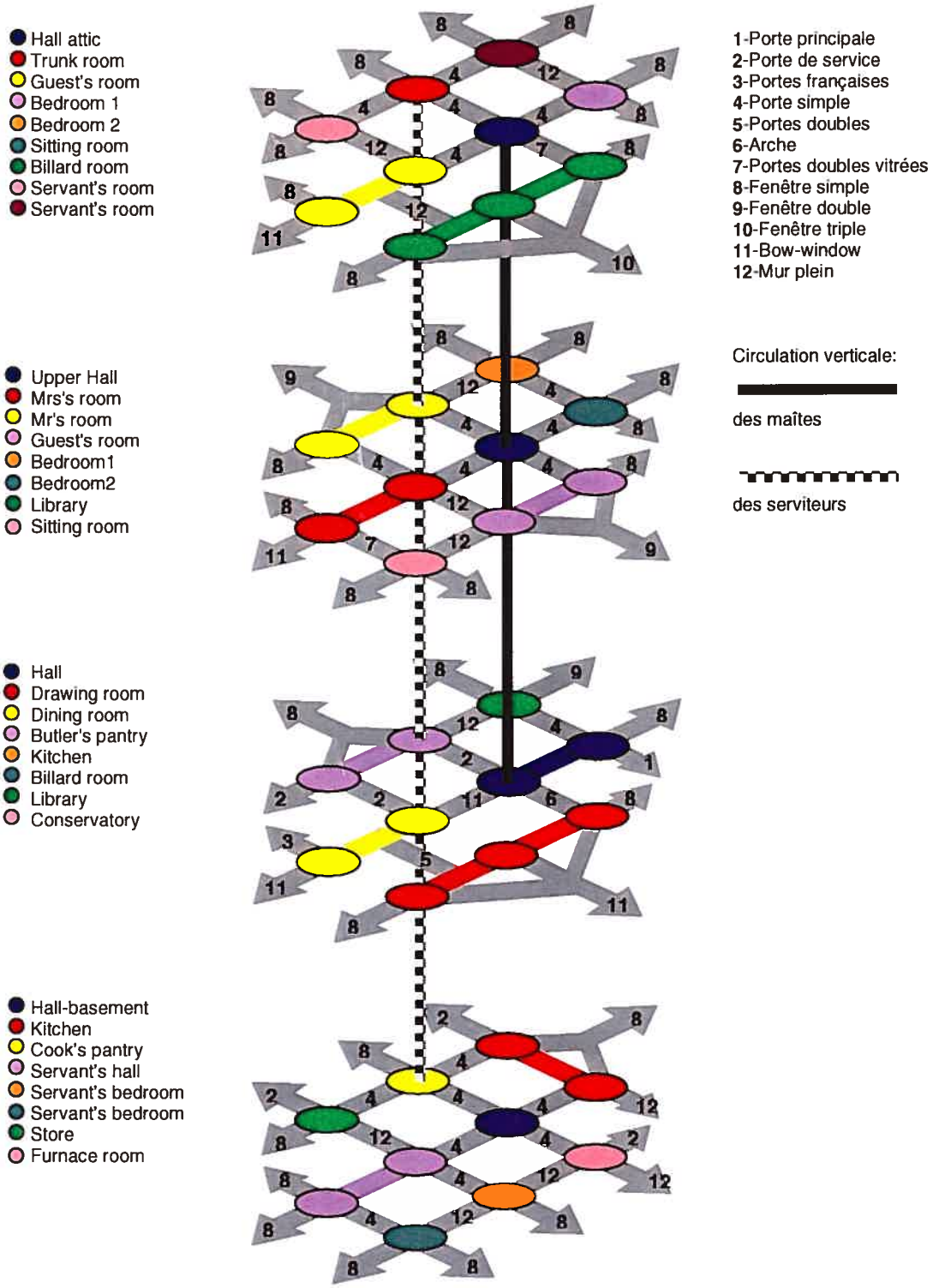

Figure 5-11 : Représentation graphique de la matrice 3D.

6. Structure du système générateur de solutions architecturales

6.1 Complexité du processus de conception architecturale

Au chapitre précédent, nous avons vu comment nous pouvions procéder progressivement à l'enrichissement de l'espace de conception. Bien entendu, plus il y aura de modules, plus il y aura d'éléments dans la liste complète des listes de répartition et plus on a de types de jonctions répertoriées, plus l'éventail des solutions partielles possibles sera varié. Il existe potentiellement une multitude de solutions architecturales partielles possibles. Nous ne pourrions toutes les énumérer. On peut donc dire qu'il ne s'agit pas d'un modèle fini qui puisse être déterminé.

Selon Jean-Louis Le Moigne (1990), pour comprendre et donc donner du sens à un système compliqué (et donc qui peut être déterminé), on peut le simplifier pour découvrir son intelligibilité, son explication. Si, par contre, il s'agit d'un système complexe, comme c'est le cas ici, on doit le modéliser pour construire son intelligibilité, sa compréhension. Mais en simplifiant un système complexe, et donc en le mutilant, on détruit a priori son intelligibilité.

Comme nous l'avons mentionné précédemment, depuis deux décennies plusieurs chercheurs ont tenté de mettre à contribution la CAO dans le processus de design. On peut cependant se demander s'ils n'ont pas, pour la plupart, abordé le problème comme s'il s'agissait d'un modèle fini. Considérer le processus de conception architecturale comme un système compliqué, plutôt que complexe, n'est pas précisément l'erreur que plusieurs d'entre eux ont commise?

Ce questionnement fait partie des préoccupations de Scott C. Chase (1999). Selon lui, le problème avec les systèmes actuels d'aide à la conception architecturale serait qu'ils ont une approche "kit-of-part". On cherche à assembler des éléments pour former un tout. Comme il est difficile d'anticiper tous les agencements possibles qu'un utilisateur potentiel pourrait souhaiter utiliser, beaucoup de restrictions et de simplifications sont imposées au processus de design pour arriver à programmer le système. L'utilisateur est donc restreint dans ses opérations parce qu'en tentant de simplifier un système complexe de conception architectural, on le mutile et on détruit son intelligibilité.

L'étude de U. Flemming (1987) sur l'ensemble patrimoniale de Shadyside, et sur la génération de modèles de maisons de style Queen Anne est, d'après nous, un exemple typique de l'approche "kit-of-part". Comme on l'a vu, on générait le plan de rez-de-chaussée en procédant par étapes. On plaçait d'abord le hall et ensuite les pièces qui l'entouraient et l'escalier principal, selon les règles pré-établies. Ce type d'approche revient non seulement à aborder le processus de design comme s'il s'agissait d'un modèle fini mais également comme s'il s'agissait d'un enchaînement linéaire d'actions.

L'approche que nous proposons, et qui consiste à avoir recours à des procédures pour sélectionner aléatoirement des éléments et à valider ces choix avant de poursuivre le processus permet, selon nous, de nous distinguer de ce type d'approche. La méthode que nous proposons nous permet d'aborder le processus de design non pas comme un enchaînement linéaire mais comme un enchevêtrement d'interactions.

Par exemple, les positions des modules Hall et Butler's Pantry (déterminant l'emplacement des deux circulations verticales) ont un impact sur la position des modules à l'étage. Inversement, ceux-ci ont un impact sur la position du Hall et du Butler's Pantry si, comme on l'a vu, la procédure nous retourne une liste vide.

Notons toutefois que bien que nous ne procédions pas à l'assemblage d'éléments, étape par étape, le système que nous avons défini constitue un générateur automatique de solutions architecturales. Tout comme le système élaboré par Flemming (1987), il n'y a pas d'interaction possible entre l'utilisateur et le système.

Il est cependant clair que si l'on souhaite explorer une nouvelle façon d'aborder le processus de design, il ne suffit pas de présenter à l'utilisateur une solution architecturale conçue de façon automatique, en laissant le système prendre toutes les décisions. Le processus de design doit être un dialogue, que ce soit un dialogue entre le designer et lui-même, dans l'approche traditionnelle, ou entre le designer et le système, dans l'approche que nous proposons.

Dans la méthode que nous avons décrite jusqu'à présent les interactions sont entièrement gérées par le système. Si nous désirons que celui-ci soit réellement un outil d'aide à la réflexion pour le designer, nous devons permettre à l'utilisateur d'interagir avec le système. Pour ce faire, nous avons pris le parti de nous orienter vers la modélisation systémique.

Le concept de base de cette approche n'est pas l'objet ou la combinaison d'objets stables (la structure) mais plutôt l'action. Alors que la modélisation analytique (comme par exemple l'approche de Flemming) part de la question "De quoi c'est fait? ", la modélisation systémique, elle, part de la question "Qu'est-ce que ça fait? ".

Aux concepts de la modélisation analytique (élément, ensemble, découpage), nous substituerons les concepts adaptés à la modélisation systémique (unité active, système, articulation). En réponse à la question "Qu'est-ce que ça fait? ", nous identifierons dans la prochaine section une série

d'unités actives. Nous verrons ensuite comment celles-ci sont mises en interrelation pour former un système complexe.

6.2 Les unités actives

Lorsque nous avons programmé notre générateur automatique de solutions architecturales sous forme matricielle, nous avons tenté de résoudre le problème en le fractionnant en résultats partiels. Chacun de ces résultats représente un progrès identifiable vers le but (c'est-à-dire générer la matrice 3D) et joue le rôle d'un sous-assemblage stable.

En choisissant d'articuler le système en différents niveaux fonctionnels par projets intermédiaires stables, nous avons pu aborder progressivement la complexité. Nous avons ainsi passé probablement plus de temps à réfléchir sur la structure du système que sur sa transcription en langage informatique.

Dans notre première version du système, c'est-à-dire la génération de la matrice 3D de forme régulière et sans information sur le type de jonctions, nous avons regroupé nos procédures Scheme en 18 fichiers. Le regroupement des procédures a été fait de façon à ce que chaque fichier regroupe des procédures qui, ensemble, arrivaient à résoudre un des sous-problèmes posés. Chacun de ces fichiers pourra ainsi être identifié comme une unité active.

Afin de bien saisir comment s'articule le système, énumérons les différentes unités actives qui le composent. Dans le tableau présenté ci-après, nous les avons nommées, numérotées et avons identifié les finalités de chacune d'entre elles. Nous fournissons également le nom du fichier Scheme afin que le lecteur puisse se référer au code des procédures fourni en annexe.

Identification	Objectif	Unités actives invoquées
Unité active # 1: Assemblage de la matrice 3D (assemblage3d.scm)	Assembler la matrice 3D représentant l'armature de la solution architecturale proposée par le système.	Sélection de la matrice de répartition pour le "attic" (2)
Unité active #2: Sélection de la matrice de répartition pour le "attic-floor" (select-mat-att.scm)	Sélectionner et valider la matrice de répartition des espaces pour le niveau "attic-floor".	Sélection des matrices de répartition des espaces pour le sous-sol et l'étage (3)
Unité active #3: Sélection des matrices de répartition des espaces pour le sous-sol et l'étage (select-mat-ss-eta.scm)	Sélectionner et valider les matrices de répartition des espaces pour le sous-sol et l'étage.	Sélection de la matrice de répartition pour le rez-de-chaussée (4)
Unité active #4: Sélection de la matrice de répartition pour le rez-de-chaussée (select-mat-rdc.scm)	Sélectionner et valider la matrice de répartition des espaces pour le rez-de-chaussée	Vérification de la concordance des proximités (5) Construction de la liste des couples de jonction (6)
Unité active #5: Vérification de la concordance des proximités (concordance.scm)	Déterminer si les jonctions d'une matrice de répartition proposée par le système sont conformes aux listes de proximités obligatoires et partiellement obligatoires de l'espace de conception.	Fonctions diverses (18)
Unité active #6: Construction de la liste des couples de jonction (prox-mat.scm)	Construire la liste des couples de jonction de la matrice de répartition à partir d'une liste contenant une matrice de répartition des espaces et le code de dimension de la matrice.	Construction de la matrice de répartition des espaces (7)
Unité active #7: Construction de la matrice de répartition des espaces (cons-mat.scm)	Construire la matrice de répartition des espaces à partir de la liste des modules et de la liste de répartition.	Choix de la liste de répartition (8)
Unité active #8: Choix de la liste de répartition (choix-ls-rep.scm)	Choisir de façon aléatoire une liste de répartition des espaces.	Génération des listes de proximités obligatoires et partiellement obligatoires (9) Triage des matrices (10)

Unité active #9: Génération des listes de proximités obligatoires et partiellement obligatoires (prox-ls-pieces.scm)	Générer pour chaque niveau une liste de 2 ou 3 sous-listes. La première sous-liste est la liste de 5 ou 6 pièces sélectionnées. La deuxième sous-liste est la liste des proximités obligatoires (code 1) pour cette liste. La troisième sous-liste regroupe les listes de proximités partiellement obligatoires (code 2 ou 3) si applicable.	Vérification des listes provisoires de pièces (12)
Unité active #10: Triage des matrices (triage-matrices.scm)	Faire le tri parmi les listes de répartition, selon la dimension de la matrice qui sera générée.	Codage des diagrammes de Mitchell (11)
Unité active #11: Codage des diagrammes de Mitchell (mat.mitchell.scm)	Transcrire les différents diagrammes de subdivision de Mitchell sous forme de listes de répartition.	aucune
Unité active #12: Vérification des listes provisoires de pièces (verif-pieces.scm)	Vérifier la cohérence entre les listes de pièces provisoires pour les quatre niveaux et retourner les listes de pièces définitives.	Génération des listes de pièces provisoires (13)
Unité active #13: Génération des listes de pièces provisoires (iste-pieces-prov.scm)	Générer pour chaque niveau une liste provisoire regroupant les 5 ou 6 pièces choisies pour cet étage.	Délimitation de l'espace de conception du sous-sol (14) Délimitation de l'espace de conception du rez-de-chaussée (15) Délimitation de l'espace de conception de l'étage (16) Délimitation de l'espace de conception du "attic" (17)
Unité active #14: Délimitation de l'espace de conception du sous-sol (DW-SS.scm)	Délimiter l'espace de conception pour le sous-sol sous forme de listes. La liste regroupant les pièces obligatoires, la liste regroupant les pièces optionnelles et la liste de l'ensemble des proximités obligatoires.	Fonctions diverses (18)
Unité active #15: Délimitation de l'espace de conception du rez-de-chaussée (DW-RDC.scm)	Délimiter l'espace de conception pour le rez-de-chaussée sous forme de listes. La liste regroupant les pièces obligatoires, la liste regroupant les pièces optionnelles et la liste de l'ensemble des proximités obligatoires.	Fonctions diverses (18)

Unité active #16: Délimitation de l'espace de conception de l'étage (DW-eta.scm)	Délimiter l'espace de conception pour l'étage sous forme de listes. La liste regroupant les pièces obligatoires, la liste regroupant les pièces optionnelles et la liste de l'ensemble des proximités obligatoires.	Fonctions diverses (18)
Unité active #17: Délimitation de l'espace de conception du "attic-floor" (DW-att.scm)	Délimiter l'espace de conception pour le niveau "attic-floor" sous forme de listes. La liste regroupant les pièces obligatoires, la liste regroupant les pièces optionnelles et la liste de l'ensemble des proximités obligatoires.	Fonctions diverses (18)
Unité active #18: Fonctions diverses (fonc-div.scm)	Regrouper différentes procédures générales fréquemment appelées.	aucune

Ce tableau nous permet de saisir non seulement le mandat de chacune des procédures mais également l'ordre dans lequel elles sont invoquées. Au chapitre 3, lorsque nous avons expliqué comment le système avait été élaboré, nous avons décrit ces procédures dans l'ordre inverse. La dernière procédure décrite, et qui avait pour fonction d'assembler la matrice 3D, est en fait celle que nous invoquons en premier. Celle-ci invoque une autre procédure qui à son tour en invoque d'autres et ainsi de suite.

Pour illustrer les interrelations entre les unités actives, nous représenterons les combinaisons d'interrelations entre les unités à l'aide de la matrice structurelle de la première version de notre système. On dira qu'il y a interrelation entre deux unités actives U1 et U2 lorsqu'une unité active en appelle une autre. Nous mettrons en ordonnée l'unité active faisant appel à une autre et en abscisse l'unité active appelée, la présence d'un "1" dans la case U1 U2 signifiant que l'interrelation U1-U2 est activée et la présence d'un zéro signifiant que cette interrelation n'est pas activée. (Voir figure 6-1.)

21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
4	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Figure 6-1 : Matrice structurelle du système

En observant cette matrice, on remarque qu'il est possible d'identifier plusieurs sous-matrices relativement denses. Ces sous-matrices, encadrées ici en trait plein, nous serviront à articuler le système en sous-systèmes. L'espace de conception et les utilitaires du système, quant à eux, sont encadrés en pointillés.

L'intérêt d'articuler entre eux une série de sous-systèmes est que cette approche nous permettrait d'introduire la notion d'arborescence qui nous

permettrait de rendre le système interactif. Le système demeurerait basé sur les mêmes procédures de sélections aléatoires. Cependant, il ne génèrerait plus automatiquement une seule matrice 3D afin de la présenter à l'utilisateur à la fin du processus. Il lui présenterait plutôt, pour chacun des niveaux, différentes possibilités acceptables de solutions architecturales partielles. C'est l'utilisateur qui sélectionnerait celles qui lui conviennent et effectuerait ainsi un parcours dans une arborescence. Ce parcours le conduirait éventuellement vers une solution architecturale "complète" tenant compte de ses préférences. Ainsi, l'utilisateur ne serait plus spectateur du processus de design, il en deviendrait le principal acteur.

6.3 L'arborescence

Comme nous l'avons vu précédemment, la matrice structurelle de la première version du système nous permet d'identifier quatre différents sous-systèmes. La finalité de chacun d'entre eux est présentée ci-dessous:

Sous-système #1	Présenter différents ensembles de listes d'espaces et faire choisir l'utilisateur
Sous-système #2	Proposer différents schémas de répartition des espaces du rez-de-chaussée et faire choisir l'utilisateur
Sous-système #3	Proposer différents schémas de répartition des espaces au sous-sol et à l'étage et faire choisir l'utilisateur
Sous-système #4	Proposer différents schémas 3D regroupant les quatre étages du bâtiment et faire choisir l'utilisateur

Dans cette optique, les unités actives seront regroupées par niveaux que nous appellerons sous-processeurs. Pour la nouvelle version du système, nous aurons le raisonnement inverse de celui représenté par la matrice structurelle de la première version. On ne cherche plus à savoir quelle unité active en appelle

une autre, mais plutôt quel sous-processeur enverra des données à quel autre. On dira qu'il y a interrelation entre deux sous-processeurs SP1 et SP2 lorsqu'un extrant ou une composante du vecteur extrant du sous-processeur SP1 sera intrant du sous-processeur SP2. L'interrelation (SP1-SP2) sera alors activée.

La figure 6-2 illustre de quelle façon les quatre sous-processeurs, les données de l'espace de conception et les procédures des fonctions utilitaires du système pourraient être mis en relation.

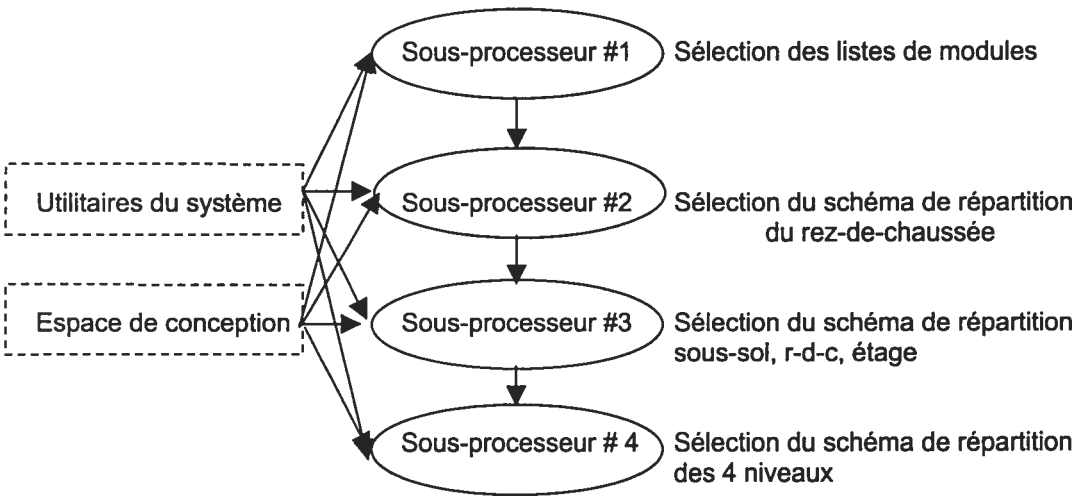

Figure 6-2 : Schéma de l'interrelation entre les processeurs

À chacun de ces niveaux, l'utilisateur pourrait effectuer des choix parmi des sous-ensembles de possibles solutions partielles, ce qui l'aiderait à progressivement définir le concept. Il s'agit d'une structure en arborescence puisque, à chaque étape, les décisions de l'utilisateur ont un impact sur les solutions présentées au projet intermédiaire stable suivant. Par exemple, selon les espaces sélectionnés par l'utilisateur, le système proposera différentes matrices de répartition des espaces. À partir de la matrice de répartition des espaces choisie par l'utilisateur pour le rez-de-chaussée, le système sera en mesure de proposer une série de matrices de répartition des espaces pour les

étages mitoyens. Et ainsi de suite... Cette structure en arborescence est présentée ci-après, en figure 6-3.

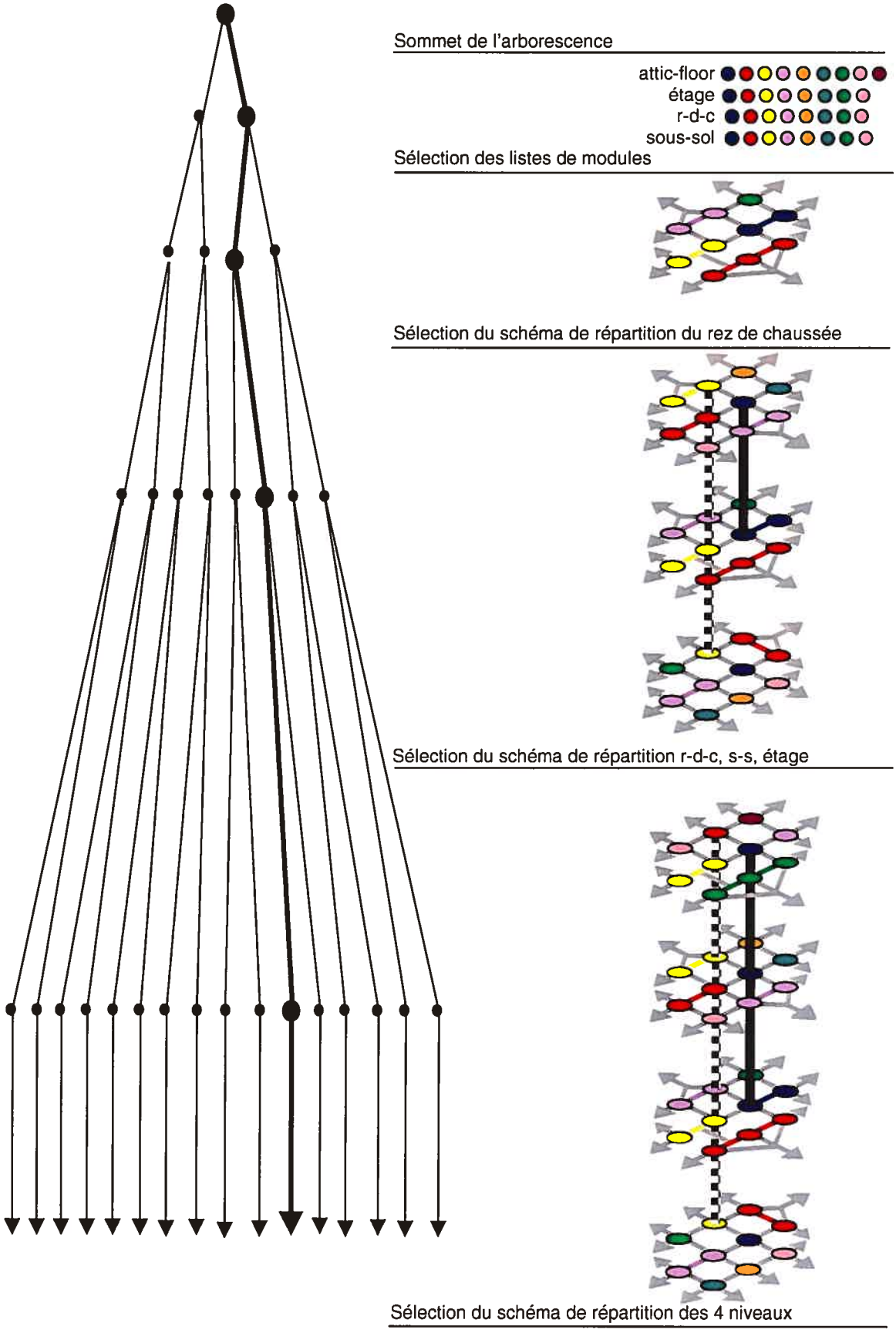

Figure 6-3 : Exemple d'un parcours dans une arborescence

L'utilisateur élaborera la solution architecturale qui lui convient en effectuant un parcours dans l'arborescence. À chaque étape, la sélection sera discrétionnaire. Il sera donc a priori impossible d'anticiper le parcours qui sera effectué dans l'arborescence puisque la notion de complexité implique celle d'imprévisible. Au dernier niveau de cette arborescence, l'éventail des solutions architecturales satisfaisantes au regard des critères établis sera multiple. La solution sélectionnée dépendra entièrement de la combinaison des choix qui auront été effectués par l'utilisateur et ne sera pas prédéterminée par le système.

Dans la modélisation analytique des systèmes compliqués, nous retrouvons la notion de "comportement optimum". Dans une situation donnée, le système est présumé avoir le choix entre diverses décisions de comportements possibles. Il doit alors pouvoir calculer la meilleure (ou l'optimum, le "one best way") en s'aidant des critères auxquels il se réfère. Il n'a pas le choix de sa décision ; celle-ci est prédéterminée.

Dans le système complexe que nous modélisons, il n'y a pas de "comportement optimum". La solution idéale n'existe pas puisque, comme on le sait, toute décision en situation complexe est une décision multicritère et il existe a priori plusieurs (souvent beaucoup) de solutions satisfaisantes possibles.

La finalité du système n'est donc pas de produire la solution optimum, mais plutôt un ensemble de solutions très soigneusement recherchées, au mieux des possibilités, satisfaisantes sous tous les critères établis et parmi lesquelles l'utilisateur pourra effectuer un choix. À chaque niveau, il se verra présenter une sélection aléatoire de solutions partielles acceptables et en sélectionnera une selon ses préférences ou demandera au système de produire un nouvel ensemble de solutions partielles possibles. Nous verrons dans la prochaine section comment le système arrivera à présenter cette sélection aléatoire de solutions partielles.

6.4 L'intelligence du système

Pour chaque projet intermédiaire stable, il existe plusieurs solutions potentielles. Nous appellerons cet ensemble de solutions potentielles "Sp". Le système générera différents éléments de cet ensemble à l'aide de procédures de sélection aléatoire. Il évaluera ensuite la validité des éléments générés à l'aide d'une procédure prédicat. Chaque fois que l'évaluation aura été positive, le système placera cette solution dans un sous-ensemble de solutions acceptables (que nous appellerons "Sa").

Lorsque Sa regroupera un nombre N d'éléments, il sera présenté à l'utilisateur afin que celui-ci effectue un choix. Dans le cas où le système aurait effectué un nombre n de tentatives (fixé à 100 dans la première version du système) et où Sa serait un ensemble vide ou ne contenant qu'un seul élément, l'utilisateur ne pourrait effectuer de choix ce qui nous empêcherait de poursuivre notre parcours dans l'arborescence et donc d'arriver à une solution satisfaisante du problème.

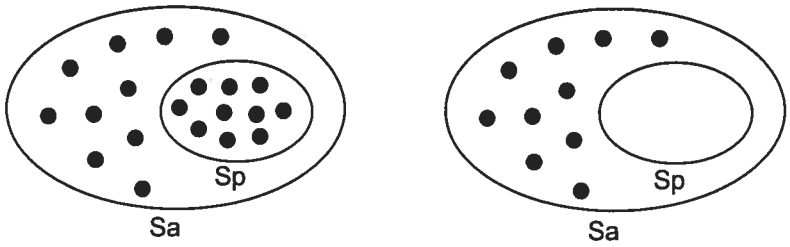

Figure 6-4 : Ensemble de solutions potentielles et sous-ensemble de solutions acceptables

Le raisonnement par tâtonnement et essais-erreurs et/ou par fractionnement en buts intermédiaires est habituellement une heuristique, c'est-à-dire un raisonnement formalisé de résolution de problème dont on tient pour plausible, mais non certain, qu'il conduira à la détermination d'une solution satisfaisante du problème. Le recours à une heuristique soulève ici la question

suivante, à savoir, le parcours de l'utilisateur dans l'arborescence nous mènera-t-il nécessairement jusqu'au dernier niveau, et ainsi jusqu'à une solution architecturale "viable"?

Il semblerait bien que la réponse soit non. En effet, dans la première version du système, lorsqu'il s'agissait de générer automatiquement une solution architecturale partielle, le système, par le biais de procédures récursives, modifiait ses choix tant et aussi longtemps qu'il n'arrivait pas à une solution acceptable. Dans notre deuxième version du système, c'est l'utilisateur qui effectue des choix et le système doit composer avec ces données. Il semble donc inévitable que certaines branches de l'arborescence conduisent à une impasse.

À chacun des niveaux, la composition du sous-ensemble Sa dépend des choix qui ont été effectués antérieurement par l'utilisateur. Le processus de sélection-validation des solutions par lequel le système construira le sous-ensemble Sa pourra s'achever de l'une ou l'autre des façons suivantes :

- Soit le sous-ensemble Sa est non-vide, c'est-à-dire que plusieurs solutions sont a priori légitimement candidates, toutes s'avérant satisfaisantes au regard des critères établis. L'utilisateur peut alors effectuer une sélection et poursuivre son parcours dans l'arborescence.
- Soit, au contraire, aucune (ou trop peu) des solutions élaborées ne s'avère satisfaisante. Le sous-ensemble Sa est vide ou contient une solution unique. L'utilisateur ne peut poursuivre son parcours dans l'arborescence.

Comme nous le savons, l'intelligence d'un système complexe est sa capacité à élaborer et à concevoir de façon endogène ou interne ses propres comportements. Lorsque le sous-ensemble Sa serait vide ou ne contiendrait qu'une solution unique, un système intelligent pourrait théoriquement modifier lui-même les matrices de restriction de l'espace de conception. Suite à ces

modifications, le sous-ensemble Sa contiendrait des éléments et l'utilisateur pourrait poursuivre son cheminement dans l'arborescence.

Mais, en tant que modélisateur du système, est-ce réellement ce que nous souhaitons? Le système ne prendrait-il pas alors le contrôle du processus de design au détriment de l'utilisateur? Certes, l'émergence de comportements nouveaux jusqu'alors non pré-programmés enrichirait la gamme des formes d'adaptation possible du système. Il nous apparaît cependant que cet "enrichissement" se ferait au détriment de l'utilisateur.

En effet, il semble souhaitable que ce soit l'utilisateur qui contrôle entièrement la délimitation de l'espace de conception. Prenons un exemple. Après que l'utilisateur ait sélectionné une matrice de distribution des espaces du rez-de-chaussée, le système doit lui présenter des matrices de répartition des espaces à l'étage qui soient compatibles. Si, en fonction de la liste des pièces et de la dimension de la matrice, le système ne peut présenter aucune solution acceptable, il pourrait théoriquement décider de modifier lui-même la matrice de restriction des proximités à l'étage et décider qu'il n'y a plus de mitoyenneté obligatoire entre Mr's room et Mrs's room, ce qui poserait un problème (!).

Lorsque l'utilisateur arrivera à une impasse à un certain niveau de l'arborescence, le système ne pourra donc pas reconsidérer localement la formulation du problème et chercher à concevoir de façon interne ses propres comportements. Deux alternatives s'offriront à l'utilisateur :

- Soit il décidera de modifier l'espace de conception,
- Soit il décidera de remonter à un niveau antérieur dans l'arborescence et d'effectuer de nouveaux choix.

Dans le premier cas, il s'agirait de modifier "manuellement" les matrices de restriction de l'espace de conception, c'est-à-dire en toute connaissance de cause et après mûre réflexion. Dans le deuxième cas, l'utilisateur pourrait

revenir à une étape antérieure du processus. Il ne s'agirait pas ici de cheminer "à contre-courant" dans l'arborescence mais bien de reprendre une partie du processus. Les données qui auraient été accumulées suite à l'étape sélectionnée seraient détruites. L'utilisateur reprendrait une partie du processus en continuant à respecter la séquence des différents niveaux de l'arborescence, tel que nous l'avons illustré à la figure 6-3. Il ferait de nouveaux choix et pourrait ainsi explorer d'autres ramifications de l'arborescence.

Dans un cas comme dans l'autre, notre système remplirait son mandat, qui est, rappelons-le, de stimuler le sens créatif de l'utilisateur. En effet, que l'utilisateur décide de remanier l'espace de conception ou qu'il décide plutôt de revenir sur certaines des décisions qu'il avait prises antérieurement et de prendre un parti différent, il pourra vérifier quelles sont les répercussions de ces modifications sur les solutions proposées par le système. Ces différentes solutions partielles seront autant d'éléments nouveaux avec lesquels le designer sera mis en contact et face auxquels il devra réagir, ce qui aura pour effet (nous l'espérons) d'enrichir sa réflexion.

7. Positionnement de la recherche

Les travaux de U. Flemming (1987), par leur similarité avec les objectifs que nous nous étions fixés, sont ceux qui nous ont le plus inspirés au moment de définir le sujet de recherche de ce mémoire. Plusieurs aspects distinguent toutefois notre approche de celle de ce chercheur.

Plutôt que d'utiliser des règles d'assemblage, nous avons considéré des règles de proximité obligatoire entre les espaces. La méthode que nous proposons permet de spécifier toutes les proximités obligatoires (et partiellement obligatoires), que ce soit au plan horizontal ou vertical. Cette approche nous a permis non seulement de mettre en relation les espaces d'un même niveau, mais également de positionner les circulations verticales.

De plus, nous n'avons pas simplifié le schéma de distribution des espaces en recopiant le même plan d'un étage à l'autre. En effet, le type d'approche que nous proposons permet de travailler non seulement avec des schémas de distribution des espaces différents selon les étages mais également avec un nombre d'espaces différent d'un étage à l'autre.

Nous avons réussi à modéliser un système capable de générer le "squelette" de solutions architecturales valides. Et, pour ce faire, nous n'avons pas eu, comme Flemming, à omettre certaines composantes parce que les règles étaient trop complexes à formuler.

Il est vrai que, dans le cadre de ce mémoire, les solutions architecturales partielles proposées par le système ne tenaient pas compte de l'aspect géométrique. Les modèles de maisons de style Queen Anne générés par Flemming étaient, quant à eux, assez détaillés à ce niveau, ce qui est nettement plus attrayant pour l'utilisateur que les listes de listes générées par notre système (dans sa version actuelle).

Nous nous sommes éloignés de l'approche "kit of part", tel que le recommandait Scott C. Chase (1999). Cependant, nous n'avons pas adopté l'approche de ce chercheur, qui consistait à définir des règles à l'aide des outils de dessin disponibles dans les logiciels dits de CAO. Pour énoncer nos règles d'une façon claire, concise et flexible, nous avons opté pour l'utilisation du langage de programmation fonctionnel : Scheme.

Nous avons contribué à démontrer, d'une part, qu'il était possible de délimiter un espace de conception au moyen de listes et, d'autre part, qu'un langage de programmation est un outil performant pour écrire des procédures capables de générer des solutions architecturales partielles valides.

Tout comme Robert J. Krawczyk (1997), nous avons opté pour la génération de solutions architecturales par le biais d'un système basé sur les règles et sur l'aléatoire, les règles que nous avons choisies d'appliquer étant définies dans l'espace de conception, et l'aléatoire, c'est-à-dire l'imprévisibilité, entrant en jeu lorsque les procédures de sélection étaient invoquées.

Nous avons opté pour l'approche dite "minimaliste" de Piazzalunga et Fitzhorn (1998), ce qui nous a permis d'aborder progressivement la complexité. Le fait d'avoir évacué, dans le cadre de ce mémoire, toute la question de l'interface et de l'algorithme géométrique pour concentrer notre attention sur la modélisation du système générateur de solutions architecturales partielles nous permet de profiter d'une grande flexibilité.

Étant donné que nous avons pris le parti de travailler, non pas sur des formes géométriques, mais plutôt sur des listes, pour développer notre système, celui-ci nous retourne des solutions architecturales sous forme de listes de listes. Comme nous le verrons dans le prochain chapitre, ces listes de listes ou matrices pourraient, dans une phase ultérieure de la recherche, être utilisées de

diverses façons. Elles pourraient être transférées à un logiciel de dessin ou de rendu ou à un moteur de modélisation volumique.

8. Développement de l'approche proposée

8.1 Pistes de recherche

8.1.1 La géométrie de la solution proposée

En aval de la recherche effectuée jusqu'à maintenant, se situe tout le travail sur la géométrie des solutions architecturales et l'exploration des différentes méthodes qui s'offrent à nous pour la visualisation de ces solutions.

Les assemblages de sphères présentés précédemment ne constituent, bien sûr, qu'une façon de représenter le "squelette" d'une solution architecturale. Dans une phase ultérieure de la recherche, il serait intéressant de mettre à contribution la grande richesse géométrique de notre échantillonnage de bâtiments pour enrichir notre espace de conception. Nous pourrions ainsi générer des solutions architecturales tenant compte de l'aspect géométrique.

L'espace de conception pourrait inclure, outre les matrices dont il a été question antérieurement, des spécifications sur les primitives géométriques utilisées. Suite à la génération de la matrice espaces-jonctions, les codes numériques représentant les différents types de jonctions pourraient faire appel à ces assemblages de primitives géométriques. Il serait ainsi possible de présenter à l'utilisateur la volumétrie de la solution architecturale proposée par le système. Il pourrait visualiser les jonctions horizontales (murs, portes, fenêtres, etc.) ainsi que les jonctions verticales (plancher, escaliers, toitures, etc.).

Les jonctions seraient d'abord traitées comme des cloisons. Les jonctions horizontales seraient donc initialement des murs pleins intérieurs et extérieurs. L'espace de conception inclurait pour chaque jonction les informations concernant d'une part la ou les primitives servant à percer la cloison et, d'autre part, les primitives servant à construire le volume qui viendrait combler cette

ouverture. En se référant à notre échantillonnage de bâtiments, il sera ainsi possible de décrire différentes portes, fenêtres, bow-windows et saillies de toutes sortes.

Les matrices 3D et les assemblages de primitives géométriques générés par le système que nous avons programmé pourraient êtres transférées à différents logiciels; à un logiciel de dessin, comme par exemple AutoCAD par le biais de AutoLisp, à un logiciel de modélisation et de rendu graphique comme POV-Ray ou encore à un moteur de modélisation volumique comme ACIS.

Nous pourrions également travailler avec SGDL, un moteur de modélisation volumique conçu par le GRCAO de l'Université de Montréal et qui permet la transmission intelligente de l'information tridimensionnelle en géométrie exacte non-polygonale, son codage, sa modélisation, son décodage et sa visualisation.

Grâce à SGDL, nous pourrions explorer d'autres avenues que le recours à la géométrie polygonale. En effet, ce moteur de modélisation volumique nous permettrait de travailler en géométrie exacte plutôt que sur une approximation des volumes. Nous pourrions également vérifier l'intérêt de travailler avec la géométrie projective puisqu'il serait possible de définir le volume de nos jonctions à l'intérieur d'hexaèdres projectifs qui seraient par la suite déformés pour présenter d'autres façons de les combiner.

Il serait également possible de travailler sur l'hétérogénéité de la matière. En assignant des densités différentes à différents assemblages de primitives, nous serions en mesure de présenter à l'utilisateur une solution architecturale où l'on pourrait visualiser la distinction entre les différents matériaux utilisés.

En somme, notre travail sur la géométrie des solutions architectural nous amènerait à étudier les différents assemblages de primitives pour chacun des

types de jonctions. Il serait également intéressant d'approfondir notre analyse et de se pencher sur l'étude de la volumétrie des détails architecturaux des éléments de notre échantillonnage de bâtiments. Il s'agirait de décomposer en différentes primitives les motifs ornementaux (fer forgé, appareillage de briques, carrelage, etc.) et d'en étudier le mode d'assemblage. Bien entendu, plus le code de programmation volumique des jonctions serait précis, plus la solution architecturale présentée à l'utilisateur aurait un haut niveau de détails.

8.1.2 L'enrichissement de l'espace de conception

Indépendamment de l'aspect géométrique, une autre façon d'enrichir notre espace de conception serait par l'ajout de matrices de restriction supplémentaires. Pour aborder progressivement la complexité, nous pourrions développer le système en incluant, à chaque étape du travail, un ensemble de matrices de plus en plus complet pour délimiter l'espace de conception. Nous pourrions évaluer ensuite la pertinence des solutions proposées et modifier la structure des matrices de restriction en conséquence. Ce travail itératif nous permettrait de raffiner progressivement le type d'informations que l'utilisateur pourrait transmettre au système.

Voyons quelle sera la nature des matrices que nous pourrions définir en vue d'enrichir notre espace de conception. En fait, une grande variété d'informations peut être traduite sous forme matricielle.

Nous pourrions avoir trois catégories de matrices. Dans notre première version du système, nous avions les matrices de restriction des proximités horizontales qui sont des matrices carrées mettant les espaces en relation avec eux-mêmes et avec l'extérieur. Il serait possible d'ajouter des matrices mettant une série d'espaces en relation avec une autre série d'espaces. Par exemple,

nous pourrions inclure une matrice d'orientation des espaces qui spécifierait, pour chaque module, s'il doit être orienté vers le nord, le sud, l'est ou l'ouest, etc.

La deuxième catégorie est constituée des matrices mettant en relation les couples de modules avec les types de jonctions répertoriées. Nous pourrions ajouter des matrices mettant en relation les types de jonctions intérieur-extérieur en relation avec eux-mêmes. Ces matrices indiqueraient, pour chaque type de jonction, quelles jonctions nous ne désirons pas juxtaposer (sur le même niveau ou verticalement) de façon à pouvoir harmoniser les façades en fonctions des critères de l'utilisateur.

Notre troisième catégorie de matrices pourraient être celles mettant en relation les différents espaces avec des valeurs numériques. En effet, nous pourrions, par exemple, avoir la matrice des dimensions minimales et maximales et des ratios longueur / largeur minimaux et maximaux selon les espaces. Ces matrices seraient fournies au système et lui serviraient de balises pour le choix des valeurs acceptables pour le dimensionnement des modules lors de la génération des différentes solutions architecturales.

Suite à la génération de la matrice de répartition des espaces, des listes de dimensions pour les travées horizontales et verticales seraient générées de façon aléatoire. Ces listes seraient ensuite testées de façon à vérifier la concordances avec les dimensionnements maximum et minimum, les ratios largeur / longueur et les superficies acceptables pour chacun des espaces.

Différentes largeurs de travées seraient donc proposées puis testées par le système. Des fonctions récursives nous permettraient de générer différentes listes de dimensions provisoires pour les travées tant et aussi longtemps que nous n'obtiendrions pas des dimensions acceptables pour chacun des modules.

L'enrichissement progressif de l'espace de conception nous permettrait de vérifier si un ensemble de matrices nous permet bel et bien de transmettre au système toutes les informations dont nous avons besoin pour générer une solution architecturale détaillée qui soit en conformité avec les spécifications de l'utilisateur.

8.1.3 L'interface

Le système que nous avons défini génère des matrices 3D servant de "squelette" aux solutions architecturales partielles proposées. Celles-ci, comme nous l'avons vu, sont générées sans l'intervention de l'utilisateur. Dans le chapitre 6, nous avons démontré comment il serait possible de permettre à l'utilisateur d'interagir avec le système en effectuant un parcours dans une arborescence, lui permettant d'arriver éventuellement à une solution architecturale complète.

Bien entendu, cette arborescence aurait de plus en plus de ramifications à mesure que l'on procéderait à l'enrichissement de l'espace de conception. Pour que l'utilisateur puisse interagir avec le système, et donc s'orienter dans cette arborescence, il faudrait développer une interface. Celle-ci pourrait, par exemple, être conçue de façon à présenter à l'utilisateur trois types d'informations distincts.

Une première fenêtre servirait à visualiser les solutions architecturales partielles en 3D. Dès les premières étapes du processus, il serait possible d'avoir recours au mode de visualisation proposé au chapitre 4. Aux premiers niveaux de l'arborescence, l'utilisateur se verrait donc proposer différents assemblages de pastilles de couleurs qui représenteraient la distribution des espaces. Aux derniers niveaux de l'arborescence, il se verrait présenter

différents assemblages de primitives géométriques représentant la volumétrie du bâtiment. À chacune des étapes, il pourrait sélectionner l'option qui lui convient.

La façon de représenter la solution partielle aurait certainement un impact considérable sur la qualité de l'évaluation et des décisions que prendrait l'utilisateur pour poursuivre le parcours dans l'arborescence. Une solution architecturale représentée de façon explicite serait plus facile à évaluer. Il serait donc intéressant que l'utilisateur puisse prendre connaissance des différentes options qui lui sont présentées par le système, et ce, de façon non seulement tridimensionnelle, mais également dynamique, et sélective.

Une seconde fenêtre de l'interface présenterait un schéma de l'arborescence. Cette fenêtre permettrait à l'utilisateur de se situer dans l'arborescence et d'indiquer au système s'il désire remonter à un niveau antérieur. Comme nous l'avons déjà mentionné, il serait important que l'utilisateur puisse revenir sur certaines des décisions qu'il avait prises antérieurement et explorer d'autres ramifications de l'arborescence.

Une troisième fenêtre de l'interface pourrait permettre à l'utilisateur de communiquer avec le système que ce soit pour la sélection des solutions partielles ou pour modifier l'espace de conception. La sélection de l'option pourrait se faire simplement en indiquant le code numérique de la solution sélectionnée. Pour ce qui est des modifications de l'espace de conception, il serait possible de présenter à l'utilisateur un tableau qui serait en quelque sorte une matrice interactive. Celle-ci lui permettrait de sélectionner les espaces qu'il veut utiliser et indiquer, par exemple, quels sont les espaces dont la proximité est obligatoire.

Le système générateur de solutions architecturales que nous avons modélisé propose à l'architecte une nouvelle façon de faire. La délimitation d'un espace de conception et le parcours dans une arborescence permettraient

d'aborder le processus de design sous un angle différent. Cependant, il est clair qu'une interface comme celle que nous venons de décrire demeure assez limitative.

En effet, l'utilisateur ne pourrait communiquer avec le système que par l'entrée de codes numériques et la modification de matrices. Il pourrait par exemple sélectionner les modules avec lesquels il veut travailler, les relations de proximité obligatoire et les types de jonctions acceptables pour chacun des couples de proximité. Il faut être conscient que la géométrie de ces différents types de jonctions serait pré-programmée par le développeur du système.

Il est cependant clair que, pour concevoir un bâtiment, il ne s'agit pas seulement de déterminer de quelle façon nous voulons mettre les espaces en relation mais également de déterminer quelles sont les primitives géométriques que nous souhaitons utiliser et de quelle façon nous voulons les combiner.

Il serait donc souhaitable que l'architecte puisse transmettre au système les informations concernant le choix et l'agencement des primitives. Pour ce faire, il devrait atteindre un excellent niveau de communication avec le système. Il ne serait plus l'utilisateur d'un outil d'aide à la conception mais deviendrait plutôt le concepteur de cet outil. Nous verrons au cours de la prochaine section ce que cette nouvelle tâche impliquerait pour l'architecte.

8.2 Vers une nouvelle perception du rôle de l'architecte?

Au cours des prochaines décennies, l'architecte désirant utiliser l'outil informatique pour l'épauler dans son travail de conception architectural pourrait choisir d'interagir avec l'interface de systèmes conçus par des développeurs mais il pourrait aussi prendre le parti d'élaborer ses propres outils d'aide à la conception.

Comme nous le savons, le problème ne réside plus aujourd'hui dans la capacité de calcul ou de mémoire de l'ordinateur mais dans notre capacité à lui fournir des instructions cohérentes et pertinentes qui seront par la suite interprétées. L'architecte, s'il décide d'élaborer ses propres outils d'aide à la conception, devrait donc chercher à actualiser ses connaissances et son savoir-faire et s'exercer à les énoncer sous forme de "règles". Dans cette optique, s'il arrive à documenter sa démarche, l'architecte pourrait par exemple (dans l'approche que nous proposons) délimiter et enrichir progressivement un espace de conception.

Il est bien évident que les architectes ne souhaitent pas se faire imposer de règles qui auraient pour effet de les brimer dans le processus de création. C'est pourquoi il faut garder en mémoire que, dans le type d'approche que nous proposons, c'est le designer lui-même qui délimite son espace de conception et donc qui dicte les "règles". Il ne s'agit pas ici d'enfermer le processus de conception dans un carcan mais plutôt de tirer profit de l'informatique pour l'enrichir.

Si l'architecte désire élaborer lui-même ses propres outils d'aide à la conception, il devra ensuite traduire, dans un langage interprétable par l'ordinateur, les "règles" qu'il aura formulées. Comme nous l'avons mentionné, un langage de programmation nous semble être l'outil le mieux adapté à cette tâche. Il s'agirait donc pour l'architecte de transcrire ses "règles" en un code qui soit concis, donc optimisé, logique et suffisamment clair pour être modifié par une tierce personne.

Pour arriver à accomplir cette tâche aujourd'hui, il faudrait, bien évidemment, que l'architecte apprenne à maîtriser un langage de programmation (ou à utiliser une interface donnant accès à un langage de programmation). Il est cependant possible que la majorité des professionnels de l'aménagement aient

une certaine réticence à entreprendre un processus d'apprentissage si éloigné de leur champ de compétences.

Quoi qu'il en soit, si l'intérêt pour l'apprentissage de langages de programmation devenait généralisé au sein de la profession, cette évolution pourrait bouleverser complètement non seulement la conception que les architectes ont du processus de design mais également les relations qu'ils entretiennent avec leurs clients et avec leurs confrères. Voyons de quelles façons ces trois aspects pourraient évoluer.

a) Une évolution de la perception du processus de design :

Si les architectes se mettaient à considérer la programmation comme une façon de développer une idée, un concept, ils pourraient, comme nous l'avons mentionné, concevoir ou modifier leurs propres outils d'aide à la conception. Par exemple, pour travailler avec le type de système que nous proposons, l'utilisateur ne se limiterait pas à utiliser exclusivement l'interface. Il pourrait d'une part répertorier de nouveaux types de jonction sous forme de procédures de programmation volumique et, d'autre part, modifier les procédures du système générateur de solutions partielles, dépendamment de sa façon de concevoir l'architecture. On peut penser ici à trois cas de figure.

Certains designers valorisent l'uniformité dans le concept architectural. Ils pourraient modifier l'espace de conception afin que, pour chaque type de jonction, l'ensemble regroupant les assemblages possibles soit divisé en différents sous-ensembles contenant chacun peu d'éléments. En faisant varier la combinaison des sous-ensembles utilisés, il pourrait générer différents types de compositions relativement uniformes.

D'autres architectes, pour leur part, s'intéressent davantage à l'effet saisissant produit par des variations libres. Pour ce type de composition, ils

pourraient programmer des procédures de sélections aléatoires qui seraient appliqués à des ensembles comptant un grand nombre d'éléments.

D'autres designers, enfin, valorisent surtout la façon dont une composition architecturale intéressante peut être conçue en faisant varier les éléments en réponse à des changements de contexte. Les programmes qu'ils écriraient pour générer des compositions de ce genre seraient riches en conditionnelles. Par exemple, si un assemblage a été choisi pour un type de jonction, on spécifiera que tel élément doit être choisi pour la jonction de l'espace mitoyen et ainsi de suite.

b) Une évolution au niveau des relations avec le client :

> Comme le fait remarquer Temy Tidafi (1996),
> "Le contrôle du processus de conception peut constituer pour un architecte une façon de délimiter un domaine de connaissance lui permettant de distinguer sa profession par rapport à d'autres professions. L'architecte est parfois associé à une image d'arrogance et d'ignorance par rapport aux besoins et aux attentes du client."

Cependant, si l'architecte élaborait des outils d'aide à la conception, il pourrait mettre à la disposition du client les outils nécessaires au processus de conception architectural. Le non-architecte pourrait être intéressé à expérimenter ce type de processus. Il pourrait effectuer des choix et en vérifier les conséquences. La tâche de l'architecte serait alors de l'assister dans sa démarche.

c) Une évolution au niveau des relations entre confrères :

Dans le type d'approche que nous proposons, à la fin du processus de design, un architecte pourrait possiblement mettre à la disposition de tous le code, c'est-à-dire les listes de l'espace de conception et les procédures du système générateur de solutions architectural, ayant été utilisé pour la conception d'un bâtiment donné. Ce code pourrait être réutilisé et éventuellement modifié par d'autres afin de générer de nouvelles solutions architecturales dans le même esprit.

Dans ce contexte, il est permis de croire que les architectes qui seraient familiers avec les langages de programmation pourraient avoir accès au code des systèmes générateurs de solutions architecturales des bâtiments conçus par leurs confrères, un peu comme il est actuellement possible d'accéder au code HTML des pages web. L'utilisation de l'informatique et notamment de la programmation volumique n'est certes pas la panacée mais elle pourrait, dans ce contexte, favoriser une circulation de l'information qui aurait peut-être pour effet d'enrichir le processus de design.

Au cours des prochaines décennies, si les professionnels de l'aménagement ne s'intéressent pas à l'informatique, et en l'occurrence à la programmation, l'architecte et le programmeur continueront à travailler séparément. Les programmeurs proposeront peut-être alors des outils d'aide à la conception permettant de générer différentes options d'aménagement à partir de contraintes pré-fixées. Ces contraintes étant différentes pour chaque projet, ces outils pourront-il être assez flexibles pour être utiles? Les architectes ne préféreront-ils pas s'en tenir au processus de design architectural traditionnel?

L'évolution de la perception que l'on a d'un outil étant un processus plus long que le développement de la technologie, il est évidemment prématuré de penser que l'ensemble des architectes s'intéresseront à la programmation, à

court terme. Il est permis de croire qu'au cours des prochaines années, les architectes continueront à utiliser les outils de dessin assisté par ordinateur que sont les logiciels dits de CAO, puisque ceux-ci font actuellement partie intégrante de la pratique en architecture.

Au cours des prochaines décennies, il serait cependant souhaitable que les architectes s'investissent progressivement à concevoir et/ou maîtriser des outils informatiques qui améliorent réellement la qualité et la diversité de la production architecturale.

Conclusion

Notre recherche a permis de démontrer que la programmation fonctionnelle peut être mise à contribution pour générer des solutions architecturales sous forme matricielle. Suite à l'élaboration de notre système générateur de solutions architecturales, nous avons généré une matrice 3D qui nous a permis de valider notre approche.

Pour assembler les modules et donc générer les solutions partielles, nous avons eu recours à des procédures de sélection aléatoire. Notre approche s'apparente à la métaphore des aimants de H. Von Foerster citée par Le Moigne (1990) dans "La modélisation des systèmes complexes" :

> *"On met dans une boîte des cubes dont certaines faces sont magnétisées positivement et négativement de façon a priori aléatoire. On l'agite de façon également désordonnée. En ouvrant la boîte, on voit qu'une structure étonnamment ordonnée a émergé. En injectant du bruit dans cette organisation initialement pauvre, on l'a apparemment enrichie. Des formes nouvelles ont émergées, que le système tient pour potentiellement signifiantes".*

Lorsque nous demandons au système de générer une matrice 3D, nous voyons apparaître des assemblages inédits qui n'auraient pas nécessairement été prévisibles par composition linéaire. C'est ce type d'effets "contre-intuitifs" qui est peut-être le plus intéressant pour le designer, dans l'outil d'aide à la conception que nous avons cherché à modéliser.

Cette première version de notre système ne génère que des solutions architecturales partielles qui ne constituent en quelques sortes que le "squelette" de la solution architecturale complète. Bien que nous ayons soumis au lecteur l'hypothèse qu'en poursuivant notre démarche, il serait possible d'arriver

éventuellement à une solution architecturale complète, nous sommes conscients du fait que cette hypothèse reste à vérifier.

Comme nous l'avons mentionné, notre approche se distingue de celle de Flemming (1987) entre autres parce que notre système ne génère pas de modèles de maisons mais plutôt des matrices. Cette distanciation par rapport à la géométrie nous a permis de centrer notre réflexion sur le processus pouvant générer des assemblages. Nous avons atteint un certain niveau d'abstraction, ce qui nous a permis d'adopter une approche plus générale.

En effet, les codes numériques réfèrent ici à différents espaces utilisés pour concevoir une solution architecturale. Cependant, le même système de génération-validation de matrices pourrait avoir différentes utilités. Il pourrait, par exemple, être utilisé en urbanisme pour proposer des schémas d'interrelations entre différents secteurs urbains ou encore, en design industriel, pour les interrelations entre les différentes composantes d'un objet. Dans ce contexte, la même approche pourrait même être utilisée pour générer un sous-assemblage à l'intérieur d'un des modules de l'assemblage.

Le but de notre recherche était de mettre à contribution l'informatique dans les premières phases du processus de design architectural. Nous avons proposé une méthode basée sur l'utilisation de procédures de sélection aléatoire pour générer des solutions partielles sous forme matricielle. Il sera intéressant de poursuivre la recherche dans ce sens, mais également dans d'autres directions puisqu'il reste évidemment beaucoup d'autres avenues à explorer pour faire de l'ordinateur un véritable outil d'aide à la conception architecturale.

Bibliographie

CHASE, Scott C. "Grammar Based Design: Issues for User Interaction Models" Acadia 1999

CHASE, Scott C. "Design Modeling With Shape Algebra and Formal Logic" Acadia 1996

DARLYRMPLE, Michael J. et Gerzso J. Micheal "Executable Drawings: The Computation of Digital Architecture" Acadia 1998

ELEB, Monique "Architectures de la vie privée. XVII^{e}-XIX^{e} siècles" Éditions Hazan, Paris, 1999.

FLEMMING, U. "More than the sum of parts: the grammar of Queen Anne houses" *Environment and Planning B: Planning and Design* 1987, volume 14, pages 323-350.

FLEMMING, U. "Get with the program: common fallacies in critiques of computer-aided architectural design" *Environment and Planning B: Planning and Design* 1994, volume 21, pages s106-s116.

FRIEDMAN, P. Daniel et FELLEISEN, Matthias, "The Little Schemer" MIT Press ed. 1996.

GAGNON-PRATTE, France "Les maisons de campagne des montréalais 1892-1924. L'architecture des frères Maxwell" Éditions du Mériridien, 1987.

KAUFMANN, A. "Des points et des flèches... la théorie des graphes" Science-Poche, Dunod 1968.

KRAWCZYK, Robert J. "Programs as Pencils: Investigating Form Generation" Acadia 1997.

LE MOIGNE, J-L, "La modélisation des systèmes complexes" Afcet Systèmes, Dunod 1990.

MITCHELL, William J. "Computer Aided Architectural Design" Petrocelli/Charter, New-York 1987.

MITCHELL, William J. "The art of computer graphics programming: a structured introduction for architects and designers" Van Nostrand Reinhold, New York 1987.

MITCHELL, William J. "The logic of architecture: design, computation and cognition" Cambridge, Mass. M.I.T. Press, 1990.

MURRAY, Irena "Edward & W.S. Maxwell. Guide du fonds." Collection d'architecture canadienne Université McGill 1986.

PIAZZALUNGA, U. et FITZHORN, P. "Note on a three dimensional shape grammar interpreter" *Environment and Planning B: Planning and Design* 1998, volume 25, pages 11-30.

RÉMILLARD, François et MERRETT, Brian "Demeures bourgeoises de Montréal. Le Mille Carré Doré 1850-1930" Éditions du Méridiens 1986.

SEEBOHM, Thomas et Wallace William. "Rule-Based Representation of Design in Architectural Practice" Acadia 1997.

SPINGER, Georges et FRIEDMAN, P. Daniel, "SCHEME and the art of programming" MIT Press, Mc Graw-Hill 1989.

STINY, George "What Designers Do That Computers Should" The Electronic design studio: architectural Knowledge and media in the computer era. Cambridge, Mass.: MIT Press 1990.

TIDAFI, Temy "Moyens pour la communication en architecture: proposition de la modélisation d'actions pour la figuration en architecture" Thèse de PH.D. Université de Montréal, 1996.

WRIGHT, Janet "L'architecture pittoresque au Canada" Parcs Canada, 1984.

"L'architecture de Edward et W.S. Maxwell" Musée des Beaux-Arts de Montréal, 1991.

TABLE DES ANNEXES

1. Fonctions diverses

```
;Nom du fichier: fonc-div.scm
;-----------------------------------------------------------------------
;Fichier qui regroupe différentes procédures générales fréquemment
appelées.
;_____________________________________________________________________

;procédure pour retirer un item d'une liste

(define remove-elem
  (lambda (item Ls)
    (cond
      ((null? Ls) '())
      ((equal? (car Ls) item) (cdr Ls))
      (else (cons (car Ls) (remove-elem item (cdr Ls)))))))
;-----------------------------------------------------------------------

;procédure pour déterminer si un item fait partie d'une liste

(define member?
  (lambda (item ls)
    (cond
      ((null? ls) #f)
      (else (or (equal? (car ls) item)
                (member? item (cdr ls)))))))
;-----------------------------------------------------------------------

;procédure pour déterminer quelle est la position d'un item dans une
liste

(define position-sim
  (lambda (ls elem n)
    (cond
      ((null? ls) '())
      ((= elem (car ls)) n)
      (else (position-sim (cdr ls) elem (+ 1 n))))))
;-----------------------------------------------------------------------

;procédure pour déterminer quelles sont les diverses positions d'un item
;dans une liste

(define position-mul

  (lambda (ls elem n)
    (cond
      ((null? ls) '())
      ((= elem (car ls)) (cons n (position-mul (cdr ls) elem (+ 1 n))))
      (else (position-mul (cdr ls) elem (+ 1 n))))))
;-----------------------------------------------------------------------

;procédure visant à mélanger aléatoirement les éléments de la liste

(define mix-1
  (lambda (L1)
```

```
    (cond
      ((null? L1) '())
      (else (let ((x (random (length L1))))
        (cons (list-ref L1 x) (mix-1 (remove-elem (list-ref L1 x)
L1))))))))
;---------------------------------------------------------------------

;procédure visant à mélanger les éléments de la liste de façon
;aléatoire tout en placant l'item dans l'une des positions acceptables
;identifiées

;ls-1 :liste des pieces
;ls-2 :liste des positions acceptables

(define mix-2
  (lambda (ls-1 item ls-2)
    (letrec
        ((ls-3 (mix-1 (remove-elem item ls-1)))
         (fonc-1
          (lambda (ls elem m n)
;m: rang choisi
;n:compteur
            (if
             (= m n) (cons elem ls)
             (cons (car ls) (fonc-1 (cdr ls) elem m (+ 1 n))))))
         (a (random (length ls-2)))
         (b (list-ref ls-2 a)))
      (fonc-1 ls-3 item b 0))))
;---------------------------------------------------------------------

;procédure qui ajoute l'extérieur (100) à la fin d'une liste de pièces.

(define ajout-100
  (lambda (ls)
    (append ls (list 100))))
;---------------------------------------------------------------------

;procédure qui fournit, pour une liste donnée, la liste des éléments
;placés aux endroits fournis par une liste de list-ref

;ls-1 : liste de base
;ls-2 : liste des list-ref

(define extraction
  (lambda (ls-1 ls-2)
    (if
      (null? ls-2) '()
      (cons (list-ref ls-1 (car ls-2))
            (extraction ls-1 (cdr ls-2))))))
;---------------------------------------------------------------------

;Procédure qui recoit la liste des pieces d'un étage (avec extérieur) et
;la matrice des mitoyennetés obligatoires et retourne une liste des
;proximités de même code

;ls: matrice des mitoyennetés obligatoires
;pie: liste des pieces
```

```
(define couple
  (lambda (ls pie x i j)
    (letrec
        ((fonc-1
            (lambda (ls-1 pie-1 x-1 i-1 j-1)
    (cond
      ((null? ls-1) '())
      ((= (car ls-1) x-1) (cons
                        (list (list-ref pie-1 i-1)(list-ref pie-1 j-1))
                        (fonc-1 (cdr ls-1) pie-1 x-1 (+ 1 i-1) j-1)))
      (else (fonc-1 (cdr ls-1) pie-1 x-1 (+ 1 i-1) j-1))))))
    (cond
      ((null? ls) '())
      (else (append (fonc-1 (car ls) pie x i j) (couple (cdr ls) pie x i
(+ 1 j))))))))
;---------------------------------------------------------------------

;Procédure qui élimine les répétitions de la liste

(define remove-inverse
  (lambda (ls)
    (cond
      ((null? ls) '())
      ((member? (reverse (car ls)) (cdr ls))
       (remove-inverse (cdr ls)))
      (else (cons (car ls) (remove-inverse (cdr ls)))))))
;---------------------------------------------------------------------
```

2. Espace de conception – sous-sol

```
;Nom du fichier: DW-SS.scm
;----------------------------------------------------------------------
;Délimitation de l'espace de conception pour le sous-sol, en terme de
;types d'espaces, dimensions des espaces et proximités obligatoires.
;----------------------------------------------------------------------
;But:
;Procédure qui produit pour le sous-sol la liste de l'ensemble des
;proximités obligatoires, la liste regroupant les pièces obligatoires
;et la liste regroupant les pièces optionnelles.
;----------------------------------------------------------------------

(load "fonc-div.scm")
;----------------------------------------------------------------------

;Liste des pièces du sous-sol (toujours placées dans le même ordre)

;0  Hall-ss
;1  Kitchen
;2  Cook's pantry
;3  Servant's hall
;4  Servant's bedroom (avec Bathroom partagé)
;5  Servant's bedroom (avec Bathroom partagé)
;6  Store (+ Cold strorage + Wine cellar)
;7  Furnace room (+ Fuel)

;100  Extérieur

;l'ensemble des pieces avec extérieur
(define ens-pieces-ss
  (list 0 1 2 3 4 5 6 7 100))
;----------------------------------------------------------------------

;matrice des proximités horizontales obligatoires
;0=permis
;1=obligatoire

(define mat-prox-obl-ss
  (list
   (list 0 0 0 0 0 0 0 0 0)
   (list 0 0 1 1 0 0 0 0 1)
   (list 0 1 0 0 0 0 1 0 1)
   (list 0 1 0 0 0 0 0 0 1)
   (list 0 0 0 0 0 1 0 0 1)
   (list 0 0 0 0 1 0 0 0 1)
   (list 0 0 1 0 0 0 0 0 0)
   (list 0 0 0 0 0 0 0 0 0)
   (list 0 1 1 1 1 1 0 0 0)))
;----------------------------------------------------------------------

;matrice des dimensions min et max (à déterminer ultérieurement)

(define mat-dim-ss
  (list
```

```
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)))
;-----------------------------------------------------------------------

;liste des proximités obligatoires pour le sous-sol
(define Ens-prox-obl-SS
  (remove-inverse
  (couple mat-prox-obl-ss ens-pieces-ss 1 0 0)))

;l'ensemble des pieces obligatoires
(define ens-pie-obl-ss
  (list 3 6 7))

;l'ensemble des pieces optionnelles
(define ens-pie-opt-ss
  (list 0 1 2 4 5))
```

3. Espace de conception – rez-de-chaussée

```
;Nom du fichier: DW-RDC.scm
;----------------------------------------------------------------------
;Délimitation de l'espace de conception pour le rez-de-chaussée, en
terme
;de types d'espaces, dimensions des espaces et proximités obligatoires.
;----------------------------------------------------------------------
;But:
;Procédure qui produit pour le r-d-c la liste de l'ensemble des
;proximités obligatoires, la liste regroupant les pièces obligatoires
;et la liste regroupant les pièces optionnelles.
;----------------------------------------------------------------------

(load "fonc-div.scm")
;----------------------------------------------------------------------

;Liste des pièces du rez-de-chaussée
;(toujours placées dans le même ordre)

;10  Hall (incluant vestibule, escalier, alcove, coat-room et lavatory)
;11  Drawing room
;12  Dinning room
;13  Butler's pantry
;14  Kitchen
;15  Billard room
;16  library
;17  Conservatory (ou loggia)

;100  Extérieur

;l'ensemble des pieces avec extérieur
(define ens-pieces-rdc
  (list 10 11 12 13 14 15 16 17 100))
;----------------------------------------------------------------------

;matrice des proximités horizontales obligatoires
;0=permis
;1=obligatoire

(define mat-prox-obl-rdc
  (list
   (list 0 1 1 1 0 1 1 0 1)
   (list 1 0 1 0 0 0 0 0 1)
   (list 1 1 0 1 0 0 0 0 1)
   (list 1 0 1 0 1 0 0 0 0)
   (list 0 0 0 1 0 0 0 0 0)
   (list 1 0 0 0 0 0 0 0 1)
   (list 1 0 0 0 0 0 0 0 1)
   (list 0 0 0 0 0 0 0 0 1)
   (list 1 1 1 0 0 1 1 1 0)))
;----------------------------------------------------------------------

;matrice des dimensions min et max (à déterminer ultérieurement)
(define mat-dim-rdc
```

```
  (list
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)))
;-----------------------------------------------------------------------

;liste des proximités obligatoires pour le rdc
(define Ens-prox-obl-rdc
  (remove-inverse
  (couple mat-prox-obl-rdc ens-pieces-rdc 1 0 0)))

;l'ensemble des pieces obligatoires
(define ens-pie-obl-rdc
  (list 10 11 12 13))

;l'ensemble des pieces optionnelles
(define ens-pie-opt-rdc
  (list 14 15 16 17))
```

4. Espace de conception – étage

```
;Nom du fichier: DW-eta.scm
;-----------------------------------------------------------------------
;Délimitation de l'espace de conception pour le premier étage, en terme
;de types d'espaces, dimensions des espaces et proximités obligatoires.
;-----------------------------------------------------------------------
;But:
;Procédure qui produit pour l'étage la liste de l'ensemble des
;proximités obligatoires, la liste regroupant les pièces obligatoires
;et la liste regroupant les pièces optionnelles.
;-----------------------------------------------------------------------

(load "fonc-div.scm")
;-----------------------------------------------------------------------

;Liste des pièces de l'étage
;(toujours placées dans le même ordre)

;20  Upper Hall
;21  Mrs's room + dressing room
;22  Mr's room + dressing room
;23  Guest's room + dressing room
;24  Bedroom1 (bathroom partagé)
;25  Bedroom2 (bathroom partagé)
;26  library
;27  Sitting room

;100  Extérieur

;l'ensemble des pieces avec extérieur
(define ens-pieces-eta
  (list 20 21 22 23 24 25 26 27 100))
;-----------------------------------------------------------------------

;matrice des proximités horizontales obligatoires
;0=permis
;1=obligatoire
;2= ensemble et/ou. Les deux chambres peuvent communiquer donc une des
;deux ou les deux doivent donner sur le hall
;3= ensemble et/ou.  Le Sitting room peut donner sur la chambre de Mrs
;ou sur le hall ou sur les deux.
;les proximités code 2 et code 3 seront appelées proximités
partiellement
;obligatoire

(define mat-prox-obl-eta
  (list
   (list 0 1 1 1 2 2 1 3 0)
   (list 1 0 1 0 0 0 0 3 1)
   (list 1 1 0 0 0 0 0 0 1)
   (list 1 0 0 0 0 0 0 0 1)
   (list 2 0 0 0 0 0 0 0 1)
   (list 2 0 0 0 0 0 0 0 1)
```

```
   (list 1 0 0 0 0 0 0 0 1)
   (list 3 3 0 0 0 0 0 0 1)
   (list 0 1 1 1 1 1 1 1 0)))
;-----------------------------------------------------------------------

;matrice des dimensions min et max (à déterminer ultérieurement)
(define mat-dim-eta
  (list
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)))
;-----------------------------------------------------------------------

;liste des proximités obligatoires pour le premier étage
(define Ens-prox-obl-eta
  (remove-inverse
  (couple mat-prox-obl-eta ens-pieces-eta 1 0 0)))

;couples de proximité et/ou
(define Ens-prox-obl-2-eta
  (remove-inverse
  (couple mat-prox-obl-eta ens-pieces-eta 2 0 0)))

(define Ens-prox-obl-3-eta
  (remove-inverse
  (couple mat-prox-obl-eta ens-pieces-eta 3 0 0)))
;-----------------------------------------------------------------------

;l'ensemble des pieces obligatoires
(define ens-pie-obl-eta
  (list 20 21 22 23))

;l'ensemble des pieces optionnelles
(define ens-pie-opt-eta
  (list 24 25 26 27))
```

5. Espace de conception – attic

```
;Nom du fichier: DW-att.scm
;-----------------------------------------------------------------------
;Délimitation de l'espace de conception pour le deuxième étage (attic),
;en terme de types d'espaces, dimensions des espaces et proximités
obligatoires.
;-----------------------------------------------------------------------
;But:
;Procédure qui produit pour le attic la liste de l'ensemble des
;proximités obligatoires, la liste regroupant les pièces obligatoires
;et la liste regroupant les pièces optionnelles.
;-----------------------------------------------------------------------

(load "fonc-div.scm")
;-----------------------------------------------------------------------

;Liste des pièces du attic
;(toujours placées dans le même ordre)

;30  Hall attic
;31  Trunk room
;32  Bedroom + dressing room
;33  Bedroom1 (bathroom partagé)
;34  Bedroom2 (bathroom partagé)
;35  Sitting room
;36  Billard room
;37  Servant's room
;38  Servant's room

;100  Extérieur

;l'ensemble des pieces avec extérieur
(define ens-pieces-att
  (list 30 31 32 33 34 35 36 37 38 100))
;-----------------------------------------------------------------------

;matrice des proximités horizontales obligatoires
;0=permis
;1= proximités obligatoires
;2= proximités partiellement obligatoires
;3= proximités partiellement obligatoires

(define mat-prox-obl-att
  (list
   (list 0 1 1 2 2 3 1 0 0 0)
   (list 1 0 0 0 0 0 0 1 1 0)
   (list 1 0 0 0 0 3 0 0 0 1)
   (list 2 0 0 0 0 0 0 0 0 1)
   (list 2 0 0 0 0 0 0 0 0 1)
   (list 3 0 3 0 0 0 0 0 0 1)
   (list 1 0 0 0 0 0 0 0 0 1)
   (list 0 1 0 0 0 0 0 0 0 1)
   (list 0 1 0 0 0 0 0 0 0 1)
```

```
   (list 0 0 1 1 1 1 1 1 1 0)))


;matrice des dimensions min et max (à déterminer ultérieurement)
(define mat-dim-att
  (list
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)
   (list 0 0)))
;---------------------------------------------------------------------

;liste des proximités obligatoires pour le attic
(define Ens-prox-obl-att
  (remove-inverse
  (couple mat-prox-obl-att ens-pieces-att 1 0 0)))

;couples de proximité et/ou
(define Ens-prox-obl-2-att
  (remove-inverse
  (couple mat-prox-obl-att ens-pieces-att 2 0 0)))

(define Ens-prox-obl-3-att
  (remove-inverse
  (couple mat-prox-obl-att ens-pieces-att 3 0 0)))
;---------------------------------------------------------------------

;l'ensemble des pieces obligatoires
(define ens-pie-obl-att
  (list 30 31))

;l'ensemble des pieces optionnelles
(define ens-pie-opt-att
  (list 32 33 34 35 36 37 38))
;---------------------------------------------------------------------
```

6. Choix des modules

```
;Nom du fichier: liste-pieces-prov.scm
;-----------------------------------------------------------------------
;Procédure qui recoit pour chaque étage la liste regroupant les pièces
;obligatoires et la liste regroupant les pièces optionnelles.
;-----------------------------------------------------------------------
;But: Retourne pour chaque étage une liste provisoire regroupant les
;5 ou 6 pièces choisies pour un étage.
;-----------------------------------------------------------------------

(load "DW-SS.scm")
(load "DW-RDC.scm")
(load "DW-eta.scm")
(load "DW-att.scm")
;-----------------------------------------------------------------------

;procédure d'aide pour construire les listes provisoires de pièces

;n: nombre de pieces pour l'étage (5 ou 6)
;ls-1: ensemble des pieces obligatoires
;ls-2: ensemble des pieces optionnelles

(define ls-pieces
  (lambda (n ls-1 ls-2)
    (letrec
        ((a (length ls-2))
         (b (list-ref ls-2 (random a)))
         (c (remove-elem b ls-2))
         (d (list-ref c (random (- a 1))))
         (e (remove-elem d c))
         (f (list-ref e (random (- a 2))))
         (g (remove-elem f e))
         (h (list-ref g (random (- a 3)))))
      (cond
        ((= (- n (length ls-1)) 1) (append ls-1 (list b)))
        ((= (- n (length ls-1)) 2) (append ls-1 (list b d)))
        ((= (- n (length ls-1)) 3) (append ls-1 (list b d f)))
        ((= (- n (length ls-1)) 4) (append ls-1 (list b d f h)))
        ))))
;-----------------------------------------------------------------------

;génération de la liste provisoire pour le rez-de-chaussée

(define ls-pieces-RDC-prov
  (lambda ()
  (list-ref (list (ls-pieces 5 ens-pie-obl-rdc ens-pie-opt-rdc)
                  (ls-pieces 6 ens-pie-obl-rdc ens-pie-opt-rdc))
            (random 2))))
;-----------------------------------------------------------------------

;génération de la liste provisoire pour le sous-sol

(define ls-pieces-SS-prov
```

```
  (list-ref (list (ls-pieces 5 ens-pie-obl-ss ens-pie-opt-ss)
                  (ls-pieces 6 ens-pie-obl-ss ens-pie-opt-ss))
            (random 2))))
;---------------------------------------------------------------------

;génération de la liste provisoire pour l'étage

(define ls-pieces-eta-prov
  (lambda ()
  (list-ref (list (ls-pieces 5 ens-pie-obl-eta ens-pie-opt-eta)
                  (ls-pieces 6 ens-pie-obl-eta ens-pie-opt-eta))
            (random 2))))
;---------------------------------------------------------------------

;génération de la liste provisoire pour le attic

(define ls-pieces-att-prov
  (lambda ()
  (list-ref (list (ls-pieces 5 ens-pie-obl-att ens-pie-opt-att)
                  (ls-pieces 6 ens-pie-obl-att ens-pie-opt-att))
            (random 2))))
;---------------------------------------------------------------------
```

7. Vérification de la cohérence du choix des modules

```
;Nom du fichier: verif-pieces.scm
;-----------------------------------------------------------------------
;Procédure qui recoit pour chaque étage la liste provisoire des pieces
;choisies.
;-----------------------------------------------------------------------
;But: Vérifier la cohérence entre les quatre étages et retourne les
;listes de pièces définitives.
;-----------------------------------------------------------------------

(load "listes-pieces-prov.scm")
;-----------------------------------------------------------------------

;Généreration de la liste des pièces du rdc qui servira de base
;de comparaison pour la génération des autres listes.

(define ls-pieces-rdc
  (lambda ()
  (ls-pieces-rdc-prov)))
;-----------------------------------------------------------------------

;Génération et validation de la liste des pièces du sous-sol

;Vérifications à faire:
;Kitchen (codé 1 au sous-sol ou 14 au rez-de chaussée) doit apparaitre
;mais ne doit pas se retrouver à plusieurs étages:
;Si il y a Kitchen (14) au rdc, il ne doit y avoir ni Kitchen (1), ni
Cook's pantry (2)
;au sous-sol,
;S'il n'y a pas Kitchen (14) au rdc, il doit aussi y avoir Kitchen (1)
et Cook's pantry (2)
;au sous-sol

;ls-1 : liste définitive des pièces du rdc
;ls-2 : liste provisoire des pièces du ss

(define verif-ls-pie-ss
  (lambda (ls-1)
    (let
        ((ls-2 (ls-pieces-ss-prov)))
    (if
     (or
      (and
       (member? 14 ls-1)
       (not (member? 1 ls-2))
       (not (member? 2 ls-2)))
      (and
       (not (member? 14 ls-1))
       (member? 1 ls-2)
       (member? 2 ls-2)))
      ls-2
     (verif-ls-pie-ss ls-1)))))
;-----------------------------------------------------------------------
```

```
;Génération et validation de la liste des pièces de l'étage
;Library  16 ou 26 est optionnelle mais ne doit apparaitre qu'a un seul
étage

;ls-1: liste définitive des pièces du rdc
;ls-2: liste provisoire des pièces de l'étage

(define verif-ls-pie-eta
  (lambda (ls-1)
    (let
        ((ls-2 (ls-pieces-eta-prov)))
    (if
      (and (member? 16 ls-1) (member? 26 ls-2))
      (verif-ls-pie-eta ls-1)
      ls-2))))
;---------------------------------------------------------------------

;Génération et validation de la liste des pièces de l'attic

;Sitting room (27 et 35) et Billard room (15 et 36) sont optionnels mais
ne doivent
;apparaitre qu'à un seul étage

;ls-1: liste définitive du rdc
;ls-2: liste définitive de l'étage
;ls-3: liste provisoire de l'attic

(define verif-ls-pie-att
  (lambda (ls-1 ls-2)
    (let
        ((ls-3 (ls-pieces-att-prov)))
    (if
     (or
      (and (member? 35 ls-3) (member? 27 ls-2))
      (and (member? 36 ls-3) (member? 15 ls-1)))
      (verif-ls-pie-att ls-1 ls-2)
      ls-3))))
;---------------------------------------------------------------------
```

8. Génération des listes de modules et de couples de proximité

```
;Nom du fichier: prox-ls-pieces.scm
;-----------------------------------------------------------------------
;Procédure qui recoit pour chaque étage la liste de 5 ou 6 pièces
sélectionnées
;et vérifiées et la liste de l'ensemble des proximités obligatoires pour
;cet étage (code 1) et des proximités partiellement obligatoires (code 2
ou 3).
;-----------------------------------------------------------------------
;But: Retourne pour chaque étage une liste de  2 ou 3 listes.
;La première sous-liste est la liste de 5 ou 6 pièces sélectionnées.
;La deuxième sous-liste est la liste des proximitées obligatoires
;pour cette liste.
;La troisième liste regroupe les listes de proximités partiellement
;obligatoires si applicable.
;-----------------------------------------------------------------------

(load "verif-pieces.scm")
;-----------------------------------------------------------------------

;Procédure qui vérifie si les 2 éléments du couple d'une proximité
;obligatoire se retrouvent dans la liste des pièces choisies et
;retourne la liste des proximités obligatoire applicables.

;ls1 liste des pièces
;ls2 ensemble des prox obligatoires pour l'étage

(define triage
  (lambda (ls ls2)
    (let ((ls1 (ajout-100 ls)))  ;ajout de de l'extérieur
    (cond
      ((null? ls2) '())
      ((and (member? (car (car ls2)) ls1)
            (member? (cadr (car ls2)) ls1))
       (cons (car ls2) (triage ls1 (cdr ls2))))
      (else (triage ls1 (cdr ls2)))))))
;-----------------------------------------------------------------------

;Quand on a une liste code 2 ou code 3, il peut arriver 3 choses
;soit les membre des deux couples se trouvent dans la liste des piéces,
;soit seulement le premier couple
;soit seulement le deuxième couple
;soit aucun des deux

;Procédure d'aide visant à déterminer quel est le cas qui s'applique
;(1 2 3 ou 4)

;ls-a :liste des pieces
;ls-b :une liste de deux couples de proximité partiellement obligatoire

(define test
  (lambda (ls-a ls-b)
     (if
     (and
```

```
      (member? (car (car ls-b)) ls-a)
      (member? (cadr (car ls-b)) ls-a)
      (member? (car (cadr ls-b)) ls-a)
      (member? (cadr (cadr ls-b)) ls-a))
     1

      (if
       (and
        (member? (car (car ls-b)) ls-a)
        (member? (cadr (car ls-b)) ls-a))
       2

       (if
       (and
        (member? (car (cadr ls-b)) ls-a)
        (member? (cadr (cadr ls-b)) ls-a))
       3
       4)))))
;---------------------------------------------------------------------

;Procédure d'aide qui retourne la liste des pièces, la liste des
proximités
;obligatoires applicables et la liste des proximités partiellement
;obligatoires si applicable

;ls-1: liste des pieces
;ls-2: liste des prox triées
;ls-3: liste des listes de proximités partiellement obligatoires
;ls-4: liste des listes de proximités partiellement obligatoires
admissibles
;qui sera retournée par la procédure

(define triage-code
  (lambda (ls-1 ls-2 ls-3 ls-4)
    (cond
      ((null? ls-3) (list ls-1 ls-2 ls-4))
      ((= 1 (test ls-1 (car ls-3)))
       (triage-code ls-1 ls-2 (cdr ls-3) (cons (car ls-3) ls-4)))
      ((= 2 (test ls-1 (car ls-3)))
       (triage-code ls-1 (cons (car (car ls-3)) ls-2) (cdr ls-3) ls-4))
      ((= 3 (test ls-1 (car ls-3)))
       (triage-code ls-1 (cons (cadr (car ls-3)) ls-2) (cdr ls-3) ls-4))
      ((= 4 (test ls-1 (car ls-3)))
       (triage-code ls-1 ls-2 (cdr ls-3) ls-4))
      )))
;---------------------------------------------------------------------

;Procédure qui retourne la liste de pièces du rdc et la liste
;des proximités obligatoires applicables

;a: liste de pieces de rdc

(define ls-prox-pieces-rdc
  (lambda (a)
  (cons a
  (list (triage a Ens-prox-obl-rdc)))))
;---------------------------------------------------------------------
```

```
;Procédure qui retourne la liste de pièces du sous-sol et la liste
;des proximités obligatoires applicables
;(Chaque fois qu'on rappelle la procédure on a une nouvelle liste de
;pieces pour le sous-sol même si on a toujours la même liste de pièces
;pour le rdc.)

;a: liste définitive de pieces rdc

(define ls-prox-pieces-ss
  (lambda (a)
    (let
        ((x (verif-ls-pie-ss a)))
  (cons x
  (list (triage x Ens-prox-obl-ss))))))
;---------------------------------------------------------------------

;Procédure qui retourne la liste de pièces de l'étage, la liste
;des proximités obligatoires applicables et la liste des listes de
;proximités partiellement obligatoires si applicable.

;a: liste définitive de pieces rdc

(define ls-prox-pieces-eta
  (lambda (a)
    (let
        ((x (verif-ls-pie-eta a)))
  (triage-code x
               (triage x Ens-prox-obl-eta)
               (list Ens-prox-obl-2-eta Ens-prox-obl-3-eta)
               '()))))
;---------------------------------------------------------------------

;Procédure qui retourne la liste de pièces de l'attic, la liste
;des proximités obligatoires applicables et la liste des listes de
;proximités partiellement obligatoires si applicable.

;a:liste définitive de pieces rdc
;b:liste définitive de pieces etage

(define ls-prox-pieces-att
  (lambda (a b)
    (let
        ((x (verif-ls-pie-att a b)))
  (triage-code x
               (triage x Ens-prox-obl-att)
               (list Ens-prox-obl-2-att Ens-prox-obl-3-att)
               '()))))
;---------------------------------------------------------------------
```

9. Définition des listes de répartition

```
;Nom du fichier: mat.mitchell.scm
;----------------------------------------------------------------------
;But: Procédure qui défini les différentes listes de répartition selon
;les diagrammes de subdivision de Mitchell.
;----------------------------------------------------------------------

;A partir de chaque de chaque diagramme de subdivision en 5 sous-
espaces,
;on dérive 5 listes de répartition en dédoublant soit la première
rangée,
;la première colonne, la dernière rangée ou la dernière colonne de façon
;à obtenir une variété de matrices 5 espaces approchant celle des
matrices
;6 espaces et d'obtenir une certaine uniformité dans les dimensions des
;matrices.

;Le premier élément d'une sous-liste est la liste de répartition et le
;deuxième, la dimension de la matrice.

(define ls-mat
  (list
   ;liste des matrices 5 espaces
    (list (list (list 0 1 2 1 3 1 4 1) "2x4")        ;1-1
          (list (list 0 1 0 1 2 1 3 1 4 1) "2x5")    ;1-2
          (list (list 0 1 2 1 3 1 4 1 4 1) "2x5")    ;1-3
          (list (list 0 0 1 2 2 1 3 3 1 4 4 1) "3x4");1-4
          (list (list 0 1 1 2 1 1 3 1 1 4 1 1) "3x4");1-5

          (list (list 0 1 2 3 0 1 2 4) "4x2")        ;2-1
          (list (list 0 1 2 3 0 1 2 3 0 1 2 4) "4x3");2-2
          (list (list 0 1 2 3 0 1 2 4 0 1 2 4) "4x3");2-3
          (list (list 0 1 2 3 3 0 1 2 4 4) "5x2")    ;2-4
          (list (list 0 0 1 2 3 0 0 1 2 4) "5x2")    ;2-5

          (list (list 0 1 2 3 0 1 4 4) "4x2")        ;3-1
          (list (list 0 1 2 3 0 1 2 3 0 1 4 4) "4x3");3-2
          (list (list 0 1 2 3 0 1 4 4 0 1 4 4) "4x3");3-3
          (list (list 0 0 1 2 3 0 0 1 4 4) "5x2")    ;3-4
          (list (list 0 1 2 3 3 0 1 4 4 4) "5x2")    ;3-5

          (list (list 0 1 2 3 0 4 4 4) "4x2")        ;4-1
          (list (list 0 1 2 3 0 1 2 3 0 4 4 4) "4x3");4-2
          (list (list 0 1 2 3 0 4 4 4 0 4 4 4) "4x3");4-3
          (list (list 0 0 1 2 3 0 0 4 4 4) "5x2")    ;4-4
          (list (list 0 1 2 3 3 0 4 4 4 4) "5x2")    ;4-5

          (list (list 0 1 2 0 3 2 0 4 2) "3x3")      ;5-1
          (list (list 0 1 2 0 1 2 0 3 2 0 4 2) "3x4");5-2
          (list (list 0 1 2 0 3 2 0 4 2 0 4 2) "3x4");5-3
          (list (list 0 0 1 2 0 0 3 2 0 0 4 2) "4x3");5-4
          (list (list 0 1 2 2 0 3 2 2 0 4 2 2) "4x3");5-5
```

```
(list (list 0 1 2 0 1 3 0 1 4) "3x3")      ;6-1
(list (list 0 1 2 0 1 2 0 1 3 0 1 4) "3x4");6-2
(list (list 0 1 2 0 1 3 0 1 4 0 1 4) "3x4");6-3
(list (list 0 0 1 2 0 0 1 3 0 0 1 4) "4x3");6-4
(list (list 0 1 2 2 0 1 3 3 0 1 4 4) "4x3");6-5

(list (list 0 1 2 3 2 4) "2x3")            ;7-1
(list (list 0 1 0 1 2 3 2 4) "2x4")        ;7-2
(list (list 0 1 2 3 2 4 2 4) "2x4")        ;7-3
(list (list 0 0 1 2 2 3 2 2 4) "3x3")      ;7-4
(list (list 0 1 1 2 3 3 2 4 4) "3x3")      ;7-5

(list (list 0 1 2 1 3 1 3 4) "2x4")        ;8-1
(list (list 0 1 0 1 2 1 3 1 3 4) "2x5")    ;8-2
(list (list 0 1 2 1 3 1 3 4 3 4) "2x5")    ;8-3
(list (list 0 0 1 2 2 1 3 3 1 3 3 4) "3x4");8-4
(list (list 0 1 1 2 1 1 3 1 1 3 4 4) "3x4");8-5

(list (list 0 0 0 1 2 3 1 2 4) "3x3")      ;9-1
(list (list 0 0 0 0 0 0 1 2 3 1 2 4) "3x4");9-2
(list (list 0 0 0 1 2 3 1 2 4 1 2 4) "3x4");9-3
(list (list 0 0 0 0 1 1 2 3 1 1 2 4) "4x3");9-4
(list (list 0 0 0 0 1 2 3 3 1 2 4 4) "4x3");9-5

(list (list 0 0 0 1 2 3 1 4 4) "3x3")      ;10-1
(list (list 0 0 0 0 0 0 1 2 3 1 4 4) "3x4");10-2
(list (list 0 0 0 1 2 3 1 4 4 1 4 4) "3x4");10-3
(list (list 0 0 0 0 1 1 2 3 1 1 4 4) "4x3");10-4
(list (list 0 0 0 0 1 2 3 3 1 4 4 4) "4x3");10-5

(list (list 0 1 2 3 0 4 2 3) "4x2")        ;11-1
(list (list 0 1 2 3 0 1 2 3 0 4 2 3) "4x3");11-2
(list (list 0 1 2 3 0 4 2 3 0 4 2 3) "4x3");11-3
(list (list 0 0 1 2 3 0 0 4 2 3) "5x2")    ;11-4
(list (list 0 1 2 3 3 0 4 2 3 3) "5x2")    ;11-5

(list (list 0 1 2 3 4 2) "3x2")            ;12-1
(list (list 0 1 2 0 1 2 3 4 2) "3x3")      ;12-2
(list (list 0 1 2 3 4 2 3 4 2) "3x3")      ;12-3
(list (list 0 0 1 2 3 3 4 2) "4x2")        ;12-4
(list (list 0 1 2 2 3 4 2 2) "4x2")        ;12-5

(list (list 0 1 2 3 1 4) "3x2")            ;13-1
(list (list 0 1 2 0 1 2 3 1 4) "3x3")      ;13-2
(list (list 0 1 2 3 1 4 3 1 4) "3x3")      ;13-3
(list (list 0 0 1 2 3 3 1 4) "4x2")        ;13-4
(list (list 0 1 2 2 3 1 4 4) "4x2")        ;13-5

(list (list 0 1 2 3 1 2 3 4 4) "3x3")      ;14-1
(list (list 0 1 2 0 1 2 3 1 2 3 4 4) "3x4");14-2
(list (list 0 1 2 3 1 2 3 4 4 3 4 4) "3x4");14-3
(list (list 0 0 1 2 3 3 1 2 3 3 4 4) "4x3");14-4
(list (list 0 1 2 2 3 1 2 2 3 4 4 4) "4x3");14-5

(list (list 0 1 2 0 1 3 0 4 4) "3x3")      ;15-1
(list (list 0 1 2 0 1 2 0 1 3 0 4 4) "3x4");15-2
```

```
(list (list 0 1 2 0 1 3 0 4 4 0 4 4) "3x4");15-3
(list (list 0 0 1 2 0 0 1 3 0 0 4 4) "4x3");15-4
(list (list 0 1 2 2 0 1 3 3 0 4 4 4) "4x3");15-5

(list (list 0 1 2 3 0 4 4 3) "4x2")         ;16-1
(list (list 0 1 2 3 0 1 2 3 0 4 4 3) "4x3");16-2
(list (list 0 1 2 3 0 4 4 3 0 4 4 3) "4x3");16-3
(list (list 0 0 1 2 3 0 0 4 4 3) "5x2")     ;16-4
(list (list 0 1 2 3 3 0 4 4 3 3) "5x2")     ;16-5

(list (list 0 1 2 3 3 2 3 3 4) "3x3")       ;17-1
(list (list 0 1 2 0 1 2 3 3 2 3 3 4) "3x4");17-2
(list (list 0 1 2 3 3 2 3 3 4 3 3 4) "3x4");17-3
(list (list 0 0 1 2 3 3 3 2 3 3 3 4) "4x3");17-4
(list (list 0 1 2 2 3 3 2 2 3 3 4 4) "4x3");17-5

(list (list 0 0 1 2 3 2 3 4) "2x4")         ;18-1
(list (list 0 0 0 0 1 2 3 2 3 4) "2x5")     ;18-2
(list (list 0 0 1 2 3 2 3 4 3 4) "2x5")     ;18-3
(list (list 0 0 0 1 1 2 3 3 2 3 3 4) "3x4");18-4
(list (list 0 0 0 1 2 2 3 2 2 3 4 4) "3x4");18-5

(list (list 0 1 1 0 2 3 0 4 4) "3x3")       ;19-1
(list (list 0 1 1 0 1 1 0 2 3 0 4 4) "3x4");19-2
(list (list 0 1 1 0 2 3 0 4 4 0 4 4) "3x4");19-3
(list (list 0 0 1 1 0 0 2 3 0 0 4 4) "4x3");19-4
(list (list 0 1 1 1 0 2 3 3 0 4 4 4) "4x3");19-5

(list (list 0 0 1 2 3 1 2 4 4) "3x3")       ;20-1
(list (list 0 0 1 0 0 1 2 3 1 2 4 4) "3x4");20-2
(list (list 0 0 1 2 3 1 2 4 4 2 4 4) "3x4");20-3
(list (list 0 0 0 1 2 2 3 1 2 2 4 4) "4x3");20-4
(list (list 0 0 1 1 2 3 1 1 2 4 4 4) "4x3");20-5

(list (list 0 1 2 0 3 2 0 3 4) "3x3")       ;21-1
(list (list 0 1 2 0 1 2 0 3 2 0 3 4) "3x4");21-2
(list (list 0 1 2 0 3 2 0 3 4 0 3 4) "3x4");21-3
(list (list 0 0 1 2 0 0 3 2 0 0 3 4) "4x3");21-4
(list (list 0 1 2 2 0 3 2 2 0 3 4 4) "4x3");21-5

(list (list 0 1 0 2 3 2 3 4) "2x4")         ;22-1
(list (list 0 1 0 1 0 2 3 2 3 4) "2x5")     ;22-2
(list (list 0 1 0 2 3 2 3 4 3 4) "2x5")     ;22-3
(list (list 0 0 1 0 0 2 3 3 2 3 3 4) "3x4");22-4
(list (list 0 1 1 0 2 2 3 2 2 3 4 4) "3x4");22-5

;on ajoute des mat 3x2 et 2x3 en raison du nombre insuffisant
(list (list 0 1 1 2 3 4) "3x2") ;23-1
(list (list 0 1 2 3 3 4) "3x2") ;23-2
(list (list 0 0 1 2 3 4) "3x2") ;23-3
(list (list 0 1 2 3 4 4) "3x2") ;23-4
(list (list 0 1 0 2 3 4) "2x3") ;23-5
(list (list 0 1 2 1 3 4) "2x3") ;23-6
(list (list 0 1 2 3 4 3) "2x3") ;23-7
(list (list 0 1 2 2 3 4) "2x3") ;23-8
(list (list 0 0 1 2 3 4) "2x3") ;23-9
)
```

```
;-------------------------------------------------------------------------
   (list
   ;liste des matrices 6 espaces
     (list (list 0 1 2 3 4 0 1 2 3 5) "5x2")      ;1
     (list (list 0 1 2 3 4 0 1 2 5 5) "5x2")      ;2
     (list (list 0 1 2 3 4 0 1 5 5 5) "5x2")      ;3
     (list (list 0 1 2 3 4 0 5 5 5 5) "5x2")      ;4
     (list (list 0 1 2 0 3 2 0 4 2 0 5 2) "3x4") ;5
     (list (list 0 1 2 0 1 3 0 1 4 0 1 5) "3x4") ;6
     (list (list 0 1 2 3 2 4 2 5) "2x4")          ;7
     (list (list 0 1 0 2 3 4 3 5) "2x4")          ;8
     (list (list 0 1 2 1 3 1 4 1 4 5) "2x5")      ;9
     (list (list 0 0 0 0 1 2 3 4 1 2 3 5) "4x3");10
     (list (list 0 0 0 0 1 2 3 4 1 2 5 5) "4x3");11
     (list (list 0 0 0 0 1 2 3 4 1 5 5 5) "4x3");12
     (list (list 0 1 2 3 4 0 5 2 3 4) "5x2")     ;13
     (list (list 0 1 2 3 4 5 2 3) "4x2")         ;14
     (list (list 0 1 2 3 4 1 5 3) "4x2")         ;15
     (list (list 0 1 2 3 4 1 2 5) "4x2")         ;16
     (list (list 0 1 2 3 4 1 5 5) "4x2")         ;17
     (list (list 0 1 2 3 4 1 2 3 4 5 5 5) "4x3");18
     (list (list 0 1 2 3 0 1 2 4 0 1 2 5) "4x3");19
     (list (list 0 1 2 3 0 1 2 4 0 1 5 5) "4x3");20
     (list (list 0 1 2 3 0 1 2 4 0 5 5 5) "4x3");21
     (list (list 0 1 2 3 4 4 2 3 5 5 5 5) "4x3");22
     (list (list 0 1 2 3 4 0 5 5 3 4) "5x2")     ;23
     (list (list 0 1 2 3 4 4 5 3) "4x2")         ;24
     (list (list 0 1 2 3 4 4 2 3 4 4 5 5) "4x3");25
     (list (list 0 0 1 1 2 3 4 3 4 5) "2x5")     ;26
     (list (list 0 1 2 2 0 1 3 4 0 1 5 5) "4x3");27
     (list (list 0 1 2 2 0 1 3 4 0 5 5 5) "4x3");28
     (list (list 0 1 2 3 0 1 4 4 0 1 5 5) "4x3");29
     (list (list 0 1 2 3 0 1 4 4 0 5 5 5) "4x3");30
     (list (list 0 0 1 2 3 4 1 2 3 4 5 5) "4x3");31
     (list (list 0 0 1 2 3 4 1 2 3 5 5 5) "4x3");32
     (list (list 0 1 2 3 4 4 4 3 5 5 5 5) "4x3");33
     (list (list 0 1 2 3 4 0 5 5 5 4) "5x2")     ;34
     (list (list 0 1 2 3 4 5 5 3) "4x2")         ;35
     (list (list 0 1 2 3 4 4 4 3 4 4 4 5) "4x3");36
     (list (list 0 0 1 2 3 2 4 2 4 5) "2x5")     ;37
     (list (list 0 1 1 1 0 2 3 4 0 5 5 5) "4x3");38
     (list (list 0 1 2 3 0 4 4 4 0 5 5 5) "4x3");39
     (list (list 0 0 0 1 2 3 4 1 2 5 5 5) "4x3");40
     (list (list 0 1 2 3 0 4 2 3 0 5 2 3) "4x3");41
     (list (list 0 1 2 3 4 2 3 5 2) "3x3")       ;42
     (list (list 0 1 2 0 3 2 0 4 2 0 4 5) "3x4");43
     (list (list 0 1 2 3 4 2 5 4 2) "3x3")        ;44
     (list (list 0 1 2 3 1 4 5 1 4) "3x3")       ;45
     (list (list 0 1 2 3 1 2 4 1 2 4 5 5) "3x4");46
     (list (list 0 0 0 1 1 1 2 3 4 2 5 5) "3x4");47
     (list (list 0 1 2 3 4 5) "2x3")             ;48
     (list (list 0 1 2 3 4 3 4 5) "2x4")         ;49
     (list (list 0 1 1 2 3 4 2 3 5) "3x3")       ;50
     (list (list 0 1 1 2 3 4 2 5 5) "3x3")       ;51
     (list (list 0 1 2 3 3 4 3 3 5) "3x3")       ;52
     (list (list 0 1 2 3 4 4 3 5 5) "3x3")       ;53
     (list (list 0 1 2 1 3 4 3 5) "2x4")         ;54
```

```
(list (list 0 1 2 1 3 4 3 5) "2x4")         ;55
(list (list 0 0 1 2 3 4 2 5 5) "3x3")       ;56
(list (list 0 1 0 2 0 3 4 3 5 5) "2x5")     ;57
(list (list 0 1 2 0 1 3 0 1 4 0 5 4) "3x4");58
(list (list 0 1 0 2 0 3 4 3 4 5) "2x5")     ;59
(list (list 0 1 2 2 3 3 3 4 3 3 3 5) "4x3");60
(list (list 0 1 0 2 3 2 4 2 4 5) "2x5")     ;61
(list (list 0 1 1 1 2 2 3 4 2 2 3 5) "4x3");62
(list (list 0 1 1 1 2 2 3 4 2 2 5 5) "4x3");63
(list (list 0 1 2 1 3 1 3 4 3 5) "5x2")     ;64
(list (list 0 0 0 1 2 3 4 4 2 3 5 5) "4x3");65
(list (list 0 0 0 1 2 3 4 4 2 5 5 5) "4x3");66
(list (list 0 0 0 1 2 3 4 2 3 5 5 5) "3x4");67
(list (list 0 1 1 1 0 2 3 4 0 5 3 4) "4x3");68
(list (list 0 0 0 1 2 3 4 5 3) "3x3")       ;69
(list (list 0 0 0 1 2 3 4 2 5) "3x3")       ;70
(list (list 0 0 0 1 2 3 4 2 3 4 5 5) "3x4");71
(list (list 0 1 1 0 2 2 0 3 4 0 5 5) "3x4");72
(list (list 0 0 0 1 2 3 1 2 4 1 5 5) "3x4");73
(list (list 0 0 0 1 2 3 4 4 3 5 5 5) "3x4");74
(list (list 0 1 1 1 0 2 3 4 0 5 5 4) "4x3");75
(list (list 0 0 0 1 2 3 4 4 3 4 4 5) "3x4");76
(list (list 0 1 1 0 2 3 0 4 3 0 4 5) "3x4");77
(list (list 0 1 1 0 2 3 0 4 3 0 5 5) "3x4");78
(list (list 0 1 2 3 4 0 1 5 3 4) "5x2")     ;79
(list (list 0 1 2 3 0 4 5 3) "4x2")         ;80
(list (list 0 1 2 3 0 1 4 3 0 1 4 5) "4x3");81
(list (list 0 1 2 3 0 1 4 3 0 5 5 5) "4x3");82
(list (list 0 1 2 3 0 4 2 3 0 4 5 5) "4x3");83
(list (list 0 1 2 3 4 2 3 4 5) "3x3")       ;84
(list (list 0 1 2 3 1 4 3 5 5) "3x3")       ;85
(list (list 0 1 2 0 1 3 4 4 3 5 5 5) "3x4");86
(list (list 0 1 2 3 0 1 2 4 0 5 5 4) "4x3");87
(list (list 0 1 2 0 1 3 4 4 3 4 4 5) "3x4");88
(list (list 0 1 1 2 2 3 4 4 3 4 4 5) "3x4");89
(list (list 0 1 1 0 2 3 4 2 3 4 5 5) "3x4");90
(list (list 0 0 1 2 3 1 2 3 4 2 5 5) "3x4");91
(list (list 0 1 2 3 1 2 4 4 2 4 4 5) "3x4");92
(list (list 0 0 0 1 1 2 3 4 2 3 5 5) "3x4");93
(list (list 0 1 1 0 2 3 0 2 4 0 5 5) "3x4");94
(list (list 0 1 2 3 0 4 4 3 0 4 4 5) "4x3");95
(list (list 0 1 2 0 3 3 4 3 3 5 5 5) "3x4");96
(list (list 0 1 2 3 0 1 4 4 0 5 4 4) "4x3");97
(list (list 0 1 2 0 3 3 4 3 3 4 5 5) "3x4");98
(list (list 0 1 1 2 1 1 3 3 4 3 3 5) "3x4");99
(list (list 0 0 1 0 0 2 3 4 2 3 5 5) "3x4");100
(list (list 0 0 1 2 3 3 4 3 3 4 5 5) "3x4");101
(list (list 0 1 1 2 1 1 2 3 4 2 5 5) "3x4");102
(list (list 0 0 1 2 1 3 4 3 5 5) "2x5")     ;103
(list (list 0 1 1 0 2 3 0 2 4 0 5 4) "3x4");104
(list (list 0 0 1 2 1 3 4 3 4 5) "2x5")     ;105
(list (list 0 1 2 2 0 3 3 4 0 5 5 5) "4x3");106
(list (list 0 1 1 2 0 3 4 4 0 5 5 5) "4x3");107
(list (list 0 1 1 2 0 3 4 2 0 5 5 2) "4x3");108
(list (list 0 1 2 3 0 1 4 3 0 5 4 3) "4x3");109
(list (list 0 1 2 0 3 2 4 3 5) "3x3")       ;110
(list (list 0 1 2 0 3 2 4 3 2 4 5 5) "3x4");111
```

```
     (list (list 0 1 2 0 1 3 0 4 3 0 4 5) "3x4");112
     (list (list 0 1 2 3 1 2 3 4 2 3 4 5) "3x4");113
     (list (list 0 1 2 0 3 2 0 3 4 0 5 4) "3x4");114
     (list (list 0 1 2 1 2 3 4 3 4 5) "2x5")    ;115
;on ajoute une mat 3x2 en raison du nombre insuffisant
     (list (list 0 1 2 3 4 5) "3x2") ;116
     )))
```

10. Classement des listes de répartition

```
;Nom du fichier: triage-matrices.scm
;-----------------------------------------------------------------------
;Procédure qui recoit l'ensemble des différentes listes de répartion
;définies selon les diagrammes de subdivision de Mitchell.
;-----------------------------------------------------------------------
;But: Fait le tri parmi les listes de répartition, selon la dimension de
la
;matrice qui sera générée et les rassemble en une liste de listes.
;-----------------------------------------------------------------------

(load "mat-mitchell.scm")
;-----------------------------------------------------------------------

;Procédure d'aide qui classifie les listes de répartition selon les
dimensions
;de la matrice

(define triage-mat
  (lambda (ls item)
    (cond
      ((null? ls) '())
      ((equal? item (cadr (car ls)))
       (cons (car (car ls)) (triage-mat (cdr ls) item)))
      (else (triage-mat (cdr ls) item)))))
;-----------------------------------------------------------------------

;Ensemble de procédures qui classifient les listes de répartition selon
;la dimension de la matrice et selon le nombre d'espaces.
;Chaque procédure génère une liste regroupant dans une première sous-
liste
;les matrices 5 espaces et dans une deuxième sous-liste les matrices 6
espaces.

(define ls-mat-3x3
  (list (triage-mat (car ls-mat) "3x3")
        (triage-mat (cadr ls-mat) "3x3")))

(define ls-mat-4x3
  (list (triage-mat (car ls-mat) "4x3")
        (triage-mat (cadr ls-mat) "4x3")))

(define ls-mat-3x4
  (list (triage-mat (car ls-mat) "3x4")
        (triage-mat (cadr ls-mat) "3x4")))

(define ls-mat-3x2
  (list (triage-mat (car ls-mat) "3x2")
        (triage-mat (cadr ls-mat) "3x2")))

(define ls-mat-2x3
  (list (triage-mat (car ls-mat) "2x3")
        (triage-mat (cadr ls-mat) "2x3")))
```

```
(define ls-mat-4x2
  (list (triage-mat (car ls-mat) "4x2")
        (triage-mat (cadr ls-mat) "4x2")))

(define ls-mat-2x4
  (list (triage-mat (car ls-mat) "2x4")
        (triage-mat (cadr ls-mat) "2x4")))

(define ls-mat-5x2
  (list (triage-mat (car ls-mat) "5x2")
        (triage-mat (cadr ls-mat) "5x2")))

(define ls-mat-2x5
  (list (triage-mat (car ls-mat) "2x5")
        (triage-mat (cadr ls-mat) "2x5")))
;-----------------------------------------------------------------------

;Procédure qui rassemble en une liste toutes les sous-listes définies
ci-dessus.

(define ls-mat-compl
  (list
   ls-mat-3x3
   ls-mat-4x3
   ls-mat-3x4
   ls-mat-3x2
   ls-mat-2x3
   ls-mat-4x2
   ls-mat-2x4
   ls-mat-5x2
   ls-mat-2x5))
;-----------------------------------------------------------------------
```

11. Sélection de la liste de répartition

```
;Nom du fichier: choix-ls-rep.scm
;-----------------------------------------------------------------------
;But: Procédure qui choisi aléatoirement une liste de répartition
;-----------------------------------------------------------------------

(load "triage-matrices.scm")
(load "prox-ls-pieces.scm")
;-----------------------------------------------------------------------

;Procédure qui chosit une matrice dans n'importe quelle sous-section
;pour le rdc selon le nombre de pieces et qui retourne la liste de
répartition
;et le code de grandeur de la matrice.

;ls:liste de toutes les grandeurs de matrices
;nbr-pie: nombre de pieces de la liste des pièces du rdc (5+ext ou
6+ext)

(define choix-1
  (lambda (ls nbr-pie)
    (letrec
        ((n (random (length ls)))
         (m (random (length (car (list-ref ls n)))))
         (o (random (length (cadr (list-ref ls n))))))
      (cond
        ((= nbr-pie 6)
         (list (list-ref (car (list-ref ls n)) m) n))
        ((= nbr-pie 7)
         (list (list-ref (cadr (list-ref ls n)) o) n))
        ))))
;-----------------------------------------------------------------------

;Procédure qui choisit une matrice d'une grandeur particulière pour un
étage
;suite à la sélection de la matrice du rdc

;ls:liste de toutes les sous-sections
;nbr-pie: nombre de pieces (5+ext ou 6+ext)
;n: code de grandeur de la matrice choisie pour le rdc

(define choix-2
  (lambda (ls nbr-pie n)
    (letrec
        ((m (random (length (car (list-ref ls n)))))
         (o (random (length (cadr (list-ref ls n))))))
      (cond
        ((= nbr-pie 6)
         (list (list-ref (car (list-ref ls n)) m) n))
        ((= nbr-pie 7)
         (list (list-ref (cadr (list-ref ls n)) o) n))
        ))))
;-----------------------------------------------------------------------
```

12. Génération de la matrice de répartition

```
;Nom du fichier: cons-mat.scm
;-----------------------------------------------------------------------
;But: Procédure qui génère la matrice de répartition des modules à
;partir de la liste des modules et de la liste de répartition
;-----------------------------------------------------------------------

(load "choix-ls-rep.scm")
;-----------------------------------------------------------------------

;Ensemble de procédures qui construisent les matrices de différentes
;grandeurs.

;ls-1:liste des pieces
;ls-2:liste de répartition

(define mat-3x3
  (lambda (ls-1 ls-2)
    (let
        ((case (lambda (n)
                 (list-ref Ls-1 (list-ref Ls-2 n)))))
      (list
       (list (case 0) (case 1) (case 2))
       (list (case 3) (case 4) (case 5))
       (list (case 6) (case 7) (case 8))
       ))))


(define mat-4x3
  (lambda (ls-1 ls-2)
    (let
        ((case (lambda (n)
                 (list-ref Ls-1 (list-ref Ls-2 n)))))
      (list
       (list (case 0) (case 1) (case 2) (case 3))
       (list (case 4) (case 5) (case 6) (case 7))
       (list (case 8) (case 9) (case 10) (case 11))
       ))))

(define mat-3x4
  (lambda (ls-1 ls-2)
    (let
        ((case (lambda (n)
                 (list-ref Ls-1 (list-ref Ls-2 n)))))
      (list
       (list (case 0) (case 1) (case 2))
       (list (case 3) (case 4) (case 5))
       (list (case 6) (case 7) (case 8))
       (list (case 9) (case 10) (case 11))
       ))))

(define mat-3x2
  (lambda (ls-1 ls-2)
    (let
```

```
        ((case (lambda (n)
                 (list-ref Ls-1 (list-ref Ls-2 n)))))
      (list
       (list (case 0) (case 1) (case 2))
       (list (case 3) (case 4) (case 5))
       ))))

(define mat-2x3
  (lambda (ls-1 ls-2)
    (let
        ((case (lambda (n)
                 (list-ref Ls-1 (list-ref Ls-2 n)))))
      (list
       (list (case 0) (case 1))
       (list (case 2) (case 3))
       (list (case 4) (case 5))
       ))))

(define mat-4x2
  (lambda (ls-1 ls-2)
    (let
        ((case (lambda (n)
                 (list-ref Ls-1 (list-ref Ls-2 n)))))
      (list
       (list (case 0) (case 1) (case 2) (case 3))
       (list (case 4) (case 5) (case 6) (case 7))
       ))))

(define mat-2x4
  (lambda (ls-1 ls-2)
    (let
        ((case (lambda (n)
                 (list-ref Ls-1 (list-ref Ls-2 n)))))
      (list
       (list (case 0) (case 1))
       (list (case 2) (case 3))
       (list (case 4) (case 5))
       (list (case 6) (case 7))
       ))))

(define mat-5x2
  (lambda (ls-1 ls-2)
    (let
        ((case (lambda (n)
                 (list-ref Ls-1 (list-ref Ls-2 n)))))
      (list
       (list (case 0) (case 1) (case 2) (case 3) (case 4))
       (list (case 5) (case 6) (case 7) (case 8) (case 9))
       ))))

(define mat-2x5
  (lambda (ls-1 ls-2)
    (let
        ((case (lambda (n)
                 (list-ref Ls-1 (list-ref Ls-2 n)))))
      (list
       (list (case 0) (case 1))
```

```
       (list (case 2) (case 3))
       (list (case 4) (case 5))
       (list (case 6) (case 7))
       (list (case 8) (case 9))
       ))))
;-------------------------------------------------------------------------

;Procédure qui construit la matrice de répartition des espaces en
fonction
;du code de grosseur fourni.

;ls-1:liste des pieces
;ls-2:liste de répartition
;n: code de grosseur de la mat

(define formation-mat
  (lambda (ls-1 ls-2 n)
    (cond
      ((= n 0) (mat-3x3 ls-1 ls-2))
      ((= n 1) (mat-4x3 ls-1 ls-2))
      ((= n 2) (mat-3x4 ls-1 ls-2))
      ((= n 3) (mat-3x2 ls-1 ls-2))
      ((= n 4) (mat-2x3 ls-1 ls-2))
      ((= n 5) (mat-4x2 ls-1 ls-2))
      ((= n 6) (mat-2x4 ls-1 ls-2))
      ((= n 7) (mat-5x2 ls-1 ls-2))
      ((= n 8) (mat-2x5 ls-1 ls-2))
      )))
;-------------------------------------------------------------------------
```

13. Génération de la liste des couples de jonctions

```
;Nom du fichier:prox-mat.scm
;-------------------------------------------------------------
;Procédure qui recoit une liste contenant une matrice de répartion
;des espaces et le code de dimension de la matrice
;-------------------------------------------------------------
;But: construit la liste des couples de jonction de
;la matrice de répartition.
;-------------------------------------------------------------

(load "cons-mat.scm")
;-------------------------------------------------------------------------

;Procédure construisant une matrice 5x5 à partir de n'importe quelle
;grosseur de matrice, en vue d'une uniformisation du processus de
;formation de la liste des couples de jonction.

;Il sera ultérieurement possible de modifier cette procédure afin de
;tenir compte de l'orientation des pièces sur les façades Nord, Sud,
;Est ou Ouest.

;ls: matrice de répartition

(define remplissage
  (lambda (ls)
    (letrec
        ((Liste (list 100 100 100 100 100))
         (fonc-1
          (lambda (ls-1)
            (cond
              ((= 2 (length ls-1)) (append ls-1 (cddr Liste)))
              ((= 3 (length ls-1)) (append ls-1 (cdddr Liste)))
              ((= 4 (length ls-1)) (append ls-1 (cddddr Liste)))
              ((= 5 (length ls-1)) ls-1))))
         (fonc-2
          (lambda (ls-2)
            (cond
              ((= 2 (length ls-2))
               (cons (fonc-1 (list-ref ls-2 0))
                     (cons (fonc-1 (list-ref ls-2 1))
                           (list Liste Liste Liste))))

              ((= 3 (length ls-2))
               (cons (fonc-1 (list-ref ls-2 0))
                     (cons (fonc-1 (list-ref ls-2 1))
                           (cons (fonc-1 (list-ref ls-2 2))
                                 (list Liste Liste)))))

              ((= 4 (length ls-2))
               (cons  (fonc-1 (list-ref ls-2 0))
                      (cons (fonc-1 (list-ref ls-2 1))
                            (cons  (fonc-1 (list-ref ls-2 2))
                                   (cons (fonc-1 (list-ref ls-2 3))
                                         (list Liste))))))
```

```
              ((= 5 (length ls-2))
               (cons (fonc-1 (list-ref ls-2 0))
                     (cons (fonc-1 (list-ref ls-2 1))
                           (cons (fonc-1 (list-ref ls-2 2))
                                 (cons (fonc-1 (list-ref ls-2 3))
                                       (cons (fonc-1 (list-ref ls-2 4))
                                             '()))))))
              ))))
      (fonc-2 ls))))
;---------------------------------------------------------------------

;Procédure qui définit toutes les 60 jonctions de la matrice 5x5

;ls: matrice de répartition avec remplissage extérieur

(define ens-jonctions
  (lambda (ls)
    (let
        ((f (lambda (liste a b)
              (list-ref (list-ref liste a) b))))
    (list
     '()
(list 100 (f ls 0 0))              ;j-1
(list 100 (f ls 0 1))              ;j-2
(list 100 (f ls 0 2))              ;j-3
(list 100 (f ls 0 3))              ;j-4
(list 100 (f ls 0 4))              ;j-5

(list 100 (f ls 0 0))              ;j-6
(list (f ls 0 0) (f ls 0 1))       ;j-7
(list (f ls 0 1) (f ls 0 2))       ;j-8
(list (f ls 0 2) (f ls 0 3))       ;j-9
(list (f ls 0 3) (f ls 0 4))       ;j-10
(list (f ls 0 4) 100)              ;j-11

(list (f ls 0 0) (f ls 1 0))       ;j-12
(list (f ls 0 1) (f ls 1 1))       ;j-13
(list (f ls 0 2) (f ls 1 2))       ;j-14
(list (f ls 0 3) (f ls 1 3))       ;j-15
(list (f ls 0 4) (f ls 1 4))       ;j-16

(list 100 (f ls 1 0))              ;j-17
(list (f ls 1 0) (f ls 1 1))       ;j-18
(list (f ls 1 1) (f ls 1 2))       ;j-19
(list (f ls 1 2) (f ls 1 3))       ;j-20
(list (f ls 1 3) (f ls 1 4))       ;j-21
(list (f ls 1 4) 100)              ;j-22

(list (f ls 1 0) (f ls 2 0))       ;j-23
(list (f ls 1 1) (f ls 2 1))       ;j-24
(list (f ls 1 2) (f ls 2 2))       ;j-25
(list (f ls 1 3) (f ls 2 3))       ;j-26
(list (f ls 1 4) (f ls 2 4))       ;j-27

(list 100 (f ls 2 0))              ;j-28
(list (f ls 2 0) (f ls 2 1))       ;j-29
```

```
(list (f ls 2 1) (f ls 2 2))      ;j-30
(list (f ls 2 2) (f ls 2 3))      ;j-31
(list (f ls 2 3) (f ls 2 4))      ;j-32
(list (f ls 2 4) 100)             ;j-33

(list (f ls 2 0) (f ls 3 0))      ;j-34
(list (f ls 2 1) (f ls 3 1))      ;j-35
(list (f ls 2 2) (f ls 3 2))      ;j-36
(list (f ls 2 3) (f ls 3 3))      ;j-37
(list (f ls 2 4) (f ls 3 4))      ;j-38

(list 100 (f ls 3 0))             ;j-39
(list (f ls 3 0) (f ls 3 1))      ;j-40
(list (f ls 3 1) (f ls 3 2))      ;j-41
(list (f ls 3 2) (f ls 3 3))      ;j-42
(list (f ls 3 3) (f ls 3 4))      ;j-43
(list (f ls 3 4) 100)             ;j-44

(list (f ls 3 0) (f ls 4 0))      ;j-45
(list (f ls 3 1) (f ls 4 1))      ;j-46
(list (f ls 3 2) (f ls 4 2))      ;j-47
(list (f ls 3 3) (f ls 4 3))      ;j-48
(list (f ls 3 4) (f ls 4 4))      ;j-49

(list 100 (f ls 4 0))             ;j-50
(list (f ls 4 0) (f ls 4 1))      ;j-51
(list (f ls 4 1) (f ls 4 2))      ;j-52
(list (f ls 4 2) (f ls 4 3))      ;j-53
(list (f ls 4 3) (f ls 4 4))      ;j-54
(list (f ls 4 4) 100)             ;j-55

(list (f ls 4 0) 100)             ;j-56
(list (f ls 4 1) 100)             ;j-57
(list (f ls 4 2) 100)             ;j-58
(list (f ls 4 3) 100)             ;j-59
(list (f ls 4 4) 100)))))        ;j-60
;-----------------------------------------------------------------------

;Ensemble de procédures qui forment les listes de jonctions pertinentes
;selon la grosseur de la matrice.

(define liste3x3
  (list 1 2 3 6 7 8 9 12 13 14 17 18 19 20 23 24 25 28 29 30 31 34 35
36))

(define liste4x3
  (list 1 2 3 4 6 7 8 9 10 12 13 14 15 17 18 19 20 21 23 24 25 26 28 29
30 31 32 34 35 36 37))

(define liste3x4
  (list 1 2 3 6 7 8 9 12 13 14 17 18 19 20 23 24 25 28 29 30 31 34 35 36
39 40 41 42 45 46 47))

(define liste3x2
  (list 1 2 3 6 7 8 9 12 13 14 17 18 19 20 23 24 25))

(define liste2x3
```

```
  (list 1 2 6 7 8 12 13 17 18 19 23 24 28 29 30 34 35))

(define liste4x2
  (list 1 2 3 4 6 7 8 9 10 12 13 14 15 17 18 19 20 21 23 24 25 26))

(define liste2x4
  (list 1 2 6 7 8 12 13 17 18 19 23 24 28 29 30 34 35 39 40 41 45 46))

(define liste5x2
  (list 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27))

(define liste2x5
  (list 1 2 6 7 8 12 13 17 18 19 23 24 28 29 30 34 35 39 40 41 45 46 50
51 52 56 57))
;-----------------------------------------------------------------------

;Procédure d'aide qui forme la liste des couples de jonction

;ls-1:matrice de répartition
;ls-2:liste des jonction utilisées pour cette grosseur de matrice

(define prox-mat
  (lambda (ls-1 ls-2)
      (if
        (null? ls-2) '()
        (cons (list-ref ls-1 (car ls-2)) (prox-mat ls-1 (cdr ls-2))))))
;-----------------------------------------------------------------------

;Procédure qui forme la liste des couples de jonction en fonction
;du code de grosseur de la taille de la matrice.

;ls-1: matrice de répartition des espaces
;n: code de dimension de la matrice

(define form-prox-mat
  (lambda (ls-1 n)
    (let
        ((ls-comp (ens-jonctions (remplissage ls-1))))
    (cond
      ((= n 0) (prox-mat ls-comp liste3x3))
      ((= n 1) (prox-mat ls-comp liste4x3))
      ((= n 2) (prox-mat ls-comp liste3x4))
      ((= n 3) (prox-mat ls-comp liste3x2))
      ((= n 4) (prox-mat ls-comp liste2x3))
      ((= n 5) (prox-mat ls-comp liste4x2))
      ((= n 6) (prox-mat ls-comp liste2x4))
      ((= n 7) (prox-mat ls-comp liste5x2))
      ((= n 8) (prox-mat ls-comp liste2x5))
      ))))
;-----------------------------------------------------------------------
```

14. Vérification de la concordance des proximités

```
;Nom du fichier: concordance.scm
;-----------------------------------------------------------------------
;Ensemble de fonctions prédicat qui déterminent si les jonctions d'une
;matrice de répartiton proposée sont conformes aux listes de proximités
;obligatoires et partiellement obligatoires.
;-----------------------------------------------------------------------

(load "fonc-div.scm")
;-----------------------------------------------------------------------

;Fonction prédicat qui détermine si chacune des proximités obligatoires
;est comprise dans la liste des jonctions

;ls-1:liste des proximités obligatoires pour la liste de pieces
;ls-2:liste des jonctions de la matrice de répartition

(define concordance
  (lambda (ls-1 ls-2)
    (cond
      ((null? ls-1) #t)
      ((not
        (or
         (member? (car ls-1) ls-2)
         (member? (reverse (car ls-1)) ls-2))) #f)
      (else (concordance (cdr ls-1) ls-2)))))
;-----------------------------------------------------------------------

;Fonction prédicat qui détermine si au moins l'un des deux couples d'une
;liste de proximité partiellement obligatoire fait partie de la liste
;des jonctions

;ls-1: liste des listes de proximités partiellement obligatoires
;ls-2: liste des jonctions de la matrice de répartition

(define concordance-2
  (lambda (ls-1 ls-2)
    (cond
      ((null? ls-1) #t)
      ((not
        (or
         (or
          (member? (car (car ls-1)) ls-2)
          (member? (reverse (car (car ls-1))) ls-2))
         (or
          (member? (cadr (car ls-1)) ls-2)
          (member? (reverse (cadr (car ls-1))) ls-2)))) #f)
      (else (concordance-2 (cdr ls-1) ls-2)))))
;-----------------------------------------------------------------------

;Fonction prédicat qui détermine si les proximités obligatoires ET les
proximités
;partiellement obligatoires sont respectées.

;ls-1: proximités obligatoires pour la liste de pièces
```

```
;ls-2: liste des listes et/ou admissibles
;ls-3: liste des proximités de la matrice de répartition

(define conc-et/ou
  (lambda (ls-1 ls-2 ls-3)
    (if
     (null? ls-2) (concordance ls-1 ls-3)
     (and
      (concordance ls-1 ls-3)
      (concordance-2 ls-2 ls-3)))))
;---------------------------------------------------------------------
```

15. Validation de la matrice de répartition du rez-de-chaussée

```
;Nom du fichier: select-mat-rdc.scm
;-----------------------------------------------------------------------
;Procédure qui sélectionne et valide la matrice de répartition des
;espaces pour le rez-de-chaussée
;-----------------------------------------------------------------------

(load "concordance.scm")
(load "prox-mat.scm")
;-----------------------------------------------------------------------

;liste: liste de pièces du rez-de-chaussée et liste des proximités
obligatoires
;pour cette liste

(define select-mat-rdc
  (lambda (liste)
    (letrec

;on choisit aléatoirement une liste de répartition
     ((x (choix-1 ls-mat-compl (+ 1 (length (car liste))))))

;on choisit aléatoirement une autre liste de répartition quand
;les précédentes n'ont pas fonctionné

;ls-p-p: liste de pièces du rez-de-chaussée et liste des proximités ;
;obligatoires pour cette liste
;c: compteur pour fonc-1
;d: compteur pour fonc-2

      (fonc-2
        (lambda (ls-p-p c d)
           (let ((w (choix-1 ls-mat-compl (+ 1 (length (car liste))))))
                  (if (< d 10)
                      (fonc-1 ls-p-p c d w)
                      '()))))

;on mélange l'ordre le la liste de pièces de façon aléatoire
;ls-p-p: liste de pièces du rez-de-chaussée et liste des
;proximités obligatoires pour cette liste
;a: compteur pour fonc-1
;b: compteur pour fonc-2
;ls-rep: liste de répartition

      (fonc-1 (lambda (ls-p-p a b ls-rep)
                (let
                    ((z (ajout-100 (mix-1 (car ls-p-p)))))
       (if (< a 10)
        (if
         (concordance (cadr ls-p-p)
          (form-prox-mat
           (formation-mat z (car ls-rep) (cadr ls-rep)) (cadr ls-rep)))
```

```
;résultat si la fonction prédicat est #t:
         (list
          (formation-mat z (car ls-rep) (cadr ls-rep))
;matrice de répartition
          (car ls-rep)
;liste de répartiton
          z
;liste des pièces mélangées
          (cadr ls-rep)
;code de grosseur le la mat
          )

;else du deuxième if
          (fonc-1 ls-p-p (+ 1 a) b ls-rep))
;else du premier if
      (fonc-2 ls-p-p (- a 10) (+ 1 b)))))))

      (fonc-1 liste 0 0 x)
          )))
;---------------------------------------------------------------------

;Procédure qui relance le processus de sélection / validation de
;la matrice de répartition du rdc tant et aussi longtemps qu'on
;n'obtient pas une liste vide.

(define mat-def-rdc
  (lambda ()
         (let
             ((x (select-mat-rdc
                  (ls-prox-pieces-rdc (ls-pieces-rdc)))))
           (if
            (null? x) (mat-def-rdc) x))))
;---------------------------------------------------------------------
```

16. Validation de la matrice de répartition du sous-sol et de l'étage

```
;Nom du fichier: select-mat-ss-eta.scm
;-----------------------------------------------------------------------
;Procédures qui sélectionnent et valident les matrices de répartition
;des espaces pour le sous-sol et l'étage
;-----------------------------------------------------------------------

(load "select-mat-rdc.scm")
;-----------------------------------------------------------------------

;Procédure qui sélectionne et valide la matrice de répartition du
;sous-sol

;ls-1:liste de répartition choisie pour le rdc
;ls-2:liste des pièces du rdc mélangées
;n: code de grosseur de la matrice choisie pour le rdc

(define select-mat-ss
  (lambda (ls-1 ls-2 n)
    (letrec

;on génère une liste des pièces du sous-sol + liste prox obligatoires
        ((w (ls-prox-pieces-ss ls-2))

;on sélectionne une liste de répartition en fonction du code de
;grosseur de la matrice du rdc
         (x (choix-2 ls-mat-compl (+ 1 (length (car w))) n))

;on génére de nouvelles listes de pièces et de répartition lorsque les
;précédentes n'ont pas fonctionné
         (fonc-2
          (lambda (c d)
            (let
                ((ww (ls-prox-pieces-ss ls-2))
                 (xx (choix-2 ls-mat-compl (+ 1 (length (car w))) n)))
              (if
               (< d 10)
               (fonc-1 ww xx c d)
               '())))))

;fonc-1 pour remélanger la liste de pièces
;ls-p-p:liste des pieces et proximités
;ls-rep:liste de répartition

         (fonc-1 (lambda (ls-p-p ls-rep a b)
                   (let
; on veux que le Servant hall #3 soit en dessous du Butler's Pantry #13
                     ((z (ajout-100
                          (mix-2 (car ls-p-p) 3 (extraction (car ls-rep)
                                                (position-mul ls-1
                                                 (position-sim ls-2 13 0)
0))))))
                      (if (< a 10)
                          (if
```

```
                              (concordance (cadr ls-p-p)
                                    (form-prox-mat
                                     (formation-mat z (car ls-rep)
                                                    (cadr ls-rep))
                                                    (cadr ls-rep)))
;résultat si la fonction prédicat est #t:
         (list
          (formation-mat z (car ls-rep) (cadr ls-rep))
;matrice de répartition
          (car ls-rep)
;liste de répartition
          z
;liste des pièces mélangées
          (cadr ls-rep)
;code de grosseur de la mat
          )


                           (fonc-1 ls-p-p ls-rep (+ a 1) b))
                      (fonc-2 (- a 10) (+ b 1))
                           )))))
      (fonc-1 w x 0 0))))
;---------------------------------------------------------------------

;procédure pour sélectionner et valider la matrice de l'étage

;ls-1:liste de répartition choisie pour le rdc
;ls-2:liste des pièces du rdc mélangées
;n: code de grosseur de la matrice choisie pour le rdc

(define select-mat-eta
  (lambda (ls-1 ls-2 n)
    (letrec

;liste des pièces du sous-sol + liste prox obligatoires
        ((w (ls-prox-pieces-eta ls-2))

;liste de répartition
         (x (choix-2 ls-mat-compl (+ 1 (length (car w))) n))

;pour générer de nouvelles listes de pièces et de répartition
         (fonc-2
          (lambda (c d)
            (let
                ((ww (ls-prox-pieces-eta ls-2))
                 (xx (choix-2 ls-mat-compl (+ 1 (length (car w))) n)))
              (if
               (< d 10)
               (fonc-1 ww xx c d)
               '())))))

;fonc-1 pour remélanger la liste de pièces
;ls-p-p:liste des pieces + liste proximités oblig + ls prox part. obl.
;ls-rep:liste de répartition

         (fonc-1 (lambda (ls-p-p ls-rep a b)
                   (let
```

```
; on veux que le Upper Hall #20 soit au-dessus du Hall #10
                    ((z (ajout-100
                         (mix-2 (car ls-p-p) 20 (extraction (car ls-rep)
                                                (position-mul ls-1
                                                 (position-sim ls-2 10 0)
0))))))
                     (if (< a 10)
                         (if
                           (conc-et/ou (list-ref ls-p-p 1)
                                       (list-ref ls-p-p 2)
                                 (form-prox-mat
                                  (formation-mat z (car ls-rep)
                                                 (cadr ls-rep))
                                                 (cadr ls-rep)))

;résultat si la fonction prédicat est #t:
          (list
           (formation-mat z (car ls-rep) (cadr ls-rep))
;matrice de répartition
           (car ls-rep)
;liste de répartition
           z
;liste des pièces mélangées
           (cadr ls-rep)
;code de grosseur de la mat
           )
          (fonc-1 ls-p-p ls-rep (+ 1 a) b)
                       )
                      (fonc-2 (- a 10) (+ b 1))
                         )))))
      (fonc-1 w x 0 0))))
;---------------------------------------------------------------------

;Procédure qui assemble les listes d'informations sur le sous-sol, le
;rdc et l'étage

;u: information sur la disposition des espaces du rdc
;      -matrice de répartition
;      -liste de répartition
;      -liste des pièces mélangées
;      -code de grosseur

(define assemb-mat
  (lambda (u)
    (let
        ((v (select-mat-ss (list-ref u 1) (list-ref u 2)
                                          (list-ref u 3)))
         (w (select-mat-eta (list-ref u 1) (list-ref u 2)
                                           (list-ref u 3))))
      (if
       (or (null? v) (null? w)) (assemb-mat (mat-def-rdc))
    (list v u w)
    ))))
;---------------------------------------------------------------------
```

17. Validation de la matrice de répartition du "attic"

```
;Nom du fichier: select-mat-att.scm
;----------------------------------------------------------------------
;But: Procédures qui sélectionne et valide la matrice de répartition des
;espaces pour le attic
;----------------------------------------------------------------------

(load "select-mat-ss-eta.scm")
;----------------------------------------------------------------------

;procédure pour sélectionner la matrice de l'attic

;ls-1:liste-base1:liste de répartition choisie pour l'étage
;ls-2:liste-base2:liste des pièces de l'étage mélangées
;ls-3:liste-base2:liste des pièces du r-d-c mélangées
;n:code-base: code de grosseur de la matrice choisie pour l'étage

(define select-mat-att
  (lambda (ls-1 ls-2 ls-3 n)
    (letrec

;liste des pièces du sous-sol + liste prox obligatoires
        ((w (ls-prox-pieces-att ls-3 ls-2))

;liste de répartition
         (x (choix-2 ls-mat-compl (+ 1 (length (car w))) n))

;pour générer de nouvelles listes de pièces et de répartition
         (fonc-2
          (lambda (c d)
            (let
                ((ww (ls-prox-pieces-att ls-3 ls-2))
                 (xx (choix-2 ls-mat-compl (+ 1 (length (car w))) n)))
              (if
               (< d 10)
               (fonc-1 ww xx c d)
               '())))))

;fonc-1 pour remélanger la liste de pièces
;ls-p-p:liste des pieces + liste proximités oblig + ls prox part. obl.
;ls-rep:liste de répartition

         (fonc-1 (lambda (ls-p-p ls-rep a b)
                   (let

; on veux que le Hall Attic #30 soit au-dessus du Upper Hall #20
                    ((z (ajout-100
                         (mix-2 (car ls-p-p) 30 (extraction (car ls-rep)
                                                (position-mul ls-1
                                                (position-sim ls-2 20 0) 0))))))
                     (if (< a 10)
                         (if
                           (conc-et/ou (list-ref ls-p-p 1)
                                       (list-ref ls-p-p 2)
                                 (form-prox-mat
```

```
                                    (formation-mat z (car ls-rep)
                                                     (cadr ls-rep))
                                                     (cadr ls-rep)))
;résultat si la fonction prédicat est #t:
         (list
          (formation-mat z (car ls-rep) (cadr ls-rep))
;matrice de répartition
          (car ls-rep)
;liste de répartition
          z
;liste des pièces mélangées
          (cadr ls-rep)
;code de grosseur de la mat
          )


                          (fonc-1 ls-p-p ls-rep (+ a 1) b))
                      (fonc-2 (- a 10) (+ b 1))
                          )))))
      (fonc-1 w x 0 0))))
;---------------------------------------------------------------------
```

18. Assemblage de la matrice 3D

```
;Nom du fichier: assemblage3d.scm
;----------------------------------------------------------------------
;But: Procédures qui assemble la matrice 3D
;----------------------------------------------------------------------

(load "select-mat-att.scm")
;----------------------------------------------------------------------

;Procédure qui relance le processus d'assemblage des matrices du
;sous-sol, rdc et étage, tant et aussi longtemps qu'on obtient une liste
;vide pour le attic

;a: information sur les matrices de répartition du sous-sol, du rdc et
;de l'étage

(define info-mat3d
  (lambda (a)
    (let
        ((x (select-mat-att
              (list-ref (list-ref a 2) 1)
;liste de répartition choisie pour l'étage
              (list-ref (list-ref a 2) 2)
;liste des pièces de l'étage mélangées
              (list-ref (list-ref a 1) 2)
;liste des pièces du r-d-c mélangées
              (list-ref (list-ref a 2) 3))
;code de grosseur de la matrice choisie

;pour l'étage
            ))
      (if
       (null? x) (sasa (assemb-mat (mat-def-rdc)))
       (list
        (list-ref a 0)
        (list-ref a 1)
        (list-ref a 2)
        x)))))
;----------------------------------------------------------------------

;Procédure qui assemble les matrices de répartition des espaces des
;quatre étages.

(define mat-3d
  (lambda ()
    (let
        ((x (info-mat3d (assemb-mat (mat-def-rdc)))))
      (list
       (list-ref (list-ref x 0) 0)
       (list-ref (list-ref x 1) 0)
       (list-ref (list-ref x 2) 0)
       (list-ref (list-ref x 3) 0)))))
;----------------------------------------------------------------------

(mat-3d)
```