Chapitre 3

Implémentation numérique

La description du modèle a montré qu'il suffisait d'un petit nombre de familles d'élément pour le construire. Les environnements orientés objet sont un outil bien adapté pour résoudre ce type de problème.

Nous voyons dans cette partie en quoi cette façon de modéliser se distingue des outils plus traditionnels avant de décrire comment la méthode zonale a été implémentée dans l'environnement SPARK [NW93].

3.1 Un environnement orienté objet

La description de phénomènes physiques par l'intermédiaire d'un outil numérique passe par deux étapes : la modélisation et la simulation. Nous allons voir comment doivent être traitées ces deux parties pour optimiser un code de calcul.

3.1.1 Modélisation

Conception d'un outil informatique

L'évolution dans le domaine numérique ces dernières années a surtout été marquée par les progrès en capacité de mémoire de machine et en vitesse de calcul. Toutefois, dans le même temps, les chercheurs se sont attachés à faire progresser les modèles décrivant les phénomènes physiques, par exemple la modélisation fine de la turbulence. De plus, la progression dans l'élaboration des algorithmes facilite l'écriture de ces modèles.

Le développement d'un outil numérique de simulation comporte les étapes suivantes décrites par Ebert [Ebe93] :

- Dans un premier temps, il s'agit d'analyser le comportement des phénomènes pour en déduire une représentation physique.
- Le problème physique est alors traduit sous forme d'équations mathématiques.
- À partir de ces équations intervient la phase algorithmique qui traduit l'enchaînement des différentes étapes de la simulation.
- Cet algorithme permet d'écrire le code de calcul par l'intermédiaire d'un langage spécifique.
- Enfin, la dernière étape consiste à exécuter le programme qui permet d'effectuer la simulation.

Cette façon de procéder qui s'est maintenant généralisée aboutit à des outils ayant de nombreux points communs au niveau théorique. L'association de ces différents outils permet de construire des outils de simulation plus puissants. Cette démarche pose cependant plusieurs difficultés qui vont maintenant être analysées.

Limites liées à la complexité des codes de calcul

Faciliter le chaînage de logiciels permet de profiter des compétences de leurs différents concepteurs sans qu'il y ait nécessairement une personne spécialisée dans l'ensemble des domaines concernés. Pour associer des codes de calcul de nature différente, il y a plusieurs solutions : Soit utiliser des fichiers intermédiaires si les entrées-sorties sont bien définies, soit passer par des bases de données, des envois de messages, des fichiers tampon....

L'inconvénient est que le code obtenu par l'association de ces logiciels est disparate et ralenti par le temps de lecture-écriture des fichiers intermédiaires. Par ailleurs, il est souvent laborieux de modifier ou améliorer un logiciel existant et c'est encore plus vrai lorsqu'il s'agit de l'association de plusieurs logiciels.

Assemblage et modification de modèles

Pour éviter ce temps perdu, des réflexions ont été menées visant non plus à développer des langages de programmation simples à écrire, facile à relire et bien structurés mais plutôt pour concevoir un environnement qui, quel que soit le langage utilisé, cumulera les propriétés de réutilisabilité et de facilité de maintenance grâce à sa modularité. Ce sont ces concepts, entre autres, que l'on place sous le terme de programmation orientée objet. Parmi les nombreux environnements manipulant ce type de concept, plusieurs sont particulièrement adaptés au cas de la thermique du bâtiment notamment CLIM2000 [BRCG93], MOTOR-2 [Ebe93], ZOOM [BJP+89], ALLAN [Fra92] ou encore IDA [Sah91].

C'est dans cet état d'esprit qu'a été écrit le logiciel SPARK [And87], en cours de développement au Lawrence National Berkeley Laboratory en Californie. C'est un outil qui, bien qu'écrit par des spécialistes de thermique, est très général et adaptable à tous types de domaines. Le principe est de construire une bibliothèque de modèles classés selon le domaine d'application et facilement adaptables entre eux.

Il a donc semblé intéressant d'implémenter le modèle zonal dans cet environnement à cause de la souplesse qu'il offre au niveau de l'implémentation ce qui permet de tester de nombreuses configurations. D'autre part, il sera possible soit de l'intégrer dans une simulation plus générale, soit de le coupler avec d'autres modèles écrits dans cet environnement.

Nécessité d'une conception hiérarchisée

Nous allons détailler les raisons qui rendent les environnements orientés objet particulièrement attrayants.

La description de la conception d'une simulation présentée au début du chapitre, montre que s'il est possible d'obtenir une description claire et structurée de l'outil informatique jusqu'à la phase algorithmique, l'écriture du code est davantage fonction de la personnalité du programmeur. Lui-même pourra, au bout de quelques mois, éprouver des difficultés à "se relire", aussi documenté et précis que soit son code. Cet état de fait est lié à la complexité des phénomènes décrits.

Le remède (suggéré par l'illustre Descartes!) consiste à décomposer un système en autant de sous-systèmes plus simples et, si nécessaire, recommencer la démarche jusqu'à aboutir à de petits systèmes élémentaires. C'est là un des points clés des environnements orientés

objet qui permet de transformer un modèle complexe en un système hiérarchisé de modèles élémentaires.

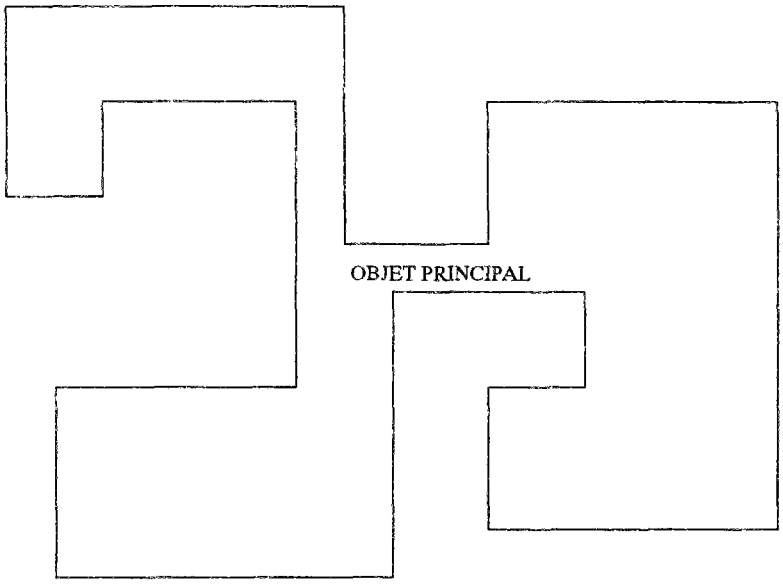

FIG. 3.1 - *Objet à modéliser*

En fait, il ne s'agit pas seulement d'assembler un ensemble de boîtes noires mais de conserver des propriétés de modularité et de réutilisabilité à tous les niveaux jusqu'aux équations les plus élémentaires.

Ce principe va être illustrer sur un exemple. La description du système présenté à la figure 3.1, quelle que soit la démarche employée, risque d'être longue, laborieuse et surtout très dépendante de la personne qui le fera. Pour simplifier les choses, il est possible par exemple le diviser en trois sous-systèmes qui seront décrits indépendamment. L'avantage est que ces objets sont plus simples et la description sera plus claire.

L'inconvénient de la méthode est qu'il faut alors décrire les liens entre ces éléments ce qui rallonge la description. C'est là une des difficultés majeures rencontrées par les concepteurs d'environnements orientés objets, ces produits étant d'autant plus intéressants que la description de ces liens est simple. La figure 3.2 montre les trois sous-objets à assembler. Leurs frontières peuvent être plus ou moins complexes.

Les objets peuvent devenir encore plus simples en poursuivant la décomposition et à ce moment, un grand nombre des objets élémentaires obtenus sont semblables à l'échelle près. Par exemple, sur la figure 3.3, les éléments P.P.3, P.T.2 et P.T.4 sont semblables tout comme P.S.1 et P.T.3.

En reprenant le cas de la méthode zonale, le système complexe peut être assimilé à un local. Ce local sera divisé en frontières et en sous-volumes. Les éléments de même type (calculs dans un sous-volume, frontières entre sous-volumes, comportement de la paroi) pourront être regroupés en quelques familles : Ce sont les trois éléments de la première décomposition. Chacun d'eux sera composé d'un certain nombre d'équations que l'on a vu au chapitre 2.1.3 et qui pourront être communes à plusieurs sous-objets. Ce sont les objets élémentaires dont plusieurs sont semblables.

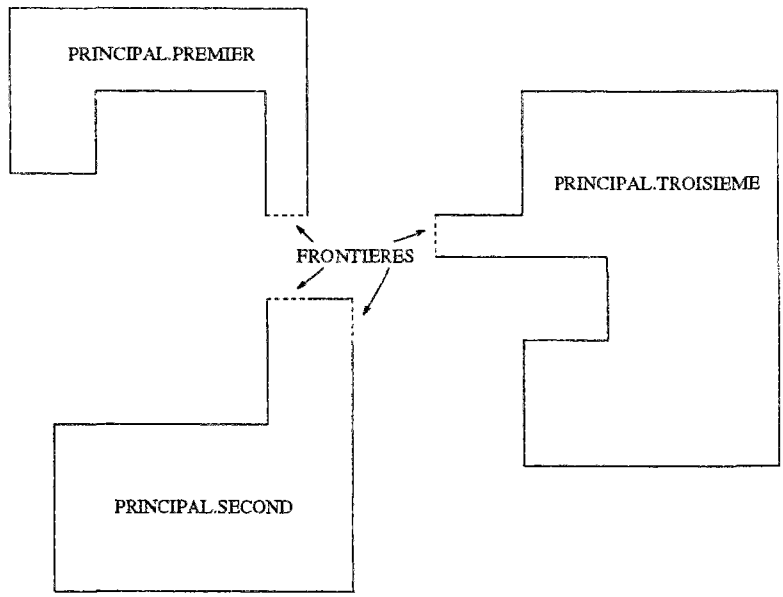

FIG. 3.2 - *Décomposition en sous-objets*

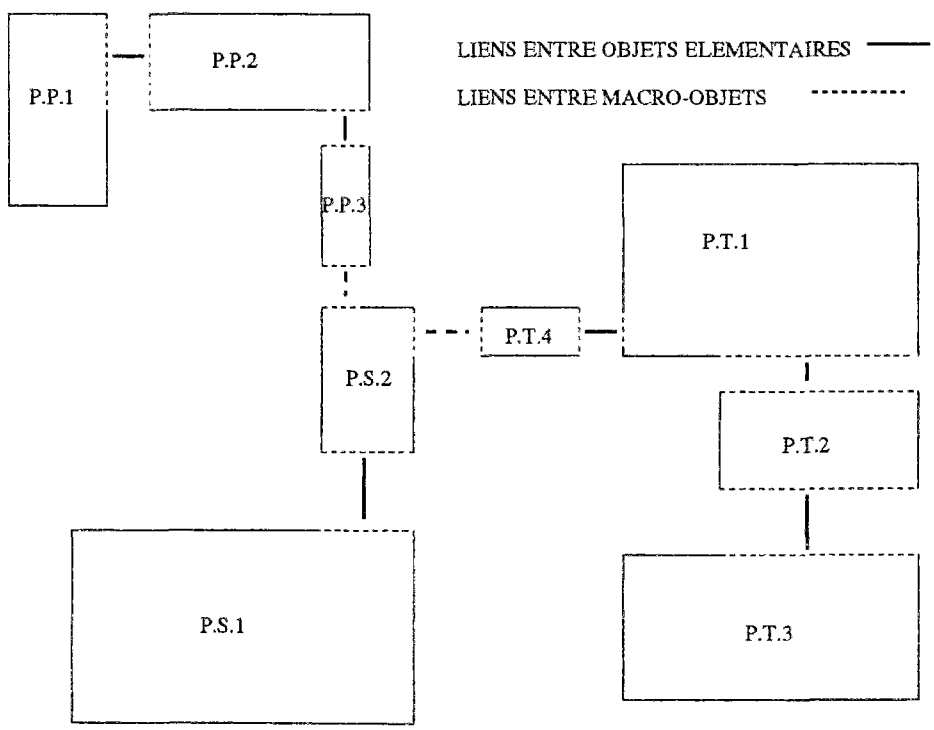

FIG. 3.3 - *Les objets élémentaires à connecter*

Pour modifier les modèles, il suffira donc d'intervenir au niveau de ces équations élémentaires. Les différentes liaisons entre tous les objets devront être précisées dans un fichier de connexion. La difficulté est de concevoir un fichier simple à écrire y compris, comme c'est le cas dans la méthode zonale, lorsqu'il y a de nombreux objets à relier.

Bien entendu, pour optimiser la modélisation, il faut adapter l'environnement numérique

aux phénomènes physiques à représenter. Il faudra donc décomposer en sous-modules le plus indépendants possibles et qui ne soient ni trop petits pour éviter un outil trop encombrant (nombre d'équation très élevées), ni trop complexes pour garder une simulation simplifiée. De plus cet outil devra être suffisamment général pour traiter un maximum de situations et facile à modifier pour ajouter ou rectifier des modèles. Enfin, il faut ajouter que toute cette implémentation interne est cachée de l'utilisateur qui n'a accès qu'aux fichiers d'entrées de données et de résultats.

3.1.2 Simulation

La deuxième étape est la simulation. Elle a un double objectif:

- Prévoir le futur comportement thermique ou aéraulique d'un système et dimensionner les éléments qui le composent.

- Effectuer un très grand nombre d'études à peu de frais pour comprendre certains phénomènes nuisibles (stratification, courants d'air).

Il faut toutefois rester prudent dans les interprétations car dans tous les cas le modèle numérique a été établi à la suite de certaines simplifications, tous les phénomènes ne pouvant être pris en compte.

Pour qu'une simulation soit efficace, il faudra résoudre le système d'équations en évitant au maximum que l'utilisateur ait à intervenir mais en veillant à ce qu'il puisse suivre l'évolution pour comprendre les éventuels dysfonctionnements (systèmes sans solution, problèmes de convergence, solution fausse, dérive numérique, coût informatique excessif, lenteur, problèmes de place mémoire).

Cette nécessité fait que la simulation ne pourra être dissocier de la modélisation au risque de mal interpréter le comportement du logiciel. En fait, l'examen des résultats de la simulation devra permettre de modifier la modélisation pour améliorer les résultats.

Parmi les environnements qui exploitent les différentes propriétés déjà décrites, on peut citer ZOOM [BJP+89] ou TRNSYS [trn84]. Ce dernier est très utilisé dans le domaine du bâtiment, avec des préprocesseurs destiné à faciliter la gestion des entrées-sorties comme CSTbât écrit par les chercheurs du C.S.T.B.

Il existe un autre type d'environnement qui génère par l'intermédiaire de préprocesseurs des fichiers source en C ou Fortran pour simplifier au maximum la tâche des utilisateurs, tels Allan-Neptunix [Fra92] [Ing88], IDA [Sah88] ou CLIM2000 [RBCG89]. C'est le cas également du logiciel SPARK [BEN+94] dont nous allons maintenant détailler le principe.

3.1.3 L'environnement SPARK

La description par objet est une manière de clarifier un langage en assemblant entre eux des éléments simples. Cela s'apparente à un jeu de construction où l'on commence par les petits éléments qui sont assemblés entre eux pour en former de plus importants et ainsi de suite jusqu'à la conception de l'ensemble. Ceci conduit à des avantages tant dans le diagnostic des erreurs que pour effectuer des modifications. Ces propriétés ont été largement exploitées dans SPARK et nous allons voir comment est modélisé un système physique.

Implémentation d'un modèle

- **Écriture du code de calcul**

  Les priorités d'un code de calcul sont la facilité de maintenance y compris lorsque l'auteur du code n'est plus présent pour intervenir et la possibilité de coupler entre eux des éléments réalisés par des auteurs différents.

  L'organisation de SPARK va donner ces possibilités aux modélisateurs à différents niveaux, le plus bas étant l'écriture des équations. Pour mieux comprendre le fonctionnement du logiciel, la description sera illustrée par des exemples simples, extraits du code décrivant les modèles zonaux.

  Les équations sont les objets élémentaires. Il s'agit de choisir judicieusement le degré de complexité pour rendre le code plus compréhensible. Les extrêmes à éviter sont les suivants :

  - On pourrait se contenter des opérations élémentaires (addition, multiplication,..) en les composant entre elles mais cela entraîne une multitude de connexions rendant ainsi le code très volumineux.
  - Une autre possibilité serait de limiter le nombre d'équations en apportant le maximum d'informations à chacune d'elles mais cela génère des équations difficiles à relire et à modifier.

  Il faut donc trouver un compromis et les objets élémentaires choisis sont les différentes équations de la méthode zonale données dans l'annexe B. Il y aura ainsi une équation pour calculer le débit, une autre pour le flux de chaleur, une autre encore pour les bilans de masse et d'énergie.

  Pour que ces équations puissent être modifiées, utilisées dans un tout autre programme et reprises par une autre personne, il faut également qu'elles soient écrites le plus clairement possible.

  Pour cela le choix a été de les écrire dans un fichier texte avant qu'un préprocesseur [NW92] ne les convertisse en langage SPARK par le biais d'environnements de calcul formel tels Maple [CGG85] ou Macsyma [MIT83].

  Le fichier contenant toutes les équations de la méthode zonale en langage Macsyma est décrit dans l'annexe B. Ce fichier a la propriété d'être facile à lire et ressemble fortement à la liste des équations. Ainsi, ces équations peuvent être modifiées, utilisées dans un autre programme ou encore reprises par une autre personne.

  Voyons, sur l'exemple du débit à travers une ouverture verticale, comment est obtenu le langage SPARK.

  Le code Macsyma est le suivant :

```
        # bilan de masse

# debit a travers l'interface horizontale de la  cellule
```

```
vert:=[[debver = rhob*k*(pb-pt)^n*x*y, (pb-pt) >= 0],
     [debver = -1*rhot*k*((-1)*(pb-pt))^n*x*y, (pb-pt) < 0]]:

makespank(eval(vert),'vert',[rhob,rhot,x,y,n,k]):
```

"makespank" est le nom de la fonction qui se charge de transformer l'expression symbolique de l'équation en code SPARK. Il donne un nom à cette équation qui sera "**vert.obj**" et indique que les variables pouvant être évaluées par cette équation seront *debver*, *pb* et *pt*. En effet x, y, n et k sont des entrées et ne doivent pas être évaluées tandis que comme *rhob* et *rhot* n'interviennent pas dans les deux lignes, il n'est pas possible d'évaluer l'une ou l'autre de ces variables pour tous les cas.

Enfin, l'équation est décomposée en deux éventualités pour que la fonction soit toujours définie. Voyons maintenant le code SPARK généré par le langage de calcul formel. Il se décompose en deux parties appelées objets et fonctions.

L'allure de l'objet appelé *vert.obj* est la suivante:

```
/*SPARK object file vert.obj*/
/*equation [[debver = k*(pb-pt)^n*rhob*x*y,pb-pt >= 0],
[debver = -k*(pt-pb)^n*rhot*x*y,pb-pt < 0]]*/
define  vert(debver,k,pb,pt,n,rhob,x,y,rhot)
double debver;
double k;
double pb;
double pt;
double n;
double rhob;
double x;
double y;
double rhot;
{
        debver=debver_vert(k,pb,pt,n,rhob,x,y,rhot);
        k;
        pb=pb_vert(debver,k,pt,n,rhob,x,y,rhot);
        pt=pt_vert(debver,k,pb,n,rhob,x,y,rhot);
        n;
        rhob;
        x;
        y;
        rhot;
}
```

Les commentaires (générés automatiquement) présentent l'équation. Le rôle de cet objet est d'indiquer quelles variables interviennent dans l'équation en précisant celles

qui doivent être évaluées.

Cette disposition permet une intervention intéressante: Il est possible par exemple d'évaluer la valeur optimale d'un coefficient empirique d'un modèle en fonction de références expérimentales ou numériques. Ces références deviennent des entrées du modèle et le coefficient empirique une variable.

Voici l'objet permettant d'obtenir la valeur de la perméabilité verticale d'une frontière en fonction du débit mesuré expérimentalement :

```
/*SPARK object file vert.obj*/
/*equation [[debver = k*(pb-pt)^n*rhob*x*y,pb-pt >= 0],
[debver = -k*(pt-pb)^n*rhot*x*y,pb-pt < 0]]*/
define  vert_ew(debver,k,pb,pt,n,rhob,x,y,rhot)
double debver;
double k;
double pb;
double pt;
double n;
double rhob;
double x;
double y;
double rhot;
{

        debver;
        k=k_vert(debver);

        pb=pb_vert(debver,k,pt,n,rhob,x,y,rhot);
        pt=pt_vert(debver,k,pb,n,rhob,x,y,rhot);
        n;
        rhob;
        x;
        y;
        rhot;
}
```

Dans ce cas *k_vert* est la fonction qui détermine le coefficient de perméabilité pour un débit vertical donné. Cette propriété commune par ailleurs à IDA et désormais dans une certaine mesure à TRNSYS permet de traiter indifféremment toute variable comme donnée ou comme inconnue, il n'y a donc pas d'entrées-sorties prédéfinies.

Par ailleurs, à chaque inconnue est associée un nom de fonction qui correspond à un fichier source écrit en C par le même préprocesseur.

Dans cet exemple ces fonctions étaient :

- *debver_vert* pour déterminer le débit vertical.
- *pb_vert* pour calculer la pression dans le sous-volume situé sous l'interface horizontale.
- *pt_vert* pour la pression au-dessus de l'interface horizontale.

Toute fonction même discontinue, non-linéaire ou différentielle pourra être créée par ce biais. La lecture de ce fichier source sera particulièrement intéressante en cas de défaillance lors de la résolution pour analyser les difficultés. Une option de "debuggage" pourra également être utilisée lors de la simulation pour faire afficher toutes les valeurs intermédiaires des variables d'itération.

Voici pour l'exemple précédent, la fonction *debver_vert.c* permettant d'évaluer le débit vertical :

```
/*SPARK function file debver_vert.c*/
#include        "val.h"
#include        <math.h>
#include        <stdio.h>


#define debver  result.dval
#define k       args[0].dval
#define pb      args[1].dval
#define pt      args[2].dval
#define n       args[3].dval
#define rhob    args[4].dval
#define x       args[5].dval
#define y       args[6].dval
#define rhot    args[7].dval

VAL
debver_vert(args)
VAL args[];
{
        VAL     result;

        if (pb-1.0*pt < 1.e-15 && pb-1.0*pt > -1.e-15 && n < 0. &&
pb-1.0*pt >= 0.0)
{fprintf(stderr,"in debver_vert exponand pb-1.0*pt is too small
and exponent n is negative. debver is set to 1\n");
```

```
                debver=1;
                return(result);
}

        if (pb-1.0*pt <= 0. && n - floor(n) > 1.0e-15 &&
 pb-1.0*pt >= 0.0)
{fprintf(stderr,"in debver_vert exponand pb-1.0*pt is negative
and exponent n is noninteger. debver is set to 1\n");
                debver=1;
                return(result);
        }

if (pt-1.0*pb < 1.e-15 && pt-1.0*pb > -1.e-15 && n < 0. &&
 pb-1.0*pt < 0.0)
{

fprintf(stderr,"in debver_vert exponand pt-1.0*pb is too small
and exponent n is negative. debver is set to 1\n");
                debver=1;
                return(result);
}

        if (pt-1.0*pb <= 0. && n - floor(n) > 1.0e-15 &&
 pb-1.0*pt < 0.0)
{fprintf(stderr,"in debver_vert exponand pt-1.0*pb is negative
and exponent n is noninteger. debver is set to 1\n");
                debver=1;
                return(result);
}
```

if (pb-1.0*pt >= 0.0) debver = k*pow(pb-1.0*pt,n)*rhob*x*y;

if (pb-1.0*pt <0.0) debver = -1.0*k*pow(pt-1.0*pb,n)*rhot*x*y;

```
#ifdef arg_dbg
        printf("in debver_vert.c debver is %lf k is %lf pb is %lf
pt is %lf n is %lf rhob is %lf x is %lf y is %lf rhot is %lf \n",
debver,k,pb,pt,n,rhob,x,y,rhot);
#endif arg_dbg
        if ( !(pb-1.0*pt >= 0.0) && !(pb-1.0*pt < 0.0) )
                printf("in debver_vert.c argument out of range:
return default\n");
        return(result);
}
```

Le contenu de cette fonction est le suivant :

- Déclaration de toutes les variables faisant partie de l'équation.
- Ensuite, vérification du domaine de validité de la fonction tout en renseignant l'utilisateur sur les problèmes éventuels lors de la simulation.
- Puis évaluation de la variable, c'est la partie écrite en caractère gras.
- La dernière partie permet éventuellement d'imprimer les valeurs intermédiaires du modèle et de retourner une valeur par défaut en cas de problème (valeur calculée non numérique).

Des modèles élémentaires sont ainsi obtenus et intégrés à l'outil de simulation. Ils alimentent une bibliothèque de modèles qui pourra servir à d'autres types de simulation. Bien entendu, chaque équation n'est écrite qu'une seule fois et sera reliée à toutes les variables de la simulation régies par la même fonction.

- **Création de macro-objets**

Lorsque les simulations deviennent conséquentes (plusieurs phénomènes physiques différents à représenter), il est nécessaire de grouper entre elles certaines équations dans ce qu'on appelle un macro-objet : Ces macro-objets regroupent plusieurs objets élémentaires représentant un même composant physique et correspondent à un système d'équations. Le choix judicieux de ces regroupements permet de construire un outil bien structuré et relativement simple, surtout si leur nombre reste limité. La structure hiérarchisée permet également le regroupement de macros pour générer une macro d'un niveau supérieur.

Dans le cas de la méthode zonale, le fait de partitionner en sous-volumes identiques permet de limiter la simulation à un petit nombre de macro-objets ce qui optimise, du point de vue du code à écrire, l'outil réalisé. Il y aura un macro-objet pour décrire les équations des sous-volumes, les autres concernant les interfaces.

Voyons cela sur l'exemple des frontières horizontales. En plus de l'objet débit vertical *vert.obj* décrit précédemment, il faut ajouter le flux de chaleur vertical *fluxv.obj* ce qui conduit à la macro suivante :

```
/*equations de liaison entre cellules reliees entre Top et Bottom
avec SPARK*/
/*interface_TB.obj*/


macro

declare vert    vert_1;
declare fluxv   fluxv_2;
```

```
link    debver(vert_1.debver,fluxv_2.debver)
link    k_tb(vert_1.k)
link    pb(vert_1.pb)
link    n_tb(vert_1.n)
link    x_tb(vert_1.x)
link    y_tb(vert_1.y)
link    rhob(vert_1.rhob)
link    rhot(vert_1.rhot)
link    pt(vert_1.pt)
link    phi_v(fluxv_2.phi_v)
link    cp_tb(fluxv_2.cp)
link    tb(fluxv_2.tb)
link    tt(fluxv_2.tt)
```

Ceci permettra d'intégrer le modèle zonal dans une simulation thermique de bâtiment, celui-ci étant considéré comme une seule macro regroupant plusieurs macro-objets.

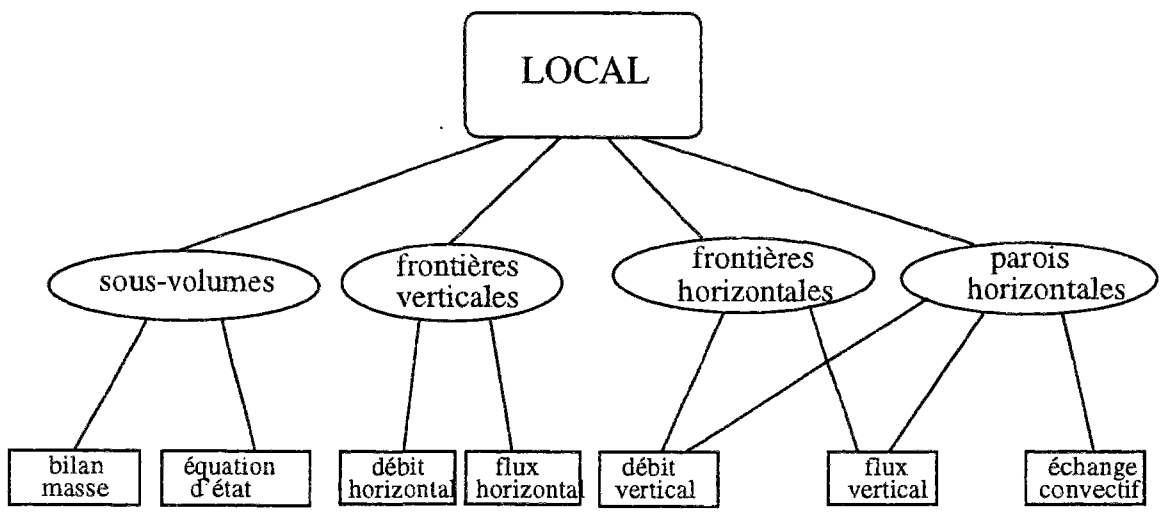

FIG. 3.4 - *Hiérarchie des différents types d'objets*

Noua avons donc présenté les trois types d'objet (équations, systèmes d'équation et simulation) qui permettent de construire une simulation. La figure 3.4 indique comment ces éléments interviennent dans la simulation d'un local. Les ovales représentent quelques-uns des macro-objets constituant le système alors que les rectangles sont les objets élémentaires. Une équation peut être appelée par plusieurs macro-objets différents. Comme chacune de ces familles n'est écrite qu'une seule fois, les seuls fichiers dont la taille dépend du nombre de partitions sont le fichier décrivant les connexions et celui des entrées de données; ils constituent l'élément LOCAL.

- **Assemblage des composants**

Le fichier de connexion contient toutes les variables de la simulation et indique à quels macro-objets elles appartiennent.

L'écriture de ce fichier qui décrit tous les liens entre les variables du problème est laborieuse et répétitive. C'est pourquoi un préprocesseur permet de le générer automatiquement à partir de la connaissance du nombre de sous-volumes dans les trois directions. Ce même préprocesseur prépare également le fichier d'entrées de données qu'il suffit de compléter.

Ce fichier de connexion, qui est la racine du programme peut devenir après quelques modifications triviales un macro-objet intégrable dans toute autre simulation. Un exemple est présenté lors de l'évaluation du confort au chapitre 5.1.

Déroulement d'une simulation

La simulation s'effectue en plusieurs étapes automatisées. Il s'agit tout d'abord de construire le solveur d'équations. Pour cela, l'environnement se sert du fichier décrivant les liens entre les variables externes et les macro-objets. À partir de là, il va associer une fonction à chaque variable pour construire un système d'équations où sont représentées toutes les équations du modèle.

Pour le modèle zonal considéré, l'écriture d'un fichier de simulation SPARK revient simplement à connecter entre eux les divers composants : sous-volumes et interfaces.

Comme vu précédemment, SPARK lit le fichier de connexion global, et l'analyse récursivement jusqu'à descendre aux composants élémentaires. Il crée ensuite une représentation interne sous forme d'hypergraphe du système d'équations, où les *équations* sont des *nœuds* et les *variables* des *arcs*: un arc connecte deux nœuds si la variable associée est commune aux deux équations correspondantes. On obtient ainsi un hypergraphe non orienté, avec autant de nœuds que d'arcs (puisque dans un problème bien posé, il y a autant d'équations que d'inconnues).

Suit alors une phase d'*appariement*, ou encore de *couplage*, dans laquelle un algorithme standard de la théorie des graphes, l'algorithme de Dinic [Baa83], est utilisé pour associer à chaque équation une de ses variables. Cela revient à apparier chaque nœud avec un arc qui lui est lié. Dès lors, chaque équation ne devra plus utiliser la méthode de résolution de SPARK que pour calculer la variable qui lui a été associée. Cet appariement revient à orienter le graphe. L'arc associé à un nœud donné est sortant (car la variable associée est une sortie de l'équation) et tous les autres arcs sont rentrants.

Dès lors on a un graphe orienté. Il s'y trouve des cycles. Un autre algorithme de la théorie des graphes, l'algorithme de Levy-Low [LL83], est alors utilisé pour déterminer un *petit* nombre d'arcs tels que l'élimination de ces arcs élimine tous les cycles. Cela revient à déterminer des variables d'itération telles que leur détermination permet d'en déduire explicitement toutes les autres inconnues du problème. La résolution optimale de ce problème est NP-complète [Baa83], l'algorithme de Levy-Low est donc heuristique, mais donne de bons résultats dans la pratique. Cette phase est dite de *réduction*.

Une fois les variables d'itération déterminées, SPARK résout le problème non linéaire réduit dont les seules inconnues sont les variables d'itération qu'il a choisies, puis propage la solution réduite à l'ensemble des inconnues du problème.

Pourquoi ces manipulations? La résolution d'un système d'équations non linéaires est classiquement résolue par la méthode de Newton Raphson, qui demande à chaque itération l'inversion de la matrice jacobienne du système, a un coût en temps calcul qui est une

fonction cubique de la taille de la matrice. La réduction de la taille du système effectuée par SPARK, par le biais d'algorithmes de la théorie des graphes, permet donc un gain significatif en temps calcul. La phase de réduction est donc primordiale à l'efficacité numérique du code de simulation généré par SPARK.

L'efficacité de cette méthode de réduction va dépendre de la nature des équations. Elle sera d'autant plus efficace que le nombre de variables communes à plusieurs équations sera grand et que le système d'équation est complexe. Dans le cas de la méthode zonale, on pourra en moyenne diviser la taille du solveur par dix.

Il faut noter que SPARK n'exploite pas nécessairement la structure hiérarchique de la modélisation pour la résolution car cela limiterait les possibilités de réduction. Il peut également mettre toutes les équations à plat et laisser à l'utilisateur le choix de la méthode de résolution par l'imposition des variables d'itération. En effet, il sera parfois plus intéressant de réduire le système d'équation initial pour obtenir un système dont les variables d'itérations sont les plus représentatives plutôt que de chercher à obtenir la plus petite simulation qui sera certes la plus rapide mais risque davantage de ne pas converger.

Cet environnement est particulièrement bien adapté pour une simulation utilisant souvent les mêmes équations si leur nombre n'est pas trop important (le cas d'un maillage éléments finis est par exemple mal adapté). La méthode zonale respecte bien ces conditions ce qui justifie le choix de l'implémenter.

Une autre caractéristique de cet environnement est le développement d'une bibliothèque de modèles, qui devrait permettre d'intégrer la méthode zonale dans d'autres simulations pour lesquelles un modèle aéraulique serait nécessaire.

Nous avons profiter de cette propriété pour générer et coupler des modèles de conduction, de rayonnement et de transport ainsi qu'un modèle de confort. Ce seront tous des éléments adaptables entre eux et avec d'autres types de modèles et qui seront intégrés à la bibliothèque.

Par ailleurs, nous avons également couplé la méthode zonale avec un modèle de plafond rayonnant développé à Berkeley [SFW95] sans apporter de modification aux codes de calculs respectifs, la seule intervention se situant au niveau du fichier de connexion. D'autres projets tels le couplage avec des modèles d'écoulement dans les sols sont envisagés.

Il faut enfin signaler qu'il existe un traducteur NMF-SPARK [Nat95] du format neutre de description de composants thermiques NMF vers l'environnement de simulation SPARK. C'est un utilitaire permettant l'élaboration automatique d'une librairie de composants pour SPARK avec des modèles développés dans d'autres types d'environnement. Cela rend donc possible l'intégration de la méthode zonale dans tout logiciel compatible avec le format NMF. Voyons maintenant comment le modèle zonal est intégré dans l'environnement SPARK.

3.2 Les modèles zonaux en environnement orienté objet

Il s'agit d'intégrer toutes les équations de la méthode données en annexe A. Pour cela, on reprend exactement la configuration décrite lors de la présentation de la méthode zonale c'est à dire qu'on distingue ce qui se passe dans un sous-volume de ce qui se passe dans les frontières.

- Chaque sous-volume contient quatre objets élémentaires : Les bilans de masse et de chaleur, l'équation des gaz parfaits et la variation hydrostatique de la pression.
- Une interface horizontale contient deux types d'objet : Le calcul du débit et celui du flux.
- Pour les interfaces verticales, il faut distinguer le débit horizontal supérieur et le débit horizontal inférieur en calculant l'ordonnée du point neutre.

La figure 3.5 reprend cette description en ajoutant les variables de chaque macro-objet.

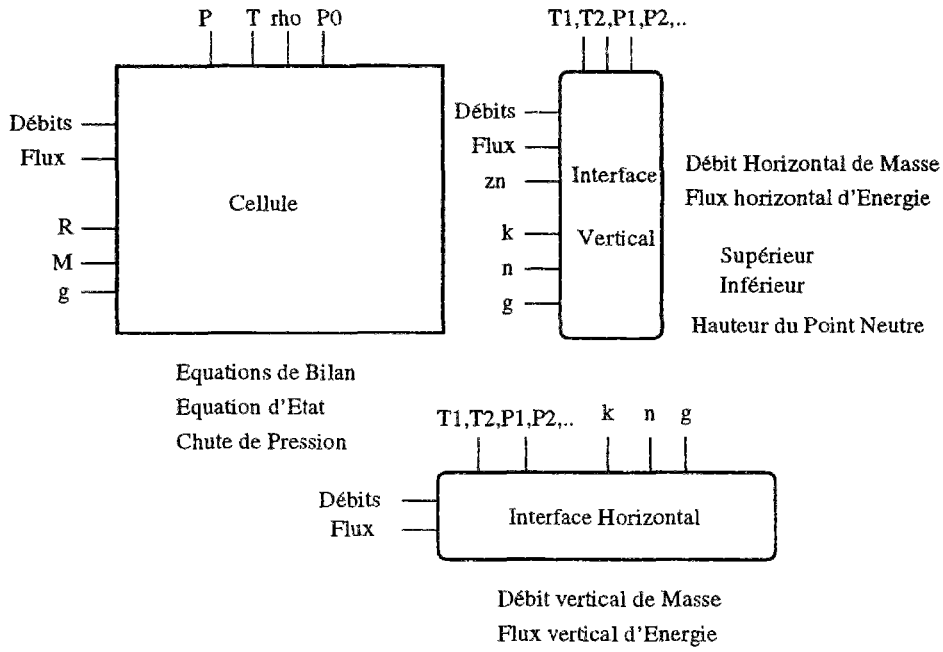

FIG. 3.5 - *Les macro-objets constituant un sous-volume*

Il faut également distinguer les frontières internes entre sous-volumes et les frontières situées à la limite de l'enceinte. Pour ces dernières on conserve une loi de débit avec un coefficient de perméabilité très faible représentant la perméabilité des parois (fissures, joints) mais il est nécessaire d'ajouter un objet calculant le flux convectif pénétrant dans l'enceinte dans le cas horizontal comme dans le cas vertical.

Une fois les différents types de modèles implémentés, il s'agit de réaliser le couplage. Celui-ci s'effectue par l'intermédiaire du fichier de connexion comportant la liste de toutes

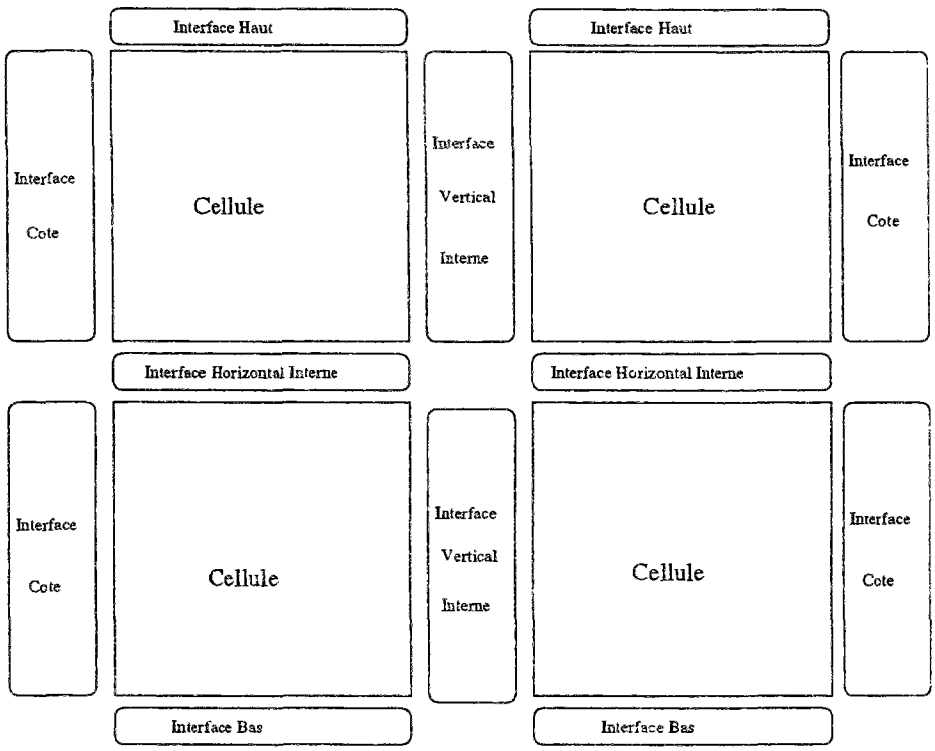

FIG. 3.6 - *Les différents macro-objets d'une simulation*

les variables de la simulation et en indiquant comment celles-ci sont reliées aux macro-objets. La figure 3.6 récapitule l'ensemble des composants d'une simulation et permet de comprendre comment la méthode zonale est implémentée dans l'environnement SPARK. Il y a donc 5 types de macro-objets différents: L'intérieur du sous-volume, les interfaces horizontales et verticales et les modèles de paroi verticaux et horizontaux.

3.3 Déroulement d'une simulation

3.3.1 Méthode de résolution

La démarche de SPARK consiste tout d'abord à recenser toutes les équations pour construire un système d'équations. A ce stade, il vérifie qu'il y a bien cohérence entre le nombre d'équations et d'inconnues.

Vient ensuite la phase de réduction. Si on laisse toute liberté au solveur, celui-ci cherchera à éliminer le maximum de variables et conduira à un système très hétéroclite où toutes sortes de variables seront représentées. Or, parmi les variables de la méthode zonale, certaines ont un poids bien plus fort que d'autres. Il s'agit d'éliminer les "mauvaises" variables.

En effet, si une variable varie peu, une des colonnes du jacobien est proche du vecteur nul: le jacobien est "presque singulier et le système ne parvient pas à s'orienter convenablement. Un autre type de mauvaise variable est une variable représentant un état discret (0-1), ou confinée dans un segment [a-b] et sans signification en dehors de ce segment. Comme une méthode numérique recalcule les variables d'itération et que la nouvelle valeur peut être a priori quelconque, une telle variable d'état confinée, recalculée, risque de tomber hors de son domaine binaire (0 ou 1) ou de son segment, et on obtiendra encore une fois des difficultés de convergence.

Dans la simulation, les températures et les pressions varient beaucoup plus que les masses volumiques par exemple. Ainsi, en cas de modification un peu trop importante de la variable masse volumique, la simulation sera bouleversée et il faudra de nombreuses itérations pour se rapprocher à nouveau de la solution. Les variables sans sens physique sont également à éviter. Comme il fallait automatiser la procédure et garantir de bonnes variables, dans tous les cas les mêmes variables d'itération seront imposées. Ce sont les pressions et les températures dans les sous-volumes ainsi que le débit vertical dans les interfaces. L'expérience a montré que ces variables étaient suffisantes pour la simulation dans l'ensemble des problèmes traités. Ainsi le système obtenu est plus sûr en contrepartie d'une taille un peu plus importante.

Cette procédure revient en fait à imposer la méthode de résolution un peu à la manière de MOTOR-2 [Ebe93] et permet de savoir comment sont effectués les calculs.

Avec ce choix de variables d'itération, la simulation obtenue, après cette réduction par substitution, contient jusqu'à dix fois moins d'équations que le problème initial. La taille de l'exécutable est donc nettement diminuée ce qui accélère d'autant la rapidité des calculs. En fait, le nombre de variables est fonction du nombre de sous-volumes. Il est égale à deux fois le nombre de sous-volumes (pressions et températures imposées) en plus du nombre d'interfaces verticales (debits verticaux). Au total, il y a donc un peu plus de trois fois le nombre de sous-volumes alors que le nombre total d'équations écrites par sous-volumes (voir annexe A) est d'environ une vingtaine.

3.3.2 Difficultés de convergence

La principale difficulté de ce type de modèle est d'obtenir la convergence des résultats. Cette situation peut paraitre étonnante étant donnée la "grossièreté" du partitionnement par rapport à des modèles fins pouvant aller jusqu'à plusieurs centaines de milliers de noeuds. (avec également des problèmes de convergence...). En fait, le principal obstacle est la pré-

sence de nombreuses *équations non-linéaires* alors que les modèles fins linéarisent toutes les équations.

On peut toutefois noter qu'avec le logiciel FLUENT, il est également difficile de trouver une solution à un problème de convection naturelle surtout lorsqu'on dispose initialement de peu d'informations sur l'écoulement.

Ce sont essentiellement les problmes de convergence qui ont freiné le développement de la méthode zonale, les utilisateurs étant souvent obligés de se limiter à des configurations bidimensionnelles, avec un très petit nombre de sous-volumes ou encore de simplifier les équations (suppression de la variation hydrostatique de la pression dans les interfaces verticales).

La robustesse du solveur de SPARK ainsi que les possibilités de réduire le système d'équation et d'imposer les variables d'itération favorisent l'obtention de la convergence.

De plus, SPARK est toujours en cours de développement et les difficultés constatées au fur et à mesure ont été prises en compte pour améliorer et accélérer les résultats.

Les difficultés de convergence peuvent être nettement réduites en prenant les précautions suivantes (qui sont offertes par l'environnement) :

- Imposer des intervalles aux variables utilisées pour éviter que le solveur ne s'écarte trop de la solution.

- Changer le coefficient de relaxation de la méthode de résolution Newton-Raphson, la valeur qui donne les meilleurs résultats est $0,5$. Cela ralentit la simulation en augmentant le nombre d'itérations nécessaires pour obtenir la convergence mais même les simulations de plus d'une centaine de sous-volumes ne prennent pas plus d'une dizaine de minutes.

- Imposer des valeurs initiales : Pour obtenir des valeurs initiales convenables, une solution est de remplacer l'exposant n égal à $0,5$ par 1 dans les lois débit-pression. Le système rendu linéaire devient plus facile à résoudre ce qui permet d'obtenir des valeurs initiales pas trop éloignées de la solution finale.

- Utiliser le mode de résolution dit des composants forts [BEN+93] qui constitue une autre option de SPARK. La simulation est décomposée en blocs qui sont résolus séparément et assemblés en dernier lieu évitant ainsi le traitement d'un gros système d'équations. Cette option sera utilisée pour le couplage avec d'autres modèles.

- Enfin, la dernière amélioration a été, entre chaque itération, de capter les variables ayant des valeurs non numériques à la suite par exemple de divisions par 0 et de les remplacer par des valeurs par défaut raisonnables.

L'objectif du travail étant la réalisation d'un outil dédié, il est nécessaire d'assurer la convergence dans tous les cas. Pour cela, il faut partir d'une solution connue pour un partitionnement donné et faire évoluer "lentement" les entrées pour obtenir le problème recherché. Cela ralentit la résolution mais a le grand avantage d'assurer la solution dans tous les cas. Cette méthode qui consiste à transformer un problème statique en problème dynamique,

dont l'état stationnaire est solution, est dite d'homotopie [EGPT83]. Le choix des variables à faire évoluer dépend du type de difficultés rencontrées :

- Sans aucune idée des résultats, la valeur de l'exposant n dans la loi débit-pression pourra être fixée à 1 avant de la faire tendre progressivement vers $0,5$.
- Une autre difficulté de convergence correspond au cas où deux parois verticales opposées sont à la même température en convection naturelle. Le solveur ne sait alors comment orienter l'écoulement et la simulation a tendance à boucler. La solution consiste à imposer tout d'abord une importante différence entre les deux faces avant de la réduire progressivement.
- Enfin, le cas de la convection forcée étant plus facile à traiter, il est possible d'imposer un débit puis de le faire tendre progressivement vers 0.

L'implémentation de ces simulations dynamiques ne pose pas de difficultés avec SPARK, il suffit de lui indiquer les entrées pour lesquelles une intervention est souhaitée et d'écrire un fichier indiquant l'évolution de ces variables.

Ces remarques montrent également que le traitement de problèmes dynamiques posera peu de problèmes de convergence car, à chaque pas de temps, les valeurs initiales (solutions du problème au pas de temps précédent) seront proche de la solution. Nous reviendrons sur ce sujet lors du couplage avec les modèles d'enveloppe.

Il reste à écrire un préprocesseur générant automatiquement ce type de fichier à partir d'une part des entrées de la simulation dont une solution est connue et d'autre part des valeurs du problème recherché.

3.3.3 Automatisation de la simulation

La figure 3.7 résume l'ensemble des opérations : un préprocesseur génère les fichiers d'entrée et de connexion à partir du nombre de sous-volumes dans les trois directions. Les objets (équations élémentaires) et les macro-objets (systèmes d'équation) ont été construits à partir de la liste des équations par l'intermédiaire d'un langage de calcul formel. Avec ces trois types de fichier (connexions, objets et macro-objets) SPARK assemble toutes les équations pour construire le solveur. Les résultats de la simulation sont ainsi obtenus sans intervention de l'utilisateur.

Enfin pour visualiser les résultats, une interface graphique a été réalisée au cours de ce travail permettant de connaître les températures, pressions et masses volumiques dans chaque sous-volume ainsi que les débits entre deux zones adjacentes. En thermique du bâtiment seuls les températures et les débits ont un intérêt.

Cette interface permet également de visualiser le confort et de connaître le champ de concentration et les débits de polluants ou de vapeur d'eau dont les calculs sont présentés dans la partie 5.

Nous décrirons cet utilitaire dans le prochain chapitre présentant la validation des résultats.

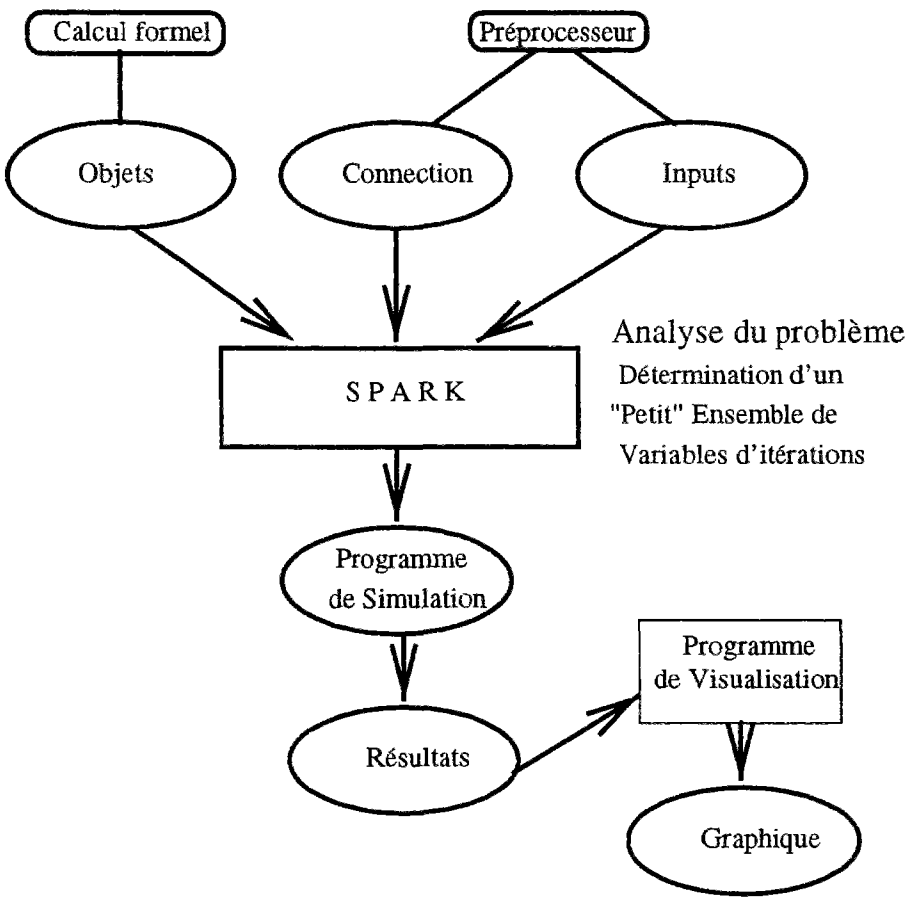

FIG. 3.7 - *Comportement d'une simulation*