Conception d'un environnement de simulation intelligent. Application à la thermique du bâtiment

Sidi Mohamed Karim El Hassar

THESE

Présentée à

L'ECOLE NATIONALE DES PONTS ET CHAUSSÉES

en vue de l'obtention du diplôme de

DOCTORAT DE L'ENPC

Spécialité : Sciences et Techniques du Bâtiment

Par

El HASSAR Sidi Mohamed Karim

CONCEPTION D'UN ENVIRONNEMENT DE SIMULATION INTELLIGENT

APPLICATION A LA THERMIQUE DU BATIMENT

Soutenue le 15 Décembre 1992

Devant le Jury composé de :

MM

A. DUPAGNE
P. DEPECKER
L. LARET
B. DELCAMBRE
R. PELLETRET
A. TROMBE

Je dédie ce travail à mes parents, et à mon frère Abdallah.

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

à M. et à Mme Fougere qui ont été les premiers à m'accueillir en France et qui ont contribué à ce que mon insertion en France se déroule de la façon la plus naturelle qui soit,

à M. Jacques Rilling pour son appui et ses conseils lors du choix du sujet de thèse,

à M. Roger Pelletret pour son aide efficace, pour la confiance qu'il m'a accordée et sans qui ce travail de thèse n'aurait jamais abouti.

Je remercie toutes les personnes qui ont accepté de participer à mon jury de thèse

M. A. Dupagne qui m'a fait l'honneur de présider le jury,
M. P. Depecker pour avoir accepté la responsabilité de rapporteur,
M. L. Laret pour avoir accepté la responsabilité de rapporteur et avec qui j'ai pu avoir des échanges scientifiques, nombreux et enrichissants,
M. B. Delcambre qui m'a fait l'honneur de faire partie du jury,
M. A. Trombe qui m'a fait l'honneur de faire partie du jury.

Je remercie toutes les personnes qui ont contribué au bon déroulement de cette thèse

M. M. Rubinstein Directeur du CSTB à Sophia Antipolis pour m'avoir admis dans le centre de recherche,
M. S. Soubra dont j'ai apprécié l'esprit d'équipe,
Mme A.M Dubois avec qui j'ai eu de nombreux échanges scientifiques.

J'adresse aussi mes remerciements à tous ceux qui travaillent au CSTB de Sophia Antipolis pour leur accueil ; je remercie surtout

M. T. Motte pour sa gentillesse,
M. et Mme Bazalgette pour leur assistance.

Table des matières

Liste des figures

Chapitre 1

Chapitre 2

Chapitre 3

Résumé

Résumé

Ce travail de thèse s'insère dans le cadre plus général du développement par le Centre Scientifique et Technique du Bâtiment d'un Environnement de Simulation Intelligent destiné à faciliter l'accès aux logiciels de calculs scientifiques dans le secteur Bâtiment.

Il s'agit d'un programme pluriannuel de recherches dont les objectifs sont de mettre à la disposition des professionnels de nouveaux outils d'aide à la conception et d'aide à l'analyse des systèmes "Bâtiments", basés sur la simulation numérique suivant divers points de vue (thermique, acoustique, structure, etc).

Le présent travail de recherche s'articule donc avec d'autres travaux menés en parallèle sur le même thème ; en particulier, un autre travail de thèse s'est attaché à définir le concept d'Environnement de Simulation Intelligent et à développer un modèle profond d'un tel environnement, modèle dont le but est de faciliter la réalisation d'applications dédiées.

Dans ce contexte, les objectifs fixés au présent travail de recherche sont de trois ordres :

- contribuer à la définition des spécifications de l'environnement de simulation générique notamment en analysant le processus de modélisation/simulation et en proposant des solutions pour la réalisation du dialogue Homme/machine ;

- réaliser une application concrète que l'on peut considérer comme une phase expérimentale de mise en œuvre des concepts développés ; cette application est réalisée autour du logiciel de calculs thermiques TRNSYS ;

- définir, pour de tels environnements, les possibilités d'utilisation des systèmes à bases de connaissances (buts et moyens, évaluation de la pertinence des solutions).

Le premier chapitre est consacré aux réflexions conceptuelles générales menées sur la base d'une analyse bibliographique et aux analyses nécessaires à la réalisation de l'application. Les étapes de développement d'un code de calcul sont rappelées (modélisation mathématique, modélisation numérique, élaboration d'une architecture informatique) ; de cette analyse, il ressort que la structure d'un code de calcul se doit d'être modulaire, l'idéal étant une claire séparation entre d'une part le solveur numérique, et d'autre part les modèles de composants physiques ; apparaît ainsi le concept de bibliothèques de modèles (modèlothèques). Par ailleurs, on s'attache à définir en quoi le système Proforma (système de documentation de la connaissance physique associée aux modèles) est utile dans un contexte d'Environnement de Simulation Intelligent et quel est le lien avec des systèmes à bases de connaissance ; il est en particulier démontré qu'il n'y a pas dichotomie mais bien interaction entre Proforma et système à base de connaissances : les informations contenues dans un ensemble de Proforma peuvent être directement utilisées pour réaliser des bases de connaissance nécessaires aux raisonnements qui permettront d'aider l'utilisateur à différents niveaux lors du processus de modélisation/simulation. Enfin le code de calcul qui doit faire l'objet de l'application est analysé du point de vue de son intégration dans un ESI.

La seconde partie du travail est consacrée à l'application proprement dite ; un environnement dénommé IISIBât (Interface Intelligente pour la SImulation dans le Bâtiment) est développé suivant les concepts précédemment définis ; dans une première phase, la stratégie de développement d'une telle interface est précisée ; le processus de conception est fondé sur la mise en œuvre séquentielle de quatre modèles : les modèles cognitif, conceptuel, structurel et perceptif. Le modèle cognitif met en évidence les

tâches que l'utilisateur est amené à réaliser dans le cadre d'un outil de simulation ; le processus de modélisation/simulation est décortiqué en ses diverses étapes ; ce travail permet de définir une interface homme/machine possible (et en particulier les outils nécessaires) spécialisée pour ce type de procèss. Le modèle conceptuel identifie les objets génériques sur lesquels se fonde l'interface, les fonctions rattachées à ces objets, et la représentation graphique de ces objets. Le modèle structurel s'intéresse à l'implémentation proprement dite. Le modèle perceptif représente la façon dont l'utilisateur perçoit le système final ; ce modèle conduit à une analyse critique des choix effectués et à la proposition d'améliorations. Au final, l'application réalisée (IISIBât-TRNSYS) permet de juger de la pertinence des concepts développés tant du point de vue du fond que des méthodes utilisées ; ainsi, l'intérêt de l'application développée pour des utilisateurs "développeurs de modèles" ou "créateurs de projets" a pu être démontré ; par ailleurs, il s'est avéré que cette application ne possédait pas toutes les fonctionnalités nécessaires à une utilisation efficace par des utilisateurs terminaux (analystes de systèmes) ; pour combler ces manques, il faut rajouter à l'interface des mécanismes de raisonnement.

La dernière partie de ce travail s'intéresse à l'introduction au sein de l'application interactive de mécanismes de raisonnement faisant appel à des bases de connaissances. Les fonctions expertes doivent permettre d'aider un utilisateur à réaliser mieux et plus vite certaines tâches qui entrent dans le processus de modélisation/simulation. Ces tâches concernent principalement le choix des modules, la connexion des variables, le choix des valeurs numériques des diverses variables. Pour réaliser en pratique de tels mécanismes de raisonnement deux architectures possibles sont évaluées : les systèmes à base de connaissances et les systèmes multi-agents. Si ces mécanismes permettent effectivement de réaliser des fonctions d'aide intelligentes, ils présentent cependant des limitations qui résultent principalement de la diversité des règles formulées et, parfois, du manque de précision de ces règles ; la mise au point de systèmes-experts pour l'aide au processus de modélisation/simulation/analyse des résultats est essentiellement un problème de formalisation des connaissances expertes et non pas un problème de technologies informatiques. D'autres possibilités innovantes de raisonnement sont évaluées comme le raisonnement par analogie ou l'utilisation des principes de raisonnement de la physique qualitative ; dans un cas comme dans l'autre, même si ces pistes présentent des potentialités, elles n'en demeurent pas moins aujourd'hui encore des thèmes de recherche.

Mots-clés

Modélisation, Simulation, Bâtiment, Environnement de Simulation Intelligent, Dialogue Homme/machine, Système expert, Programmation Orientée Objet

Introduction

Introduction

Contexte

La présente recherche vise à contribuer au processus d'informatisation des professions du bâtiment. Elle s'insère dans le cadre du projet ESI : Environnement de Simulation Intelligent [SOUBRA 92]. Le besoin de créer des interfaces "intelligentes" est motivé par le souci d'élargir l'audience de certains logiciels de simulation développés dans le monde du bâtiment, en facilitant leur utilisation.

Conçus pour la recherche, ces logiciels sont basés sur des modèles et algorithmes performants ; cependant leur utilisation est réservée à un cercle restreint d'utilisateurs, du fait essentiellement de difficultés liées à la description des projets, à la modification des problèmes à simuler, et à l'analyse des résultats. Ces difficultés ont des origines diverses :

- ces codes de simulation sont souvent bâtis autour de modèles de composants complexes, dont les paramètres ne sont pas toujours adaptés aux données couramment utilisées dans le monde professionnel ;
- ces codes de calcul ont souvent été construits sans tenir compte de l'efficacité de l'interface entre l'homme et la machine ;
- ces différents codes de calcul ont été développés indépendamment les uns des autres, sans possibilité de communication aucune.

Des travaux de recherche récemment menés et pour certains d'entre eux encore en cours, apportent des éléments de réponse aux difficultés ci-avant mentionnées :

- d'importants travaux sont menés pour le développement de modèles adaptés [LARET 2-89] ;
- des progrès ont été réalisés dans l'ergonomie des logiciels, et il existe des outils de développement qui facilitent la réalisation d'interfaces Homme/Machine performantes ;
- la standardisation des structures de données [DUBOIS 92] peut favoriser la communication entre les codes de simulation. Par ailleurs, des efforts de standardisation concernant la documentation des modèles devraient permettre de constituer des banques de modèles de composants ; cette documentation standardisée contribuera à faciliter l'échange des modèles entre développeurs. De plus, il est important de réaliser le lien entre la documentation standardisée des modèles et la structure des données des projets des bâtiments.

L'idée est de développer des environnements de simulation qui facilitent les études techniques, qu'elles soient conceptuelles ou d'analyse, et qui en améliorent la qualité tout en augmentant le niveau de productivité. Pour ce faire, nous avons développé le concept d'Environnement de Simulation Intelligent (ESI), qui repose sur les idées suivantes :

- accéder aux codes de simulation performants ;
- faciliter la description des entités physiques à étudier ;
- faciliter la tâche du créateur de modèles pour l'intégration des nouveaux modèles dans un environnement de simulation ;
- faciliter la préparation des données de simulation ; en effet chaque code de simulation possède son propre langage langage d'entrée qui demande un temps d'apprentissage long et difficile ;
- faciliter la lecture et l'interprétation des résultats des simulations ;
- réaliser un ou plusieurs systèmes capables d'assister l'utilisateur en lui proposant des aides, des vérifications de cohérence, des tâches automatisées ;

- réaliser une structure de données commune à plusieurs codes de simulation, de façon à faire communiquer différents codes de simulation ;
- permettre la réutilisation des données, tout en s'assurant de la cohérence des descriptions techniques du projet considéré à travers différents points de vue.

Par ailleurs, le concept d'ESI repose sur une approche orientée objet dans laquelle l'environnement de simulation finalement réalisé n'est qu'une instance d'un modèle générique.

Objectifs

Ce travail a pour objectif d'une part de contribuer aux réflexions générales concernant le concept d'ESI, et d'autre part d'apporter des éléments de réponse à un certain nombre de questions :

- faisabilité du couplage entre des codes de simulation écrits dans un langage de programmation algorithmique avec des langages utilisés pour le développement d'un ESI ;
- proposition de solutions concrètes pour améliorer le dialogue Homme/Machine ;
- proposition de solutions qui aident à la modélisation-simulation, notamment grâce à l'utilisation de bases de connaissances.

Méthodologie adoptée

La méthode suivie pour remplir les objectifs ci-avant exprimés est la suivante :

- réalisation d'une étude bibliographique autour des thèmes "modélisation/simulation", "programmation orientée objets" et "systèmes experts". Il ne s'agit pas de faire une étude exhaustive de chacun de ces thèmes, mais de relever pour chacun d'entre eux les aspects utiles à nos objectifs ;
- réalisation pour un code de calcul existant d'un ESI qui mette en œuvre les idées proposées ;
- évaluation de la pertinence des idées proposées ;
- proposition de nouvelles lignes directrices.

Plan du document

Le chapitre 1 est consacré à la présentation du contexte de ce travail, dont les aspects sont nombreux :

- présentation des principes généraux qui guident l'élaboration d'un logiciel ;
- présentation de l'environnement informatique pour la conception de l'interface (codes de simulation, langage de programmation) ;
- explication du concept d'aide à la modélisation/simulation.

Le chapitre 2 est consacré à l'interface réalisée. Sont présentées :

- la méthodologie de développement adoptée ;
- la structure de données qui a été mise en place ;
- une évaluation du prototype logiciel.

Le chapitre 3 est consacré à la recherche de l'augmentation de la productivité de l'utilisateur, notamment en introduisant les systèmes à base de connaissances.

Chapitre 1 : Contexte-Préliminaires-Choix effectués

Devant la complexité du problème posé par le développement d'un Environnement de Simulation Intelligent, il s'agit tout d'abord de définir le cadre de travail de thèse (paragraphe 1-1) et d'opérer des restrictions quant à nos objectifs. Il s'agit pour nous plutôt de raisonner sur un cas particulier d'ESI, c'est à dire sur une instance d'ESI, puis à essayer de tirer des généralisations du résultat obtenu. Cette démarche nous permet de choisir un niveau d'abstraction où les concepts qui seront mis en œuvre seront plus simples à manipuler. Pour ce faire, il s'agit tout d'abord de rappeler les différentes étapes du processus de conception d'un logiciel de simulation (paragraphe 1-2). La connaissance des aspects principaux de ce processus va influencer la conception de l'application interactive. Il s'agit ensuite de dégager les critères de qualité auxquels doit répondre un produit logiciel (paragraphe 1-3), ce qui va nous permettre de définir les caractéristiques que doit posséder l'environnement informatique du développement de l'ESI : choix du système d'exploitation, du type de langage de développement. Cependant, l'interface n'est pas tout, le produit interfacé doit être aussi une bonne application. Une étude des codes de calcul nous a permis d'opérer un choix parmi les codes existants (paragraphe 1-4). Il nous est aussi apparu qu'un Environnement de Simulation Intelligent doit incorporer en son sein des mécanismes d'aide. En effet, dans le cas d'une application qui entre dans le cadre de la modélisation/simulation, l'utilisateur non expert ne connait pas, a priori, la procédure à suivre au sein du système informatique : l'utilisateur est généralement démuni pour contrôler le processus de modélisation/simulation et pour en apprécier les résultats (multiples, méconnus et qui ne sont pas prédéterminés). Le paragraphe 1-5 dresse une première liste d'applications possibles dont l'objectif est l'assistance de l'utilisateur.

1-1 Positionnement

Le projet ESI a pour objectif de développer un environnement de simulation générique permettant l'intégration de logiciels de simulation pour des domaines d'applications variés (le bâtiment pour ce qui nous concerne). Il s'agit d'autre part de réaliser des applications, qui se situent à plusieurs niveaux et qui concernent des publics différents [PELLETRET 1-92].

Les travaux réalisés dans le cadre de cette thèse consistaient à contribuer au développement du concept d'ESI, notamment par la réalisation et le test d'une instance de ce concept. Pour ce faire, nous avons réalisé un environnement fondé sur le principe de la modélisation par assemblage, cet environnement est appelé IISIBât (Interfaces Intelligentes pour la Simulation dans le Bâtiment). Dans la suite de ce document, nous ferons référence essentiellement à cette application du concept générique d'ESI.

Les travaux menés dans le cadre du projet IISIBât portent principalement sur le développement d'un certain nombre de fonctions d'aide à la modélisation, à la simulation et à l'analyse des résultats ; ces fonctions utilisent les informations relatives aux modèles (consignées sous forme de PROFORMA [DUBOIS 90]). Dans ce cadre, une des tâches du projet IISIBât consiste à envisager le couplage avec les banques de données de composants manufacturés. Par ailleurs, IISIBât a pour objectif d'intégrer un modèle de données des projets de bâtiments, adapté aux "points de vue" des différents évaluateurs incorporés dans cet environnement. Une tâche importante du projet IISIBât est de faire le lien entre le mode standardisé d'archivage des connaissances sur les modèles (PROFORMA) et le modèle de données interne à IISIBât. Une autre tâche importante est de démontrer la pertinence de l'utilisation couplée de divers logiciels, ce qui, a priori,

permet d'accroître les potentialités de l'environnement de simulation en termes d'objectifs et de précision des calculs [PELLETRET 1-92].

Les applications en développement actuellement sont :

- l'application **IISIBât/TRNSYS**[1] qui intéresse principalement les laboratoires de recherche ; elle constitue elle-même un nœud à partir duquel d'autres applications spécialisées peuvent être développées (**IISIBât/CA-SIS**[2]).

- l'application **IISIBât/COMIS**[3] qui s'adresse également aux centres de recherche. Comme pour IISIBât/TRNSYS, il est tout à fait possible de dériver des sous-applications spécialisées.

Les applications IISIBât/TRNSYS et IISIBât/COMIS peuvent être aussi couplées avec des banques de données "constructeurs", de façon à les rendre opérationnelles dans des bureaux d'études.

D'autres applications sont envisagées, à moyen terme :

- l'application **IISIBât/TRNSYS/COMIS** s'inscrit dans l'idée d'un couplage entre logiciels de calcul, ce couplage permettant de couvrir différents aspects des performances d'un bâtiment à partir d'une même application interactive.

- L'application **IISIBât/SPARK**[4] va permettre d'utiliser un solveur numérique performant (SPARK [NATAF-WINKELMANN 91]). Ce solveur permet une distinction claire entre modèle physique et résolution numérique. Cette séparation entre objets représentant des composants technologiques ou des phénomènes physiques, et des méthodes de résolution confère à SPARK une grande modularité et de réelles capacités d'adaptation à n'importe quel domaine d'application. Il est donc possible de dériver aisément l'application IISIBât/SPARK vers n'importe quel domaine d'application.

- L'application **IISIBât/CSTBât**[5] va permettre de valoriser des modèles complexes développés au sein de la structure du code de calcul TRNSYS [LARET 1-89].

---

1 TRNSYS est un logiciel de calculs thermiques pour le bâtiment ; il a été développé à l'Université de Wisconsin-Madison [TRNSYS 90].

2 CA-SIS est un logiciel dérivé de TRNSYS, centré sur les performances des installations de climatisation, développé par EDF.

3 COMIS est un logiciel de calculs aérauliques, développé par le Lawrence Berkeley Laboratory [COMIS 91].

4 SPARK est un solveur numérique développé par le Lawrence Berkeley Laboratory.

5 CSTBât est un logiciel dérivé de TRNSYS, développé par le CSTB (Centre Scientifique et Technique du Bâtiment) [LARET 1-89].

Le projet IISIBât peut se résumer par la figure 1-1. Ce travail de thèse s'inscrit principalement dans le cadre du développement de l'application IISIBât/TRNSYS.

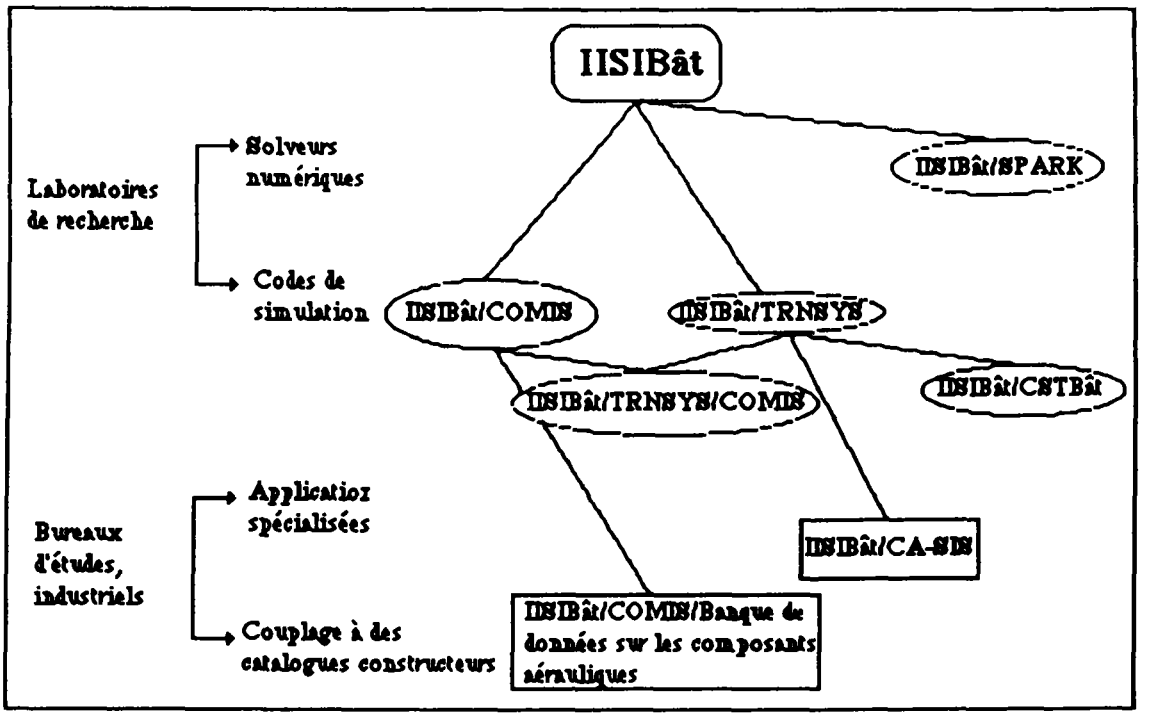


Figure 1-1 : Exemples des différents niveaux d'application du projet IISIBât

1-2 La simulation

L'objectif général d'un programme de simulation est d'analyser et de prédire le comportement d'une réalité externe, appelée aussi système physique. Beaucoup de ces systèmes physiques se prêtent à la simulation par événements discrets. Celle-ci consiste à décrire des systèmes physiques dont les états changent à la suite d'événements individuels, qui se produisent à des instants précis. En entrée de la simulation, il y a une suite d'événements ou de données avec leurs date de réalisation. Cette suite est obtenue en général grâce à des mesures sur de vrais systèmes physiques (lorsque la simulation est utilisée pour analyser des phénomènes passés). Elle est aussi obtenue grâce à des générateurs de nombres aléatoires conformément à certaines lois statistiques.

Un modèle à événements discrets doit notamment observer le temps du système physique, appelé *temps simulé*. Ce temps simulé ne doit pas être confondu avec le temps de calcul nécessaire pour exécuter le code de simulation. En général, le temps simulé est pour le programme de simulation une variable qui augmente par sauts discrets (variable *time* pour le code TRNSYS).

1-2-1 Objectifs de la Simulation

La simulation numérique donne à l'utilisateur la possibilité de faire varier différents paramètres d'un système en vue d'observer son comportement. La simulation numérique est généralement utilisée pour des phénomènes dont l'expérimentation réelle est dangereuse, impossible ou coûteuse.

Les objectifs peuvent être multiples :

- l'étude du comportement d'un système dans différentes circonstances ;
- l'évaluation de diverses stratégies pour le fonctionnement d'un système ;
- la détermination des limites de fonctionnement d'un système ;
- la prévision ;
- la vérification de théorie ;
- la mise au point de systèmes ;
- la compréhension des phénomènes physiques ;
- la compréhension du fonctionnement des systèmes physiques.

Ces différents objectifs vont avoir une incidence directe sur la conception d'une interface pour un outil de simulation.

Un utilisateur qui s'intéresse aux données et à leur interprétation doit avoir à sa disposition des outils pour l'aider dans sa tâche : traceur de courbes, logiciels de mise en forme de données statistiques. Par contre, un utilisateur dont l'objectif est la modélisation peut avoir besoin d'accéder aux fichiers sources, pouvoir en modifier certains et tester les incidences de sa modification. Dans ce cas, l'interface doit proposer à l'utilisateur des outils qui lui permettront d'accélérer la mise au point de ses programmes.

1-2-2 La conception d'un logiciel de simulation

Deux grandes tâches sont à prévoir lors de la réalisation d'un outil de simulation :

- la modélisation ; toute conception d'un logiciel de simulation passe par un travail de modélisation ; celle-ci consiste à construire des modèles qui sont des représentations abstraites des phénomènes-objets réels ; la modélisation est un travail qui comprend deux étapes, la modélisation mathématique et la modélisation numérique ;
- la mise en place d'une structure logicielle.

La modélisation mathématique consiste, après avoir fait une analyse physique du système à étudier, à dégager les flux jugés essentiels dans le cadre du problème à étudier et à les traduire sous forme d'équations (algébriques ou différentielles). Cette étape est délicate, il s'agit souvent de générer le modèle le mieux adapté au problème posé. Cette difficulté peut être contournée en proposant plusieurs modèles pour un même système, d'où la notion de bibliothèques de modèles (on parle de *modèlothèque*). Un modèle reste une représentation approchée, il représente certains comportements d'un objet ou système physique, avec un certain nombre d'approximations. Au cours d'une modélisation par assemblage, ces approximations peuvent expliquer des résultats de simulation inattendus : un effet indésirable dû à une approximation pouvant être propagé ou amplifié dans l'assemblage au cours d'une simulation.

La conception d'une interface doit tenir compte du grand nombre d'objets existant dans une modèlothèque ; elle doit permettre à l'utilisateur l'accès rapide aux modèles, ainsi qu'une aide pour leur choix dans le cadre d'un projet. A cet effet, une attention toute particulière doit être accordée aux fonctions relatives à l'archivage des composants dans une modèlothèque (cf. § 2-3-1-1, § 2-4-3).

La modélisation numérique consiste à traduire les équations définies lors de la modélisation mathématique en algorithmes de calcul. La modélisation numérique va elle

aussi introduire une approximation par rapport à la modélisation mathématique. De plus, l'algorithme de calcul doit être adapté aux possibilités de l'ordinateur.

Dans le monde de la thermique, les systèmes différentiels linéaires occupent une place importante ; ils sont généralement résolus à l'aide de méthodes qui discrétisent le temps (méthode d'Euler). D'autres méthodes de résolution discrétisent et le temps et l'espace : méthodes aux différences finies et méthodes aux éléments finis.

Remarque : il n'est pas possible de tout résoudre numériquement ; certaines équations instables sont difficiles, voire impossibles à résoudre numériquement (cf Annexe 2).

Principes de la modélisation mathématique

Dans le monde du bâtiment, la modélisation mathématique des phénomènes repose essentiellement sur des principes de conservation et sur des équations de transferts de deux types [LEBRUN 85] :

- transfert des flux d'énergie échangés entre sous-systèmes. On distingue les transferts qui ont lieu sans transfert de fluide, sous la forme de chaleur (modélisation de la conduction) ou de travail, et les transferts qui s'opérent avec transfert de fluide sous forme d'énergie interne, d'énergie cinétique, d'énergie potentielle, etc...

- transfert des flux de matière, comme les débits d'air, les débits d'eau.

Dans la modélisation mathématique, ce sont principalement les équations de transfert qui introduisent le plus grand nombre d'incertitudes et d'imprécisions. D'autres imprécisions sont introduites par les définitions de quantités telles que la chaleur massique, la conductivité thermique d'un corps. Dans la plupart des cas, on se contente de fabriquer des modèles simplifiés, à base d'hypothèses simplificatrices dont certaines reviennent souvent :

- isotropie et homogénéité d'un milieu ;
- régime permanent ;
- conduction unidimensionnelle ;
- tout transfert thermique peut s'exprimer linéairement en fonction des températures d'air (intérieures, extérieures), et des températures de parois ;
- les propriétés thermophysiques des matériaux varient très peu ;
- les échanges convectifs et radiatifs, bien que non linéaires, peuvent se linéariser.

La connaissance d'informations concernant la modélisation mathématique peut aider l'utilisateur à opérer des choix dans une bibliothèque de composants, et à interpréter des résultats. Ces informations peuvent être contenues dans des fiches, appelées fiches Proforma (cf. § 1-5-3). Si l'utilisateur sait que la modélisation d'un composant s'est faite à partir d'équations de transferts, il peut effectuer des déductions sur ce modèle quant à sa précision et au degré d'incertitude qui l'entoure.

1-2-3 Modularité et structure informatique des codes de simulation

Il existe dans le monde de la thermique du bâtiment un certain nombre de logiciels qui offrent la possibilité de simuler des phénomènes physiques. Ces outils diffèrent par les techniques numériques et logicielles employées, le niveau de détail des systèmes thermiques décrits et la convivialité des interfaces. Ces outils sont généralement

incompatibles entre eux, et l'étude complète d'un problème complexe de bâtiment nécessite l'emploi de plusieurs logiciels ; le chercheur est même parfois obligé d'insérer au sein d'un programme principal des morceaux de code spécifiques, ce qui a pour effet de rendre rapidement les programmes difficiles à maintenir.

Par ailleurs, les outils de simulation sont caractérisés par le domaine traité : on parle d'outil "général" et d'outil "dédié". Les développeurs sont confrontés au dilemne suivant : un outil "général" est mal adapté à une application particulière, un outil "dédié" ne permet pas de traiter de problèmes, qui relevent d'un autre domaine que celui pour lequel il a été conçu. Le principe de la modularité permet de contourner cette difficulté : il est possible au sein d'une architecture modulaire d'insérer des modules écrits pour des applications a priori différentes ; certains modules peuvent être polyvalents ; d'autres peuvent être très généraux, d'autres encore peuvent être spécifiques (modules "dédiés"). Ainsi, la modularité permet d'avoir "au sein d'un seul logiciel plusieurs logiciels".

On peut donc distinguer les codes de simulation par leur structure informatique :

- certains possèdent une structure modulaire ;
- d'autres possèdent une "structure globale".

Ces derniers permettent en général une simulation globale du bâtiment et de ses équipements ; l'utilisateur doit décrire obligatoirement les caractéristiques de son bâtiment et de ses équipements pour pouvoir simuler. Ces logiciels opèrent des restrictions importantes quant aux variantes du système, et dans la prise en compte des interactions entre le bâtiment et ses équipements.

Par opposition à ces codes de simulation, il existe des logiciels permettant une simulation à *la carte*. Ces logiciels ont la particularité de posséder une importante bibliothèque de composants et de sous-systèmes, qui doivent être assemblés par l'utilisateur pour effectuer une simulation. Cet assemblage est entièrement dirigé par l'utilisateur, à ses risques et périls. Ces logiciels demandent donc un peu de dextérité de la part de l'utilisateur pour leur utilisation.

Ces logiciels sont généralement de type *modulaire* et sont structurés en *routines*, celles-ci modélisent les processus de façon autonomes. Chaque processus se caractérise par un cycle de vie bien défini : une routine appelle une autre, qui s'exécute complètement et renvoie le contrôle à l'envoi de l'appel. Ainsi chaque processus suit sa propre vie et s'interrompt pour fournir ou recevoir de l'information d'un autre. Le schéma global de l'architecture informatique est complètement décentralisé : chaque processus s'occupe de sa propre tâche, avec une interférence limitée avec les autres.

1-3 Eléments de base pour la conception d'une application interactive

Avant de concevoir une interface, il paraît important de répertorier les exigences que doit satisfaire un produit logiciel. Ces exigences vont permettre de donner une appréciation sur sa qualité.

On peut définir neuf qualités essentielles d'un logiciel [MEYER 90] :

- la *validité* ; c'est l'aptitude du logiciel à réaliser exactement les tâches définies par sa spécification ;
- la *robustesse* ; c'est l'aptitude du logiciel à fonctionner même dans des conditions anormales. Il faut s'assurer que le logiciel, pour des tâches non spécifiables, puisse

quand même fonctionner, ou du moins ne pas déclencher des actions irréparables (destruction de fichiers de données par exemple).
- *L'extensibilité* ; c'est la facilité d'adaptation d'un logiciel aux changements de spécification ;
- la *réutilisabilité* ; c'est l'aptitude d'un logiciel à être réutilisé en tout ou partie pour de nouvelles applications ;
- la *compatibilité* ; c'est l'aptitude d'un logiciel à pouvoir être compatible avec d'autres logiciels ;
- *l'efficacité* ; c'est l'aptitude d'un logiciel à utiliser de façon optimale les ressources du matériel ;
- la *portabilité* ; c'est l'aptitude d'un logiciel à s'adapter à différents environnements ;
- *l'intégrité* ; c'est l'aptitude d'un logiciel à protéger ses composantes contre des modifications non autorisées ;
- la *facilité d'utilisation* ; c'est la facilité avec laquelle les utilisateurs vont apprendre à utiliser un logiciel.

En pratique, il est impossible de respecter simultanément tous les critères cités précédemment. A titre d'exemple, une efficacité optimale qui exige l'adaptation parfaite à un certain environnement est en contradiction avec le principe de la portabilité. Il faut donc effectuer un compromis.

Dans le cadre du projet IISIBât, la continuité est une préoccupation importante. Il ne s'agit pas de ne réaliser qu'une première version, il s'agit tout au contraire de penser à son cycle de vie qui peut s'étendre sur différentes phases d'adaptation et d'évolutions. Cette remarque conduit à accorder une attention particulière à l'extensibilité et à la réutilisabilité.

Il faut retenir deux principes qui permettent d'assurer l'extensibilité à un logiciel :

- la simplicité de l'architecture informatique ;
- la modularité de l'architecture informatique ; plus les modules de l'architecture logicielle sont autonomes, plus il est probable qu'une modification n'affectera qu'un nombre restreint de modules.

La compatibilité est à prendre en compte. En effet, les interfaces produites dans le cadre d'IISIBât doivent pouvoir communiquer avec le monde des outils de simulation (monde de TRNSYS) et interagir avec eux.

La facilité d'utilisation est un critère important dans le cadre d'IISIBât : il s'agit d'offrir aux utilisateurs une interface simple à appréhender et conviviale.

En résumé, les qualités essentielles à respecter dans le cadre d'IISIBât sont les suivantes : validité, robustesse, extensibilité, réutilisabilité, compatibilité, facilité d'utilisation.

1-3-1 Extensibilité, réutilisabilité et approche orientée objet

Pour la conception de l'interface, l'approche orientée objet répond aux critères énoncés plus haut. Elle répond notamment aux soucis de ***réutilisabilité et d'extensibilité***. Les programmes sont de nature modulaires grâce à la création d'entités cohérentes et facilement manipulables : les objets.

La question essentielle à laquelle est confrontée un développeur est la suivante "Quels objets sont concernés ? ". Cette question touche deux aspects ; d'un côté, elle concerne les objets appartenant à la réalité physique environnante (pompes, régulateurs, bâtiments...) ; d'un autre côté elle concerne les représentations informatiques appropriées, appelées aussi objets. On parlera dans la suite de ce document d'objets externes et internes. Au stade de la conception, il s'agit plutôt de passer en revue les classes d'objets externes dont le système veut modéliser le comportement ; au stade de l'implémentation, les systèmes à modéliser doivent être écrits comme des objets internes.

Chaque objet appartient à une classe qui détermine son fonctionnement et les attributs statiques le caractérisant. Une classe est un objet "abstrait" capable d'engendrer d'autres objets appelés *instances*. Une instance est un objet "concret" ; l'instanciation se fait par duplication de toute la structure de l'objet abstrait : les champs ou méthodes définis au niveau d'une classe se retrouvent dans tous les objets concrets instanciés à partir de cette classe. Les objets appartenant à une même classe ont la même forme, ont un comportement commun, mais sont différents par la valeur de leurs champs. Appliquer une opération à une instance consiste à lui envoyer un *message* contenant le nom de la méthode devant être activée.

Associée à ce concept de classe se rattache une notion importante : l'héritage de classe. Beaucoup d'objets peuvent être définis par rapport à d'autres, seules quelques caractéristiques les différencient. Une classe peut dériver d'une ou plusieurs classes appelées *superclasses*, dont elle hérite les variables et les méthodes. L'ensemble des classes forment ce qu'on appelle un *graphe d'héritage*. En général, l'héritage des variables est statique : les variables héritées sont recopiées dans la structure représentant la classe. Par contre, l'héritage des méthodes est dynamique : les méthodes héritées ne sont pas recopiées et la méthode à appliquer lors d'un envoi de message est recherchée dynamiquement dans le graphe d'héritage.

Une application est utilement formulée par un langage orienté objet quand elle en met à profit les caractéristiques. La structuration en objets se justifie lorsqu'on a à représenter des entités ayant des données propres en quantité suffisante et des comportements propres. L'instanciation est particulièrement utile dans des applications ayant de nombreux objets du même type. L'héritage trouve son utilité dans un environnement qui nécessite la construction de nouveaux types d'objets à partir de ceux existants. Ceci fait gagner en productivité en évitant la création inutile de code similaire.

La conception d'une hiérarchie de classes constitue le problème central. Dans le cadre d'un outil de simulation, la conception par objets est naturelle : il suffit de décrire les objets à simuler, et par conséquent la modularité d'un langage orienté objet sera mise à profit. L'instanciation et l'héritage seront également mis à profit, ainsi que les interactions entre les objets.

L'approche orientée objets n'est pas seulement utile pour le développeur de l'environnement de simulation mais elle peut aussi être utile pour celui qui développe et intègre de nouveaux modèles au sein de l'environnement de simulation. En effet, il existe au sein de l'ESI des fonctions d'aide à la modélisation et la simulation qui utilisent des propriétés et des informations définies au niveau des modèles. En utilisant la programmation orientée objet (POO), on crée des bibliothèques de modèles qui se présentent sous forme arborescente (cf. § 2-4-3) ; les nœuds de l'arbre constituent des classes auxquelles il est possible de rattacher des propriétés, communes à plusieurs modèles. Les modèles attachées à une classe peuvent hériter des propriétés définies pour cette classe (c'est le principe d'héritage propre à la POO). Ainsi, s'il existe une ou des

bibliothèques de modèles déjà structurées et implémentées sous formes d'objets, le concepteur d'un nouveau modèle pourra, en rangeant son modèle dans une bibliothèque, c'est à dire en rattachant ce modèle à une classe de modèle, bénéficier des propriétés génériques définies au niveau de la classe ou des éventuelles classes-mères.

Par exemple, supposons qu'il existe une classe "Pompe de Circulation" sous laquelle sont déjà rattachés des modèles de pompe de circulation. Admettons que nous ayons défini, parmi d'autres, comme propriété générique d'une pompe de circulation la propriété "possède en sortie un débit" et que cette propriété soit utilisée par un module expert pour réaliser des connexions automatiques entre modèles. Supposons alors qu'un créateur de modèle développe un nouveau modèle de pompe de circulation qui, par exemple, fasse le distinguo entre les pertes thermiques du moteur vers le fluide caloporteur et vers l'ambiance. Une fois son modèle créé, il le rattache à la classe "Pompe de Circulation" ; cette action induit l'héritage par le nouveau modèle des propriétés préalablement définies au niveau de la classe "Pompe de Circulation". Le nouveau modèle hérite du fait qu'il possède un débit en sortie. De plus, toutes les fonctions de l'environnement de simulation qui utilisent ce champ dans leur raisonnement deviennent opérationnelles pour le nouveau modèle.

1-3-2 Plate-forme logicielle retenue pour la conception de l'interface

Pour notre étude, l'environnement informatique est constitué du système d'exploitation UNIX. Celui-ci autorise la communication entre programmes par fichiers interposés, ce qui assure la compatibilité entre les différents logiciels ; de plus, il offre la possibilité de concevoir des interfaces graphiques conviviales, à base de menus déroulants, d'icônes, de fenêtres, pilotées par une simple souris, grâce à l'environnement X-Window (appelé aussi X11 ou tout simplement X).

Celui-ci peut être qualifié de standard "de fait", car implicitement adopté à la fois par les constructeurs de stations de travail et par les programmeurs. En général, X-Window est livré avec un gestionnaire de fenêtres, chargé d'habiller les fenêtres et de les contrôler ; on peut citer : UWM (Unix Window Manager), TWM (Tiled Window Manger), MWM (Motif Window Manager), OWM (OpenLook Window Manager).

Les avantages présentés par X-Window sont nombreux ; il ne privilégie aucun environnement de programmation, il est extensible et reconfigurable. Seule son implantation diffère d'une machine à l'autre.

X-Window s'appuie sur une librairie de fonctions (XLIB pour le langage C), mais celle-ci n'est pas simple à mettre en œuvre directement. Un certain nombre de langages de haut niveau qui facilitent le travail pour le programmeur sont apparus ces dernières années ; il s'agit principalement de langages interprétés et compilés (langage LISP[6]) et de langages compilés (langage C). Ils permettent de définir des objets graphiques et leurs comportements. Les langages interprétés permettent de tester rapidement des morceaux d'interface, les langages compilés sont plus rapides à l'exécution mais plus difficiles à implanter.

---

6 Dans un environnement LISP, il s'agit d'une compilation "au sens LISP", c'est à dire une transformation en un langage intermédiaire plus rapide (LLM3 pour Le-LISP).

Dans le cadre de ce travail, nous avons opté pour l'utilisation du langage Le-Lisp[7], enrichi d'une extension capable de gérer des objets graphiques : la boîte à outils Aïda[8].

Le-Lisp est un descendant du langage LISP, qui fut développé au laboratoire d'Intelligence Artificielle du Massachussets of Technology (MIT). Celui-ci donna lieu à la production de nombreux dialectes, Le-Lisp s'en distinguant par le fait qu'il s'appuie sur une machine virtuelle indépendante du système d'exploitation. De plus, Le-Lisp (version 15-25) est muni d'une extension objet (Ceyx), permettant une programmation orientée objet.

La boîte à outils Aïda est liée à la bibliothèque X-Window ; elle offre la possibilité de programmer des interfaces graphiques à partir de composants élémentaires, en utilisant les techniques d'héritage. Elle se compose d'une bibliothèque de plus de 100 classes de composants graphiques prédéfinis simples (boutons poussoirs, boutons radio, cases à cocher) ou complexes (barres de défilement, éditeurs d'arbres, éditeurs de texte, éditeurs de courbes).

L'utilisation de la boîte à outils Aïda (version 1.65) consiste à définir des structures, sous-structures des composants graphiques cités plus haut. La technique de l'héritage permet d'éviter de redéfinir les comportements des nouvelles applications.

Le-Lisp fournit des mécanismes puissants pour appeler des programmes écrits en d'autres langages (comme C, fortran ou pascal) ; cela permet de séparer de façon claire le programme (écrit en Le-Lisp, en C, ou en fortran) de l'interface graphique construite à partir de l'environnement Aïda.

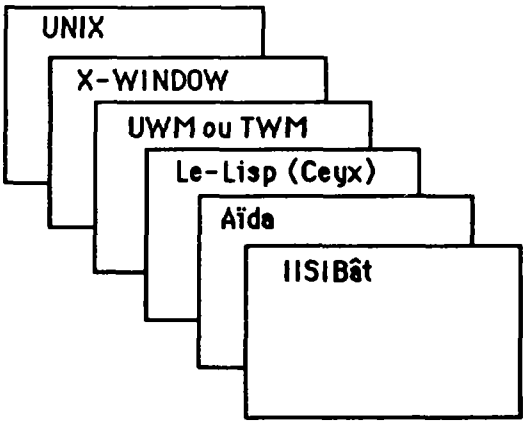


Figure 1-2 : Couches logicielles pour IISIBât

Remarque : **l'extension objet Ceyx sera probablement modifiée, au profit d'une autre extension objet (TELOS) ; certaines fonctions développées au cours de ce travail devront probablement être réécrites (une autre solution consiste à réécrire l'ensemble du code dans un langage indépendant de la couche objet -plate-forme MIPS [POYET 1-92]-).**

---

7 **Le-Lisp est un système LISP, développé à l'INRIA.**

8 **Aïda est un système de construction d'interfaces graphiques pour le système Le-Lisp ; il a été développé par la société ILOG.**

1-4 Code de calcul choisi

Notre choix s'est porté sur l'exploitation du code simulation TRNSYS [TRNSYS 90]. Plusieurs avantages sont présentés par ce logiciel :

- sa portabilité ;
- sa modularité ;
- sa bibliothèque de modules ;
- sa place dans le domaine des logiciels d'études thermiques qui en fait un outil de référence dans le monde de la thermique du bâtiment

Réalisé en Fortran de base, le code TRNSYS est un logiciel modulaire : il est composé d'un programme principal et d'un certain nombre de sous-programmes (ou modules). On distingue :

- les programmes qui modélisent un composant thermique ou un sous-système thermique ;
- les programmes utilitaires qui assurent la lecture des données, l'intégration numérique, l'impression des résultats, etc...

1-4-1 Principe de fonctionnement de TRNSYS

Pour simuler le fonctionnement d'un système global, l'utilisateur doit déterminer quels composants sont présents, à quels modules ils correspondent et quelles relations existent entre eux. La simulation s'effectue à partir du système ainsi reconstitué par modules et connexions.

Un module décrivant un composant du système est caractérisé par un certain nombre de paramètres et d'entrées-sorties connectées sur des entrées-sorties d'autres modules. Ainsi, le module "capteur solaire" est symbolisé de la façon suivante :

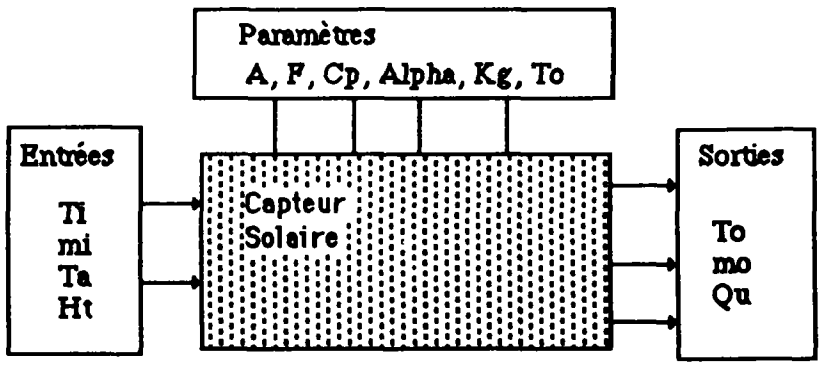


Figure 1-3 : module "capteur solaire" de TRNSYS

Les paramètres pour le module "capteur solaire" de TRNSYS sont :

- la surface (A) ;
- l'efficacité d'absorbeur (F) ;
- la chaleur spécifique du fluide (Cp) ;
- l'absorptivité de la plaque (Alpha) ;
- le coefficient global de déperditions (Kg) ;
- la transmittivité de la plaque (To).

Les entrées pour le module "capteur solaire" de TRNSYS sont :

- la température d'entrée du fluide (Ti) ;
- le débit du fluide (mi) ;
- la température ambiante (Ta) ;
- le rayonnement total incident (Ht).

Remarque : pour une simlation spécifique, une entrée peut être déclarée constante.

Les sorties pour le module "capteur solaire" de TRNSYS sont :

- la température de sortie du champ de capteurs (To) ;
- le débit du fluide (mo) ;
- l'énergie utile recueillie par le fluide (Qu).

Remarque : ces trois sorties (To, mo, et Qu) vont servir à leur tour d'entrées à des modules divers.

Pour certains modules, il faut ajouter aux entrées, sorties, paramètres la notion de "dérivatives" ; les "dérivatives" permettent de spécifier les valeurs initiales des dérivées premières par rapport au temps de certaines variables d'état.

La simulation d'un système complet nécessite la résolution numérique d'équations différentielles du premier ordre couplées. L'algorithme d'intégration utilisé est une méthode d'Euler modifiée, qui introduit un critère de convergence. S'il n'y a pas convergence au bout d'un certain nombre d'itérations au moins sur une variable, la simulation est avortée. Il faut noter que c'est l'utilisateur qui fixe le pas de temps et la tolérance de convergence.

1-4-2 Le fichier DECK

Une simulation est décrite par le fichier de simulation appelé "DECK". Ce fichier est composé de cartes alphanumériques : les cartes de contrôle et les cartes "Unit".

Chaque module mis en jeu au cours d'une simulation est décrit par une carte "Unit". Celle-ci contient les informations suivantes :

- n° d'unité dans la simulation et n° du type du module correspondant ;
- déclaration des paramètres (nombre, valeurs) ;
- déclaration des entrées (nombre, provenance, valeurs initiales) ;
- déclaration éventuelle des "dérivatives" ;
- déclaration éventuelle de cartes spéciales (par exemple la spécification du format de lecture et d'écriture des variables).

Les cartes de contrôle concernent les aspects généraux de la simulation. Les cartes de contrôle les plus importantes sont :

- **la simulation card** (définit le début, la fin et le pas de temps de la simulation) ;
- **la tolérance card** (définit la tolérance sur la convergence) ;
- **la limits card** (définit le nombre maximum d'itérations à chaque pas de temps).

1-4-3 Analyse de TRNSYS

La structure modulaire est le point fort de TRNSYS : celle-ci assure l'extensibilité de la bibliothèque de modules. Il est assez aisé de modifier un module ou même d'en créer de nouveaux.

L'autre point fort de TRNSYS est sa portabilité.

Par contre, l'exécution d'une simulation nécessite des va et vient permanents d'un sous-programme à un autre, faisant ainsi de TRNSYS un code gourmand en temps de calcul.

TRNSYS pose des problèmes de convergence : l'utilisateur doit choisir le pas de temps de la simulation, et la tolérance de convergence de manière à minimiser le temps de calcul tout en gardant une précision acceptable pour les résultats.

1-5 Aide à la Modélisation/Simulation

Le concept d'aide à la modélisation passe par le développement des axes suivants :

- mise en place d'une interface conviviale, qui tient compte de l'efficacité du dialogue homme / machine (cf. § 1-5-1) ;
- mise en place d'un système d'aide concernant le mode de fonctionnement de l'application interactive, et l'outil de simulation (cf. § 1-5-2) ;
- mise en place d'un système d'aide concernant l'utilisation des composants des bibliothèques (cf. § 1-5-3),
- mise en place de mécanismes intelligents qui guident l'utilisateur vers des solutions satisfaisantes, grâce notamment à la méthodologie des systèmes à base de connaissances (cf. § 1-5-4). Ces mécanismes concernent :

  - la mise en place d'un système d'aide à l'introduction, dans une bibliothèque de composants, de nouveaux modèles ;
  - la mise en place d'un système d'aide à la modification des modèles ;
  - la mise en place d'un système d'aide au choix de modules adaptés aux besoins des utilisateurs, en fonction d'un assemblage préexistant ou en fonction de l'objectif de la simulation ;
  - la mise en place d'un système d'aide au choix de méthodes numériques adaptées au problème posé par l'utilisateur ;
  - la mise en place d'un système d'aide à l'analyse des résultats de simulation.

1-5-1 Efficacité du dialogue Homme/Machine

Tout système d'interface est déformant vis à vis de l'information qu'il véhicule. Il est indispensable de développer autour du clavier et de l'écran des systèmes de dialogue qui soient de nature à favoriser le dialogue Homme/Machine. Ceci doit être réalisé dans des conditions qui respectent le plus le mode naturel de langage ou de concept mis en œuvre par l'homme avec les possibilités du système informatique.

L'homme a mis en œuvre la communication, caractérisée par une relative ambiguïté et une certaine redondance. Les systèmes informatiques, quant à eux, utilisent leurs propres langages où redondance et ambiguïté sont absents. Cette distance qui sépare ces deux systèmes d'informations pose problème : qui doit s'adapter à l'autre ?

1-5-1-1 Modélisation de l'interaction Homme/Machine

Plusieurs approches ont été développées, avec pour objectif la modélisation de l'interaction Homme/Machine. Dans le cadre de cette étude, nous rappellerons les aspects principaux des deux modèles suivants : le modèle STI (Système de Traitement de l'Information [BONNET 85]) et les modèles centrés objet [GOLDBERG 84].

Le modèle STI introduit la notion de mémoires. Il distingue :

- la mémoire à court terme, appelée aussi mémoire de travail, qui contient les résultats intermédiaires du raisonnement ;
- la mémoire à long terme qui stocke les connaissances pour une utilisation future, sous forme d'unités de connaissances.

L'augmentation de la productivité d'un utilisateur passe par une utilisation rationnelle de ces deux types de mémoires. Il faut retenir les principes suivants pour la conception d'une interface :

- éviter de surcharger la mémoire de travail de l'utilisateur ; cela consiste à le soulager, par exemple, de la manipulation d'informations telles que les domaines de validité des variables utilisées dans le cadre d'une simulation ; la solution consiste en général à introduire ces données dans une mémoire permanente sous forme d'unités de connaissances ; il s'agit ensuite d'associer à ces unités des mécanismes automatisés de *remémoration* ;

- proposer à l'utilisateur un environnement connu. L'utilisateur doit pouvoir poser ses problèmes et retrouver ce qui est nécessaire à son processus de réflexion, tout en utilisant des formalismes correspondant aux concepts qu'il a l'habitude de manipuler. Ainsi l'utilisateur consacrera l'essentiel de ses efforts à résoudre le problème physique posé et non pas à essayer de dialoguer avec la machine.

Les utilisateurs de TRNSYS travaillent généralement en réalisant un diagramme des flux (établissement des connexions entre les modules) avant d'en faire une traduction informatique. Au niveau de l'interface, l'utilisateur doit pouvoir constituer ce diagramme en manipulant des graphismes. Ceci permettra d'établir un dialogue purement conceptuel entre l'utilisateur et le système.

Les modèles centrés objets introduisent, quant à eux, le principe de la conception modulaire des composants des applications interactives. Cette approche sépare les deux notions suivantes :

- l'application qui rassemble les fonctions propres au domaine ;
- l'interface qui assure la communication entre l'utilisateur et l'application interactive.

La fonction de l'interface consiste à gérer les images à l'écran et à lire les données d'entrée. L'application quant à elle gère le contexte des actions de l'utilisateur.

L'aspect le plus important de cette approche est la répartition du dialogue Homme/Machine, chaque composant dialogue avec l'utilisateur suivant un langage et un formalisme qui lui est propre ; cette répartition du dialogue assure à l'interface extensibilité et réutilisabilité. Dans le cadre d'IISIBât, le dialogue est réparti entre les différents types de fenêtres, chaque type de fenêtre permet de gérer une activité ; malgré

tout, des efforts ont été menés pour rapprocher les différents formalismes associés aux activités en cours au sein de chaque type de fenêtre (cf. § 2-3-3).

1-5-1-2 Les différents types de dialogue Homme/Machine

Les premieres interfaces conçues étaient en général des interfaces textuelles, où l'utilisateur dialoguait avec le système via un système de questions/réponses prédéfini ; ce type d'interface est mal adapté à la description rapide et précise d'un bâtiment, du fait du grand nombre des informations à introduire lors de la conception des projets de bâtiment, de leur caractère hétérogéne et de la difficulté de description des systèmes techniques en langage naturel (il est difficile par exemple de décrire en une seule phrase un pont thermique). Ces dix dernières années, on a vu apparaître des interfaces, pour ce qui concerne le secteur bâtiment, où le dialogue Homme/Machine s'effectue via le dessin du bâtiment et de ses équipements : il s'agit des outils de CAO (Conception Assitée par Ordinateur). On peut penser qu'ils sont bien adaptés au mode de travail des différents intervenants du bâtiment ; en effet, à tous les niveaux de l'étude d'un projet de bâtiment est produit un plan graphique (structure, architecture, électicicité, ...). Cependant, la plupart de ces outils véhiculent une sémantique pauvre, ils ne permettent de manipuler que des données dimensionnelles, et des données touchant à la composition des éléments entrant dans la conception du bâtiment. Ils ne permettent pas de définir tous les comportements (thermiques, acoustiques...) des "objets" constituant le bâtiment, et les interactions de ces "objets" entre eux. Pour y arriver, il faudrait disposer d'un modeleur 3D, capable de générer un modèle de données intégré du bâtiment et de ses équipements (un tel modeleur est en cours de développement pour complèter la plate-forme MIPPE POYET 2-92]).

Dans l'attente de la mise au point d'interfaces graphiques basés sur des outils de CAO qui véhiculeraient une sémantique riche, construits à partir d'une approche orientée objets, il convient de développer des solutions intermédiaires pour le dialogue Homme/Machine ; dans le domaine qui nous intéresse, nous retiendrons comme possibilités :

- le dialogue fondé sur l'assemblage de boîtes noires ;
- le dialogue fondé sur l'assemblage de composants électroniques (résistances, capacités,...) ;
- le dialogue fondé sur l'écriture d'un système d'équations, selon une certaine syntaxe.

Chacun de ces types de dialogue présente avantages et inconvénients, en fonction des utilisateurs, des domaines d'application, et des codes de calcul. Par exemple, pour un mathématicien chevronné, s'exprimer au travers d'équations ne pose guère de difficultés ; quand on traite de modélisation de circuits électriques, décrire le problème par assemblage de composants électroniques est tout naturel.

D'une façon générale, pour appréhender un problème, l'être humain utilise couramment la technique de décomposition d'un système complexe en sous-systèmes plus simples, jusqu'à arriver à des briques élémentaires qu'il sait maîtriser. La description d'un problème par assemblage de composants épouse cette logique.

Il est parfois utile au sein d'un environnement de simulation de combiner divers modes de dialogue. S'il nous paraît aujourd'hui naturel de dialoguer par assemblage de composants, ce type de dialogue étant connu et facile à appréhender par les utilisateurs, il peut être parfois utile de mettre à la disposition de l'utilisateur des outils qui permettent par exemple de décrire une partie précise du problème, sous la forme d'un réseau RC.

Prenons un exemple concret pour étayer notre propos ; supposons la modélisation thermique d'une pièce sous la forme de l'assemblage suivant (figure 1-4-a).

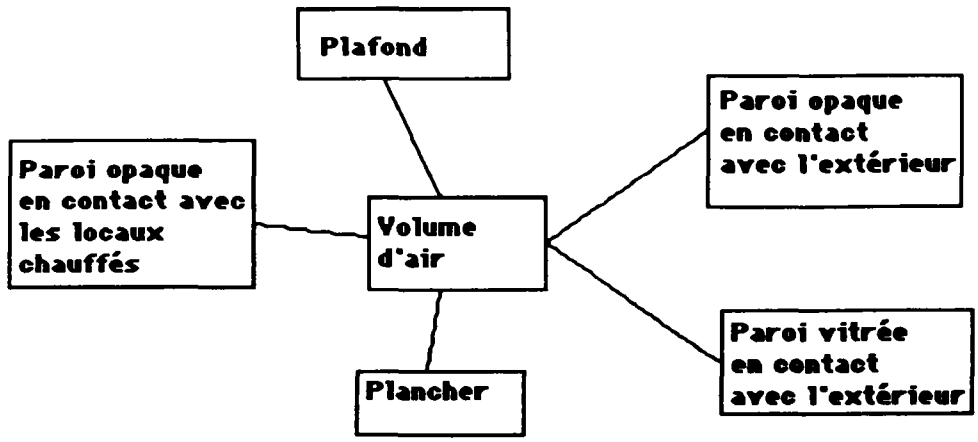


Figure 1-4-a : Modélisation thermique d'une pièce sous la forme d'un assemblage

Il est tout à fait possible que le modèle "Paroi extérieure", ait été conçu de façon à permettre une discrétisation aussi fine que souhaitée. On peut alors imaginer que, pour décrire cette discrétisation, on mette à la disposition de l'utilisateur une interface spécialisée qui lui permette d'assembler des conductances, des capacités, des générateurs de courant, des générateurs de tension...(figure 1-4-b)

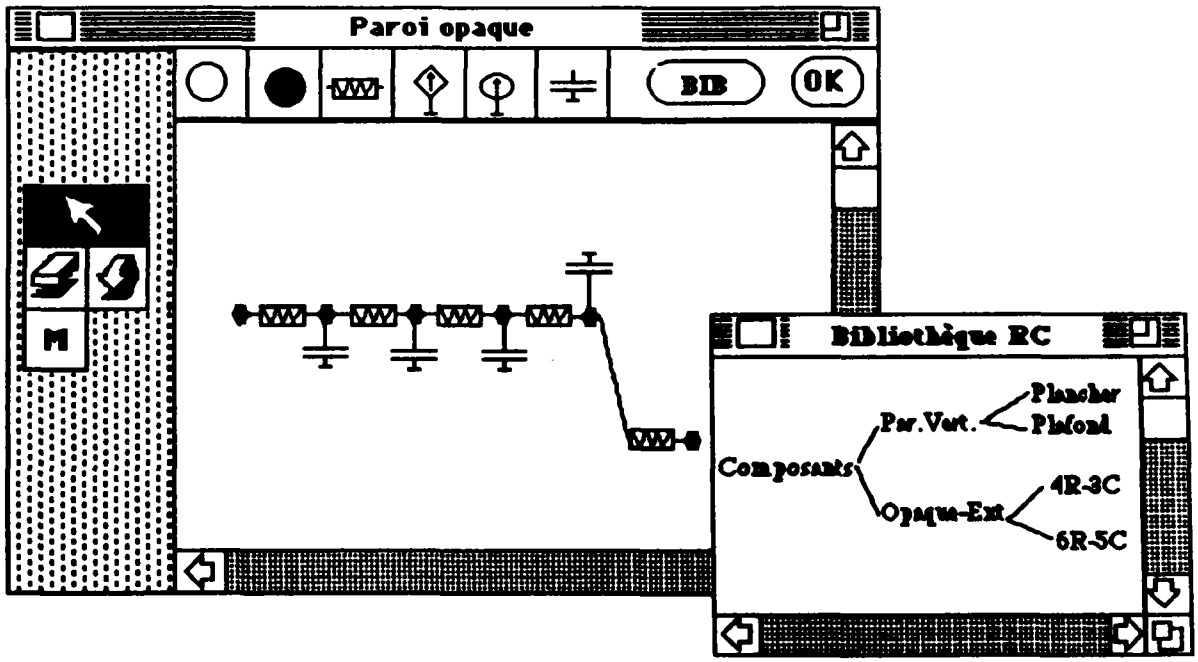


Figure 1-4-b : Sous-interface "Réseau RC"

La sous-interface "Réseau-RC" est constituée :

- d'un bouton, en haut à gauche, qui permet de quitter la session en cours ;
- d'un bouton "OK" qui permet de valider le réseau ;

- d'un bouton "BIB" qui permet d'ouvrir la bibliothèque de réseaux RC prédéfinis ;
- de composants élémentaires. Ils sont au nombre de six : le nœud qui représente une condition limite (cercle vide), le nœud sans inertie (cercle plein), la conductance, le générateur de courant, le générateur de tension et la capacité ;
- d'une boîte à outils ;
- d'une zone de travail.

Les outils qui permettent d'exploiter cette interface spécialisée sont :

-  permet d'instancier l'objet couramment sélectionné (en inverse-vidéo) qui peut être soit un composant élémentaire, soit un réseau prédéfini dans la bibliothèque RC ; il permet aussi de déplacer un objet dans la zone de travail, et de spécifier les valeurs numériques (double-clic) ;
-  permet d'effacer un composant ;
-  permet de revenir à l'état précédent ;
-  permet de stocker dans la bibliothèque RC le réseau qui se trouve dans la zone de travail.

1-5-2 Mise en place d'un système d'aide

Dans un outil logiciel, l'aspect "aide" est important ; il va permettre à un utilisateur de "mieux utiliser" le logiciel. Pour y arriver, l'aide doit concerner :

- l'interface et ses fonctionnalités ; l'utilisateur doit notamment être informé, sur la façon de manipuler les différentes fenêtres, sur les objets manipulables au sein de ces fenêtres, et sur la manière d'utiliser les différents outils. Les informations concernant l'interface, dans le cas idéal, n'ont pas lieu d'être (interface si simple que son utilisation est immédiate) ; dans le cadre d'IISIBât, l'une des exigences à respecter est la facilité d'utilisation de l'interface ; par conséquent, un effort tout particulier doit être fourni pour concevoir une interface simple d'utilisation, qui n'aurait donc pas besoin d'un support documentaire important ;
- l'outil de simulation ; l'utilisateur a besoin à ce niveau d'informations concernant le mode d'emploi du code de calcul. Ce mode d'emploi doit notamment faire apparaître des informations sur la manière de constituer de la façon la plus productive possible le fichier d'entrée pour la simulation (fichier DECK pour TRNSYS), en mettant l'accent sur les options existantes et sur les "subtilités" du code de calcul (par exemple les astuces pour faire converger plus vite les calculs...) ;
- la physique, c'est à dire les modèles existants au sein de l'environnement du code de calcul, grâce notamment à une description standardisée des modèles (fiches Proforma).

Par ailleurs, nous défendons l'idée d'un système d'aide intégré au logiciel, le support papier devant être à terme inutile. L'aide peut apparaître sous plusieurs formes, elle peut être :

- **ciblée ou globale** ;
- **statique ou interactive** ;

- **adaptée** à l'utilisateur ou non.

L'aide ciblée consiste pour le système à donner des informations qui ne concernent que l'objet de la demande de l'utilisateur, par exemple lorsque celui-ci déplace son dispositif de pointage (souris) sur des objets graphiques. L'aide globale consiste quant à elle à afficher un document complet plus ou moins astucieusement administré (structuration en chapitres, sous-chapitres, fonctions de requête), et qui concerne tout le domaine d'étude. L'aide globale posséde deux inconvénients : d'une part, elle oblige l'utilisateur à faire le tri entre les informations qui concernent son problème, et les autres. D'autre part, elle demande à l'utilisateur d'établir la correspondance entre la terminologie employée au sein de l'aide, et l'objet de son problème (qui peut être un bouton, un menu, une icône, une action -dupliquer, gommer-).

L'aide statique est une aide invariante, ceci quel que soit le contexte d'utilisation. Elle peut apparaître sous une forme textuelle ou graphique, pouvant comporter plusieurs niveaux (Hypertext). Pour ce qui concerne l'aide interactive, il s'agit dans la plupart des cas de mini-séquences didactiques, qui permettent d'expliquer le fonctionnement de tel ou tel outil ; l'utilisateur s'entraîne à "blanc", pendant qu'un module expert espionne ses faits et gestes, et lui explique ses erreurs éventuelles. Pour cela, le module expert doit connaître la façon de manipuler les différents outils, et doit être capable, d'une part d'analyser les actions de l'utilisateur, et d'autre part de générer des commentaires adaptés aux erreurs commises, et aux succès de manipulation.

Une aide adaptée à l'utilisateur est fondée sur le principe suivant : "à utilisateur différent aide différente". Il faut donc disposer d'un modèle performant de l'utilisateur, qui puisse évaluer son niveau de compétence. Illustrons ces propos par un exemple ; supposons que l'objectif de l'utilisateur soit l'accomplissement de l'action "Tout Effacer" ; une façon optimale de réaliser cette action consiste à combiner l'action "Tout Sélectionner" et l'action "Gommer". Si le système d'aide estime que l'utilisateur maîtrise l'action "Tout Sélectionner" (parce que le système sait que l'utilisateur en question a utilisé plusieurs fois déjà "Tout Sélectionner"), le message d'aide insistera plutôt sur l'explication du fonctionnement de l'outil "Gomme". La mise en place à l'heure actuelle d'un modèle de l'utilisateur est difficile, d'autant plus qu'il faut tenir compte de certains facteurs psychologiques tel que la notion de dégradation de compétence entre deux manipulations, distantes dans le temps.

Dans le cadre de ce travail, pour les raisons invoqués ci-avant, nous avons opté pour un système d'aide non adapté à l'utilisateur, tous les autres types d'aide (ciblé, global, statique et intéractif) peuvent être insérés au sein de l'environnement de simulation.

1-5-3 La fiche PROFORMA

Dans le cadre des travaux liés à l'analyse des phénomènes physiques, la documentation sur les modèles utilisés est de qualité variable et de présentation différente. Elle se retrouve parmi des documents hétéroclytes tels que des manuels d'utilisateurs, des thèses, des articles, des ouvrages spécialisées. En 1984, sous l'appellation de "Data Processor Proforma", le laboratoire de Physique du Bâtiment de LIEGE (B), avait proposé, avec le groupe ABACUS de l'Université de Stathclyde (GLASCOW-UK), une fiche descriptive des modèles de thermique. Cette première version a été appliquée dans les travaux de l'Agence Internationale de l'Energie, en liaison avec le CSTB. Ce dernier au sein du GER Almeth reprenait et développait le concept, en collaboration avec la FNB. Un projet, testé notamment à l'INSA de Lyon en 1985 et au CSTB, a abouti à la proposition du Proforma, cadre permettant à tous d'archiver de façon standard les

connaissances relatives aux modèles. Depuis, les réflexions sur le Proforma continuent au sein du GER-Almeth ; par ailleurs, les idées à la base du Proforma ont été reprises par différents laboratoires qui développpent des modèles (DETN-GDF, département ADE à EDF).

L'idée initiale du Proforma est d'offrir un outil pour faciliter la compréhension, et l'échange des modèles, grâce à une standardisation de la documentation associée aux modèles. Cette standardisation va permettre aussi d'assurer une certaine survie du modèle à son auteur. Depuis 1990, dans le cadre du développement du concept d'ESI, le Proforma est utilisé pour documenter sous forme de fichiers informatiques les modèles. Les informations contenues dans les Proformas sont utilisées pour divers processus de raisonnements intégrés dans les ESI.

Remarque : la nécessité de disposer de certaines informations spécifiques pour conduire des raisonnements au sein de l'ESI peut amener des modifications du Proforma.

A l'heure actuelle, les fiches Proforma sont utilisées pour :

- la documentation. Il s'agit de renseigner les utilisateurs sur les bases théoriques des modèles, sur les principales hypothèses qui ont guidé la modélisation, sur les méthodes de traitement utilisées,...
- l'aide à l'utilisation des modèles au sein des environnements de simulation ; une description simple et claire de ce que fait le modèle, de ses entrées, de ses sorties, de ses paramètres figure au sein de la fiche Proforma. Cette description permet à l'utilisateur de faire le choix du modèle le mieux adapté à son problème ;
- l'aide à l'interprétation des résultats ; les informations qui concernent les applications possibles du modèle, les limites de validité, les problèmes rencontrés permettent d'expliquer des anomalies éventuelles constatées au cours de l'utilisation. Elles permettent par exemple de comprendre pourquoi des valeurs calculées sortent de l'intervalle de validité donné dans les fiches Proforma.

Dans le cadre de notre travail, nous avons choisi de documenter les modèles de TRNSYS en gardant l'essentiel de la structure du Proforma. Les informations essentielles pour l'aide à l'utilisation des modèles sont :

- les hypothèses physiques sous-jacentes à la modélisation (linéarité, isotropie, etc...) ;
- le type du modèle (équationnel, algorithmique, expérimental, logique, qualitatif) ;
- les limites de validité (elles sont décrites essentiellement dans les domaines de valeur des variables) ;
- la liste des applications (un même modèle peut par exemple aussi bien représenter un refend, une paroi extérieure ou un plancher) ;
- les modèles amonts (modèles connectables en amont du modèle considéré) ;
- la liste des variables utilisées avec leurs attributs (nom, valeurs par défaut, unité, ...).

La fiche Proforma est structurée en cinq chapitres :

- le premier donne une information synthétique à l'aide d'un nom, d'un résumé et de quelques données sur les méthodes utilisées ;
- le second apporte la description formelle du modèle ;
- le troisième décrit le champ d'application du modèle par le biais de conditions de validité et de règles d'usage ;

- tout ce qui touche aux essais de validation est rassemblé dans le quatrième chapitre ;
- le cinquième rassemble les références bibliographiques.

Nous nous sommes posés les questions suivantes :

- Les rubriques sont-elles remplies et correctement remplies ?
- Que faut-il faire pour inciter les utilisateurs à remplir toutes les rubriques ?

Le travail de documentation est un travail pénible, la description d'un modèle dans le formalisme des fiches Proforma demande un effort non négligeable. Le temps de rédaction d'une fiche Proforma peut varier entre une journée et trois semaines, selon que le rédacteur possède ou non une bonne connaissance de la méthodologie du Proforma, selon qu'il soit ou non l'auteur du modèle et selon la complexité du modèle lui-même. Ainsi, il n'est pas rare de trouver des Proformas incomplets ou incorrectement remplis. Pour aider un utilisateur à écrire des Proformas, il serait souhaitable de disposer d'un logiciel ; un premier travail en ce sens a été mené au CSTB, il s'agit du logiciel PROFSYS [ESCUDIER 92] (PROFSYS est un prototype logiciel, implanté sur MacIntosh ; il utilise l'application HyperCard). Ce travail a permis une certaine standardisation du vocabulaire : on y retrouve des listes préétablies d'hypothèses, d'objets physiques, de phénomènes rencontrés en thermique du bâtiment. Il permet, d'une part de réaliser un gain de temps dans l'écriture des fiches Proforma (certaines tâches sont automatisées, comme par exemple la génération automatique des schémas-blocs), et d'autre part d'homogénéiser la présentation des documents papiers et du vocabulaire.

Il serait possible d'améliorer cette version, notamment en introduisant des vérificateurs :

- vérificateurs de formules, du point de vue des unités (équations aux dimensions) ;
- vérificateurs de cohérence ; par exemple déclarer des modèles amonts et pas de variables d'entrée.

Cependant, la version sur Macintosh n'est pas directement utilisable au sein de l'environnement Aïda, il serait plus pertinent de développer une version compatible avec le modèle abstrait des fiches Proforma utilisées au sein de l'ESI (les fiches Proforma des modèles incorporées dans les environnements de simulation sont des instances de ce modèle abstrait).

Remarque : quand on instancie le modèle abstrait de fiche Proforma, on l'adapte aux spécifités de description des modèles dans les différents codes de calcul considérés ; par exemple, le logiciel TRNSYS nécessite la définition de dérivatives (cf § 1-4-1) ; COMIS ignore cette notion, tout comme il ignore la notion de variables d'entrée/sortie.

Pour inciter un auteur à remplir une rubrique et à la remplir correctement, il faut non seulement lui donner des aides, mais lui expliquer aussi à quoi vont servir les informations. Si l'on explique au développeur de modèles que les informations contenues dans les Proformas seront manipulées dans le cadre d'un système expert (connexions automatiques, aide au choix de modules, aide à l'introduction des cartes de contrôle, aide au choix des valeurs des paramètres), cela l'incitera à remplir les fiches Proforma, car il prendra conscience de la valorisation qu'apporte à son modèle cet ensemble d'informations.

Un ensemble de fiches Proforma constitue une base de données à partir de laquelle il est possible d'élaborer des bases de connaissances, et donc des systèmes experts. Il faut

pour cela exploiter les informations contenues dans les fiches pour élaborer des règles de production. L'exploitation informatique des fiches Proforma n'est donc pas réduite à une simple manipulation de document. Elle peut s'effectuer autour des axes suivants :

- les connaissances qui donnent lieu à l'expression de règles de production ;
- les connaissances qui permettent d'automatiser certaines tâches, par exemple celles relatives à l'introduction de valeurs numériques (valeurs par défaut) ;
- les connaissances qui permettent de réaliser des tests de vérification (comparaison des valeurs des paramètres avec leur domaine de validité).

1-5-4 Mise en place d'une couche experte

L'étude bibliographique concernant les systèmes experts, en particulier centrée sur l'utilisation des systèmes experts dans le domaine de la modélisation/simulation a montré :

- qu'il s'agit encore de travaux de recherche ;
- que les applications sont peu nombreuses et très focalisées [ROBIN 91], [BAER 92] ;
- que le problème essentiel se situe au niveau de la constitution de la base de connaissances ; il existe peu de connaissances expertes correctement formalisées pour l'aide à la modélisation des bâtiments et pour l'aide à la simulation de leurs comportements.

D'une façon générale, la mise en place d'une couche experte a pour rôle de prendre en charge tout ou partie des mécanismes de raisonnement, qui sont utilisés habituellement dans le cadre d'un outil de simulation. Ces mécanismes font souvent appel à des connaissances éparses, parcellaires souvent d'origine expérimentale.

Ces différents raisonnements peuvent être reproduits en faisant appel à la méthodologie des systèmes experts. En effet, leur structure et leur fonctionnement se prêtent bien à la reproduction des facultés humaines de décision.

Le concept de système expert regroupe deux notions qui sont les connaissances nécessaires pour résoudre un problème et les mécanismes de raisonnement exploitant ces connaissances, appelés aussi moteurs d'inférences.

Les systèmes à base de connaissances peuvent être utiles dans le cadre d'un ESI. Illustrons nos propos : un utilisateur de TRNSYS doit, pour pouvoir simuler, construire un diagramme de flux, c'est à dire qu'il doit choisir les modules constituant son assemblage ainsi que les connexions à réaliser ; ces connexions obéissent à des règles, qui sont implicites pour un utilisateur chevronné. Par exemple, l'objet "capteur solaire" possède comme entrées :

- la température extérieure "Te" ;
- le rayonnement total incident "Ht" sur la surface du capteur ;
- le débit du fluide "mi".

L'utilisateur chevronné sait que la variable "Te" est issue directement d'un fichier météorologique, que le rayonnement total incident "Ht" est soit issu d'un fichier météo, soit issu d'un objet de type "processeur météo" et que le débit du fluide "mi" est une des sorties d'un objet de type "pompe". Ce savoir peut être formalisé par des règles de production qui permettront, à tout le moins des vérifications concernant l'exactitude et la

complétude des assemblages, et à terme d'automatiser la construction de schémas (cf. Chapitre 3).

1-5-4-1 Connaissances dans un système expert

Les connaissances peuvent être permanentes, temporaires ou une hypothèse émise au cours d'un raisonnement. Dans de nombreux systèmes experts, une distinction est faite entre la base de connaissances contenant la connaissance permanente et la base de faits contenant des connaissances et hypothèses initiales.

La connaissance peut être qualifiée de différentes façons :

- par sa précision ;
- par la certitude avec laquelle elle est donnée ;
- par son évolution au cours du temps ;
- par la finesse de détail de la description du problème à traiter.

En Intelligence Artificielle, il est indispensable de préciser clairement les limites du domaine d'application auquel se rattache les connaissances manipulées, on parle de *monde clos*.

Le mode de représentation des connaissances doit posséder un certain nombre de propriétés :

- celui-ci doit rester simple ;
- le temps de calcul doit rester un facteur à ne pas négliger.

Notons qu'il existe deux grands types de représentation de connaissances :

- les représentations déclaratives dans lesquelles les connaissances sont décrites indépendamment de leur exploitation ultérieure ;
- les représentations procédurales, qui contiennent la façon dont les connaissances doivent être utilisées.

1-5-4-2 Le raisonnement en Intelligence artificielle

Il existe une grande diversité de modes de raisonnement en IA, on peut citer :

- le raisonnement formel fondé sur la manipulation syntaxique de structures symboliques à l'aide de règles ;
- le raisonnement procédural dans lequel toutes les connaissances, la façon de les utiliser et la façon de conduire le raisonnement sont entièrement figés ;
- le raisonnement par analogie consistant à mettre en correspondance deux ou plusieurs situations ; on peut introduire à ce niveau de la notion de raisonnement par cas ;
- le raisonnement qualitatif basé sur la description physique d'un système.

Objectifs et conduite du raisonnement

L'objectif du raisonnement reste la résolution de problèmes. Raisonner nécessite le plus souvent l'examen d'un ensemble de solutions possibles. Ce parcours s'effectue à l'aide des méthodes classiques : en largeur, en profondeur ...

Il existe deux grands types de démarche en résolution de problèmes pour conduire un raisonnement face à un certain but recherché :

- partir des données disponibles sur l'état courant du problème à résoudre et utiliser les connaissances pour progresser vers une ou plusieurs solutions. Cette démarche est qualifiée d'ascendante, elle correspond au chaînage avant des règles ;
- partir du but et des connaissances disponibles pour transformer ce but en sous buts pour vérifier une situation ou une hypothèse sur les données du problème. Cette démarche est quant à elle qualifiée de descendante, elle correspond au chaînage arrière des règles.

Il est possible d'adopter une démarche bidirectionnelle combinant les deux démarches précédentes (chaînage mixte).

Difficultés dans la conduite d'un raisonnement

Un des problèmes auquel chacun est confronté reste l'imprécision des connaissances et des données sur lesquelles il raisonne. La conception des systèmes à base de connaissances performants nécessite donc de disposer de mécanismes définissant l'imprécision, et qui en tienne compte lors de sa propagation au cours des étapes de raisonnement.

Ce problème est résolu partiellement en faisant des suppositions sur les connaissances imprécises ou manquantes. La source la plus habituelle des suppositions est constituée de *valeurs par défaut* et de *domaines de validité* provenant de nombreuses origines :

- l'hypothèse du monde clos conduit à ne pas initialiser ou à ne pas accepter par exemple dans le cadre d'un projet ayant pour monde clos un bâtiment une température de zone de 150 °C ;
- le contexte du raisonnement peut permettre de deviner ou de calculer des informations manquantes (par exemple, le site d'implantation du bâtiment, centre urbain, rase campagne, bord de mer, permet de calculer le coefficient d'exposition au vent, lequel permet de calculer le débit de perméabilité à l'air des façades).

En plus de l'imprécision qui les entachent, les connaissances sont souvent caractérisées par leur évolution au cours du temps. Il est courant d'avoir à réviser au cours d'un raisonnement une décision prise antérieurement. Cela implique de disposer de mécanismes de gestion des hypothèses selon l'évolution des choses. Cela introduit donc les concepts suivants [HATON 91] :

- le concept d'évolution ; des faits nouveaux viennent contredire des faits acquis ;
- le concept de non-monotonie ; l'apport de nouvelles connaissances vient modifier des hypothèses formulées précédemment.

1-5-4-3 Réalisation pratique d'un système expert

La réalisation pratique d'un système IA sur un ordinateur pose des problèmes d'architecture matérielle et logicielle. L'un des problèmes reste la lenteur du moteur d'inférence interprétant une base de règles. Cette lenteur peut être gênante lorsque les contraintes de temps deviennent fortes (cas des interfaces). Dans le cycle de fonctionnement d'un moteur d'inférence, une part majeure du temps est consacrée à l'étape de recherche des régles applicables (parfois plus de 90%).

Système à base de règles

Le noyau d'un système à base de connaissances utilisant des règles comprend trois parties :

- une base de règles constituant la connaissance permanente ;
- un moteur d'inférence qui est le mécanisme de raisonnement exploitant ces règles ;
- une base de faits qui représente la mémoire de travail du système.

Une règle est généralement rédigée de la façon suivante :

[Si **Conditions** alors **Conclusions** (ß)]

La partie *conditions* doit être vérifiée pour que la régle s'applique. La partie *conclusions* correspond à l'action déclenchée lors de l'activation de la règle. Le coefficient ß traduit l'incertitude sur la connaissance exprimée par la règle, appelé degré de vérité ou coefficient de vraisemblance.

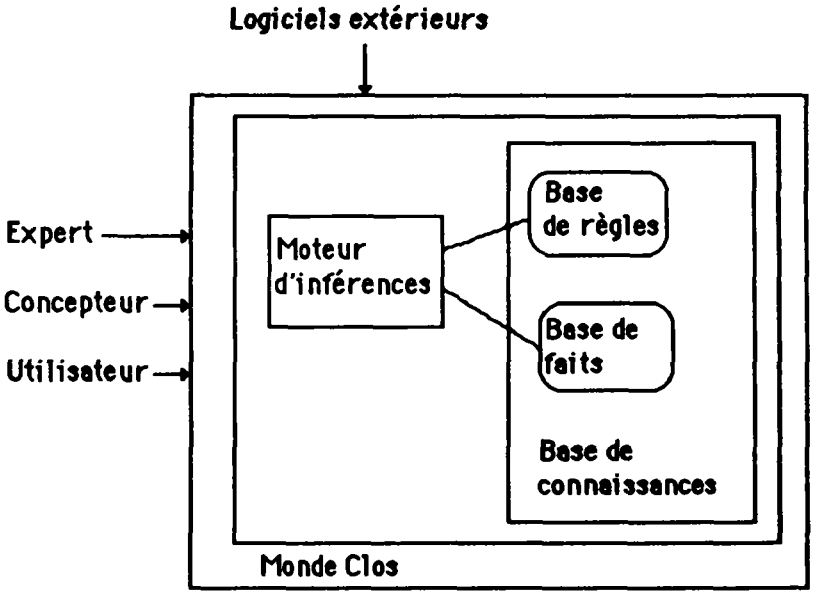


Figure 1-5 : Organisation d'un système à base de connaissances

Fonctionnement d'un moteur d'inférence

Le fonctionnement d'un moteur d'inférence comprend deux phases : la phase d'évaluation et la phase d' exécution. La phase d'évaluation comprend trois étapes : la sélection, le filtrage et la résolution de conflits.

La sélection détermine à partir d'un état présent ou passé de la base de faits et d'un état présent ou passé de la base de règles les faits et les règles qui méritent d'être comparées lors de l'étape de filtrage.

Le filtrage détermine les règles dont les conditions de déclenchement ont été jugées satisfaisantes.

La résolution de conflits détermine les règles qui doivent être effectivement déclenchées, ainsi que l'ordre de leur déclenchement, dans la phase d'éxécution

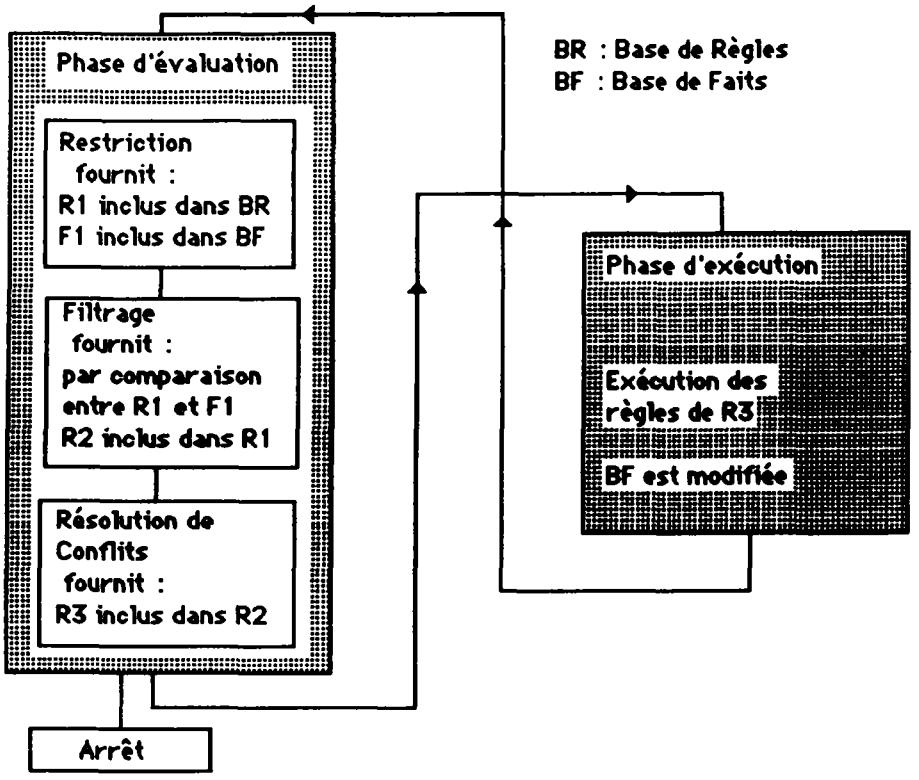


Figure 1-6 : Principe de fonctionnement des moteurs d'inférence [FARRENY 87]

Conclusion du chapitre 1

Nous avons définis au paragraphe 1-1 les grandes lignes (objectifs) des environnements de simulation que nous entendons développer. Pour que ces environnements soient à la fois faciles à faire évoluer, et qu'ils permettent d'offrir à l'utilisateur final toute une panoplie d'aides au processus de Modélisation/Simulation/Analyse des résultats, la programmation orientée objet nous est apparue comme la solution logicielle la mieux adaptée (cf. § 1-3-1). Elle est atout pour le développeur au sens où elle permet une programmation très bien structurée (définition de classes et d'un graphe d'héritage). Cette structuration rend le logiciel particulièrement facile à modifier et à faire évoluer. Pour l'utilisateur final, outre le fait qu'il peut bénéficier à moindre coût de versions améliorées, il peut éventuellement tirer parti des propriétés d'héritage. Les bibliothèques de modèles peuvent (doivent) être structurées selon une approche orientée objets ; ainsi, les propriétés génériques (attributs, méthodes) définies pour une classe seront automatiquement héritées par tout nouveau composant créé par un utilisateur final, et rangé par lui sous cette classe.

Nos réflexions ont ensuite porté sur le choix d'un environnement logiciel : nos analyses nous ont conduit à choisir le langage Le_Lisp, enrichi de la boîte à outils Aïda, au sein de l'environnement UNIX/X-Window (cf. § 1-3-2).

Un de nos objectifs est de démontrer à travers une application concrète la pertinence des concepts génériques que nous développons dans le cadre du projet ESI. Pour réaliser cette première application, nous avons recherché un logiciel modulaire, qui possédait déjà une bibliohèque de composants relativement étendue ; nous avons choisi le code de calcul TRNSYS (cf. § 1-4), qui est largement utilisé dans le monde de la recherche. Cette application spécifique dénommée IISIBât (Interfaces Intelligentes pour la Simulation dans le Bâtiment) est fondée sur la modélisation de systèmes complexes par assemblage de composants élémentaires. Les travaux relatifs au développement de cette application sont exposés au chapitre 2.

Cette application (IISIBât) doit permettre l'amélioration de la productivité des utilisateurs des codes de simulation ; nous pensons que cette amélioration doit passer par :

- la mise en place d'une interface conviviale, où l'efficacité du dialogue Homme/Machine est prise en compte (cf. § 1-5-1) ; pour cela, nous avons retenu deux approches qui permettent de modéliser l'interaction Homme/Machine, il s'agit du modèle STI (Système de Taitement de l'Information) et des modèles centrés objets ;

- la mise en place d'un système d'aide, intégré à l'application interactive, qui concerne le fonctionnement de l'application interactive et l'outil de simulation ; nous pensons qu'un système qui comporte en son sein plusieurs types d'aide (ciblé, global, statique, interactif, et non adapté à l'utilisateur -cf. § 1-5-2-) permet de répondre de façon efficace aux attentes des utilisateurs ;

- la mise en place d'un système d'aide qui concerne l'utilisation des composants des bibliothèques ; nous pensons que la structure du Proforma (cf. § 1-5-3) est bien adaptée à la documentation des composants des bibliothèques ;

- la mise en place de mécanismes fondés sur la manipulation de règles expertes qui guident l'utilisateur vers des solutions satisfaisantes ; ces règles expertes peuvent être écrites à partir des informations contenues dans les fiches Proforma ; en effet, un ensemble de fiches Proforma constitue une base de données à partir de laquelle il est possible d'élaborer des bases de connaissances, et donc des systèmes experts. Ces systèmes pourraient concerner :

    - l'aide au choix des systèmes, en fonction par exemple des objectifs de la simulation ;
    - l'aide au choix des modules, en fonction des objectifs de la simulation ou en fonction de modules constituant un début de solution ;
    - l'aide à la connexion des variables entre modules ;
    - l'aide à la manipulation des données numériques (vérification de validité, aide au choix de valeurs manquantes) ;
    - l'aide à l'introduction des paramètres de la simulation (pas de temps, début et fin de la simulation, tolérance de convergence) ;
    - l'aide à l'analyse des résultats (aide au choix de valeurs sensibles).

A priori, pour le type de connaissances que nous serons amenés à manipuler, et pour le type de raisonnement que nous comptons éventuellement introduire, la méthodologie des systèmes à base de connaissances semble bien adaptée (cf. § 1-5-4). Ainsi au chapitre 3, nous développons les divers types de raisonnement qui peuvent aider les utilisateurs dans leurs démarches de Modélisation/Simulation/Analyse des résultats.

Chapitre 2 : Présentation de l'interface réalisée

Chapitre 2 : Présentation de l'interface réalisée

Les aspects logiciels pour la conception d'une application interactive viennent d'être résolus, il existe aujourd'hui des outils qui, d'une part réduisent le temps de programmation d'une interface (Aïda pour ce qui nous concerne), et d'autre part permettent d'obtenir des applications cohérentes, puissantes et simples d'usage. Certains de ces outils logiciels sont à l'heure actuelle encore en plein développement ; l'approche orientée objets est fondamentale pour ce domaine. Cependant, concevoir une interface en ne tenant compte que de l'aspect logiciel conduit à des échecs pour le concepteur ; il faut aussi tenir compte des avis de personnes travaillant dans des mondes divers : ergonomes, cogniticiens, graphistes, spécialistes de la communication.

La construction d'interfaces Homme/Machine passe notamment par une bonne connaissance des utilisateurs potentiels, connaissance exprimée en termes d'habitudes de travail. L'objectif est de concevoir des interfaces qui répondent aux besoins des utilisateurs, et faciles à appréhender.

Concevoir une interface est une tâche délicate dans la mesure où un même système est destiné à plusieurs catégories d'utilisateurs. Les catégories de pensée, les concepts et la terminologie auxquels chaque groupe se réfère sont tributaires du métier exercé, du degré de spécialisation, mais aussi de la culture propre d'une entreprise. Pouvoir élaborer des versions différentes d'un même système constitue une solution pour s'adapter à l'utilisateur. L'aspect générique de l'ESI trouve là tout son sens, il doit permettre d'obtenir rapidement des interfaces différentes, qui répondent aux habitudes de la plupart des catégories d'utilisateurs.

Le chapitre 2 est structuré de la façon suivante. Le paragraphe 2-1 décrit de façon générale le processus de conception d'une application interactive. Le paragraphe 2-2 présente le modèle cognitif sur lequel se fonde le développement de l'interface. Le paragraphe 2-3 est consacré à la présentation du modèle conceptuel de l'application interactive. Le paragraphe 2-4 présente le modèle structurel de l'interface. Le paragraphe 2-5 présente une évaluation du travail effectué.

2-1 Le processus de conception d'une application interactive

De façon générale, le processus de conception se divise en quatre étapes [BEAUDOIN 91] ; à l'issue de chaque étape un modèle est produit. On obtient ainsi les modèles suivants :

- le modèle cognitif ;
- le modèle conceptuel ;
- le modèle structurel ;
- le modèle perceptif.

Le modèle cognitif consiste en l'analyse des tâches de l'utilisateur et prend en compte les facteurs humains généraux, ainsi que ceux propres à l'application envisagée. Il s'agit dans ce modèle de définir les habitudes de travail, les besoins des utilisateurs, et les contraintes techniques.

Le modèle conceptuel a pour objectif de rationaliser les résultats issus du modèle cognitif. La description du modèle conceptuel passe par la décomposition de l'application interactive en plusieurs fonctions d'interaction : gestion des données (sauvegarde, chargement), présentation des objets graphiques à l'écran (création de fenêtres, défileurs...). Une autre phase consiste à identifier les objets, leurs propriétés, et les opérations qui leur sont associées.

Le modèle structurel de l'interface s'intéresse à l'implémentation proprement dite, à partir des boîtes à outils et outils existants, en respectant les guides de style qui leur sont associés.

Le modèle perceptif représente la façon dont l'utilisateur perçoit le système final. Si les modèles conceptuel et perceptif correspondent, l'utilisateur peut anticiper le fonctionnement du système et donc l'utiliser de façon efficace. Si les deux modèles sont trop éloignés l'un de l'autre, l'utilisateur peut ne pas comprendre les réactions du système interactif. La manière la plus adéquate pour évaluer le modèle perceptif est de réaliser des expérimentations auprès des utilisateurs. Il s'agit en fait de réaliser des prototypes, en associant les utilisateurs potentiels au sein du processus de conception, l'objectif étant l'amélioration de la qualité du logiciel.

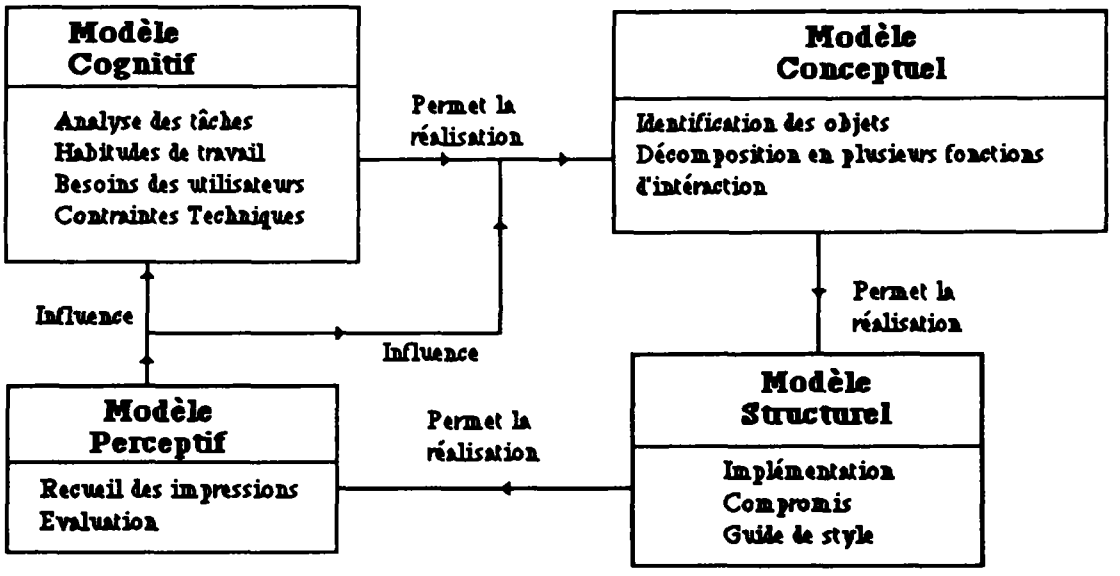


Figure 2-1 : Processus de conception d'une interface

2-2 Modèle Cognitif

Il s'agit pour nous de dégager, d'une part les méthodes de travail couramment suivies pour exploiter TRNSYS et les scénarios d'utilisation les plus fréquents, et d'autre part un certain nombre d'idées clés concernant le dialogue Homme/Machine, notamment en faisant référence aux environnements informatiques réputés les plus conviviaux. Les résultats de ce travail préliminaire vont nous permettre de proposer un environnement intégrant des méthodes qu'ont coutume d'employer les différents utilisateurs de TRNSYS ; par conséquent, cela va favoriser l'appropriation de ce nouvel outil par les nombreux anciens utilisateurs de ces codes de simulation, par le fait que l'environnement

proposé ne leur sera pas totalement inconnu (notons qu'en général, un utilisateur préfère utiliser un système dépassé dont il a l'habitude plutôt qu'un nouveau logiciel, a priori plus convivial, plus performant, mais également inconnu et qu'il doit donc commencer à apprendre).

De plus, l'analyse des méthodes de travail utilisées pour exploiter TRNSYS va nous permettre de mettre en évidence les facteurs qui freinent l'utilisation du code de calcul, ainsi que les phases qui ralentissent le travail des utilisateurs, et parmi celles-ci celles qui peuvent être automatisées toutes ou partie.

2-2-1 Identification des tâches

Rappellons que les utilisateurs de TRNSYS procédent par la mise en place d'un diagramme de flux (cf. § 1-4-1) : il s'agit pour eux de constituer un système composé de modules, représentés par des boîtes noires, et de connecter les variables entrées/sorties qui doivent l'être. Pour certains modules, la représentation sous forme de boîtes noires ne suffit plus, l'utilisateur doit constituer parfois un réseau RC (module bâtiment du code de calcul CSTBât [LARET 1-89] par exemple - cf. § 1-5-1-2 -).

Les principales tâches qu'accomplit un utilisateur de TRNSYS sont les suivantes :

- introduction de nouveaux objets dans une bibliothèque (tâche A) ;
- modification d'un objet dans une bibliothèque (tâche B) ;
- échange de modèles entre utilisateurs (tâche C) ;
- préparation des données de simulation ; elle consiste pour l'utilisateur à faire le choix des modules (tâche D1), à réaliser les liaisons entre modules (tâche D2), à connecter les variables d'entrée/sortie (tâche D3), à introduire les valeurs numériques pour les paramètres (tâche D4) et à spécifier les cartes de contrôle (tâche D5) ;
- lancement de simulations (tâche E) ;
- analyse de résultats (tâche F).

Dans la plupart des cas, l'enchaînement des tâches, pour un but donné, est classique : préparation des données de simulation, lancement de simulations, analyse de résultats (on parle dans ce cas d'enchaînement critique). L'enchaînement des tâches, dans le cas où l'utilisateur veut étudier un système dont tous les composants existent dans une de ses bibliothèques, est schématisé par la figure 2-2.

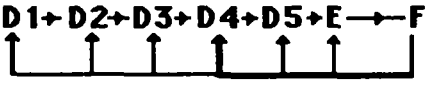


Figure 2-2 : Enchaînement des tâches

La boucle D1, D2, D3, D4, D5, E, F, D4, D5, E, F, D4, D5, E, F... constitue l'enchaînement le plus probable. Ce scénario très fréquent dans l'utilisation d'un code de simulation consiste pour l'utilisateur, une fois un système conçu, à faire varier un ou plusieurs paramètres. L'objectif peut être l'observation du comportement du système (pour étudier par exemple le système dans les conditions limites de fonctionnement) ; l'objectif peut aussi consister à faire tendre la valeur d'une variable vers une valeur numérique spécifiée dans un cahier des charges (température de zone appartenant à un certain intervalle par exemple). Cela nécessite d'une part de choisir les "bons paramètres", de leur affecter les "bonnes valeurs", et d'autre part de lancer plusieurs simulations. L'interface doit répondre à cette attente (cf. § 3-2-4) en proposant des aides pour le choix

des paramètres sensibles, et pour l'automatisation de certaines tâches, comme le lancement automatique de plusieurs simulations.

Par ailleurs, l'étude de l'étape qui consiste à rechercher un module approprié (tâche D1) est intéressante ; elle peut aboutir aux scénarios suivants (figure 2-3) :

- soit l'utilisateur trouve le module dans une librairie qui lui appartient ;
- soit l'utilisateur trouve un module proche de ses exigences, l'interface doit permettre alors à l'utilisateur de modifier facilement le module ;
- soit l'utilisateur trouve le module dans une librairie d'un autre utilisateur, il devra alors pouvoir copier ce module dans une de ses librairies ;
- soit l'utilisateur ne trouve pas de modules appropriés, il devra alors le créer (il devra développer alors du code Fortran).

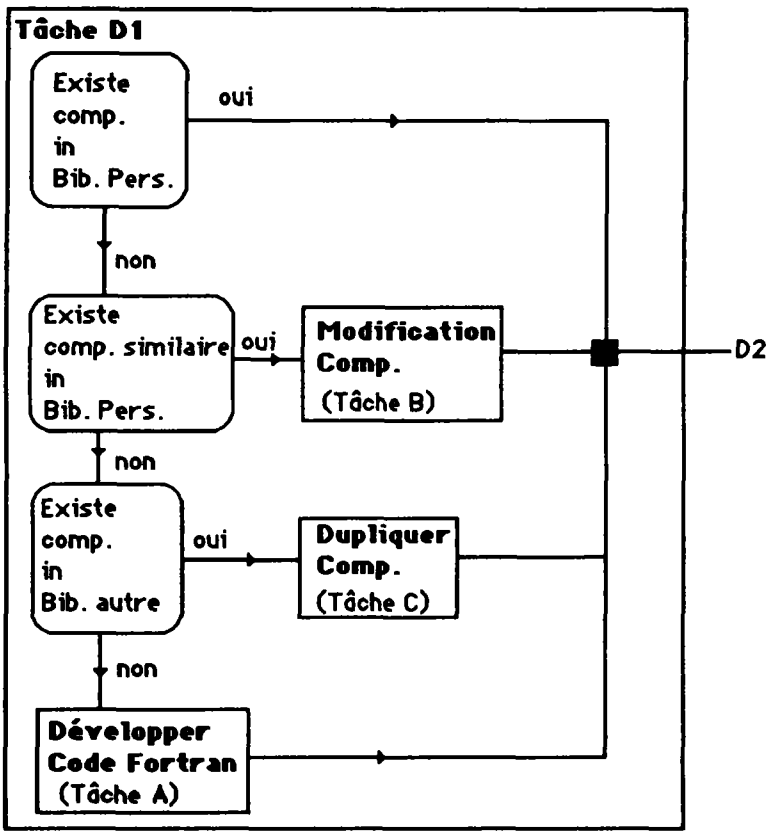


Figure 2-3 : Tâche D1 (choix de modules)

2-2-2 Catégories d'utilisateurs

Pour optimiser le travail de modélisation/simulation/analyse des résultats, il est intéressant de distinguer différentes catégories d'utilisateurs, les membres de chacune de ces catégories étant des spécialistes d'un certain type de travail. On retrouve ce type d'organisation aussi bien dans les bureaux d'études, que dans les centres de recherche. L'intérêt des catégories d'utilisateur est qu'ainsi il est possible d'adapter l'interface au type d'utilisateur, afin de lui en faciliter l'usage ; concrètement, il s'agit de mettre à sa disposition les outils nécessaires au type de travail qu'il aura à accomplir et de "cacher"

ceux dont il n'a pas l'usage. L'objectif recherché est de rendre l'interface plus souple, d'apparence plus simple et donc plus compréhensible.

Nous avons répertorié trois préoccupations majeures, pour les utilisateurs :

- la création des bibliothèques de modèles.

  Il s'agit d'un travail de haut niveau, qui demande des compétences importantes en physique (thermique, acoustique, aéraulique...), et nécessite une capacité de programmation (fortran, C...). Ce type de travail est réalisé par un utilisateur que nous appellerons "Ingénieurs Concepteur". Il s'agit en général du responsable de l'équipe d'études dont le rôle est, outre la préparation des "briques" de base, de gérer le travail d'équipe ; c'est lui qui décidera de la répartition des tâches au sein de l'équipe, et donc attribuera à chacun de ses collaborateurs une "clé d'entrée". Ce type d'activité (la création de bibliothèques de modèles) n'est pas l'activité principale des ingénieurs en bureau d'études, il s'agit plutôt d'activité relevant des chercheurs et universitaires. Il n'est donc pas certain qu'il soit nécessaire de mettre à la disposition des bureaux d'études cette catégorie d'utilisateurs.

- la création de projets.

  C'est une activité essentielle pour les bureaux d'études qui vient en amont de l'activité "analyse des systèmes". L'activité "création de projets" consiste à analyser un projet sous un certain point de vue (par exemple analyse thermique d'un projet de bâtiment en vue de définir ses équipements). Pour ce faire, le "créateur de projets" doit conceptualiser le projet réel sous la forme d'un assemblage de modèles élémentaires ; il dispose pour cela des modèles élémentaires stockés dans les bibliothèques auxquelles il a accès. D'autre part, le "créateur de projets" (le "créateur de projet sera appelé dans la suite de ce document "Ingénieur Assembleur") spécifie le type d'analyse qu'il attend de l'utilisateur final. Ainsi, des outils qui interdisent l'effacement intempestif de modèles ou de variables, et qui laissent libres les variables pour lesquelles l'analyse paramétrique doit être menée, doivent exister au sein de l'interface.

- l'analyse de systèmes.

  Pour définir le système le mieux adapté au problème à traiter, l'ingénieur chargé d'étude travaille par analyse de sensibilité. Il s'agit en théorie de réaliser une optimisation, c'est à dire de définir pour n variables d'entrée l'ensemble des valeurs numériques qui permet de répondre au mieux à l'objectif fixé. Ces études paramétriques sont confiées à ce que nous appellerons un "Ingénieur Analyste". Pour réaliser l'étude paramétrique, l'ingénieur analyste travaille sur des assemblages (des projets) déjà définis (par l'utilisateur "Ingénieur Assembleur"). Ainsi, pour un analyste seuls quelques outils sont nécessaires ; il est inutile de mettre à sa disposition des outils pour la création d'assemblage, ou des outils pour la création de macros ou de projets.

2-2-3 Habitudes de travail liées à l'environnement informatique

Les habitudes de travail liées à l'environnement informatique sont basées sur le fenêtrage et sur la manipulation directe des objets graphiques. Les objets graphiques les plus connus sont les boutons poussoirs, les icônes, les éditeurs de texte, les éditeurs d'arbres, les défileurs.

Une analyse des techniques ergonomiques [MEKOUAR 91] et sur laquelle nous nous sommes basés nous a amené à retenir les principes suivants pour la conception de l'interface ; ils concernent :

- **l'homogénéité** ; il faut faire en sorte que les mêmes actions produisent les mêmes effets ;
- **la concision** ; il s'agit d'éviter à l'utilisateur de mémoriser des informations longues et nombreuses ;
- **le feed-back** ; il doit être aussi immédiat que possible. L'utilisateur doit être informé rapidement et de façon adéquate sur l'incidence de ses actions ;
- **la charge informationnelle** ; il convient de minimiser le nombre d'opérations à effectuer pour l'utilisateur ainsi que le temps de traitement ;
- **la flexibilité** ; le logiciel doit pouvoir concerner diverses populations, de culture scientifique différente et qui n'ont pas le même niveau de compétence ;
- **la gestion des erreurs** ; il s'agit de prévoir les erreurs, de donner à l'utilisateur la possibilité de détecter et de corriger ses erreurs, ou de permettre au système de reprendre en main les opérations en cas d'erreur grave de manipulation ;
- **le multifenêtrage** ; les fenêtres fournissent un modèle intuitif pour gérer plusieurs activités en parallèle, chaque activité ayant lieu dans une fenêtre ;
- **le contrôle explicite** ; l'interface doit apparaître comme étant sous le contrôle de l'utilisateur ;
- **la correspondance entre les objets manipulés et les objets réels** ;
- **l'intégration de l'aide au sein de l'application interactive.**

2-3 Modèle Conceptuel

B. Meyer dans son livre [MEYER 90] pose la question suivante : *le concepteur d'un logiciel doit faire face à un choix fondamental ; doit-il développer son produit sur les actions (fonctions ou processus), sur les données (objets), ou doit-il opérer une approche combinant données et actions ?*

Nous allons dans ce qui suit comparer les deux approches qui guident le développement d'un produit logiciel : l'approche qui est fondée sur les fonctions et l'approche qui est fondée sur les données. Le choix d'une approche doit être guidée par le souci d'assurer la continuité pour le projet IISIBât, cela implique de réaliser une architecture informatique qui sait s'adapter aux changements de spécifications.

Comparaison de l'approche fonctionnelle descendante et de l'approche guidée par les données (approche fonctionnelle ascendante)

La méthode fonctionnelle descendante s'appuie sur l'idée que le logiciel s'obtient par une décomposition pas à pas de la fonction abstraite du système (fonction principale). Chaque pas va diminuer le niveau d'abstraction des éléments manipulés, jusqu'à obtenir des éléments de niveau d'abstraction suffisamment bas pour permettre l'implémentation dans le langage de programmation disponible. Ce processus descendant peut être représenté comme le développement d'un arbre. Les nœuds de l'arbre représentent les actions issues de la décomposition, les branches de l'arbre représentent, quant à elles, des structures de contrôle.

La figure 2-4 donne un exemple d'approche descendante ; pour réaliser la fonction principale "Simuler", deux actions sont nécessaires : réaliser un diagramme de flux et introduire les cartes de contrôle. Pour réaliser le diagramme de flux, il faut :

- choisir des modules ;
- réaliser des liaisons ;
- introduire les valeurs des paramètres des modules.

Pour réaliser certaines actions, il s'agit d'exécuter des applications spécialisées. Le lancement de telles applications est soit dirigé par l'utilisateur (introduction des valeurs des paramètres), ou automatiquement déclenché par le système en fonction des actions de l'utilisateur (l'application qui permet de spécifier les cartes de contrôle apparaît à l'écran lorsque l'utilisateur exécute TRNSYS).

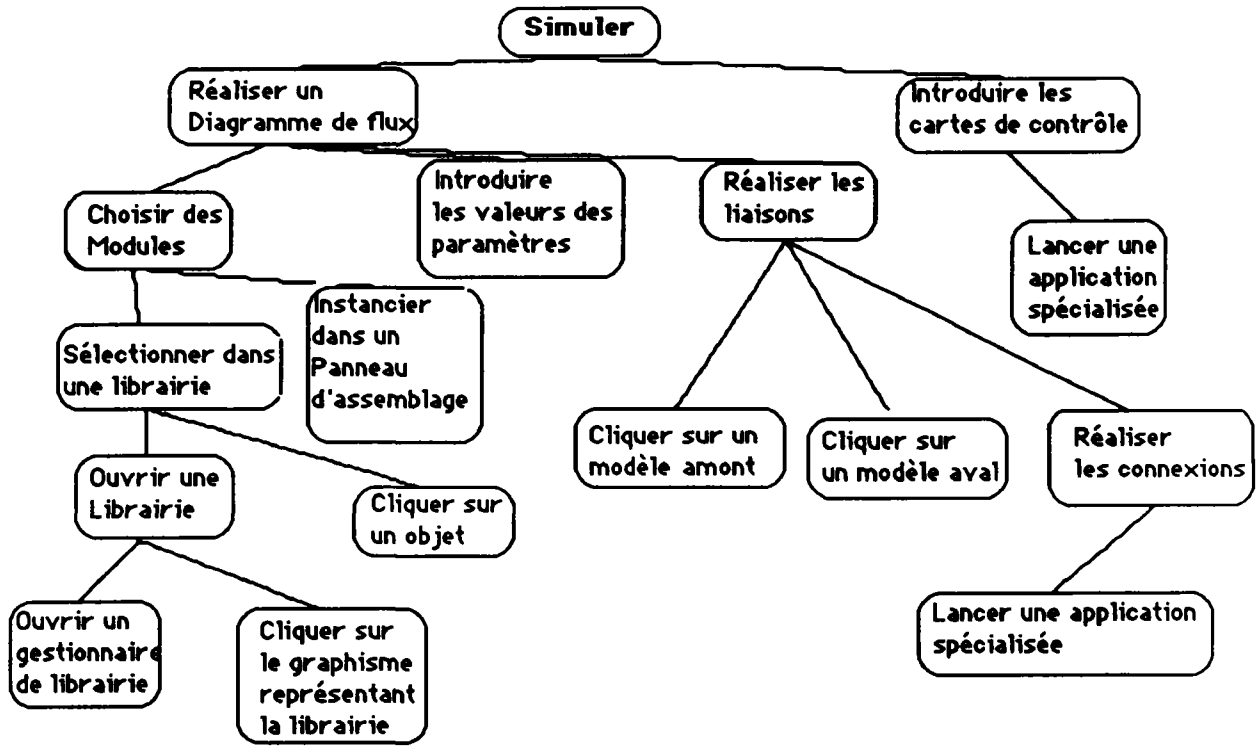


Figure 2-4 : Application de l'approche descendante

L'approche fonctionnelle descendante présente plusieurs inconvénients :

- l'accent dans cette approche est mis sur l'interface externe, la question clé est : que doit faire le système ? Or dans l'évolution d'un système, les fonctions représentent en fait la partie la plus volatile. Celles-ci se caractérisent par leur nature évolutive, les nouvelles fonctions sont soit dérivées des anciennes, soit peuvent changer radicalement. Un outil qui à l'origine était destiné à interfacer un code de calcul devra, peut-être, interfacer un autre type de logiciel, et donc résoudre de nouvelles tâches (simulation graphique, animation...).

- Cette approche introduit le risque de négliger la structure de données de l'environnement de simulation ;

- on introduit aussi le risque d'obtenir une interface pauvre en fonctions annexes. Définir un logiciel à partir d'une ou plusieurs fonctions principales (partir par

exemple de la fonction abstraite "Simuler") n'est pas toujours la bonne méthode. Dans le cas d'un outil de simulation, il ne s'agit pas de le voir comme une simple fonction qui génère des résultats numériques, mais il faut prendre en compte toutes les fonctions annexes (stockage des objets, gestion des comptes, fonctions faisant appel à des bases de connaissances), qui sont susceptibles d'évoluer.

L'approche par les données pose le problème différemment : que peut faire le système avec les données ?

Elle a le mérite de fonder l'architecture d'un système sur sa substance, et non sur sa forme externe. Les données sont en effet moins changeantes : un outil de modélisation/simulation possède toujours comme données de base des modèles.

L'architecture par les données se base sur ce principe : elle consiste à constituer des structures de données suffisamment stables, dans la mesure où elles sont vues avec un niveau d'abstraction suffisamment élevé, et d'adopter une conception fonctionnelle ascendante (partir des données). Elle permet aussi de séparer totalement l'interface du reste du système.

Dans le cadre d'IISIBât, la façon de procéder qui a été retenue est la suivante :

- identification des objets de l'interface ;
- identification des fonctions rattachées aux objets de l'interface ;
- partir de cette double identification pour concevoir les fonctionnalités de l'interface ;
- utilisation de la méthode fonctionnelle descendante pour de petits programmes et des algorithmes indépendants.

2-3-1 Identification des objets de l'interface

Trois types d'objets, qui permettent la préparation des données de simulation, ont été définis pour IISIBât : **les modèles, les macro-modèles et les projets.**

Les modèles sont les différents modules de l'outil de simulation. Ils sont caractérisés par un numéro de type et correspondent à un sous-programme Fortran.

La notion de macro-modèle est une notion spécifique d'IISIBât, cette notion n'existe pas au sein de TRNSYS. Les macro-modèles représentent des assemblages réalisés par l'utilisateur. Ces assemblages font intervenir des modèles et/ou des macro-modèles. Ce type d'objet trouve son utilité dans la mesure où il correspond aux besoins suivants :

- il permet la réalisation de morceaux d'assemblages, qui reviennent souvent dans des systèmes complexes, par exemple le sous-système pompe/capteur solaire/régulateur ;
- il donne la possibilité de travailler par intermittence ; l'utilisateur commence un assemblage, et pour une raison quelconque doit l'interrompre. Avant de quitter une session de travail, il stocke son assemblage sous la forme d'un macro-modèle.

Un projet représente un assemblage, en principe prêt à la simulation, qui est composé :

- de modèles et/ou de macro-modèles ;
- des cartes de contrôle nécessaires pour la réalisation d'une simulation.

Un projet est en principe un assemblage qui a donné satisfaction à l'utilisateur et qui correspond soit à ses besoins professionnels soit à ses besoins de modélisation. Il peut être utilisé de plusieurs façons :

- soit pour réaliser des simulations, en faisant varier les valeurs de paramètres sensibles ;
- soit pour réaliser un nouvel assemblage, à partir d'un assemblage considéré comme sûr.

Pour compléter les objets de base (modèle, macro-modèles et projets), il faut définir trois autres types d'objets, qui obéissent aux objectifs suivants :

- l'archivage des objets de base ;
- la communication entre les utilisateurs d'IISIBât (échanges d'objets) ;
- l'assemblage des objets de base.

Le premier type d'objet est appelé **gestionnaire de bibliothèques**, il contient les **bibliothèques** où sont définis les modèles, macro-modèles et projets manipulables par un utilisateur.

Le deuxième type d'objet est appelé **gestionnaire des comptes** : il contient, pour tout utilisateur, ses caractéristiques (nom, mot de passe, répertoire de travail), ainsi que les noms des autres utilisateurs dont il peut lire les gestionnaires de bibliothèques et dont il peut copier le contenu. Le gestionnaire de comptes n'est accessible qu'aux utilisateurs de type Ingénieur Concepteur (cf. § 2-2-2).

Le troisième type d'objet est appelé **panneau d'assemblage** ; un panneau d'assemblage est vide à l'état initial. Il correspond en fait à la feuille de papier où un concepteur pose et résout son problème.

2-3-1-1 Identification des opérations associées aux objets

Les gestionnaires de comptes doivent permettre pour un utilisateur de type Ingénieur Concepteur :

- d'ajouter un nouvel utilisateur au sein de l'environnement de simulation ;
- de spécifier les utilisateurs avec lesquels il veut communiquer ;
- de modifier les caractéristiques d'un compte (nom, mot de passe, catégorie).

Les gestionnaires de bibliothèques doivent permettre pour tout utilisateur :

- de fabriquer de nouvelles bibliothèques ;
- de dupliquer des bibliothèques ;
- d'échanger des bibliothèques avec les utilisateurs associés.

Dans le cadre d'IISIBât, les bibliothèques, qui contiennent les objets de base (modèles, macro-modèles, projets), reproduisent en fait un graphe d'héritage. Elles sont constituées de classes et de sous-classes qui sont de deux types :

- les classes intermédiaires, qui représentent des objets ou phénomènes physiques décrits de façon générale ;
- les classes finales qui représentent des objets directement utilisables par le code de calcul (modèles, macro-modèles et projets).

Ces deux types de classes, afin éviter toute confusion pour l'utilisateur, doivent être représentés à l'écran de façon différente. Les classes intermédiaires apparaissent sous la forme de chaînes de caractères, les classes finales apparaissent sous la forme d'icônes (cf. § 2-4-3).

L'intérêt de construire des classes réside, d'une part dans le fait que les bibliothèques obtenues sont mieux structurées et permettent donc à l'utilisateur d'accéder rapidement aux modèles concernant son problème, et d'autre part dans le fait qu'il est possible de bénéficier des propriétés d'héritage. On peut définir deux approches permettant d'obtenir des bibliothèques structurées :

- une approche non orientée objet (où la notion d'héritage n'existe pas) ; dans cette approche, le scénario le plus fréquent pour la construction rapide d'une bibliothèque est le suivant. Le développeur de modèles (DM) a pour objectif par exemple l'insertion de deux modèles M2.0 et M2.1, qui ont la particularité de se ressembler, au sein d'une bibliothèque. Pour y arriver, il doit d'abord créer le modèle M2.0, il crée ensuite un modèle M2.1 duplicata du modèle M2.0, puis adapte le modèle M2.1 ; afin de mieux structurer sa bibliothèque, le DM peut créer un nœud M2, sous lequel sont rattachés les modèles M2.0 et M2.1. L'inconvénient dans cette approche réside dans la modification d'objets existants : si l'utilisateur veut modifier une propriété commune à tous les objets rattachés au nœud M2, il devra réaliser autant d'opérations "Modification d'objet" qu'il y a d'objets rattachés au nœud M2.

- Une approche orientée objet ; dans cette approche, l'utilisateur se pose la question suivante : quelles sont les propriétés (attributs et méthodes) communes à tous les modèles d'un même groupe que je peux définir au niveau d'une classe ? La réponse à cette question doit conduire le DM à concevoir la classe M2. Ensuite, le DM crée les modèles M2.0 et M2.1 et bénéficie de tout ce qui a été préalablement défini pour la classe M2, à partir du moment où le DM rattache M2.0 et M2.1 à M2. Cette approche demande un effort de synthèse de la part du développeur de modèles, si celui-ci veut utiliser de façon optimale les propriétés d'héritage.

Nous pensons, quant à nous, que pouvoir bénéficier des propriétés d'héritage dans les bibliothèques est un avantage pour le développeur de modèles ; cela va notamment permettre d'assurer l'extensibilité du logiciel, en autorisant des modifications rapides et sûres des objets stockés. De plus, l'héritage permet une manipulation "globale" d'objets de même type : par exemple, il est possible d'exprimer une règle experte pour toutes les pompes, sans avoir à l'exprimer autant de fois qu'il existe d'objets de type "pompe".

Les bibliothèques devront donc posséder des outils qui permettent la construction rapide du graphe d'héritage ; elles doivent notamment permettre :

- la constitution de nouvelles classes intermédiaires et finales ; cela sous-entend que la nouvelle classe introduite dans le graphe d'héritage doit pouvoir hériter des méthodes et attributs de sa classe mère ; cela sous-entend aussi que l'utilisateur doit pouvoir spécifier de nouveaux attributs pour la nouvelle classe introduite ;
- la modification de classes existantes ;
- la fabrication et la modification des liens d'héritage ;
- la duplication et le collage d'une partie du graphe d'héritage.

Les méthodes associées aux objets de base (modèles, macro-modèles et projets) sont les suivantes :

- l'instanciation ;
- l'accès aux attributs ;
- la modification des valeurs des attributs ;
- l'information ; l'utilisateur doit pouvoir s'informer à tout moment sur le modèle, macro-modèle ou projet qu'il utilise ;
- l'assemblage avec d'autres objets de base ;
- l'effacement ;
- le déplacement ;
- la visualisation des macro-modèles et des projets sous leur forme éclatée.

2-3-1-2 Implémentation informatique

La représentation informatique des objets (modèles, macro-modèles, projets, gestionnaire de bibliothèques, gestionnaire de comptes) de l'interface est totalement codifiée de façon homogène : toutes les données sont représentées par des arbres binaires (appelés listes en LISP). L'intérêt de la liste en tant que structure de données, par rapport par exemple à la structure d'une table, est principalement sa souplesse. Celle-ci vient directement de son organisation en tant que chaîne de pointeurs, directement accessibles et modifiables. Le maillon élémentaire de la structure sous forme de listes est un couple de pointeurs :

- l'un accède à l'élément de la liste associé à ce maillon ;
- l'autre accède au reste de la liste.

L'insertion ou la suppression d'un élément quelconque dans une liste se fait en une seule opération, tandis que l'insertion ou la suppression dans une table nécessite que celle-ci soit recopiée entièrement. Par contre, pour accéder au nième élément d'une liste, il faut n opérations, alors qu'une seule suffit dans une table.

Tous les types d'objets sont représentés dans des fichiers ASCII, ce qui donne à IISIBât la possibilité d'utiliser le système de protection des fichiers du système d'exploitation UNIX (l'une des exigences que doit respecter un logiciel est l'aptitude à protéger ses composantes contre des modifications non autorisées : exigence d'intégrité cf. § 1-3).

Tout utilisateur possède, associé à son identification, un numéro de compte et appartient à un groupe particulier. Ainsi pour un fichier donné, les utilisateurs du système sont classés en trois catégories :

- le propriétaire du fichier ;
- les membres d'un groupe particulier ;
- les autres utilisateurs.

Il existe par ailleurs un utilisateur privilégié disposant de tous les droits (super utilisateur).

Trois opérations élémentaires sont contrôlées par le système : la lecture, l'écriture, l'exécution. Pour un fichier ordinaire, le droit de lecture permet de lire le contenu d'un fichier, le droit d'écriture permet de modifier le fichier.

Associé à ce système de protection, IISIBât propose un système de protection complémentaire basé sur la reconnaissance de l'utilisateur (nom et mot de passe).

Remarque : le code ASCII assure une totale lisibilité, mais n'assure aucune garantie sur la confidentialité, dans le cas où le système de protection du système d'exploitation ne suffit pas.

La base de données, qui constitue le coeur d'IISIBât, est donc composée des fichiers représentant tous les types d'objets (modèles, macro-modèles, projets, gestionnaire de bibliothèques, gestionnaire de comptes). Ces fichiers peuvent être de deux types :

- les fichiers *statiques*, qui renferment des connaissances considérées constantes dans le temps. Cette notion d'invariance dans le temps n'est pas absolue : un modèle évolue sous l'action des mises à jour. On parle plutôt de connaissances figées pendant "un certain temps", ce temps étant au moins égal à une session de travail ;
- les fichiers *dynamiques*, qui renferment des connaissances qui évoluent pendant une session de travail.

2-3-1-3 Organisation hiérarchique

Dans le cadre d'IISIBât, l'organisation hiérarchique des fichiers (figure 2-5) est la suivante :

- désignation d'un "répertoire racine" pour IISIBât ; cela permet l'installation d'IISIBât à n'importe quel niveau de l'arborescence d'un système ;
- création pour chaque utilisateur d'un catalogue qui lui est propre (qui porte son nom), appelé "catalogue utilisateur", et qui contient trois sous-répertoires spécifiques : **modèles**, **macros** et **projets** ; le répertoire **modèles** contient tous les objets de type modèle, le répertoire **macros** contient tous les objets de type macros, le répertoire de type **projets** contient tous les objets de type projets (notons qu'il existe autant de "catalogues utilisateurs" qu'il existe d'utilisateurs d'IISIBât) ;
- création pour chaque utilisateur de deux fichiers **biblios.ll** et **users.ll**, qui sont stockés au niveau du "catalogue utilisateur", et qui représentent respectivement les objets de type gestionnaire de bibliothèques et gestionnaire de comptes, relatifs à l'utilisateur du "catalogue utilisateur" ;
- création d'un répertoire de nom IISIBât qui contient les objets et procédures communs à tous les utilisateurs (icônes graphiques, procédures d'installation), et qui est obligatoirement un sous-catalogue du "répertoire racine".

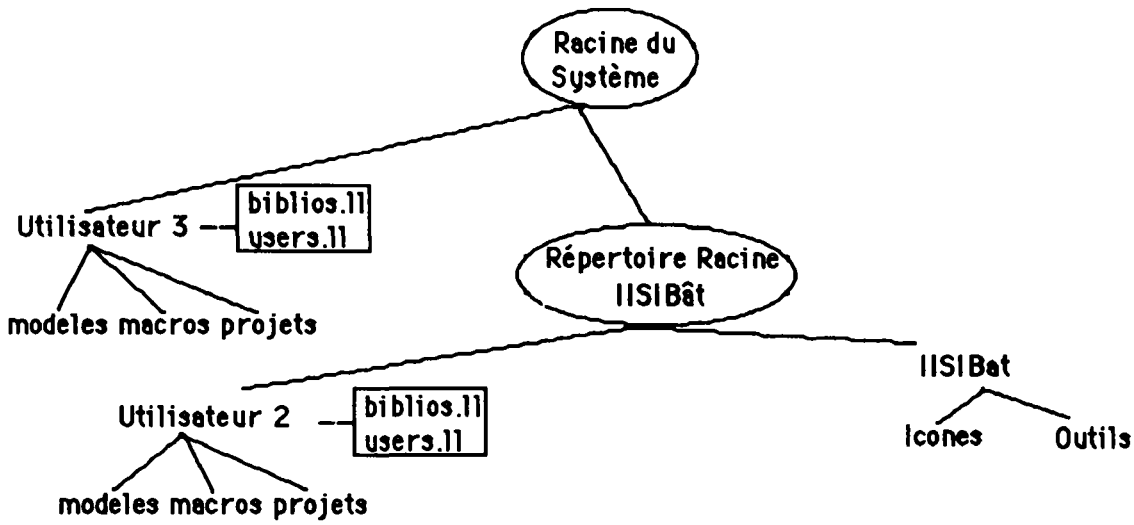


Figure 2-5 : Organisation hiérarchique de la structure de données

2-3-2 Représentation graphique des objets de la simulation : le modèle iconique

Les objets qui constituent un système physique peuvent se définir comme "un ensemble d'élèments en interaction dynamique en fonction d'un but". La représentation de tels objets peut être vue comme une représentation physique ou mentale d'un système réel, et s'exprime en général sous forme graphique ou mathématique. Le problème qui se pose pour nous est la représentation des objets, ou des phénomènes physiques au sein de l'interface.

Dans ce cadre, il s'agit de définir un modèle d'interaction entre l'utilisateur et la machine. Les modèles d'interaction les plus adaptés aux applications interactives sont fondés sur la manipulation directe. Il s'agit de représenter les objets de l'interaction, et d'utiliser des actions physiques sur ces objets (généralement à l'aide de la souris). Les effets de ces actions doivent être visibles et facilement réversibles. Ces modèles doivent aussi permettre d'interrompre une tâche pour en effectuer une autre, de mener des tâches en parallèle, ou d'effectuer une sous-tâche. Le succès d'un système à manipulation directe est fortement lié à l'analogie avec le monde réel.

Le modèle d'interaction directe le plus connu est le modèle iconique, popularisé par le Macintosh (principe WYSWYG : What You See What You Get). Il se base sur la représentation des objets, tels que l'on peut les trouver sur un bureau, par des images de taille réduite appelées icônes. L'utilisateur peut manipuler ces icônes par quelques actions simples (sélectionner, effacer, déplacer, activer une commande sur les icônes sélectionnées) depuis un menu et/ou depuis le clavier.

Un inconvénient des modèles iconiques est la combinatoire des opérations possibles qui augmente avec le nombre de types d'icônes disponibles, chaque type n'autorisant pas les mêmes opérations. Ce problème peut être contourné en rendant le fonctionnement intuitif grâce à l'aspect graphique des icônes (couleur, forme, indicateurs particuliers). Le modèle iconique ne devient performant que si les icônes correspondent à des objets concrets, ou reconnus de l'utilisateur.

Les icônes ne suffisent pas en général à interfacer l'ensemble des fonctionnalités de l'interface. Cependant, le modèle iconique peut s'adapter à de nombreux environnements. Les fonctions classiques de l'icône sont la représentation des documents, des dossiers pour classer des documents, des applications et des volumes disques. Il nous est aussi apparu que le modèle iconique est bien adapté à la préparation des données de simulation (fichier DECK pour le code de calcul TRNSYS), notamment grâce à la manipulation d'objets de deux types :

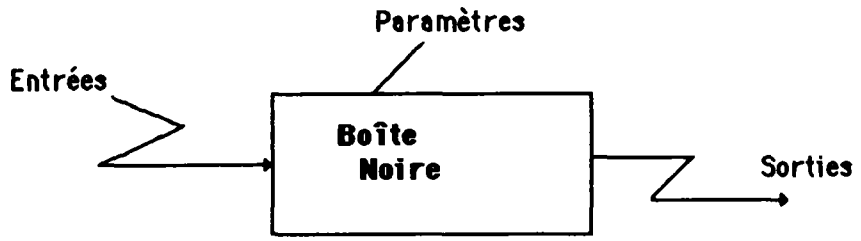
- les icônes ;
- les liens entre les icônes, qui définissent les interactions entre des couples d'objets ; dans le cadre d'IISIBât, cet objet sera appelé "liaison".

Les objets qui nous intéressent (bâtiment, zones, parois, équipements) sont plutôt vus par les utilisateurs des codes de simulation pour les "services" qu'ils peuvent rendre. L'utilisateur ne s'intéresse aux objets que pour constituer un réseau, le plus judicieux possible, à partir de processeurs élémentaires dont il connaît le comportement. En général, l'utilisateur n'est pas le développeur du modèle et n'a pas les compétences nécessaires pour comprendre la structure interne du modèle : cet objet lui apparaît donc comme une boîte noire. La représentation des objets sous forme de boîtes noires,

familière dans la pratique, devient alors commode à utiliser : pour un utilisateur d'un code de simulation, un objet réel est en fait une boîte qui a un certain comportement.

Remarque : la structure interne des objets n'intéresse que les utilisateurs qui créent de nouveaux objets à partir d'objets existants ("Ingénieurs Concepteurs" cf. § 2-2-2).

Les utilisateurs ont toujours un point de vue sur les objets , ils les considèrent à travers un filtre qui correspond à leurs préoccupations. Par exemple, l'objet "fenêtre" sera considéré par un thermicien comme une simple résistance thermique, bien plus faible que celle d'un mur. Si le thermicien a une certaine habitude des caméras infra-rouge, il pourra aussi voir l'objet "fenêtre" comme un objet déperditif rouge, par comparaison au mur froid et donc bleu. Le même objet sera vu par un spécialiste de sécurité incendie comme une masse consumable possible, qui présente dans certaines conditions données, une certaine durée de résistance au feu.



2-6 : Représentation d'un objet sous forme d'une boîte noire

La représentation des objets sous la forme de boîtes noires n'est en fait qu'un macro-langage qui permet de dialoguer, de comprendre ce que veut exprimer l'utilisateur, et de le traduire en terme de fichiers d'entrées (fichier DECK pour le code de calcul TRNSYS), et ceci quelque soit le code choisi.

2-3-2-1 Modèle iconique et outils de simulation

Pour illustrer nos propos, nous nous sommes intéressés à quatre codes de calcul :

- TRNSYS (Transient System Simulation Program) développé à l'université de Madison (Wisconsin) [TRNSYS 90] ;
- COMIS (Conjunction of Multizone Infiltration Specialists) [COMIS 91] ;
- SPARK (Simulation Problem Analysis Kernel) développé à l'université de Berkeley [NATAF-WINKELMANN 91] ;
- CSTBât développé au CSTB (Centre Scientifique et Technique du Bâtiment) [LARET 1-89].

Dans le cadre de TRNSYS, la représentation des objets sous forme de boîtes noires est toute naturelle. Ces boîtes possèdent des pattes qui représentent les entrées, les sorties et les paramètres. Deux boîtes communiquent entre elles par le biais des connexions entre les variables d'entrée/sorties. L'objet[9] de type "DEPERDITIONS" modélisé par la formule D = GV (Ti - Te) est représenté de la façon suivante :

9 Il ne s'agit pas ici d'objets au sens Programmation Orientée Objets, mais de composants appartenant à une bibliothèque

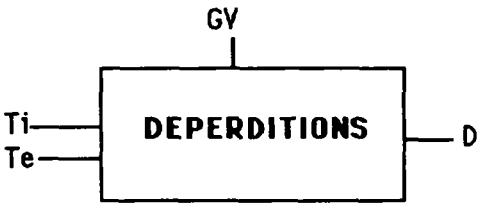


Figure 2-7 : Objet de type "DEPERDITIONS"

COMIS est un code de calcul, dont les objectifs principaux à l'heure actuelle sont :

- calcul des distributions de pression dans un bâtiment ;
- dimensionnement des composants de ventilation dans un projet ;
- détermination de la distribution des polluants ;
- estimation des déperditions de chaleur dues à la ventilation et aux infiltrations d'air ;
- estimation de l'efficacité du système de ventilation.

La conception informatique de ce code n'est pas modulaire : il existe certes des sous-programmes correspondants aux composants physiques modélisés (par exemple, tel sous-programme modélise un ventilateur, tel autre un système de contrôle des débits, etc), mais la structure informatique même du code fait qu'il est très difficile de le modifier, d'ajouter ou de supprimer des sous-programmes ; cela est dû à l'option initiale choisie par les développeurs : elle consiste à déclarer des liaisons entre sous-programmes au sein même du code ; de ce fait, un module "COMIS" ne possède ni entrées, ni sorties connectables ; il possède par contre des paramètres, et des variables internes qui peuvent être déclarées comme sorties pour l'affichage des résultats. C'est là une limitation du logiciel, il est très difficile de personnaliser la bibliothèque des modèles "COMIS" par adjonction de nouveaux modèles sauf à parfaitement connaître la structure informatique interne du code Fortran. Cependant, du point de vue du développeur de projets ou de l'analyste, une telle structure offre des avantages au sens où, lorsqu'on réalise un assemblage, il n'est nul besoin de spécifier les connexions internes aux liaisons entre composants. Rappelons qu'avec TRNSYS, un de nos objectifs futurs est de développer une fonction qui permette la connexion automatique entre variables pour précisément affranchir l'utilisateur de cette tâche de définition des connexions entre entrées et sorties. Une telle fonction, surtout si on la veut générique, est difficile à développer, mais une fois réalisée, les performances de l'interface avec TRNSYS deviennent égales à ce qu'elles sont avec COMIS (puisqu'alors, dans un cas comme dans l'autre, il suffit de positionner les composants et de déclarer les liaisons entre composants, les connexions internes étant, dans le cas de TRNSYS, réalisées par la fonction "connexion automatique" et, dans le cas de COMIS, les connexions étant déclarées de façon interne au programme), mais, avec la structure de TRNSYS, on conserve la modularité nécessaire à l'extension aisée des bibliothèques de modèles. Cependant, il faut noter qu'avec TRNSYS et la fonction "connexions automatiques", un certain travail peut être nécessaire lors de l'implémentation d'un nouveau composant pour enrichir la base de connaissances qui permet à la fonction "connexion automatique" d'effectuer les liaisons adéquates (cf. § 3-2-3).

Quoi qu'il en soit, pour l'utilisateur terminal, c'est à dire pour le créateur de projets ou pour l'analyste, l'incorporation de COMIS au sein d'un Environnement de Simulation Intelligent comme IISIBât, lui donne l'illusion de la modularité et de l'aspect "orienté objets" du code, en ce sens qu'il manipule à l'écran des icônes représentant des objets

physiques indépendants ; mais il ne s'agit là que d'une apparence puisque derrière, les sous-programmes Fortran sur lesquels pointent ces "objets" sont étroitement dépendants les uns des autres.

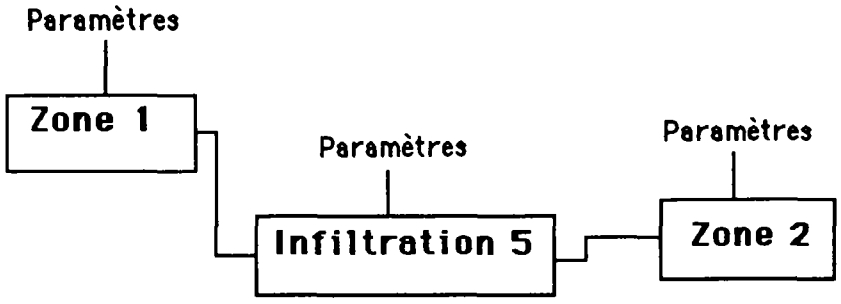


Figure 2-8: Boîte noire COMIS

SPARK est un solveur numérique associé à une bibliothèque de modèles. Un objet SPARK est représenté par un ensemble d'équations, qui font intervenir toutes les variables utilisées : ainsi toute variable peut être soit une variable de sortie, soit une variable d'entrée, selon le cas.

Soit l'équation suivante D - GV (Ti - Te) = 0. L'objet SPARK qui en découle est caractérisé par les quatres équations suivantes :

$$\begin{cases} D = GV(Ti-Te) \\ GV=\dfrac{D}{(Ti-Te)} \\ Ti=\dfrac{D}{GV} + Te \\ Te = Ti - \dfrac{D}{GV} \end{cases}$$

Cet objet SPARK peut être représenté par une boîte noire (figure 2-9), possédant quatre pattes, chacune d'entre elles représente une variable pouvant être calculée en fonction des trois autres. Les boîtes communiquent par le biais de connexions entre les variables. Il n'y a pas , a priori, de variables d'entrée/sortie ; c'est le contexte de la simulation (exprimé en termes de données disponibles et d'objectifs) qui fournira les paramètres, les entrées et les sorties.

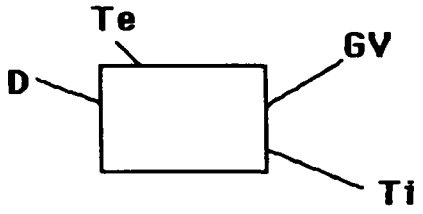


Figure 2-9 : Boîte noire SPARK

Remarque : l'objet TRNSYS, modélisé par la formule D = GV (Ti - Te) est un objet SPARK spécifique, orienté conception. L'objet TRNSYS, modélisé par la formule GV = D / (Ti - Te) est un objet SPARK spécifique, orienté audit (figure 2-10).

Figure 2-10 : Boîte noire SPARK orientée AUDIT

CSTBât utilise la structure informatique du logiciel de simulation TRNSYS. Il s'articule autour de la modélisation analogique du bâtiment. Là aussi, tous les objets ou phénomènes physiques peuvent être représentés par des boîtes noires. Les boîtes noires rapportées au modèle de bâtiment (type 50 [LARET 1-89]) ne sont pas sur le même plan que les précédentes ; elles représentent en fait un réseau de conductances et de capacités.

L'utilisateur peut vouloir affiner sa modélisation (ajouter des ponts thermiques par exemple) ; il devra alors manipuler des objets tels que des capacités, des conductances, des générateurs de courant (cf. § 1-5-1-2). De plus, une interface experte devra proposer des aides pour la discrétisation de l'espace modélisé et pour le calcul des valeurs des conductances.

2-3-2-2 Définition d'un objet "icône"

La représentation des objets sous forme de boîtes noires reliées entre elles peut être généralisée à un grand nombre de codes de calcul. Elle donne à l'utilisateur la possibilité de structurer sa réflexion à propos d'une modélisation d'un système physique, tout en lui assurant une liberté d'action, du fait de l'aspect modulaire de l'approche.

Les objets graphiques (boîtes noires -icônes- et liaisons) manipulées au sein de l'interface n'ont pas la même signification selon le code de simulation choisi (pour TRNSYS, l'icône est orientée, elle pointe sur des sous-programmes indépendants ; pour COMIS, l'icône ne possède ni entrées ni sorties, elle pointe sur des sous-programmes dépendants ; pour SPARK, l'icône, au niveau le plus générique de SPARK, ne possède que des variables, elle pointe sur des "objets SPARK" indépendants). L'idée est de trouver un moyen pour représenter ce qu'est une icône de la façon la plus générale possible, de telle sorte qu'il soit facile d'adapter cette forme aux spécificités de différents codes de simulation. Pour ce faire, l'approche orientée objets présente tous les avantages nécessaires : elle permet de créer un objet générique qui par spécification peut facilement permettre de créer des icônes bien adaptées à la manipulation des composants technologiques modélisés par le code considéré. Les icônes ainsi créées, du fait de l'utilisation d'une technique de programmation orientée objets, hériteront de toutes les caractéristiques et méthodes attachées à l'objet générique "icône" ; ainsi, lors de l'incorporation dans l'environnement de simulation intelligent, d'un nouveau code de calcul, il ne sera pas nécessaire de redéfinir une icône par une fonction nouvelle et spécifique, mais il suffira de spécifier l'icône générique existante, au niveau abstrait, au sein de cet environnement ; ainsi le travail du développeur de l'environnement de simulation s'en trouvera grandement facilité. Par ailleurs, cette technique assure une certaine cohérence d'aspect et de comportement entre les différentes représentations des objets modélisés par les différents codes de calcul, puisque ces représentations dérivent d'un objet unique ; cette cohérence d'aspect facilite la prise en main des logiciels par les utilisateurs terminaux.

Nous nous sommes donc attachés à définir un modèle profond de ce qu'est une icône (ou boîte noire) au sein de l'environnement IISIBât.

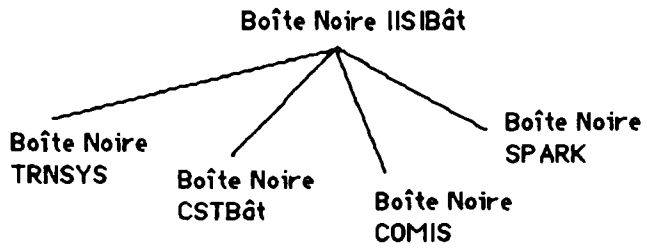


Figure 2-11 : Hiérarchie des boîtes noires

Pour assurer un fonctionnement correct de l'outil de simulation, les boîtes noires doivent posséder :

- un cadre ;
- un dessin, qui doit représenter soit un objet du monde réel, soit être un graphique habituellement utilisé dans le domaine du code de simulation ;
- des zones activables, c'est à dire des zones à partir desquelles il est possible pour l'utilisateur de lancer des applications spécialisées ;
- des liaisons ;
- des graphismes particuliers, qui se superposent au dessin de l'icône.

Par exemple, la boîte noire IISIBât (racine du graphe d'héritage) possède les caractéristiques suivantes :

- le cadre ;
- le dessin.

La boîte noire TRNSYS possède les caractéristiques spécifiques suivantes :

- les liaisons amonts ;
- les liaisons avals ;
- la zone activable paramètre ;
- la zone activable entrée ;
- la zone activable sortie ;
- la zone activable dérivative ;
- la zone activable centrale (qui va permettre le déplacement de l'icône) ;
- un graphisme particulier indiquant si l'objet est verrouillé ou non.

La boîte noire SPARK possède les caractéristiques spécifiques suivantes :

- les liaisons associées ;
- un graphisme particulier indiquant si l'objet est verrouillé ou non.

Modèle HBDS de l'objet "icône"

Le modèle HBDS (Hypergraph Based Data System) fournit un cadre formel permettant de représenter des structures de connaissances, de façon non ambiguë. Cette méthode doit permettre de transcrire de façon automatique [POYET 1-92] la représentation des structures de connaissances en une forme directement utilisable par la machine (transcription orientée objets pour ce qui nous concerne). Avant de présenter le modèle abstrait de l'objet icône, quelques informations concernant le formalisme HBDS sont nécessaires.

Le modèle HBDS utilise les notions de classes et d'hyper-classes :

- les classes sont des objets structurés ; elles sont formées d'un certain nombre de champs (ou attributs) nommés ; des méthodes peuvent être rattachées aux classes (elles se déclenchent à réception d'un message donné) ; des démons peuvent être rattachés aux champs (ils permettent en particulier de spécifier les contraintes à vérifier lors de l'instanciation de l'objet) ; à chaque attribut peut être rattaché une facette qui spécifie la liste des valeurs possibles pour l'attribut ; en formalisme HBDS, une classe est représentée par un ovale ; les attributs sont rattachés en un même point à la classe ;

- une hyper-classe est une classe possédant des sous-classes ; comme pour les classes, on peut définir pour les hyper-classes des attributs, des méthodes, des démons et des relations rattachées à l'hyper-classe ; les sous-classes héritent de tous les champs de l'hyper-classe, mais aussi des éventuelles méthodes définies au niveau de l'hyper-classe et des relations de celles-ci avec les autres classes ou hyper-classes. En formalisme HBDS, une hyper-classe est représentée par un carré aux coins arrondis.

Des liens sémantiques (relations) peuvent exister entre les classes et/ou les hyper-classes, ils définissent les interactions entre les classes ; ces relations sont représentées par des traits orientés entre objets (classe ou hyper-classe). Des démons peuvent éventuellement être rattachés aux relations.

Une instance d'une classe est représentée par un double ovale contenant le nom de l'instance et le nom de la classe à laquelle l'instance appartient.

Nous avons défini un modèle abstrait de l'objet "icône", il est est décrit dans la figure 2-12-a en formalisme HBDS ; ce modèle aurait pu être différent, nous avons effectué des choix qui sont expliqués ci-après.

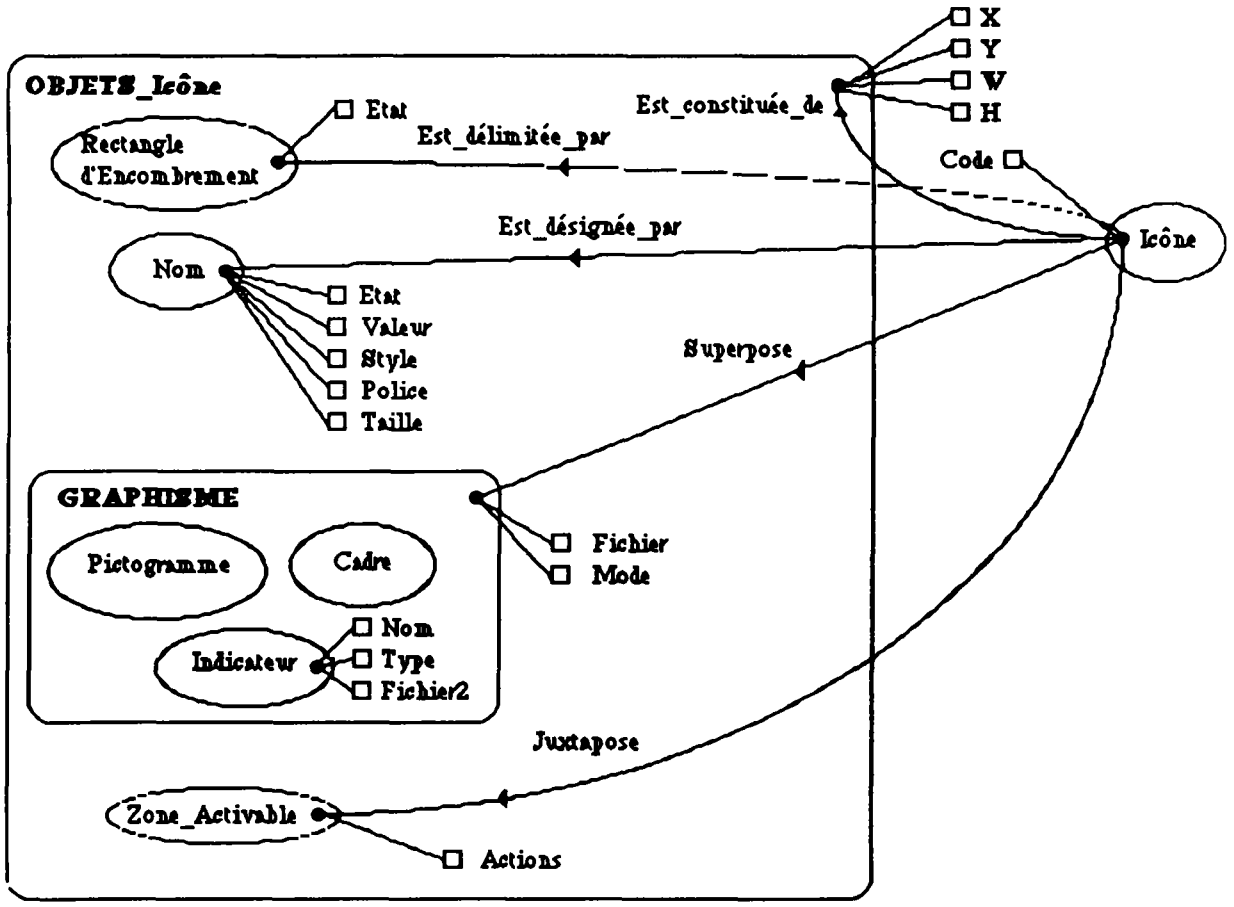


Figure 2-12-a : Modèle HBDS de l'objet "icône"

• Classe **Icône**

- champ **Code** : il prend pour valeur le nom du code de simulation (TRNSYS, COMIS, CSTBât ...) pour lequel le modèle représenté par l'icône est utilisable (rappelons qu'à chaque icône correspond un sous-programme -fortran, pascal, C, etc...- "valide" pour un et un seul code de simulation). Il est implémenté sous forme d'un symbole. Le champ **Code** est utilisé par des procédures de contrôle pour s'assurer de la cohérence des bibliothèques et des assemblages (il est illicite de connecter une icône de type "TRNSYS" avec une icône de type "COMIS", lorsqu'il sera possible de lancer plusieurs codes de calcul à partir d'une même application interactive).

•• Hyper-Classe **OBJETS_Icône**

Une icône est constituée de plusieurs éléments, sous-classes de l'hyper-classe **OBJETS_Icône** ; tous les objets qui permettent de définir une icône partagent les attributs suivants :

- champs **X** et **Y** ; il s'agit des coordonnées de l'espace occupé par l'objet en question (s'expriment en pixels) ; ces coordonnées sont soit des coordonnées absolues (nombre de pixels par rapport au coin supérieur gauche de la fenêtre

considérée), soit des coordonnées relatives ; pour l'objet **Rectangle d'Encombrement**, il s'agit de coordonnées absolues, dans la suite du document on les appellera Xréf, Yréf ; pour tous les autres objets, il s'agit de coordonnées relatives par rapport aux coordonnées Xréf et Yréf ;
- **W** et **H** ; il s'agit de la largeur et de la hauteur de l'espace occupé par l'objet en question (s'expriment en pixels) ;

Remarque : les deux dimensions (largeur, hauteur) d'une image ne doivent pas dépasser 32000 pixels. Cela induit d'une part que X doit être inférieur à 32000 pixels, et d'autre part que X+W doit être inférieur à 32000 pixels (même raisonnement pour Y et H).

• Classe **Rectangle_d'Encombrement**

Une icône est délimitée par une zone d'encombrement qui ne peut être que rectangulaire (ceci est un choix du modèle) ; la définition géométrique de la zone d'encombrement permet de gérer graphiquement le positionnement des icônes les unes par rapport aux autres, en évitant par exemple les recouvrements. La zone d'encombrement délimite le dessin du composant, mais également (éventuellement) le nom, si celui-ci est prévu sous le dessin. La limite du rectangle d'encombrement n'est pas nécessairement visible. La classe **Rectangle_d'Encombrement** possède l'attribut spécifique suivant :

- **Etat** ; Etat prend sa valeur dans la liste suivante (0, trait-simple, trait-double, pointillés) ; si Etat = 0, le rectangle d'encombrement n'est pas visible, sinon il est visible ; si Etat = "pointillés", le rectangle d'encombrement apparaît en pointillés.

• Classe **Nom**

La signification d'une icône peut être précisée par un nom (ceci est utilisé dans IISIBât/TRNSYS pour différencier à l'écran les modèles élémentaires des macro-modèles) ; ce nom est nécessairement positionné sous le cadre (il s'agit d'un choix arbitraire) ; les champs sont :

- **Etat** ; aspect de la chaîne de caractères (0, standard, gras, souligné, etc...) ; si **Etat** vaut 0 (c'est à dire si le nom du modèle n'apparaît pas à l'écran), les champs suivants n'ont à être renseignés ;
- **Valeur** ; nom du modèle (chaîne alphanumérique limitée à 12 caractères) ;
- **Police** ; nom de la police de caractères ;
- **Taille** ; taille de la police de caractères ;
- **Style** ; façon dont le nom est positionné par rapport au cadre ; ce champ prend sa valeur dans la liste (centré, aligné_à_gauche, aligné_à_droite).

• Hyper-Classe **GRAPHISME**

Le graphisme est la partie essentielle de l'objet icône pour ce qui concerne le dialogue Homme/Machine dans le sens Machine/Homme (compréhension non ambiguë, aussi immédiate que possible, par l'utilisateur de l'objet réel représenté par l'icône). Le graphisme est constitué de la superposition (fonction "view" dans Aïda) d'un cadre, d'un pictogramme et de plusieurs indicateurs.

L'hyper-classe **GRAPHISME** possède les sous-classes suivantes :

- la classe **Cadre** ; cet objet permet de définir le cadre (visible) du graphisme ;

- la classe **Pictogramme** ; cet objet permet de définir le dessin représentatif de l'objet réel ;
- la classe **Indicateur** ; lors du processus de modélisation d'un projet par assemblage de composants, il est parfois nécessaire de signaler les options (souvent en relation avec le code de simulation utilisé) retenues pour la simulation ; par exemple dans TRNSYS, il est possible de "tracer" chaque composant ; il s'agit d'une option strictement liée à TRNSYS, elle permet d'obtenir des informations détaillées (par exemple, l'évolution temporelle des valeurs des entrées) sur le déroulement de la simulation pour le composant considéré ; pour signaler à l'utilisateur qu'un modèle est "tracé", on surcharge l'icône d'un indicateur particulier (en l'occurrence un "T") ; autre exemple, dans TRNSYS certains modèles sont polyvalents ; ils permettent de représenter une grande variété de composants technologiques réels d'une même famille ; le distinguo entre ces possibilités s'effectue à l'aide d'un paramètre spécial dénommé "mode" ; il est impératif de renseigner ce paramètre avant de pouvoir utiliser de tels modèles (en effet, le paramètre "mode" influe sur la liste même des paramètres, des entrées et des sorties) ; pour signaler cette particularité à l'utilisateur, on surcharge l'icône d'un indicateur spécial (en l'occurrence un "?") ; dans le cadre d'IISIBât/TRNSYS, l'introduction des données numériques qui concernent un composant (valeurs des paramètres, valeurs initiales des entrées, valeurs initiales des "dérivatives") s'effectue lorsque l'utilisateur "clique" dans le rectangle d'encombrement ; il peut s'avérer utile d'indiquer à l'utilisateur quels sont les données numériques qu'il doit manipuler (rôle du "créateur de projets" -cf. § 2-2-2-) en surchargeant l'icône d'indicateurs spéciaux (indicateurs que nous appellerons "pattes"). Ces "pattes" servent aussi à indiquer à l'utilisateur où il doit positionner sa souris pour avoir accès aux données numériques.

Les classes **Cadre**, **Pictogramme** et **Indicateur** héritent des champs définis au niveau de l'hyper-classe **GRAPHISME**. L'hyper-classe **GRAPHISME** possède les champs spécifiques suivants :

- Fichier ; nom du fichier contenant le dessin bitmap (défini pixel par pixel) du graphisme ; il est implémenté sous la forme d'une chaîne de caractères n'excédant pas 11 caractères (certaines machines n'acceptent pas des noms de fichiers possédant plus de 11 caractères).
- Mode ; il indique comment les points se combinent[10].

La classe **Indicateur** possède les attributs spécifiques suivants :

- **Nom** ; utile pour le développeur afin de connaître le type d'indicateur rattaché à l'icône (indicateur de type "verrou", de type "trace", etc...) ; il est implémenté sous la forme d'une chaîne de caractères ;

---

10 Sur un affichage monochrome, le mode de combinaison définit quelle couleur (noir ou blanc) résultera de la superposition d'une couleur d'encre sur un fond. Par, exemple, le mode XOR est défini par le tableau 2-1.

Encre	Fond	Résultat
blanc	blanc	blanc
blanc	noir	noir
noir	blanc	noir
noir	noir	blanc

Tableau 2-1 : Mode de combinaison XOR

- **Type** ; il indique le type d'indicateur et prend sa valeur dans la liste suivante ("Toujours", "ON/OFF", "Bascule") ; un indicateur de type "Toujours" est toujours présent dans le graphisme quelles que soient les manipulations de l'utilisateur ; un indicateur de type "ON/OFF" peut disparaître du graphisme associé à l'icône (c'est le cas de l'indicateur "Trace", l'annulation par l'utilisateur de la "trace" d'un module fait disparaître l'indicateur en forme de "T") ; un indicateur de type "Bascule" est toujours présent dans le graphisme, mais peut apparaître sous deux formes distinctes (par exemple, un indicateur désignant une icône verrouillée pourrait être 🔒, un indicateur désignant une icône déverrouillée pourrait être 🔓) ;
- **Fichier2** ; nom du fichier contenant le deuxième dessin Bitmap ; ce champ doit être renseigné dans le cas où l'indicateur est de type "Bascule".

Remarque

Pour construire une image, deux possibilités existent :

- soit charger une mémoire de points stockée dans un fichier ; cette approche permet de stocker des images complexes, et permet la modification des images sans avoir à redévelopper du code ; cette technique a été utilisée pour l'affichage des icônes qui concernent les boîtes à outils, et pour l'affichage des icônes dans les bibliothèques (modèles, macro-modèles et projets) ; ces icônes sont susceptibles d'évoluer (modification des modèles, des macro-modèles et des projets) ; ainsi, si l'on veut modifier le "look" des modèles, il suffit de modifier les fichiers contenant les mémoires de points (idem pour le développeur qui souhaite améliorer le "look" des outils) ; il faut noter que la présence des fichiers est obligatoire pour que le logiciel puisse fonctionner (si des fichiers sont détruits, le logiciel ne peut plus démarrer) ;

- soit exécuter une fonction qui utilise les primitives graphiques du Bitmap Virtuel et qui affiche les points de l'image pixel par pixel ; cette technique pourrait être utilisée pour compléter le dessin de l'icône ; elle concernerait alors des images simples et qui ont peu de chances d'évoluer au cours du temps (dessin des indicateurs, dessin du cadre). Exécuter des fonctions rend le logiciel indépendant vis à vis des effacements de fichiers.

Pour ce qui nous concerne, nous avons opté pour la solution qui consiste à charger des mémoires de points à partir de fichiers. Cette façon de procéder convient à l'organisation du travail : l'aspect graphique de l'interface est confié à des graphistes professionnels qui génèrent des fichiers et non pas du code.

• Classe **Zone_Activable**

Les zones actives sont les zones de l'icône dans lesquelles un événement "souris" déclenche une action (par exemple positionner une zone active dans la partie supérieure de l'icône ; un clic souris dans cette partie déclenche alors l'ouverture de la fenêtre d'introduction des valeurs des paramètres -cf. figure 2-22-). Toute action déclenchée est fonction de l'outil sélectionné dans la boîte à outils, qui est associée à la fenêtre où se trouve l'icône (ainsi, une même zone activable peut engendrer des actions différentes).

La classe **Zone_Activable** possède le champ spécifique suivant :

- **Actions** ; liste des associations "Outil, Actions" relatives à la zone ; par exemple, la liste (("Multifonctionnel" , "Déplace") ("Gomme" , "Efface")) signifie que si l'outil "Multifonctionnel" est sélectionné et si un clic souris se produit dans la zone activable, l'icône subira un déplacement, si l'outil "Gomme" est sélectionné et si un clic souris se produit dans la zone activable, l'icône sera effacée de la fenêtre de travail.

Définition des démons

Nous avons défini les démons rattachés aux relations ou aux classes ; ces démons se déclenchent lors de l'instanciation des objets constituant l'icône, ils vérifient que les valeurs affectées aux attributs sont cohérentes avec les relations (relations "Juxtapose", "Superpose", "Est_constituée_de") qui doivent exister entre les objets.

Désignons par :

- $X_{réf}$, $Y_{réf}$, $W_{réf}$, $H_{réf}$ l'espace occupé par le rectangle d'encombrement ;
- X_{cadre}, Y_{cadre}, W_{cadre}, H_{cadre} l'espace occupé par le cadre ;
- $X_{pictogramme}$, $Y_{pictogramme}$, $W_{pictogramme}$, $H_{pictogramme}$ l'espace occupé par le pictogramme ;
- X_{nom}, Y_{nom}, W_{nom}, H_{nom} l'espace occupé par le nom donné à l'icône ;
- $X_{indicateur}$, $Y_{indicateur}$, $W_{indicateur}$, $H_{indicateur}$ l'espace occupé par l'indicateur ;
- $X_{zone\_activable}$, $Y_{zone\_activable}$, $W_{zone\_activable}$, $H_{zone\_activable}$ l'espace occupé par une zone_activable ;
- $W_{caractère}$, la largeur d'un caractère dans la police courante ;
- $H_{caractère}$, la hauteur d'un caractère dans la police courante.

On obtient les démons suivantes :

$$\begin{cases} 0 \le X_{réf} + W_{réf} \le 32000 \\ 0 \le Y_{réf} + H_{réf} \le 32000 \end{cases}$$

$$\begin{cases} 0 \le X_{cadre} + W_{cadre} \le W_{réf} \\ 0 \le Y_{cadre} + H_{cadre} \le H_{réf} \end{cases}$$

$$\begin{cases} 0 \le X_{pictogramme} + W_{pictogramme} \le X_{cadre} + W_{cadre} \\ 0 \le Y_{pictogramme} + H_{pictogramme} \le Y_{cadre} + H_{cadre} \end{cases}$$

$$\begin{cases} W_{nom} = W_{caractère} * \text{nombre de caractères} \\ 0 \le X_{nom} + W_{nom} \le W_{réf} \\ Y_{cadre} + H_{cadre} + H_{caractère} + 1 \le Y_{nom} \le H_{réf} \end{cases}$$

$$\begin{cases} 0 \le X_{indicateur} + W_{indicateur} \le W_{réf} \\ 0 \le Y_{indicateur} + H_{indicateur} \le H_{réf} \end{cases}$$

$$\begin{cases} 0 \le X_{zone\_activable} + W_{zone\_activable} \le W_{réf} \\ 0 \le Y_{zone\_activable} + H_{zone\_activable} \le H_{réf} \end{cases}$$

Pour deux zones activables contiguës i et j, une de ces deux relations doit être vérifiée :

$$\begin{cases} X_{zone\_activable_i} + W_{zone\_activable_i} + 1 = X_{zone\_activable_j} \\ Y_{zone\_activable_i} + H_{zone\_activable_i} + 1 = Y_{zone\_activable_j} \end{cases}$$

2-3-2-3 Exemples d'instanciation d'icône

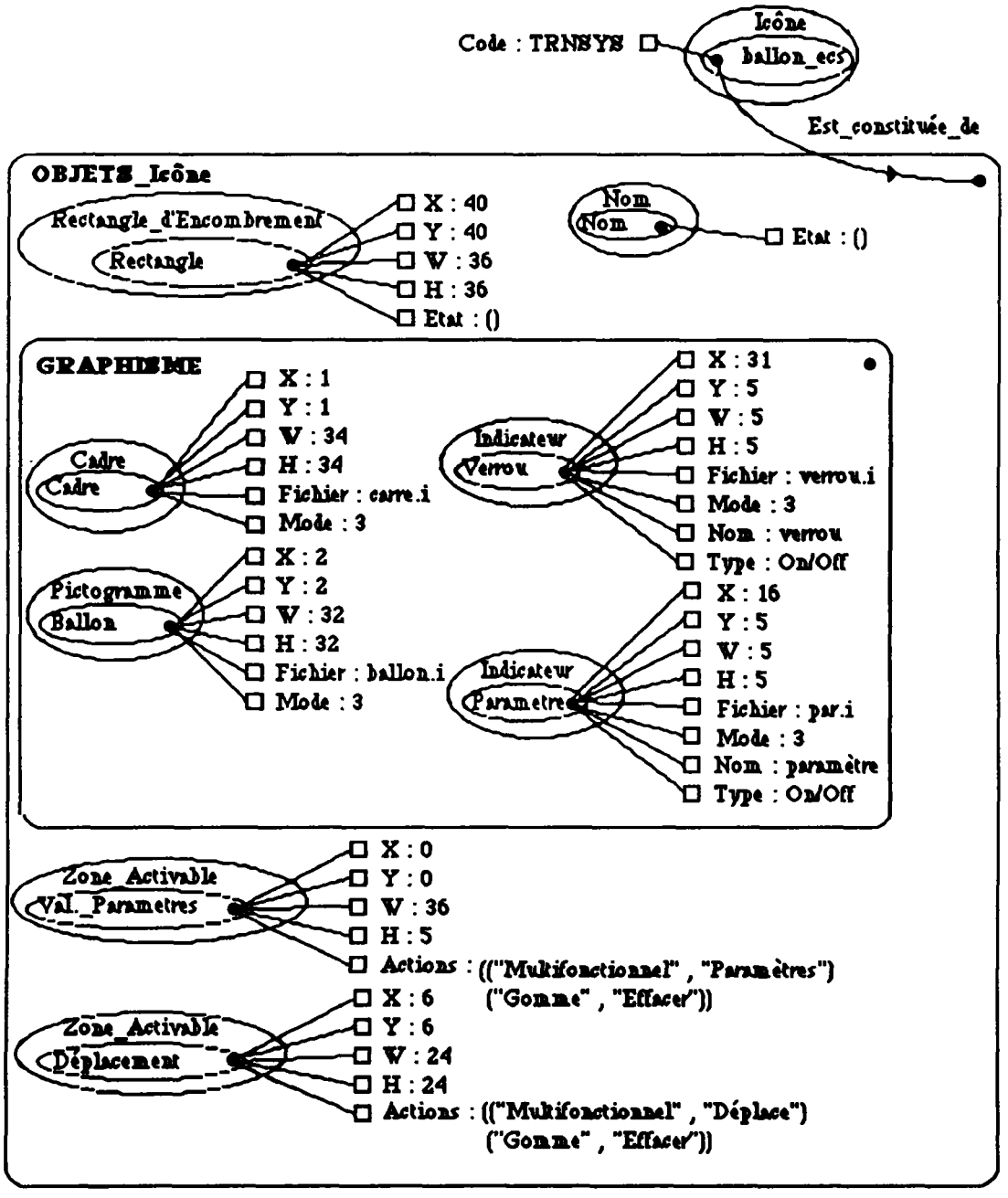


Figure 2-12-b : Icône Ballon de stockage - Instanciation des classes

La figure 2-12-b donne un exemple d'instanciation des classes qui concernent le modèle HBDS de l'objet "icône". Il s'agit d'une icône (instance de nom "ballon_ecs") qui représente un modèle de ballon de stockage ; le rectangle d'encombrement (instance de nom "rectangle") est invisible, le cadre (instance de nom "cadre") est un rectangle visible, un indicateur de type "patte" (instance de nom "paramètre") est présent et indique à l'utilisateur la nécessité pour la cohérence de son assemblage, d'introduire les valeurs des paramètres ; cette "patte" est un demi-cercle visible ; d'autre part, il existe un indicateur

qui montre que l'icône est verrouillée (instance de nom "verrou"). Cette icône possède deux zones activables ; l'une (instance de nom "Val._parametre") permet à partir d'un clic (outil "Multifonctionnel") l'ouverture d'une fenêtre spécialisée pour l'introduction des valeurs des paramètres ; l'autre (instance de nom "Déplacement") permet le déplacement de l'icône dans la fenêtre où elle est positionnée (outil "Multifonctionnel").

La figure 2-12-c illustre l'aspect graphique de l'icône "ballon_ecs" ; la figure 2-12-d montre les zones activables de l'icône "ballon_ecs".

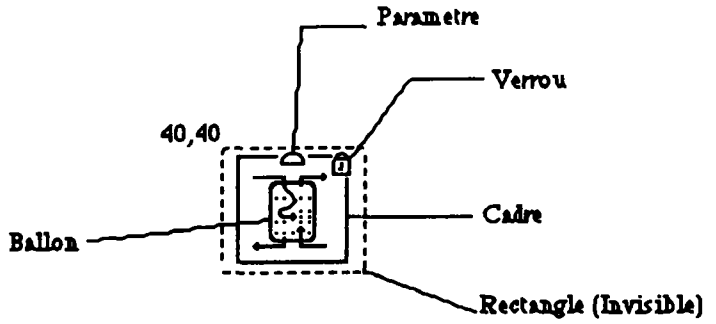


Figure 2-12-c : Icône Ballon de stockage - Aspect graphique

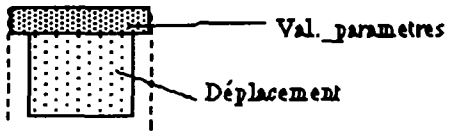


Figure 2-12-d : Icône Ballon de stockage - Zones activables

La figure 2-12-e représente un ballon de stockage ; le cadre est un rectangle visible, les "pattes" sont des formes triangulaires visibles et il existe un indicateur qui montre que l'icône est verrouillée.

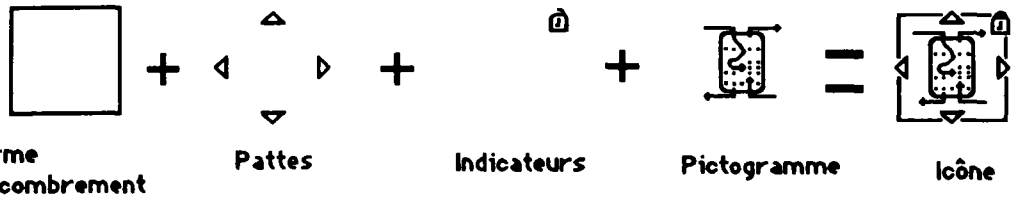


Figure 2-12-e : Icône Ballon de stockage - Pattes triangulaires

2-3-3 Le modèle de fenêtrage

L'aspect externe d'IISIBât est constitué de fenêtres multiples, pilotées par un dispositif de pointage (souris). Les fenêtres sont des objets structurés typés, qui ont des représentations graphiques sur l'écran, sous la forme de cadre rectangulaire muni d'un titre.

Les fenêtres peuvent se superposer, une fenêtre peut être partiellement ou complètement cachée par une autre ; elles peuvent être déplacées, redimensionnées, ouvertes ou fermées (représentation sous forme réduite).

Les fenêtres peuvent contenir deux types d'objets : les images et les applications. Une image est un objet qui se dessine sur un support, il s'agit d'un élément purement graphique ; une application est un objet qui contient des attributs graphiques (police de caractères, couleur de fond, style de ligne, curseur, motif de remplissage), et qui possède un comportement au sein de l'interface.

Les applications reçoivent des messages lorsque l'utilisateur réalise des opérations avec la souris sur la zone rectangulaire associée. Les actions de l'utilisateur sont appelées événements, dont les principaux sont :

- un bouton de la souris a été enfoncé (clic) ;
- déplacement de la souris en maintenant un bouton enfoncé ;
- relâchement d'un bouton ;
- un caractère au clavier a été frappé ;
- la souris entre ou sort d'une fenêtre.

Il est possible de gérer des événements particuliers, issus de la combinaison des événements cités plus haut : par exemple, déplacer la souris en maintenant un bouton et un caractère au clavier enfoncés. Dans le cadre de ce travail, la gestion de tels événements est exclue, ils compliquent l'utilisation du logiciel (difficulté de mémoriser les touches).

Les fenêtres fournissent un modèle intuitif pour gérer plusieurs activités en parallèle, chaque activité ayant lieu dans une fenêtre.

Les types de fenêtres développés dans le cadre de ce travail correspondent aux quatres entités principales :

- les fenêtres de type *gestionnaire des comptes* ;
- les fenêtres de type *gestionnaire de bibliothèques* ;
- les fenêtres de type *bibliothèque* ;
- les fenêtres de type *panneau.*

La fenêtre de type *gestionnaire des comptes* est la traduction graphique de l'objet de même nom (cf. § 2-3-1).

Les fenêtres de type *gestionnaire de bibliothèques* et de type *bibliothèque* représentent les objets de type *gestionnaire de bibliothèques* (cf. § 2-3-1).

La fenêtre de type *panneau*, quant à elle, représente la zone de travail de l'utilisateur : cette fenêtre va lui permettre de réaliser un diagramme de flux et de simuler son assemblage.

Ces quatre types de fenêtres sont conçus de façon homogène, et possèdent des protocoles d'accès standard. Elles sont toutes structurées de la façon suivante ; elles ont :

- un titre ;
- un carré de fermeture ;
- un bloc-note ;
- une zone de travail ;
- une boîte à outils associée.

Le carré de fermeture a pour fonction essentielle la suppression de la fenêtre et, suivant le type de la fenêtre, peut modifier les données stockées dans les fichiers associés. Dans le cas de la fenêtre de type *gestionnaire de bibliothèques*, le carré de fermeture modifie la structure de données contenue dans le fichier *biblios.ll*, si la modification est autorisée.

Le bloc-note apparaît suite à une action sur un bouton ; il permet à un utilisateur d'introduire ses remarques.

La zone de travail est une application qui réagit aux événements réalisés par l'utilisateur. Ces événements déclenchent des actions qui sont fonction de l'outil sélectionné de la boîte à outils.

La composition des boîtes à outils dépend du type d'utilisateur (nous avons distingué trois catégories d'utilisateurs (cf. § 2-2-2) : les "Ingénieurs Concepteurs", les "Ingénieurs Assembleurs" et les "Ingénieurs Analystes"). La séparation en groupes d'utilisateurs permet de disposer du produit, selon le niveau de compétence de l'utilisateur, afin de limiter les risques de fausses manipulations. Cette méthode de travail implique que chaque groupe de travail n'a accès qu'à certaines fonctions ; chaque groupe peut effectuer les tâches qui lui sont spécifiques mais aussi celles des groupes inférieurs. La décomposition du processus de Modélisation/Simulation en plusieurs métiers va permettre aussi de réduire la taille des boîtes à outils. Une boîte à outils riche et dont les outils ne répondent pas aux mêmes objectifs (une boîte à outils qui permettrait par exemple de concevoir des modèles, d'assembler, de lancer des simulations et d'analyser les résultats de simulation) aura tendance à handicaper l'utilisateur terminal.

2-3-3-1 Les boîtes à outils

Pour la conception des boîtes à outils, deux approches peuvent être adoptées :

- soit de réaliser une boîte à outils générale, pour toutes les fenêtres de travail ;
- soit d'associer une boîte à outils à chaque fenêtre.

La première approche part de l'idée qu'il faut éviter la redondance. En effet, certaines fonctions utilitaires reviennent souvent : copier, coller, sélectionner, effacer, modifier, créer, déplacer. Cependant, certaines de ces fonctions sont trop abstraites pour générer les mêmes actions dans tous les types de fenêtres. En effet, créer un compte dans une fenêtre de type *gestionnaire de comptes* ne correspond pas à la même action que créer un macro-modèle dans une fenêtre de type *panneau*, ou créer un modèle à partir d'une fenêtre de type *bibliothèque*. Un même outil qui regrouperait la fonction "créer" risque de compliquer l'apprentissage d'un utilisateur débutant. Par contre, la fonction "effacer" correspond à la même action dans tous les types de fenêtres : on sélectionne l'outil *gomme*, puis on pointe sur l'objet à effacer.

Cette approche possède aussi l'inconvénient d'augmenter le nombre d'informations échangées entre les modules de l'architecture logicielle (figure 2-13).

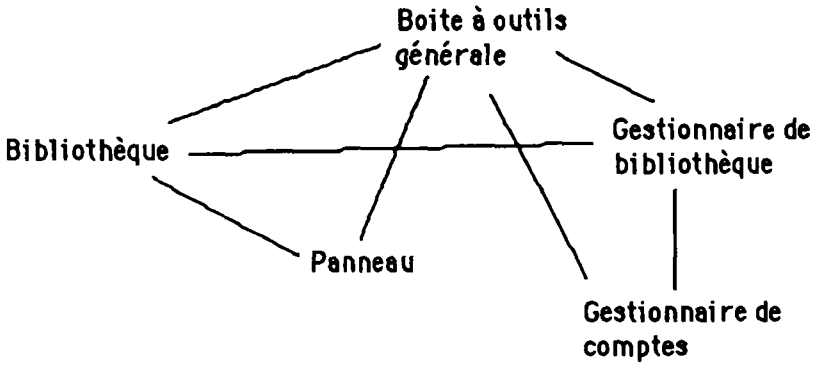


Figure 2-13 : Communication entre modules

Pour que l'objet de type *bibliothèque* fonctionne, il doit communiquer avec le module de type *boîte à outils* et avec le module de type *gestionnaire de bibliothèques*, qui lui-même doit communiquer avec le module de type *boîte à outils*.

S'il y a trop de relations entre les modules, tout changement ou toute erreur peut se propager à un grand nombre d'entre eux. Pour qu'un module soit utilisable dans un nouvel environnement, il ne doit pas dépendre d'un trop grand nombre de modules. De plus, si deux modules communiquent entre eux, ils doivent échanger aussi peu d'informations que possible. L'objectif qui consiste à atteindre le nombre minimum de liens entre modules peut être obtenu de deux façons :

- soit en adoptant une structure totalement centralisée ; un module communique avec tous les objets (figure 2-14-a) ;
- soit en adoptant une structure où un module ne communique qu'avec ses voisins immédiats (figure 2-14-b).

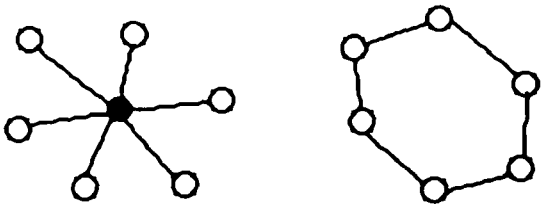


Figure 2-14 : Les différents types de communication entre modules

Une boîte à outils associée à chaque type de fenêtre permet d'obtenir une structure avec très peu de communication entre modules (figure 2-15). Cette façon de procéder est en accord avec le principe de la répartition du dialogue (cf. § 1-5-1-1). De plus, dans le cadre d'IISIBât, très peu d'outils sont véritablement redondants, comme l'outil gomme.

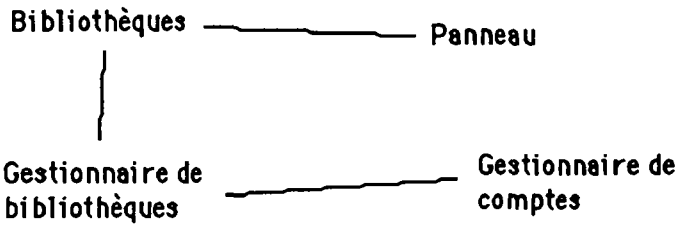


Figure 2-15 : Communication entre modules adopté pour IISIBât

2-3-3-2 Utilisation des outils

Il existe trois façons d'utiliser un outil :

- soit on sélectionne d'abord les objets sur lesquels on veut agir, puis on sélectionne l'outil, qui va déterminer les actions sur les objets sélectionnés : modification de la valeur d'un champ en général (approche objets) ;
- soit on sélectionne d'abord l'outil, puis l'objet sur lequel on veut agir (approche fonctionnelle) ;
- soit on sélectionne l'outil, et l'action associée à cet outil est déclenchée : création automatique d'un fichier DECK, lancement de la simulation.

La première approche (approche objets) est utilisée essentiellement dans le cadre d'une boîte à outils générale. Cette approche est celle de l'environnement du bureau MacIntosh. L'utilisateur choisit d'abord les objets sur lesquels il veut agir. Les outils qui ne peuvent s'appliquer à cet objet apparaissent sous une forme grisée et ne sont plus sélectionnables par l'utilisateur (cela permet d'opérer un filtre, dans le cas où la boîte à outils est très riche).

La seconde approche (approche fonctionnelle) sous-entend que l'utilisateur raisonne plutôt sur les actions à réaliser : choix de modules, réalisation de connexions, stockage. Pour gommer un objet dans une fenêtre, l'utilisateur choisit d'abord l'outil "gomme" dans la boîte à outils, puis pointe sur les objets qu'il veut effacer dans la fenêtre. Nous avons opté pour cette approche dans le cadre d'IISIBât (cette approche est celle d'outils de dessin comme "SuperPaint" dans l'environnement MacIntosh).

Remarque : l'action "sélectionner" est, elle aussi, dirigée par l'utilisateur dans la seconde approche ; il doit d'abord choisir l'outil "sélectionner" avant d'effectuer sa sélection.

Nous n'excluons pas de construire des boîtes à outils pour d'autres applications à partir de l'approche objets (exemple : IISIBât/COMIS).

2-4 Modèle Structurel

Nous présentons ci-après l'aspect visuel des différentes applications développées ; ces applications constituent une partie importante dans le cadre du travail de cette présente recherche ; ce travail peut s'apparenter à une expérimentation, c'est à dire à une application pratique des concepts développés (ESI). Cette application est d'ailleurs suivie d'un travail d'évaluation (cf. § 2-5).

Il s'agit ici de déterminer comment mettre en œuvre le modèle conceptuel à partir de la boîte à outils Aïda. Sont présentées dans ce paragraphe les sous-applications suivantes :

- le gestionnaire des comptes ;
- le gestionnaire des bibliothèques ;
- la bibliothèque ;
- le panneau d'assemblage ;
- la fenêtre mère ;
- l'application spécialisée dans le tracé des courbes ;
- l'application spécialisée dans l'introduction des valeurs des paramètres ;
- l'application spécialisée dans l'introduction des connexions.

2-4-1 Le gestionnaire des comptes

La gestion des comptes n'est autorisée que pour les utilisateurs principaux. Le gestionnaire des comptes (figure 2-16) permet de consulter et de modifier les caractéristiques d'un compte (nom, mot de passe et groupe d'utilisateur) ; il permet aussi la communication entre utilisateurs. Un gestionnaire des comptes appartient à l'utilisateur propriétaire du répertoire où se trouve ce gestionnaire.

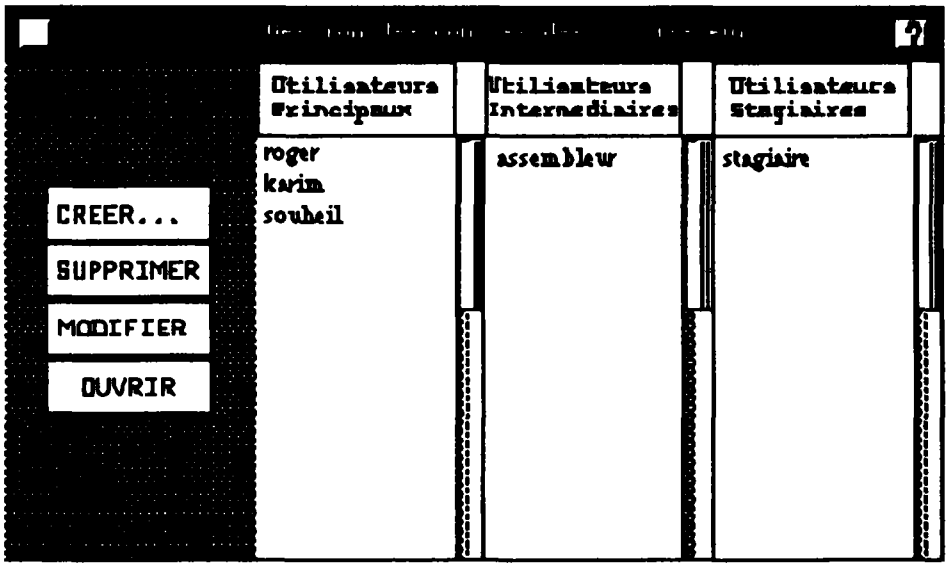


Figure 2-16 : Gestionnaire des comptes

CREER... permet d'ajouter un nouvel utilisateur, en l'insérant dans une des listes d'utilisateurs (utilisateurs principaux, utilisateurs intermédiaires, utilisateurs stagiaires) ; le propriétaire du gestionnaire des comptes pourra ensuite visualiser et copier le contenu du gestionnaire de bibliothèques relatif à l'utilisateur ajouté. La procédure pour utiliser cet outil est la suivante : l'utilisateur de l'interface sélectionne l'outil, une fenêtre apparaît (cf. figure 2-17) dans laquelle il doit spécifier la catégorie, le nom, le mot de passe et la référence (chemin d'accès) du nouvel utilisateur.

SUPPRIMER permet de supprimer un utilisateur ; l'outil MODIFIER permet de modifier les caractéristiques (catégorie, nom, mot de passe et référence) d'un utilisateur ; l'outil OUVRIR permet d'ouvrir le gestionnaire de bibliothèques rattaché à un utilisateur. Pour ces trois outils, la procédure d'utilisation est la suivante : l'utilisateur de l'interface

sélectionne l'outil en question, et pointe (clic) sur une des chaînes de caractères représentant un nom d'utilisateur.

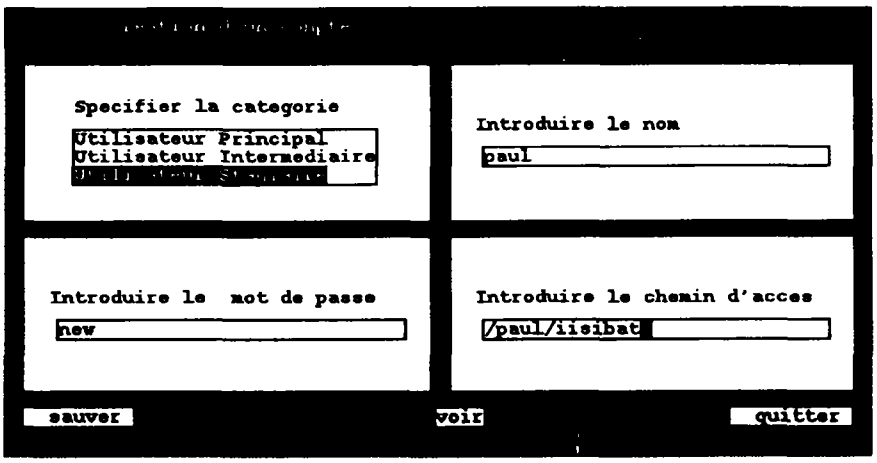


Figure 2-17 : Fenêtre de création de comptes

2-4-2 Le gestionnaire de bibliothèques

La gestion générale des bibliothèques couvre deux champs distincts ; d'une part, la gestion générale des bibliothèques (à ce niveau les bibliothèques sont considérées comme des contenants vides ou non et caractérisés par un nom) ; d'autre part, la gestion du contenu des bibliothèques (modèles élémentaires, macro-modèles ou projets).

Remarque : le titre de la fenêtre représentant un gestionnaire de bibliothèques désigne le propriétaire de ce gestionnaire.

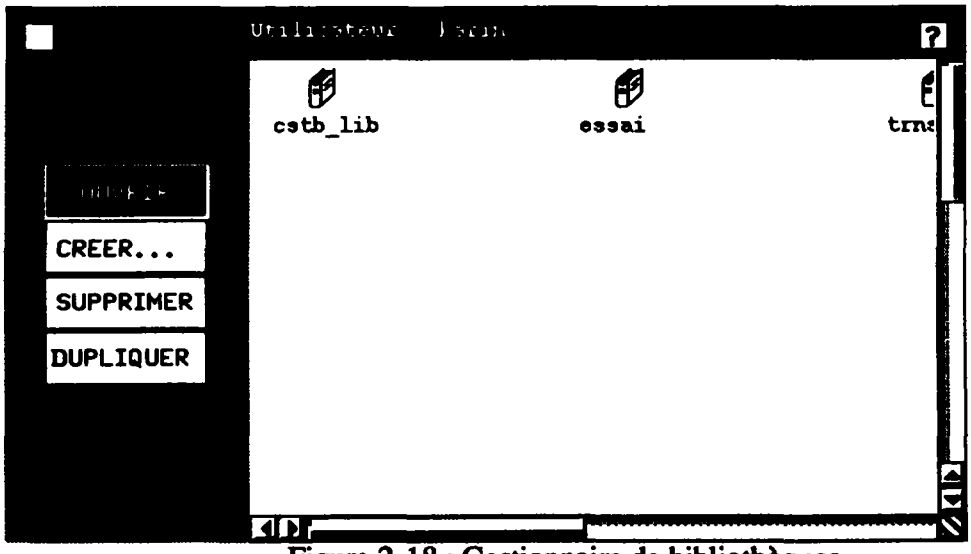


Figure 2-18 : Gestionnaire de bibliothèques

CREER... permet de créer une nouvelle bibliothèque ; la procédure d'utilisation est la suivante : l'utilisateur de l'interface sélectionne cet outil, une fenêtre apparaît à l'écran dans laquelle il doit spécifier le nom de la nouvelle bibliothèque, ainsi que le type de la

bibliothèque, type qui peut être soit "bibliothèque standard", soit "bibliothèque de projets". Une "bibliothèque standard" ne peut contenir que des modèles et des macro-modèles, une "bibliothèque de projets" ne peut contenir quant à elle que des projets. La nouvelle bibliothèque créée possède comme racine son propre nom ; ce nom apparaît aussi sous la forme d'une chaîne de caractères, juste en dessous de l'icône en forme de dossier (icône qui est une instance de l'objet générique "icône" défini au paragraphe 2-3-2-2).

OUVRIR permet de visualiser une bibliothèque ; SUPPRIMER permet de supprimer une bibliothèque. Pour ces deux outils, la procédure est la même : l'utilisateur de l'interface sélectionne l'outil en question et pointe sur l'icône en forme de dossier représentant une bibliothèque. L'action de supprimer une bibliothèque n'est possible que si celle-ci est vide.

DUPLIQUER permet de copier une bibliothèque : l'utilisateur sélectionne cet outil, il sélectionne ensuite une bibliothèque dans le gestionnaire de bibliothèques "source" ; un message apparaît où l'utilisateur désigne le gestionnaire de bibliothèques "cible". La copie d'une bibliothèque sous-entend une copie des fichiers relatifs à la bibliothèque sélectionnée (fichiers de description des modules "TRNSYS", fichiers de description des macro-modèles et des projets, fichiers de description des icônes, fichiers de description des fiches Proforma) dans le répertoire privé du propriétaire du gestionnaire de bibliothèques "cible".

2-4-3 Les bibliothèques

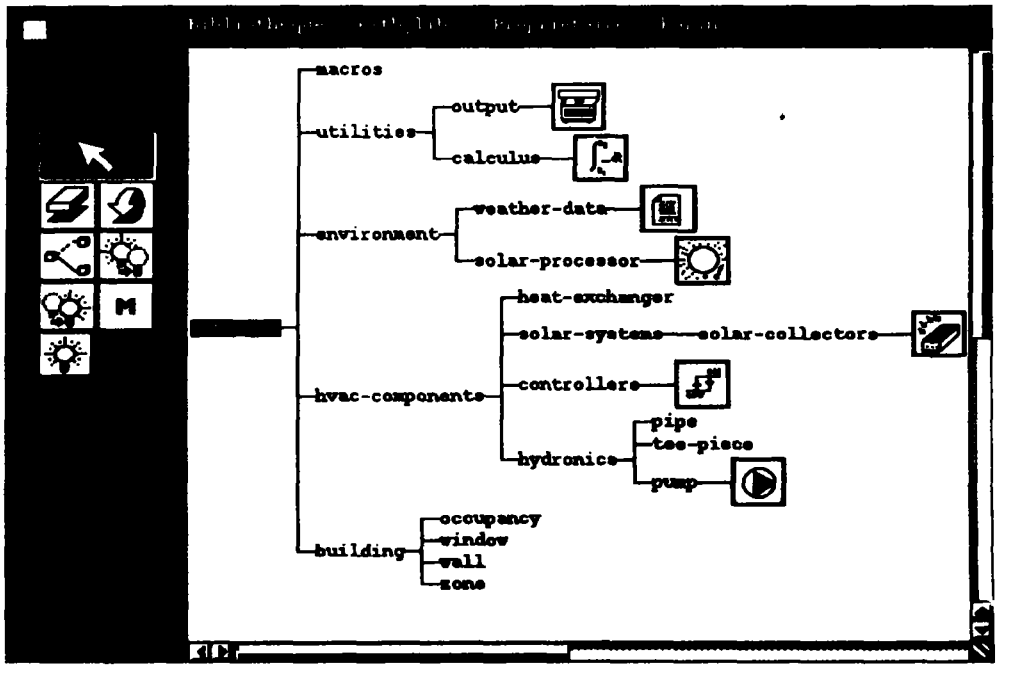


Figure 2-19 : Aspect des bibliothèques

Les bibliothèques se présentent sous la forme d'arbres (figure 2-19), dont les nœuds sont soit des dossiers, soit des objets de type modèles, macro-modèles ou projets. Les

dossiers apparaissent sous la forme d'une chaîne de caractères, les objets de la structure de données quant à eux ont la forme d'icônes graphiques. Cette représentation présente plusieurs avantages :

- elle simplifie l'accès aux objets, par la visualisation de la totalité de la bibliothèque ; le nombre d'opérations nécessaires à un utilisateur avec le dispositif de pointage pour accéder à un objet est faible ;
- elle permet d'obtenir des bibliothèques structurées et personnalisées assez rapidement (cf. § 2-3-1-1).

 outil de nom "Sélection" (c'est l'outil par défaut de la fenêtre "Bibliothèque") ; il permet de pointer sur des objets de la bibliothèque : dossier ou icône. On ne peut sélectionner qu'un objet à la fois, même si plusieurs bibliothèques sont ouvertes en même temps. La procédure d'utilisation est la suivante ; l'utilisateur de l'interface sélectionne cet outil et pointe sur un des objets de la bibliothèque ; celui-ci passe alors en inverse-vidéo.

 outil de nom "Gomme"; il permet de supprimer un objet de la bibliothèque ; la procédure d'utilisation est la suivante : l'utilisateur de l'interface sélectionne cet outil et pointe sur un des objets de la bibliothèque. On ne peut supprimer que des objets se situant aux extrémités de l'arbre ; si l'utilisateur veut supprimer un modèle ou un macro-modèle, l'interface vérifie si celui-ci n'est pas utilisé par ailleurs (dans un assemblage).

 outil de nom "Undo" ; il permet de revenir à l'état précédent, en annulant la dernière action ; la procédure d'utilisation est la suivante : l'utilisateur de l'interface sélectionne cet outil.

 outil de nom "Dossier" ; il permet de créer un nouveau dossier ; la procédure d'utilisation est la suivante : l'utilisateur de l'interface sélectionne cet outil, et pointe sur un dossier déjà existant ; une fenêtre apparaît où l'utilisateur introduit le nom du nouveau dossier. Dans un des prototypes logiciels, l'utilisateur introduit aussi une fiche "Proforma" relative au nouveau dossier ; ainsi tout nouvel objet rattaché à ce dossier héritera des informations contenues dans cette fiche "Proforma".

 outil de nom "Copier" ; il permet de stocker dans un tampon un sous-arbre appartenant à la bibliothèque. La procédure d'utilisation est la suivante : l'utilisateur de l'interface sélectionne cet outil, et pointe sur un objet de la bibliothèque. Le sous-arbre copié possède comme racine l'objet de la bibliothèque sélectionné par l'utilisateur.

 outil de nom "Coller" ; il permet de coller le sous-arbre stocké dans le tampon. La procédure d'utilisation est la suivante : l'utilisateur de l'interface sélectionne cet outil et pointe sur un dossier.

 outil de nom "Macro" ; il permet de visualiser un macro-modèle sous sa forme éclatée dans un nouveau panneau d'assemblage. La procédure d'utilisation est la

suivante : l'utilisateur de l'interface sélectionne cet outil et pointe sur un objet de type macro-modèle (les macro-modèles apparaissent dans les bibliothèques standard sous la forme d'une icône plus une chaîne de caractères et sont donc différentiables par l'utilisateur).

 outil de nom "Créer" ; il permet de créer un nouveau modèle et de l'insérer dans la bibliothèque, ou de modifier un modèle existant. La procédure d'utilisation est la suivante : l'utilisateur de l'interface sélectionne cet outil et pointe sur un dossier pour créer un nouveau modèle ; s'il veut modifier un modèle existant, il doit pointer sur une icône représentant un modèle.

2-4-4 Le panneau d'assemblage

Les panneaux d'assemblage ont l'aspect de la figure 2-20.

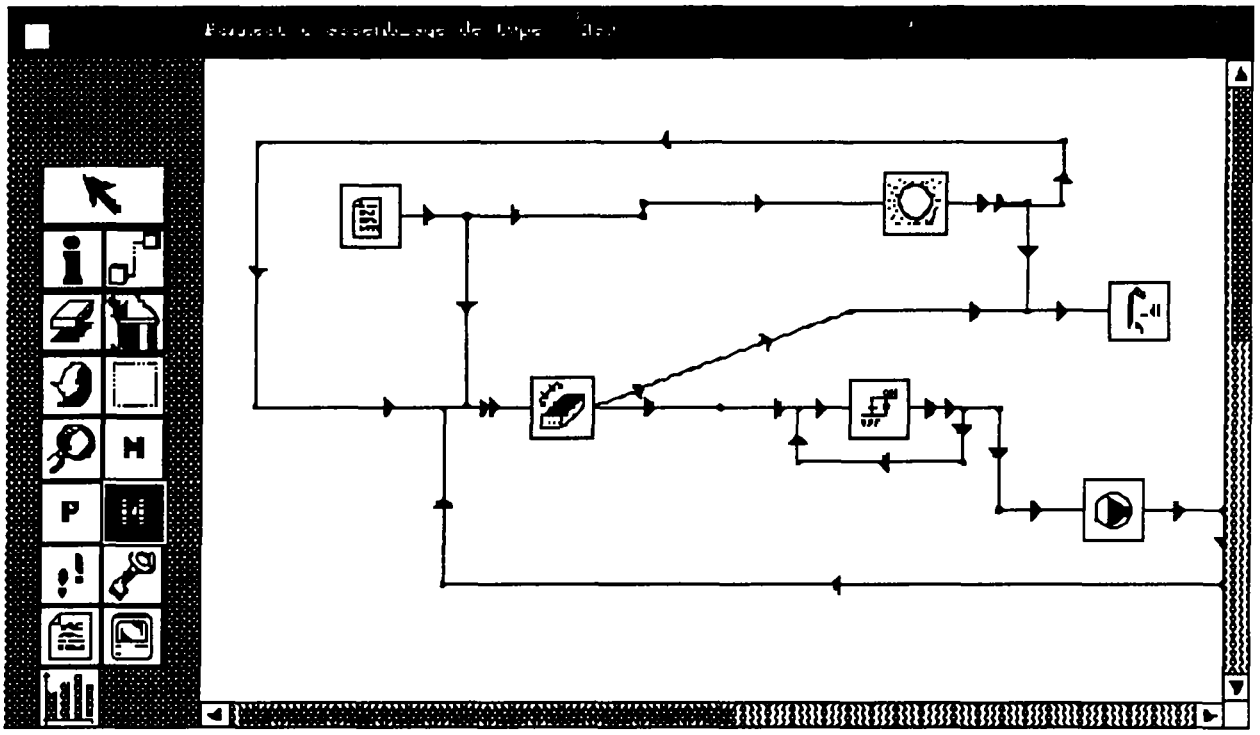


Figure 2-20 : Panneau d'assemblage

 outil de nom "Multifonctionnel" (c'est l'outil par défaut de la fenêtre "Panneau d'assemblage") ; il permet :

- d'instancier des modules ; pour cela l'utilisateur doit d'abord choisir un objet (modèle, macro-modèle ou projet) dans une bibliothèque, puis cliquer dans le panneau d'assemblage ;
- de déplacer des objets (icônes ou points activables des liaisons) dans le panneau d'assemblage ; pour déplacer une icône, l'utilisateur doit cliquer dans la partie centrale de l'icône (partie 1 sur la figure 2-21) et déplacer la souris sans la lâcher ;

pour déplacer des points activables des liaisons, l'utilisateur doit cliquer sur l'un d'entre eux et déplacer la souris sans la lâcher ;

- de visualiser et d'introduire des données numériques (paramètres, valeurs initiales des entrées, valeurs initiales des "dérivatives", liste des sorties) ; si l'utilisateur veut spécifier les valeurs des paramètres (cf. figure 2-22), il doit cliquer sur la partie supérieure de l'icône (partie 2 sur la figure 2-21) ; si l'utilisateur veut spécifier les valeurs initiales des entrées, il doit cliquer sur la partie gauche de l'icône (partie 4 sur la figure 2-21) ; si l'utilisateur veut spécifier les valeurs initiales des "dérivatives", il doit cliquer sur la partie inférieure de l'icône (partie 3 sur la figure 2-21) ; enfin si l'utilisateur veut visualiser la liste des sorties, il doit cliquer sur la partie droite de l'icône (partie 5 sur la figure 2-21).

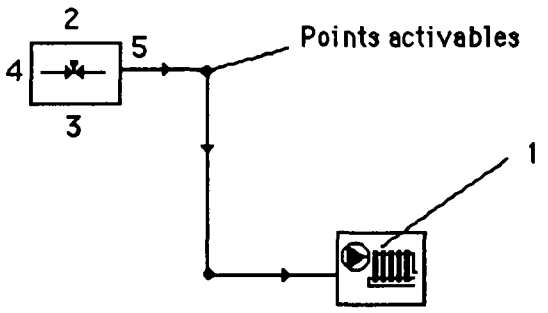


Figure 2-21 : Zones activables des objets manipulés pour l'outil "Multifonctionnel" du panneau d'assemblage

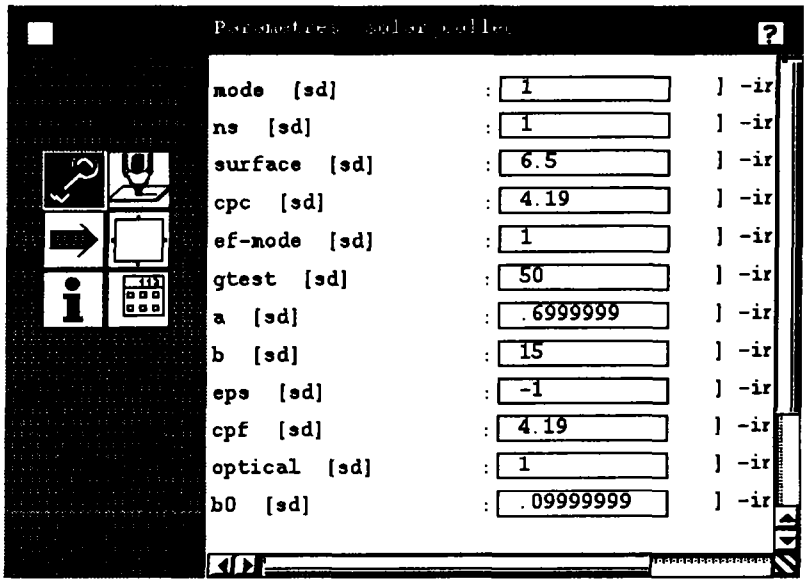


Figure 2-22 : Fenêtre d'introduction des paramètres

 outil de nom "Info" ; il permet d'afficher à l'écran la fiche Proforma (cf. figure 2-23) associée aux modèles ; la procédure d'utilisation est la suivante : l'utilisateur doit cliquer sur une icône représentant un modèle dans le panneau d'assemblage.

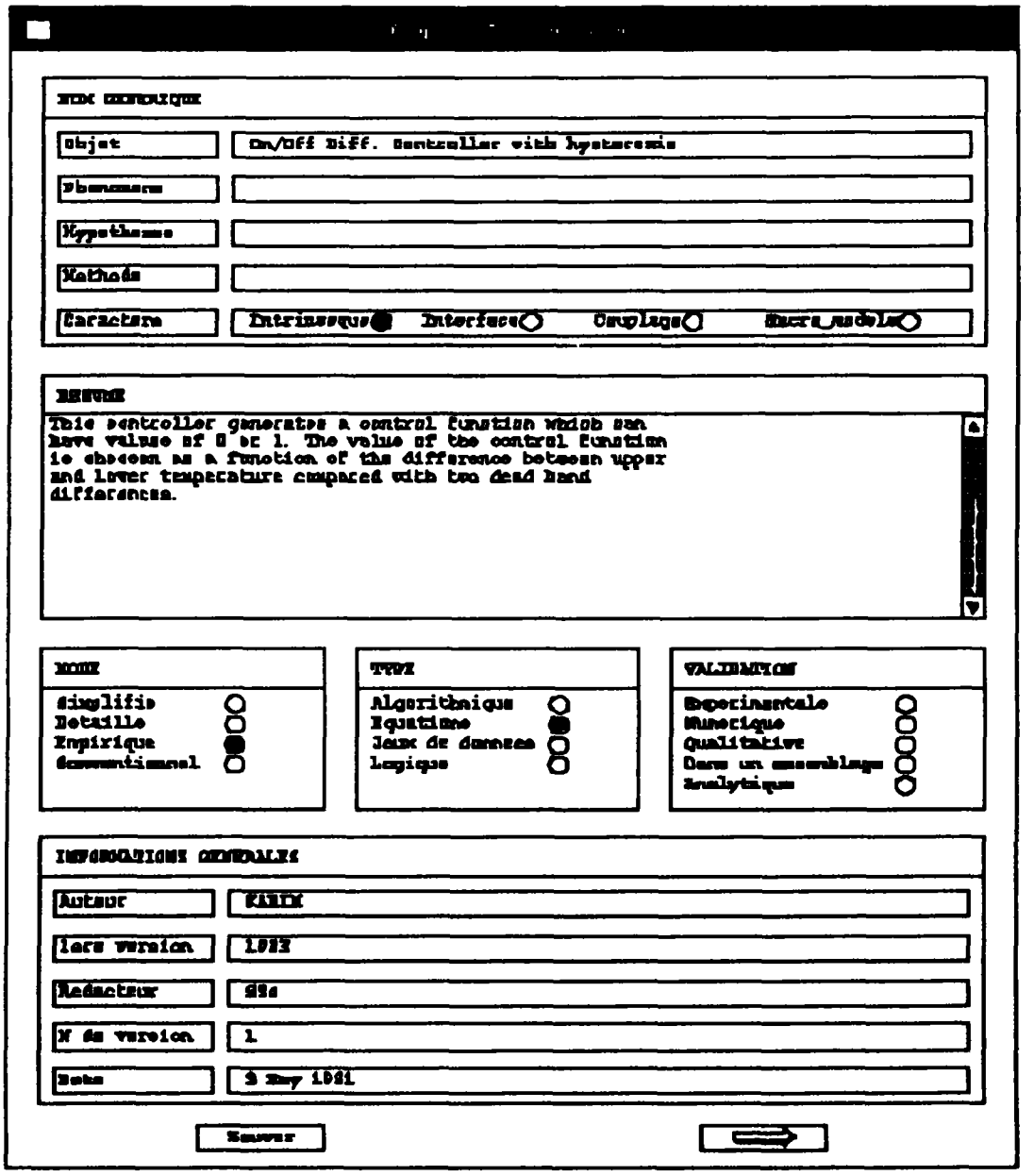


Figure 2-23 : Aspect des fiches Proforma sous IISIBât/TRNSYS

 outil de nom "Lier" ; il permet :

- de lier des modèles (les liaisons entre modèles sont représentées par des segments contenant des points activables) ; deux procédures d'utilisation existent ; soit

l'utilisateur clique sur l'icône amont, puis sur l'icône aval, alors les points activables de la liaison se mettent en place de façon automatique ; soit l'utilisateur clique sur l'icône amont, puis clique sur le panneau d'assemblage et enfin clique sur le modèle aval ; cette approche permet à l'utilisateur de mettre en place manuellement les points activables de la liaison

- d'afficher les fenêtres d'introduction des connexions de variables (cf. figure 2-24) ; la procédure d'utilisation est la suivante ; l'utilisateur clique sur un point activable d'une liaison.

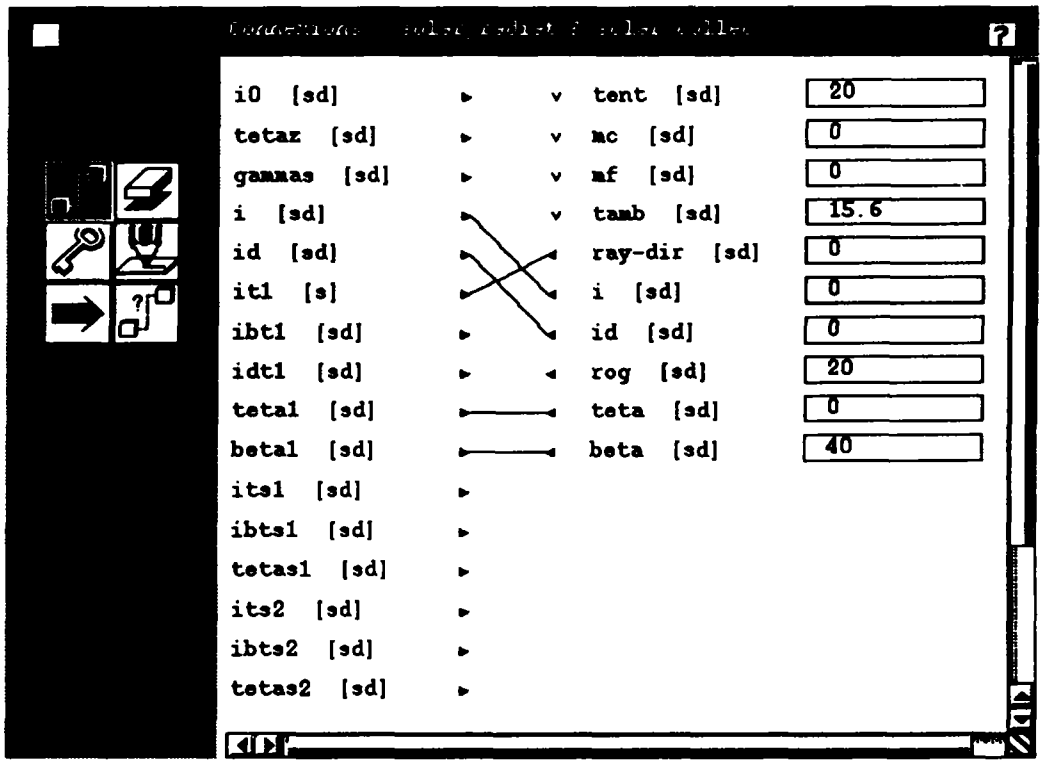


Figure 2-24 : Fenêtre d'introduction des connexions

 outil de nom "Gomme" ; il permet d'effacer des objets du panneau d'assemblage ; pour effacer une icône, l'utilisateur pointe sur cette icône ; pour effacer une liaison, l'utilisateur pointe sur un point activable de cette liaison.

 outil de nom "Clear" ; il permet d'effacer tous les objets du panneau d'assemblage (par simple sélection de cet outil).

 outil de nom "Undo" ; il permet d'annuler la dernière action (par simple sélection de cet outil).

 outil de nom "Sélection" ; il permet de sélectionner (ou de désélectionner) des icônes dans le panneau d'assemblage ; les icônes sélectionnées passent alors en inverse-vidéo ;

deux procédures d'utilisation existent : soit l'utilisateur clique sur les icônes, soit il clique sur le panneau d'assemblage et déplace la souris de façon à former un rectangle qui englobe les objets de sa sélection.

 outil de nom "Loupe" ; il permet de visualiser un macro-modèle sous sa forme éclatée dans une fenêtre spécialisée ; pour cela l'utilisateur doit cliquer sur un macro-modèle.

 outil de nom "Macro" ; il permet de lancer une procédure de stockage d'un macro-modèle en bibliothèque ; la procédure d'utilisation est la suivante : l'utilisateur doit d'abord sélectionner les objets (modèles, macro-modèles et liaisons) composant son nouveau macro-modèle, ensuite il doit sélectionner un dossier dans une bibliothèque standard et enfin sélectionner cet outil.

 outil de nom "Projet" ; il permet de lancer une procédure de stockage d'un projet en bibliothèque ; la procédure d'utilisation est la suivante : l'utilisateur doit d'abord sélectionner les objets (modèles, macro-modèles et liaisons) composant son nouveau projet, ensuite il doit sélectionner un dossier dans une bibliothèque de projets et enfin sélectionner cet outil.

 outil de nom "Eclate" ; il permet de visualiser un macro-modèle sous sa forme éclatée dans le panneau d'assemblage (en vue de le modifier par exemple) ; pour cela l'utilisateur doit cliquer sur un macro-modèle.

 outil de nom "Trace" ; il permet d'indiquer qu'un modèle ou un macro-modèle est "tracé" (un macro-modèle "tracé" signifie que tous les modèles élémentaires le constituant sont "tracés") ; pour cela l'utilisateur doit cliquer sur un modèle ou un macro-modèle ; un indicateur en forme de "T" surcharge alors le dessin de l'icône.

 outil de nom "Verrou" ; il permet de verrouiller (ou de déverrouiller) un modèle ou un macro-modèle (un macro-modèle vérrouillé signifie que tous les modèles élémentaires le constituant sont vérrouillés) ; pour cela l'utilisateur doit cliquer sur un modèle ou un macro-modèle ; un indicateur en forme de verrou surcharge alors le dessin de l'icône.

 outil de nom "DECK" ; il permet de générer le fichier DECK relatif à l'assemblage présent dans le panneau d'assemblage (par simple sélection de cet outil).

 outil de nom "Run" ; il permet de lancer en arrière-plan la simulation (par simple sélection de cet outil).

Remarques : ces deux derniers outils pourraient être regroupés en un seul. D'autre part, dans la version actuelle, il n'existe pas de procédure qui indique à l'utilisateur que la simulation est terminée.

 outil de nom "Courbe" ; il permet de visualiser les résultats de la simulation ; pour cela l'utilisateur doit cliquer sur un utilitaire de type "imprimante", les résultats apparaissent alors dans une fenêtre de type "Editeur de courbes" (cf. figure 2-25).

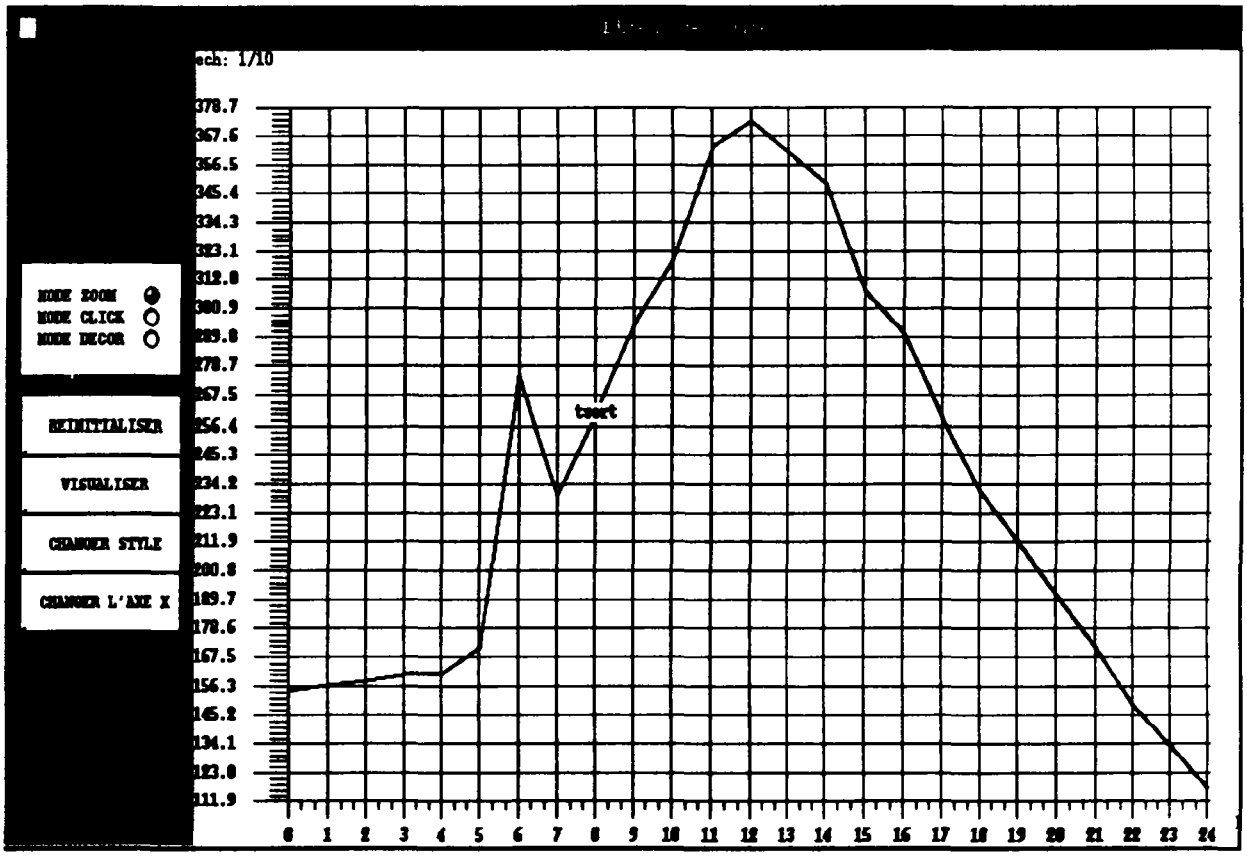


Figure 2-25 : Module traceur de courbes

2-4-5 La fenêtre mère

La fenêtre mère (figure 2-26) a pour objectif principal la gestion des fenêtres d'IISIBât. Elle permet :

- d'instancier un panneau d'assemblage en donnant à l'utilisateur le choix des outils associés au panneau (bouton "Panneau d'assemblage", figure 2-26) ;
- de visualiser le gestionnaire des comptes (bouton "Gestionnaire des comptes", figure 2-26);
- d'activer l'aide manipulatoire (bouton "Aide manipulatoire", figure 2-26) ; cela consiste à donner à l'utilisateur de l'information sur les fenêtres de travail et sur les outils qui y sont associés. Pour cela, l'utilisateur doit déplacer le dispositif de pointage, l'information apparaît dans une fenêtre de dialogue à chaque fois que l'utilisateur pointe sur un objet de l'interface graphique (événement de type "un bouton de la souris a été enfoncé") ;
- de changer la langue des messages (bouton "Preferences", figure 2-26) ;

- d'ajouter dans les panneaux d'assemblage des boutons qui utilisent dans leur action associée des connaissances stockées sous forme de règles expertes (bouton "Module Expert", figure 2-25).

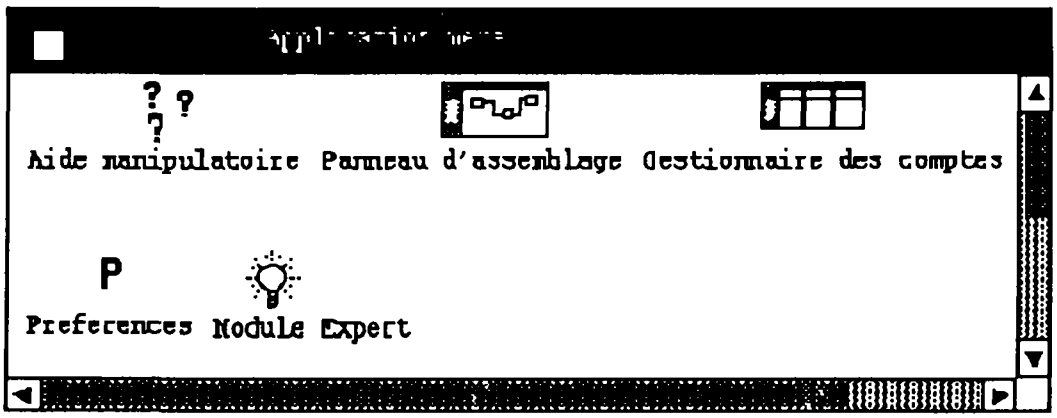


Figure 2-26 : Fenêtre mère

2-5 Evaluation de l'interface réalisée

L'évaluation de l'interface a été effectuée à partir de sessions "type" comme celle présentée à l'annexe 3. Cette évaluation concerne à la fois, la confrontation des performances de l'interface réalisée par rapport aux exigences de qualité définies au paragraphe 1-3, et l'évaluation par divers utilisateurs de l'ergonomie de l'interface (modèle perceptif).

Le logiciel réalise les tâches spécifiées dans le cahier des charges en respectant les exigences de validité et de robustesse.

L'exigence d'extensibilité est atteinte dans la mesure où le principe de la modularité de la conception a été respecté.

On peut aussi considérer que l'exigence de la réutilisabilité a été atteinte. Celle-ci, en fait, touche deux aspects :

- la réutilisation du code source, démarche qui est fréquente dans un environnement LISP ;
- la réutilisation de la conception.

La conception informatique d'IISIBât/TRNSYS peut être généralisée à d'autres interfaces, qui font intervenir des codes de calcul qui fonctionnent par assemblage de modules. Des objets tels que les modèles, macro-modèles, projets, gestionnaires de bibliothèques, gestionnaires de compte, bibliothèques, panneau d'assemblage sont tout à fait réutilisables dans d'autres environnements.

Les exigences de compatibilité et d'intégrité sont parfaitement réalisées : les produits logiciels (TRNSYS, LISP, UNIX, IISIBât) interagissent les uns avec les autres sans conflits, tout en protégeant leurs différentes composantes contre les accès et les modifications non autorisés. L'exigence de portabilité est atteinte dans les limites fixées au départ, à savoir l'implémentation de l'interface dans l'environnement UNIX.

Une efficacité optimale n'a pu être atteinte, cela exigerait d'une part, l'adaptation parfaite du logiciel à un certain environnement matériel et logiciel (contraire à l'exigence de portabilité), d'autre part l'adaptation parfaite à une spécification particulière (contraire de l'exigence de l'extensibilité).

L'application réalisée a été implémentée au département Applications de l'Electricité au sein de EDF (Electricité de France) ; elle a été par ailleurs testée par divers utilisateurs au sein du CSTB. Ces différents tests avaient pour but principal d'évaluer pour divers types d'utilisateurs, la facilité d'utilisation de l'interface. Nous définissons la facilité d'utilisation de l'interface par les degrés de difficultés pour les utilisateurs d'apprendre :

- à lancer le logiciel ;
- à travailler au niveau des bibliothèques (créer de nouveaux composants) ;
- à créer un nouveau projet par assemblage ;
- à obtenir des résultats et à les interpréter ;
- à lancer la simulation ;
- à opérer des modifications en cas d'erreur.

Sans entrer dans les détails de ces différents tests, il apparaît que :

- lancer le logiciel, comprendre comment fonctionnent les boîtes à outils ne posent guère de problèmes pour les utilisateurs (quels que soient leurs droits d'accès) ;
- la création de projets (choix des modules, connexions des variables, introduction des valeurs des paramètres, introduction des cartes de contrôle) reste une tâche délicate ;
- pour les utilisateurs finaux (analystes), certaines notions restent difficiles à appréhender (cartes de contrôle "ACCELERATE", "TOLERANCE"...).

En conclusion, il est nécessaire pour améliorer la productivité du travail à l'aide de cette interface, d'automatiser encore un certain nombre de tâches. Ces tâches concernent notamment la connexion des variables et la génération de certaines cartes de contrôle. Pour automatiser ces tâches, il convient d'introduire au sein de l'application des modules experts (cf. chapitre 3).

Le tableau 2-2 résume les résultats de l'évaluation de l'interface.

	Exigence moyennement atteinte	Exigence considérée atteinte	Exigence parfaitement atteinte
Validité			X
Robustesse		X	
Extensibilité		X	
Réutilisabilité		X	
Compatibilité			X
Efficacité	X		
Portabilité		X	
Intégrité			X
Facilité d'utilisation	X		

Tableau 2-2 : Evaluation de l'interface

Conclusion du chapitre 2

La conception d'un logiciel extensible passe par la mise en place d'une architecture informatique basée sur les données. Dans le cadre d'IISIBât, la structure de données de base est constituée de trois types d'objets : modèles, macro-modèles et projets. Les modèles correspondent aux différents modules de l'outil de simulation ; les macro-modèles correspondent à des assemblages (constitués de modèles et de macro-modèles) réalisés par l'utilisateur ; les projets représentent quant à eux des assemblages prêts à la simulation (cf. § 2-3-1).

La structure de données de base est complétée par des objets qui permettent l'archivage des données de base ; il s'agit d'objets de type "gestionnaire de comptes" et "gestionnaire de bibliothèques" ; un autre objet essentiel modélise la feuille de papier où le concepteur pose et résout son problème : il s'agit du panneau d'assemblage.

Une attention toute particulière a été portée sur l'implémentation des bibliothèques ; celles-ci ont un rôle d'archivage et contiennent donc pour tout utilisateur les objets (modèles, macro-modèles et projets) auxquels il a accès. Ces bibliothèques ont été organisées en une structure arborescente dont les "nœuds" sont des "classes", et les feuilles (ou classes terminales) sont des modèles, des macro-modèles ou des projets. Cette organisation sous forme arborescente autorisera l'utilisation des propriétés d'héritage, dans des versions ultérieures. Ces propriétés possèdent un double intérêt : d'une part, elles permettent à l'utilisateur d'effectuer des modifications rapides et sûres des bibliothèques ; d'autre part, elles lui permettent une manipulation globale d'objets de même type (c'est à dire appartenant à une même classe, cf. § 2-3-1-1).

Par ailleurs, le succès d'une application interactive passe par la mise en place d'un environnement informatique basé sur le fenêtrage et la manipulation directe des objets graphiques ; le modèle iconique constitue un modèle d'interaction adapté (cf. § 2-3-2). Ce modèle est fondé sur la manipulation d'objets graphiques de deux types : les icônes et les liens entre les icônes. Les icônes correspondent à des objets concrets qui sont les objets de la structure de données de base (modèles, macro-modèles et projets). Le modèle iconique peut être généralisé à un grand nombre de codes de calcul (COMIS, SPARK, CSTBât, cf. § 2-3-2-1) ; pour ce faire, nous avons défini un objet "icône" générique (cf. § 2-3-2-2), qui par spécification crée des icônes bien adaptées à la manipulation des composants modélisés au sein du code de calcul. La définition de cet objet "icône" générique contribue au développement du modèle profond d'ESI.

L'interface réalisée à partir de cette conception, après évaluation, répond à la plupart des exigences de qualité. Par contre, elle ne répond que partiellement à l'exigence de la facilité d'utilisation. Le choix des modules, la connexion des variables, le choix des valeurs numériques constituent toujours des difficultés pour l'utilisateur non expert. L'introduction de bases de connaissances et de mécanismes de raisonnement peut rendre le logiciel plus simple à utiliser.

Chapitre 3 : Système Expert et outil de simulation

Chapitre 3 : Système Expert et outil de simulation

L'intelligence artificielle (IA) a pour ambition d'automatiser les divers mécanismes de raisonnement utilisés par l'être humain. Le raisonnement, en général, permet de trouver la solution d'un problème en enchaînant une succession d'étapes, les points de départ de chaque étape étant les résultats des étapes précédentes. La technique la plus utilisée, pour conduire un raisonnement de façon automatique, consiste à manipuler des systèmes à base de connaissances.

Dans le cadre d'IISIBât, l'objectif consiste à voir dans quelle mesure il est possible d'améliorer la productivité de l'utilisateur d'un code de calcul (TRNSYS pour ce qui nous concerne), notamment en utilisant des techniques d'IA. Ces techniques doivent pouvoir automatiser certaines des étapes conduisant à simuler correctement un problème de thermique du bâtiment.

Le paragraphe 3-1 est consacré à l'analyse du processus global de conception d'un projet de bâtiment. Le paragraphe 3-2 montre qu'il est possible d'automatiser la conception de systèmes thermiques dans le cadre de IISIBât/TRNSYS, en utilisant des techniques classiques de raisonnement, fondées sur le raisonnement logique formalisé par les règles de production. Le paragraphe 3-3 est consacré aux différents courants de pensée novateurs comme le raisonnement par analogie ou le raisonnement distribué.

3-1 Positionnement par rapport au processus global de conception

Il n'est pas dans notre objectif de proposer un environnement informatique qui automatise toutes les phases du processus de conception d'un projet de bâtiment, il s'agit plutôt de concevoir des outils permettant de faciliter, voire d'automatiser certaines tâches qui entrent dans ce processus de conception.

Pour ce qui concerne la conception des équipements des bâtiments (quels qu'ils soient : chauffage, climatisation, VMC,...), la démarche est toujours la même :

- formulation du problème à résoudre en termes d'objectifs à atteindre, de données disponibles, de solutions partielles et de contraintes (techniques, juridiques...) ; l'énoncé du problème diffère selon le type d'utilisateur, dans la plupart des cas cet énoncé est posé soit de manière incomplète soit de manière inadaptée à l'environnement de simulation ;
- proposition de plusieurs principes d'installation (ou de plusieurs installations) possibles ; cette étape est généralement effectuée grâce à la manipulation de connaissances expertes, propres aux bureaux d'études ;
- comparaison des solutions et choix d'une installation.

La formulation d'un problème

Dans le cadre du bâtiment, la formulation d'un problème s'exprime au minimum par trois questions ; la première définit le contexte de la simulation et vise dans la plupart des cas :

- soit un objet physique (pompe, capteur solaire par exemple) ;
- soit un sous-système physique, pouvant représenter un début de solution ;

- soit un système physique complet (l'objectif de la simulation dans ce cas peut consister à visualiser l'incidence d'un paramètre sur le comportement du système).

La définition du contexte de la simulation va permettre une décomposition d'un système thermique complexe en composants élémentaires, et de relier chaque composant à un module existant dans TRNSYS. Pratiquement, ce lien est établi à l'aide d'une numérotation (n° du type TRNSYS), qui permet de relier le composant identifié par l'utilisateur (ou par l'ESI) au modèle approprié existant dans la bibliothèque TRNSYS. Prenons l'exemple suivant : l'utilisateur veut créer un assemblage qui représente "une installation solaire de production d'eau chaude". Il a le choix entre deux stratégies :

- l'utilisateur décompose son installation ; il a besoin de quatre modules essentiels, qui sont, un capteur solaire (type 1), un régulateur (type 2), une pompe (type 3), et un ballon de stockage (type 4) ; il a aussi besoin de deux utilitaires, qui sont, un lecteur de données (type 9), et un processeur d'ensoleillement (type 16) ;
- l'utilisateur cherche dans la bibliothèque un module représentant une installation solaire de production d'eau chaude, déjà prête à l'utilisation (type 23).

La deuxième question vise les données disponibles (origine, forme, date de réalisation), et les données dont a besoin l'utilisateur pour traiter ses problèmes. Il s'agit pour l'utilisateur d'établir le lien entre les données disponibles et les données nécessaires à la conduite de la simulation. Parfois, établir ce lien n'est pas direct et nécessite le développement de modules spécifiques : par exemple, supposons que l'utilisateur dispose d'un module modélisant le bâtiment en réseau [LARET 1-89], avec une interaction entre le rayonnement extérieur et le bâtiment par le biais d'affectation de puissances thermiques sur des nœuds de surface ; supposons aussi que l'utilisateur dispose de données telles que des irradiations totales et diffuses horizontales ; pour pouvoir transformer ces irradiations en puissances thermiques, il devra alors introduire dans la modélisation de son système, un module solaire qui effectue les calculs des puissances solaires à affecter sur les nœuds capacitifs [EL HASSAR 90].

La troisième question consiste à définir l'objectif de la simulation : il s'agit en fait de définir les variables, appelées quantificateurs, qui permettent d'évaluer si l'objectif de la simulation est atteint ou non ; ces quantificateurs, une fois identifiés, vont guider l'utilisateur dans le choix des modules : ces modules seront choisis, soit parce qu'ils possèdent les quantificateurs comme variables de sortie, soit parce qu'ils possèdent des variables permettant de calculer les quantificateurs. Prenons l'exemple suivant ; l'objectif de l'utilisateur consiste "à vérifier si, dans un bâtiment, la concentration de certains polluants (NO2, CO2, SO2) est acceptable pour les occupants". L'assemblage constituant la solution à ce problème devra être composé d'utilitaires qui calculent le taux de polluant absorbé par un occupant, et de modèles qui traitent du transport des polluants au sein d'un bâtiment. Un ESI doit donc proposer ces utilitaires et ces modèles à l'utilisateur, par exemple en les instanciant de façon automatique dans le panneau d'assemblage.

On voit bien que, dans le cadre d'un système d'aide à la décision, la formulation du contexte de la simulation, d'un objectif de simulation, des données disponibles peut être utile pour automatiser certaines tâches, comme le choix des modèles ou des utilitaires, encore faut-il que l'ESI comprenne ce qu'exprime l'utilisateur. Pour y arriver, le langage utilisé par le système doit se rapprocher le plus possible de celui employé par les utilisateurs ; la difficulté réside dans le fait qu'un outil logiciel est destiné à des catégories socioprofessionnels différentes, qui n'utilisent pas forcément la même terminologie, ni les mêmes unités ; pour contourner ce type de problème, on peut être amené à constituer des "dictionnaires" (thesaurus), établissant des correspondances terminologiques.

Le choix d'une installation

Afin d'opter pour une installation, les bureaux d'études passent par les étapes suivantes :

- vérification dans un premier temps que les installations, solutions possibles du problème posé, sont conformes aux cahier des charges ;
- comparaison des différentes installations, comparaison exprimée en termes de coûts initiaux et de coûts de fonctionnement ; le choix final d'une installation est fonction des résultats de cette comparaison.

La simulation numérique va permettre d'affiner la comparaison entre les différentes installations, elle va surtout permettre de garantir les solutions, notamment en permettant une estimation assez précise des coûts de fonctionnement.

Pour pouvoir exploiter numériquement chaque solution (exprimée en termes d'installation ou de principe d'installation), les tâches suivantes doivent être réalisées :

- traduire les principes d'installation en termes compréhensibles par l'outil de simulation (modules, liaisons entre modules, connexions entre variables, valeurs numériques des paramètres) ;
- compléter l'assemblage pour le rendre opérationnel dans le cadre de l'outil de simulation (introduction des cartes de contrôle, modification de la valeur de certains paramètres, introduction de modules) ; à l'issue de cette étape, le fichier permettant la simulation (fichier DECK pour TRNSYS) est généré ;
- simuler l'assemblage obtenu, en faisant varier les paramètres sensibles, c'est à dire ceux qui ont une importance dans le cadre du projet.

La figure 3-1 résume les différentes phases du processus de conception que nous venons de décrire en détail. Dans le cadre de ce travail, nous nous sommes restreints à proposer de l'aide à l'utilisateur, à partir du moment où il connaît une partie de son assemblage, c'est à dire à partir du moment où il a constitué un macro-modèle (cf. modules "Aide" sur la figure 3-1).

3-2 Automatisation de la conception de systèmes thermiques - Logique des propositions

Afin que l'outil de simulation TRNSYS fonctionne, il faut mettre en place un diagramme de flux, c'est à dire définir les modules à utiliser ainsi que les connexions entre les variables d'entrées et les variables de sorties (cf. § 1-4-1). Dans l'univers TRNSYS, certaines liaisons n'ont pas de sens, comme par exemple lier directement un module "mur" avec un module "régulateur". De plus il n'existe pas de structure de contrôle qui assure à l'utilisateur la pertinence d'une liaison.

Il s'agit de voir dans quelle mesure il est possible d'automatiser l'étape du processus de modélisation/simulation, qui consiste à rechercher les modules les mieux adaptés au problème posé. Cette recherche doit aboutir à la mise en place d'un ou plusieurs systèmes thermiques qui pourront satisfaire les exigences des utilisateurs.

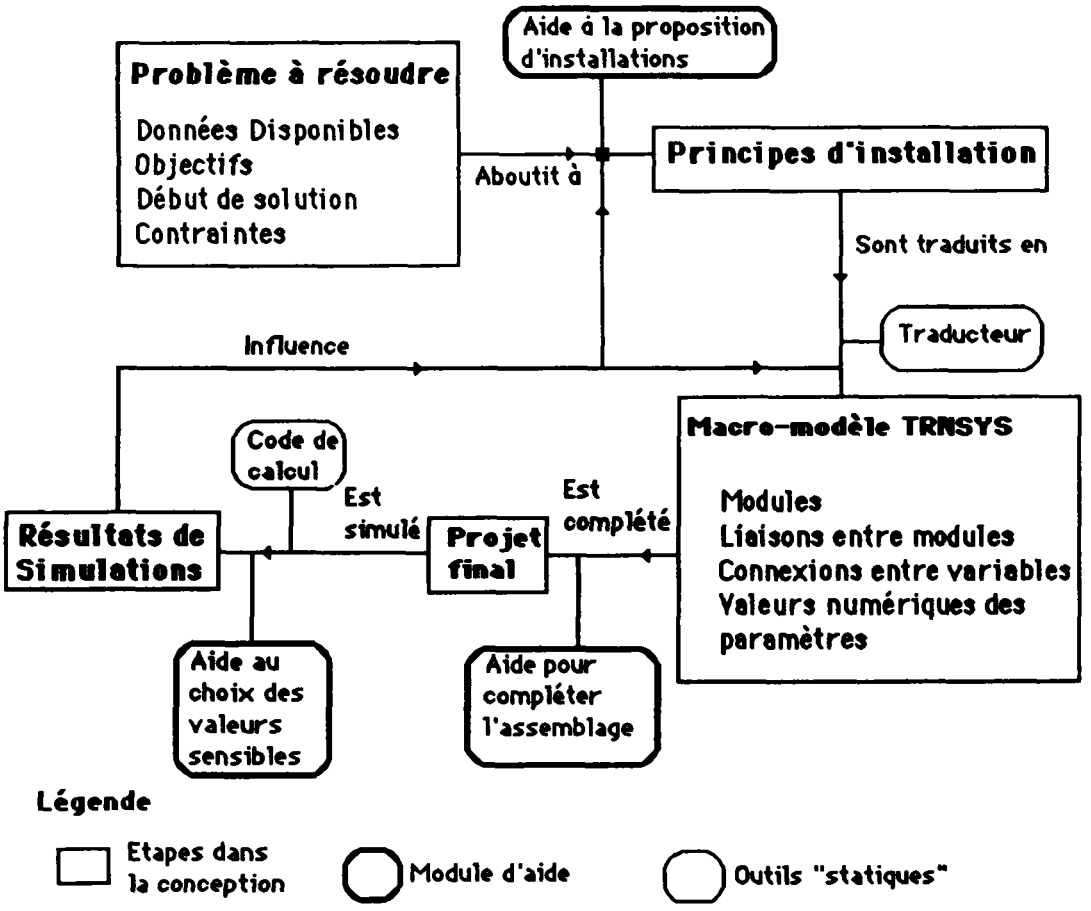


Figure 3-1 : Phases dans la conception d'un projet

Dans le cadre de l'aide à la conception, l'ambition d'IISIBât consiste à automatiser :

- le choix des modules ;
- les liaisons entre modules ;
- les connexions entre variables ;
- l'introduction des valeurs des paramètres.

3-2-1 Logique des propositions

La logique propositionnelle est la composante la plus simple de la logique classique. Les objets de cette logique sont des énoncés susceptibles d'être vrais ou faux. Elle exprime une connaissance générale, proche du sens commun. Le cadre formel de la logique classique permet de manipuler certaines notions cognitives élémentaires, et peut aider à la prise de décision ainsi qu'à l'automatisation des tâches précitées.

L' aspect syntaxique de la logique propositionnelle est composé :

- d'un ensemble dénombrable de symboles, appelés propositions ;
- de cinq symboles : ¬, et, ou, =>, <=>, appelés connecteurs logiques qui sont respectivement négation, conjonction, disjonction, implication et équivalence.

La partie "conditions" d'une règle de production est composée en général d'un ensemble de conditions élémentaires reliées par des connecteurs *et*, *ou*. La partie "conclusions" peut produire plusieurs types d'effet :

- introduction d'un nouveau fait dans la base de faits ;
- émission d'une hypothèse ;
- déclenchement d'une procédure.

Le problème essentiel dans le cadre de la logique propositionnelle concerne la maintenance de la base de connaissances, qui passe par l'insertion de nouvelles règles. Cette remise à jour de la base de connaissances peut engendrer des incohérences :

- risque de répéter une règle existante ;
- risque de redondance, de conflits ;
- prémisses de règles insuffisamment réduites.

Pour résoudre cette difficulté, il s'agit d'associer à la base de connaissances une procédure d'insertion de règles dont l'objectif est de détecter les conflits, et de les signaler au concepteur des règles expertes. Ceci se fait d'autant plus difficilement qu'il est possible qu'une base de connaissances engendre des résultats contradictoires (ce qui est en contradiction avec l'hypothèse du monde clos), suivant l'ordre d'examen des règles, ou suivant l'état de la mémoire de la machine.

La vérification de la non redondance des prémisses, lors de l'introduction d'une règle, peut être une solution : elle consiste à comparer la prémisse de la règle à ajouter avec les prémisses des règles existantes. Le problème se pose en ces termes :

si
Prem1 est la prémisse de la règle à ajouter ("Règle Nouvelle")
et
Prem2 est une prémisse d'une règle de la base de connaissances ("Règle Ancienne")
et
Prem1 est équivalent sémantiquement à Prem2
alors
ne pas ajouter la règle ("Conclusion 1")
ou
ajouter la règle et effacer la règle existante ("Conclusion 2")

Les deux règles ("Règle Nouvelle" et "Règle Ancienne") ne peuvent exister simultanément dans une même base de connaissances (redondance) ; le développeur de la base de connaissances doit opter pour "Conclusion 1" ou "Conclusion 2". Il faut donc définir une relation d'équivalence sémantique entre deux formules : deux prémisses peuvent être logiquement équivalentes, et donc avoir la même sémantique, mais peuvent être syntaxiquement distinctes. Par exemple, la prémisse "*A et B* " est équivalente à la prémisse "*B et A*".

Certaines équivalences logiques sont particulièrement importantes, comme celles qui précisent les propriétés sémantiques des connecteurs :

- (A et B) equiv (B et A) ;
- (A et A) equiv A ;

- (A ou B) equiv (B ou A) ;
- (A ou A) equiv A
- (A et (B ou C)) equiv ((A et B) ou (A et C) ;
- (A ou (B et C)) equiv ((A ou B) et (A ou C) ;
- ¬¬ A equiv A ;
- ¬ (A ou B) equiv (¬A et ¬B).

Il reste à définir la méthode qui permet de juger que deux propositions ont la même sémantique (A equiv B) ; l'aspect syntaxique, dans ce cas, est fondamental.

La comparaison des prémisses est elle aussi insuffisante, elle ne permet pas d'obtenir toujours des bases de règles cohérentes. Illustrons ces propos par l'exemple suivant :

- R1 : A et B => F ;
- R2 : B et C => D et A ;
- R3 : C et D => ¬F.

Si la base de faits initiale contient B et C, on obtient simultanément F et ¬F au cours du raisonnement ; les deux propositions F et ¬F ne pouvant exister simultanément, le raisonnement aboutit à une impasse, bien que les règles R1, R2, et R3 possèdent des prémisses distinctes.

Prise en compte de l'incertitude

Dans le cadre de la logique propositionnelle, une proposition est vraie ou fausse, et donc la notion de probabilité est absente. Dans le cadre d'un outil de simulation, le raisonnement s'appuie fréquemment sur des connaissances incertaines. Cette incertitude a pour origine la modélisation des phénomènes physiques : un modèle est une représentation approchée de la réalité ; pour représenter tous les comportements d'un système technologique, l'utilisateur a le choix entre plusieurs modèles. Le problème est de savoir quel est le modèle qui résout au mieux le problème posé par l'utilisateur, et avec quel degré de confiance.

Le raisonnement à mettre en jeu doit prendre en compte cette incertitude, les objectifs pouvant être multiples :

- aide à la prise de décision ;
- interprétation des résultats de simulation.

Concevoir des mécanismes basés sur le raisonnement *approximatif* oblige à sortir du cadre de la logique mathématique classique. Il s'agit :

- de définir une représentation de l'incertitude ;
- étendre les schémas de raisonnement pour prendre en compte ces nouveaux aspects ;
- propager l'approximation au cours des étapes de raisonnement.

Une première approche consiste à étendre la logique binaire classique en introduisant une représentation numérique de l'approximatif (par exemple le coefficient ß d'une règle de production).

Cette démarche résulte des travaux concernant le système MYCIN (cf. Annexe 4) dans lequel a été mis en place un mécanisme empirique de gestion de *facteurs de certitude*.

3-2-2 Aide au choix de modules

L'idée pour aider l'utilisateur à choisir les modules les mieux adaptés à son objectif de simulation, et à réaliser des liaisons entre modules consiste à répertorier pour chaque modèle ses "modèles amonts" et ses "modèles avals" (la rubrique qui contient l'information concernant les modèles amonts existe dans la version actuelle des fiches Proforma).

Un modèle amont j d'un objet i autorise la liaison de l'objet i avec l'objet j dans le sens (j, i).

Un modèle aval j d'un objet i autorise la liaison de l'objet i avec l'objet j dans le sens (i, j).

Il est possible de constituer une base de connaissances regroupant tous les cas possibles de modèles amonts et avals pour la plupart des objets du monde de TRNSYS. Pour illustrer ces propos, prenons l'exemple de l'objet "capteur solaire" dans TRNSYS.

Dans la plupart des cas (cas 1 sur la figure 3-2), le capteur solaire possède simultanément comme modèle amonts :

- une pompe ; cet objet est nécessaire pour faire circuler le fluide caloporteur. Les deux variables de sortie utilisées comme entrées par le capteur solaire sont la température et le débit massique du fluide à la sortie de la pompe ;
- un lecteur de données ; celui-ci contient en général les données concernant la température ambiante, directement utilisable par le capteur solaire, et les rayonnements ;
- un processeur d'ensoleillement ; les irradiations solaires sont généralement disponibles à l'échelle de l'heure et sur un plan horizontal. Dans la plupart des simulations de TRNSYS, on a besoin d'estimer l'irradiation solaire sur un plan incliné par rapport à l'horizontale, et/ou sur des échelles de temps différentes de l'heure, ce que réalise le processeur d'ensoleillement. Si on dispose de l'irradiation globale sur un plan horizontal, l'irradiation sur un plan incliné est calculée en deux étapes ; d'abord l'irradiation globale horizontale est décomposée en une irradiation directe et une irradiation diffuse horizontales, qui sont ensuite transformées en irradiations directe et diffuse sur la paroi inclinée.

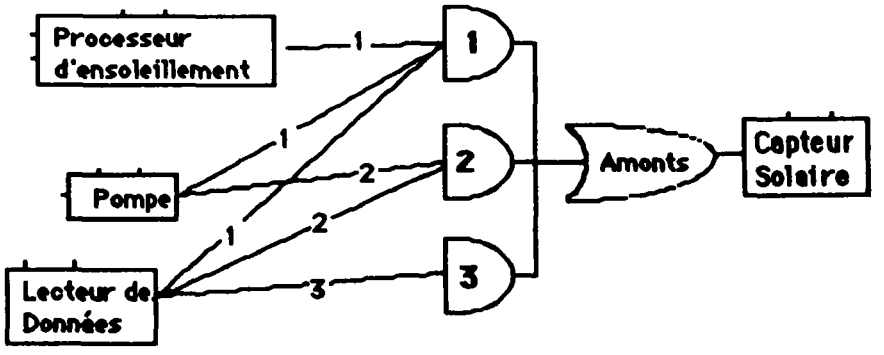


Figure 3-2 : Modèles amonts possibles d'un capteur solaire

Dans le cas où les données générées par un processeur d'ensoleillement ont déjà été calculées par ailleurs (et sont donc stockées dans un fichier), l'utilisation d'un processeur d'ensoleillement ne se justifie plus au niveau de l'assemblage (cas 2 sur la figure 3-2). De plus, si le débit et la température du fluide à l'entrée du capteur sont déclarés constants, l'utilisation d'un module "pompe" (et d'un module "régulateur") ne se justifie plus (cas 3 sur la figure 3-2). Un utilisateur de TRNSYS a très peu de chances de rencontrer ces deux derniers cas (cas 2 et cas 3 sur la figure 3-2).

3-2-2-1 Méthode pas à pas

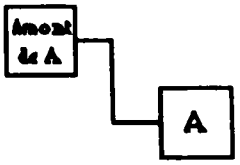
Pour mettre en œuvre concrètement la méthode "pas à pas", nous avons créé un outil spécifique, qui a été rajouté dans la boîte à outils de la fenêtre d'assemblage. La logique d'utilisation de cet outil est la suivante :

- l'utilisateur positionne dans la fenêtre d'assemblage un premier composant ;
- il sélectionne alors l'outil "Modèles amonts/Modèles avals" ;
- il désigne la partie gauche (ou droite) de l'icône déjà instanciée dans la fenêtre d'assemblage ; la partie gauche induit l'ouverture d'une fenêtre qui contient la liste des modèles amonts possibles pour le composant considéré (un clic dans la partie droite induit l'ouverture d'une fenêtre qui contient la liste des modèles avals pour le composant considéré) ;
- l'utilisateur désigne dans la liste des modèles amonts le modèle amont qu'il souhaite voir lier au composant déjà positionné ; cette simple action de désignation du modèle amont induit la fermeture de la fenêtre contenant la liste des modèles amonts, l'instanciation dans la fenêtre d'assemblage du modèle amont sélectionné et la liaison de ce modèle amont avec le composant initial ;
- l'utilisateur dispose alors d'un assemblage de deux composants ; il peut continuer sa démarche pas à pas, soit en recherchant un autre modèle amont, ou un modèle aval, pour le composant initial, ou un modèle amont, ou aval, du nouveau composant instancié.

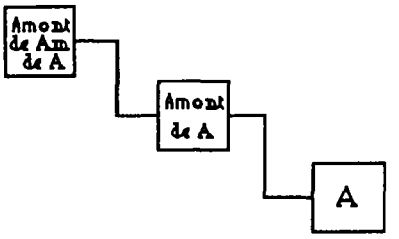
Illustration des cheminements possibles avec la méthode pas à pas

Instanciation du composant initial (composant A) :

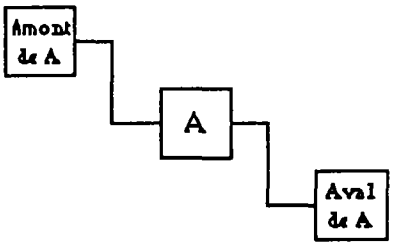
Choix d'un modèle amont pour le composant A :



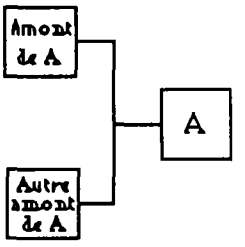
1ère possibilité : choix d'un modèle amont pour le modèle amont de A qui vient d'être instancié :



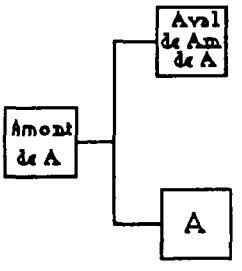
2ème possibilité : choix d'un modèle aval pour A :



3ème possibilité : choix d'un autre modèle amont pour A :



4ème possibilité : choix d'un modèle aval pour le modèle amont de A qui vient d'être instancié :



L'intérêt de la démarche pas à pas réside dans le gain de temps qu'elle procure à l'utilisateur ; dans un processus sans assistance au choix des modèles à l'aide de cette méthode, l'utilisateur aurait dû :

- répondre (mentalement) à la question "quels sont les modèles amonts du composant A ?" ou "ce modèle (en bibliothèque) est-il bien connectable en amont de A ?" ;

- rechercher et sélectionner en bibliothèque le modèle amont de A qu'il souhaite connecter à A (c'est cette phase qui en général demande le plus de temps aux utilisateurs) ;

- positionner le modèle "amont de A" dans la fenêtre d'assemblage ;

- changer d'outil pour réaliser la liaison "A" ↔ "Amont de A"

3-2-2-2 Automatisation de l'approche Amont/Aval

L'automatisation de l'approche amont/aval peut conduire à proposer à l'utilisateur tous les systèmes possibles à partir d'un problème posé par lui. Le raisonnement fondé sur l'automatisation de l'approche amont/aval, ne peut s'effectuer que si l'utilisateur possède un début de solution (en fonction d'un assemblage préexistant dans un panneau d'assemblage par exemple).

L'exploitation de la connaissance des modèles amonts et avals pour chaque modèle permet de réaliser un système à base de connaissances utilisant des règles de production, obéissant à un raisonnement déductif. Ce système expert est basé sur le développement d'une arborescence. Les règles ont la forme suivante :

- **si l'objet i existe dans la base de faits alors ajouter l'objet j, amont de i, dans la base de faits.**

Si l'objet du problème est l'objet i, le moteur d'inférences ajoute dans la mémoire de travail les objets amonts de i qui deviennent objets du problème. Le moteur d'inférence va rechercher les modèles amonts des amonts de i, et ainsi de suite jusqu'à trouver des objets qui n'ont pas de modèles amonts.

Le rôle du moteur d'inférence consiste donc à développer une arborescence, à partir d'une base de faits contenant à l'état initial l'objet du problème, et à proposer comme solution l'ensemble des chemins de l'arbre.

Les règles sont rédigées de façon associative, c'est à dire que chaque règle ignore l'existence effective des autres. On présume seulement que chaque règle introduite dans la base de connaissances comporte des expressions de conditions qui permettront d'évaluer si la règle est applicable. On présume aussi que les faits introduits comme effets d'une règle peuvent appeler d'autres règles, via leurs filtres. La rédaction associative permet d'ajouter et de retirer des règles sans être obligé d'étudier les conséquences des ajouts et des suppressions, ce qui donne au système expert une certaine modularité.

La phase d'évaluation du moteur d'inférence consiste dans l'observation de l'état présent de la base de faits : il s'agit de sélectionner le dernier élément ajouté dans la base de faits et de chercher toutes les règles qu'il est possible de déclencher à partir de cet élément. L'exécution de la règle consiste à ajouter un ou plusieurs objets dans la base de faits. Tous les faits établis ne sont jamais remis en cause. Les critères d'arrêt consistent à ne plus développer un chemin de l'arborescence lorsque :

- un modèle n'a pas de modèles amonts ou avals ;
- deux modèles de même type sont dans la base de faits, ceci afin d'éviter les boucles infinies.

Considérons l'exemple représentée par le tableau 3-1 (modèles amonts).

Problème à traiter	Amonts
A	B, C, D
B	E, F
D	-
E	-
F	-
C	D

Tableau 3-1 : Base de connaissances "Amonts"

Si l'utilisateur veut concevoir un assemblage à partir de l'objet A, il commence par positionner l'objet A dans la fenêtre d'assemblage, puis il doit sélectionner l'outil "génération automatique d'assemblages : méthode amont/aval" et enfin il doit cliquer sur la partie droite ou gauche de l'icône représentant l'objet A. Si l'utilisateur clique sur la partie gauche de l'icône, les assemblages proposés correspondent aux sous-systèmes qu'il est possible de placer en amont de l'objet A, si l'utilisateur clique sur la partie droite de l'icône, les assemblages proposés correspondent aux sous-systèmes qu'il est possible de placer en aval de l'objet A.

La base de faits initiale contiend l'objet A (A).

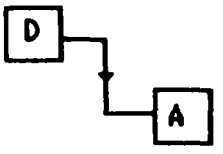
Le déclenchement des règles produit les cycles suivants :

- cycle 1:

  système n° 1 : (A, B)
  système n° 2 : (A, C)
  système n° 3 : (A, D)

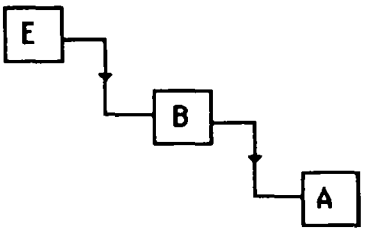
- cycle 2 :

  système n° 11 : (A, B, E)
  système n° 12 : (A, B, F)
  système n° 21 : (A, C, D)
  système n° 31 : (A, D) qui est une solution :

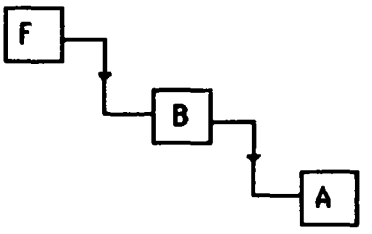


- cycle 3 :

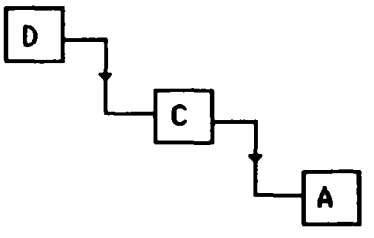
système n° 111 : **(A, B, E)** qui est une solution :



système n° 121 : **(A, B, F)** qui est une solution :



système n° 211 : **(A, C, D)** qui est une solution



Evaluation

D'une façon générale, le nombre de solutions peut être estimé par la formule suivante, K^n, avec :

- K, nombre de directions possibles à un certain niveau ;
- n, nombre de niveaux.

A titre illustratif, supposons qu'à chaque niveau de l'arborescence, il y ait cinq chemins possibles et qu'il y ait environ une dizaine de niveaux avant d'atteindre les extrémités de l'arborescence ; alors $K^n = 5^{10}$ soit environ dix millions de solutions, ce qui représente une terrible explosion combinatoire. En effet la résolution d'un problème aussi modeste, en utilisant un ordinateur rapide qui prendrait comme temps de calcul 25ms par solution, nécessite environ trois jours de temps machine !

Il apparaît donc que le développement d'un outil qui se voudrait général, c'est à dire un outil qui effectuerait une recherche systématique de toutes les solutions, est en pratique impossible (sauf si le nombre de directions à chaque niveau et le nombre de niveaux restent faibles, ce qui est en contradiction avec une base de connaissances évolutive).

De plus proposer à un utilisateur plus d'une dizaine de solutions n'est pas toujours souhaitable.

Pour résoudre ce problème d'explosion combinatoire, une solution consiste à orienter la recherche.

3-2-2-3 Amélioration de l'approche amont-aval : recherche ordonnée

La stratégie de recherche d'une solution peut s'articuler autour des repères suivants :

- construction d'un état unique, et cohérent de l'ensemble des déductions produites par le moteur d'inférence ;
- maintien du réseau de dépendance entre les déductions. Ainsi, lorsqu'une déduction est invalidée, le système expert peut retrouver et supprimer du contexte courant l'ensemble des déductions qui en découlent ;
- mise en place d'un mécanisme de retour arrière, dirigé par l'utilisateur, pour permettre de défaire le raisonnement jusqu'à un point de reprise.

Cette démarche a le mérite de prendre en compte une facette importante du processus de modélisation/simulation, qui est la possibilité de réviser les faits établis. En effet, un concepteur de systèmes physiques doit passer par plusieurs boucles rétroactives pour mettre au point sa solution finale.

La construction d'un état, ou solution, consiste à atteindre le plus rapidement possible une des extrémités de l'arbre. Cette démarche s'appuie sur une recherche en profondeur. Elle s'articule autour des étapes suivantes :

- développement de toutes les alternatives issues d'une étape ;
- choix d'une alternative en fonction d'une relation d'ordre entre les différentes alternatives ; cette relation d'ordre peut être liée par exemple à l'objectif de simulation ;
- repérage de cette étape et de l'alternative choisie pour un retour éventuel.

Exemple

Soit la base de connaissances représentée par le tableau 3-2 :

Problème à traiter	Amonts dans le cadre de l'objectif n°1	Amonts dans le cadre de l'objectif n°2
A	B, C, D et F	B
B	D, C	D, C
C	E	-
D	F, E	-
E	-	-
F	-	-

Tableau 3-2 : Base de connaissances "Amonts-Objectifs"

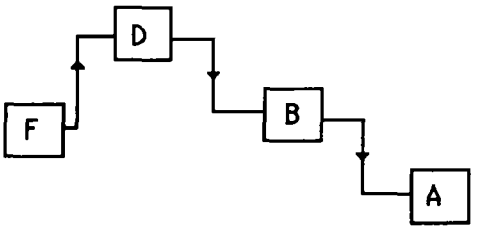
Le tableau 3-2 se lit de la façon suivante :

Le modèle A a
- dans le cadre de l'objectif de simulation n°1

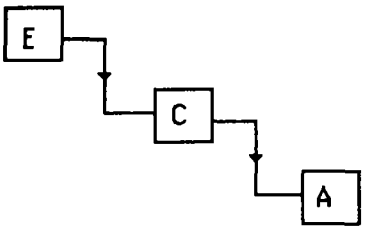
pour modèle amonts
d'abord B
ensuite C
ensuite D et F

- dans le cadre de l'objectif de simulation n°2
  pour modèle amont
  B

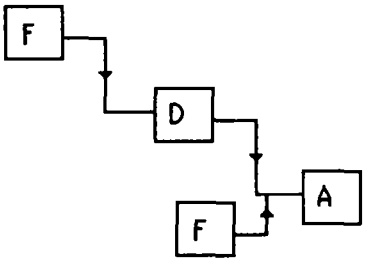
Si l'objet du problème est A, et si l'objectif de la simulation est de type "objectif n° 1", la solution générée par le moteur d'inférence est représentée par la liste : (A, B, D, F).



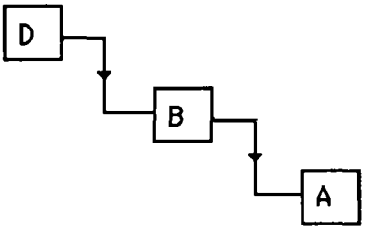
Si l'utilisateur remet en cause B, la nouvelle solution est : (A, C, E).



Si l'utilisateur remet en cause C, la solution générée est : (A, F, (D, F)).



Si l'utilisateur change d'objectif, la solution générée est : (A, B, D).



Evaluation de cette démarche

Les qualités de cette démarche sont nombreuses :

- elle est cohérente avec la démarche itérative d'un concepteur ;
- la base de connaissances est facilement extensible ;
- il est possible de changer la relation d'ordre en fonction de la stratégie de résolution, en introduisant des métarègles ;
- tous les cas possibles de systèmes peuvent être fabriqués automatiquement, tout en évitant l'explosion combinatoire.

Si la relation d'ordre est parfaite, c'est à dire si elle s'appuie sur un travail exhaustif concernant le domaine d'application du code de calcul, le retour arrière sera rarement utilisé (la base de connaissances correspondra aux cas les plus utilisés dans la pratique).

3-2-2-4 Approche globale

Le raisonnement logique présente des avantages importants pour la réalisation d'un système expert. La puissance d'expression de ce formalisme permet notamment de prendre en compte des aspects complexes du processus de modélisation/simulation. Dans le cadre d'un outil de simulation, la prise en compte de ces différents aspects conduit à réaliser des règles qui combinent :

- la connaissance des modèles amonts des modules de TRNSYS ;
- la connaissance du champ d'application des modèles ; celle-ci fait intervenir le mode d'approche d'un modèle (simplifié, détaillé), le type des modèles (équationnel, algorithmique, expérimental, logique ou qualitatif) ;
- la connaissance de l'objectif de simulation ;
- la connaissance des données disponibles exprimée en termes de variables d'entrée des modules (TRNSYS) ou en termes de scénarios (COMIS) ;
- la connaissance de certaines techniques d'optimisation.

Un exemple de rédaction de règles de production combinant propriétés à vérifier et données initiales est représenté par le tableau suivant :

	P1		P2		pas de prop.	pas de donnée
A	d1	d2	d2	d3	B	Message
	B	B , C	C	D		

Tableau 3-3 : Base de connaissances "Approche Globale"

Le tableau 3-3 se lit de la façon suivante : par exemple, si la propriété P1 est vérifiée et si des données de type d1 existent, l'objet A possède comme modèle amont un objet de type B. Dans le cas où des données de type d2 existent, l'objet A possède comme modèle amont d'abord un objet de type B, ensuite un objet de type C. Dans le cas où aucune propriété n'est à vérifier, l'objet A possède comme modèle amont un objet de type B.

Cependant, cette démarche présente des inconvénients majeurs :

- le manque de structuration des connaissances exprimées ;

- il est difficile de fabriquer une base de connaissances qui tienne compte de la plupart des cas, ceci nécessiterait un gros effort en profondeur sur l'outil de simulation ;
- TRNSYS étant sujet à évolution, l'enrichissement de la base de connaissances n'est pas toujours possible, ce qui pose le problème de la maintenance des systèmes de grande taille.

Il est possible de remédier au manque de structuration en associant au raisonnement logique la présence de plusieurs bases de connaissances spécialisées, différentes par le domaine d'application des règles qui sont mises à la disposition de l'utilisateur. Dans ce cas, la phase de conception du moteur d'inférence s'articule tout d'abord autour du choix de la base.

Une structuration possible consiste à regrouper les connaissances par contexte :

- usage de l'habitation, collectif, individuel ;
- type d'énergie utilisée, électrique, gaz, fioul.

Le grand nombre de cas peut conduire à restreindre l'espace de recherche en figeant la base de connaissances. Le problème qui se pose est de répertorier les règles les plus "générales", c'est à dire celles qui sont susceptibles d'être le plus utilisées.

Cette façon de procéder n'est pas toujours souhaitable car cela limite les possibilités de création du concepteur.

Pour réduire l'espace de recherche, une autre solution consiste à associer aux règles logiques un ensemble de prédicats particuliers à vérifier, introduits par l'utilisateur, représentant des contraintes d'utilisation. La recherche d'une solution est alors guidée par des propriétés qui représentent les contraintes que doit vérifier la solution.

Un exemple d'application de vérification de propriétés consiste à construire une solution seulement avec des modèles simplifiés, ou seulement avec des modèles appartenant à une librairie définie par l'utilisateur, ou avec les modèles les plus récents.

La création d'une base de connaissances prenant en compte les aspects complexes du raisonnement constitue le problème principal ; elle nécessite un travail cognitif important, voire impossible, dépassant le cadre de cette thèse.

3-2-2-5 Exemple d'une base de règles pouvant s'appliquer à TRNSYS

Cet exemple n'est donné qu'à titre indicatif, il ne prétend pas automatiser l'étape du choix des modules, passage obligé pour un utilisateur de TRNSYS.

Supposons que l'utilisateur possède une bibliothèque composée des cinq modèles suivants : un lecteur de données, un processeur d'ensoleillement, un capteur solaire, un régulateur et une pompe. Les règles qui permettent d'obtenir une installation solaire de production d'eau chaude (exemple 1 du manuel TRNSYS [TRNSYS 90]) sont illustrées dans la figure 3-3.

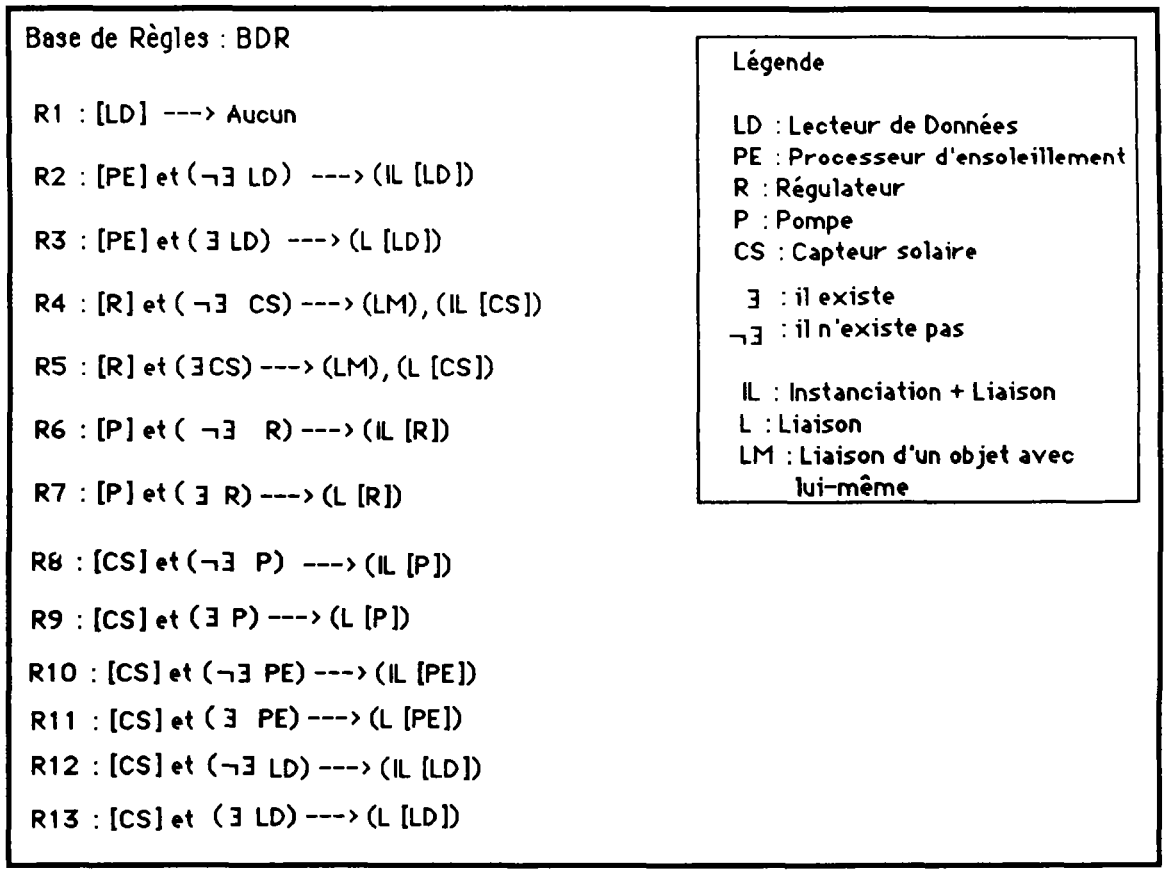


Figure 3-3 : Base de Règles BDR

Les prémisses sont formées d'un ensemble de conditions élémentaires ; entre crochets apparaît le type de l'objet sur lequel s'applique la règle, c'est à dire l'objet dont on recherche les modèles amonts ; les autres conditions vérifient l'existence d'autres types d'objets, dans le panneau d'assemblage.

La partie "conclusions" des règles peut induire les effets suivants :

- instanciation comme amont de l'objet dont le type est défini entre crochets dans la prémisse, d'un module dont le type est défini entre crochets dans la partie "conclusions", et liaison de ces deux modules (effet désigné par "IL") ;
- liaison de l'objet dont le type est défini entre crochets dans la prémisse, avec un module, déjà présent dans le panneau d'assemblage, dont le type est défini entre crochets dans la partie "conclusions" (effet désigné par "L") ;
- liaison de l'objet dont le type est défini entre crochets dans la prémisse avec lui-même.

La règle R2 se lit de la façon suivante : si l'objet du problème est un processeur d'ensoleillement, et s'il n'existe pas dans le panneau d'assemblage de lecteur de données, alors instancier un lecteur de données et le positionner comme amont du processeur d'ensoleillement.

La règle R5 se lit de la façon suivante : si l'objet du problème est un régulateur, et s'il existe dans le panneau d'assemblage un capteur solaire, alors relier les entrées du

régulateur avec ses propres sorties, et positionner le capteur solaire comme amont du régulateur.

Si l'utilisateur démarre avec dans son panneau d'assemblage un capteur solaire, les règles seront déclenchées dans cet ordre : R8, R10, R12 (effet représenté dans la figure 3-4-a), R6 (effet représenté dans la figure 3-4-b), R5 (effet représenté dans la figure 3-4-c), R3 (effet représenté dans la figure 3-4-d), R1 et R1 (pas d'effet induit par le déclenchement de cette règle).

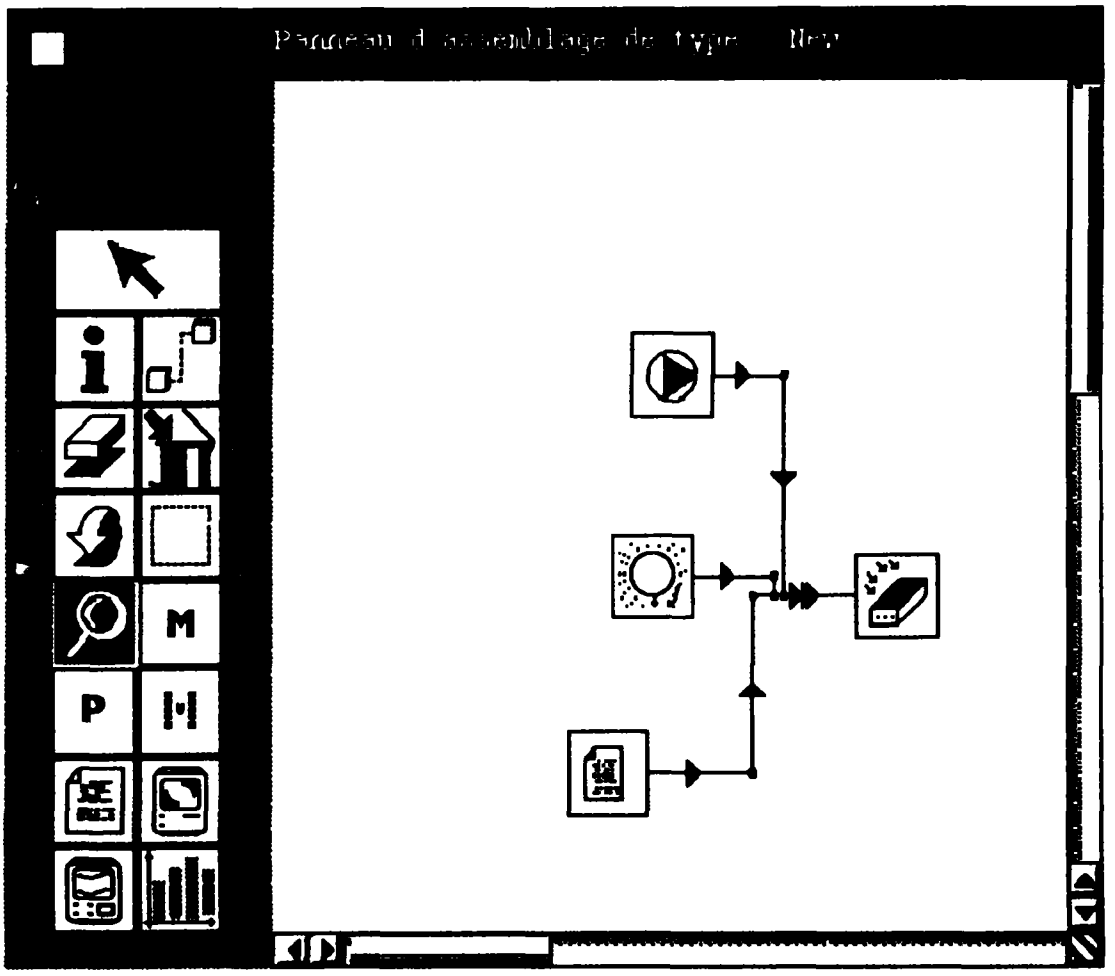


Figure 3-4-a : Effet induit par le déclenchement de R8, R10 et R12

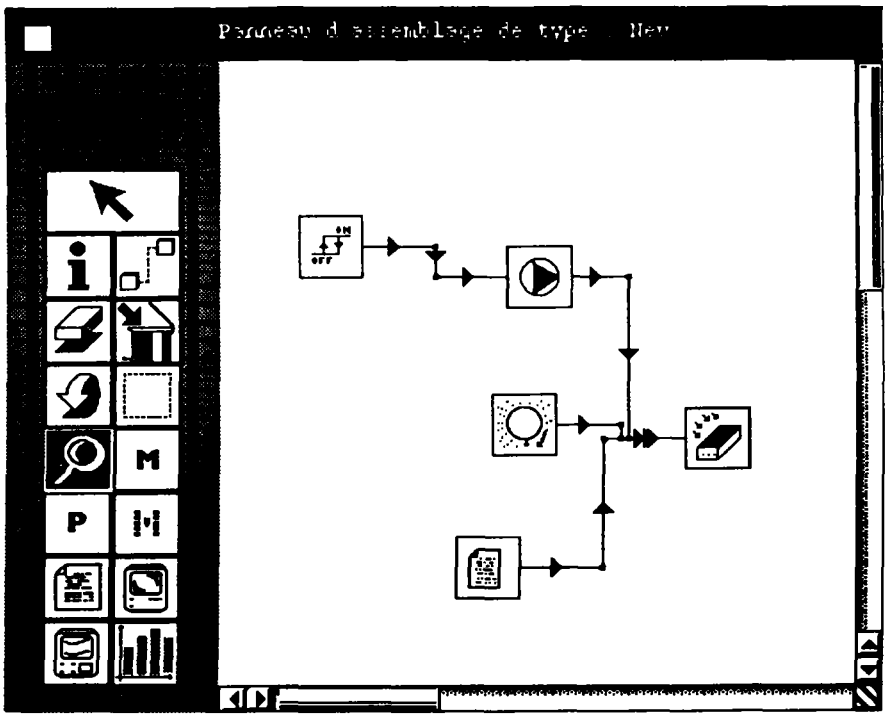


Figure 3-4-b : Effet induit par le déclenchement de R6

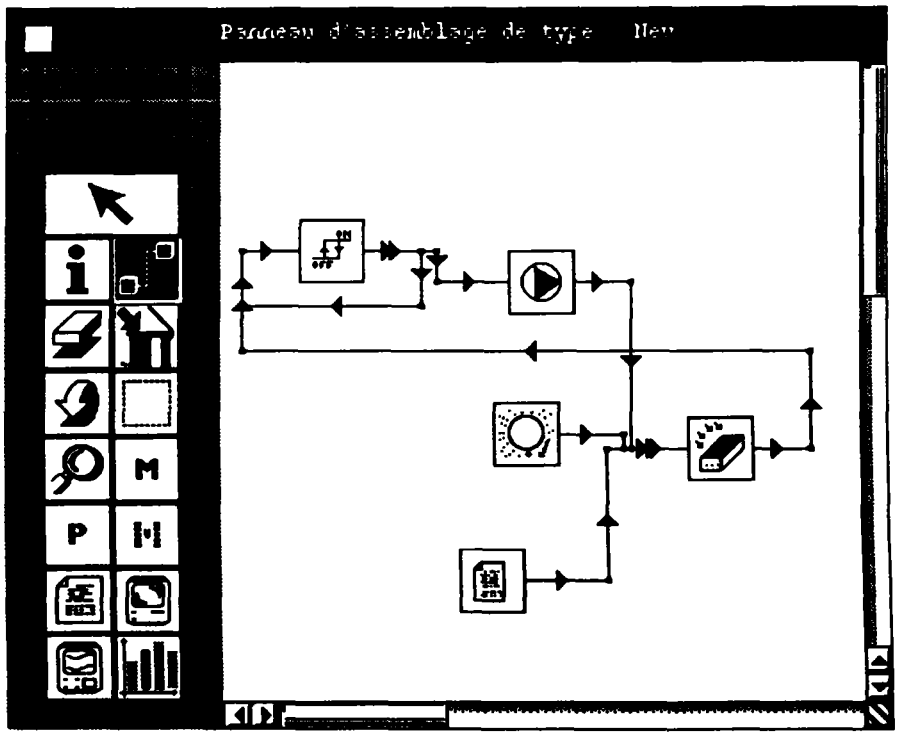


Figure 3-4-c : Effet induit par le déclenchement de R5

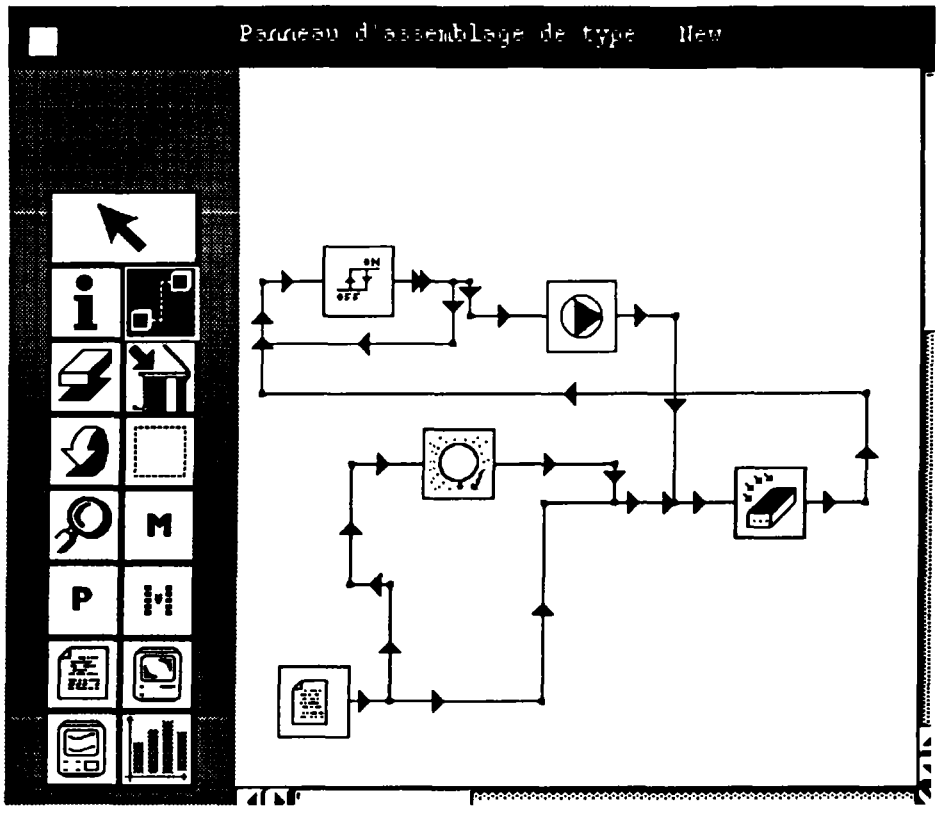


Figure 3-4-d : Effet induit par le déclenchement de R3

Si l'utilisateur ajoute dans sa bibliothèque un objet de type "Ballon de stockage" (BDS), il devra ajouter les règles suivantes :

R14 : [R] et ($\neg\exists$ BDS) ---> (IL [BDS])
R15 : [R] et ($\exists$ BDS) ---> (L [BDS])
R16 : [P] et ($\neg\exists$ BDS) ---> (IL [BDS])
R17 : [P] et ($\exists$ BDS) ---> (L [BDS])
R18 : [BDS] et ($\neg\exists$ CS) ---> (IL [CS])
R19 : [BDS] et ($\exists$ CS) ---> (L [CS])

La manipulation d'une base de règles comme BDR nécessite une collaboration entre l'utilisateur et le système. En effet, le système ne peut appliquer la règle R5 seul ; si par exemple il existe dans le panneau d'assemblage plusieurs capteurs solaires, l'utilisateur devra dialoguer avec le système pour lui indiquer lequel des capteurs solaires est amont du régulateur ; il faut alors ajouter une règle qui a la forme suivante : "s'il existe plusieurs solutions possibles, alors dialoguer avec l'utilisateur".

3-2-3 Automatisation des connexions

La deuxième phase dans l'utilisation de TRNSYS, après avoir choisi les composants du système à simuler, consiste à connecter les variables d'entrée et de sortie des modules reliés. Cette phase est beaucoup plus délicate que la précédente. En effet, la fabrication d'un système fait intervenir un nombre fini de types d'objets physiques connus par les

utilisateurs, comme par exemple : pompes, régulateurs, vannes, chaudières, tuyaux, zones.

Par contre, les utilisateurs ont toujours du mal à connecter les variables et à affecter à chaque connexion une valeur initiale. En moyenne, un assemblage fait intervenir une dizaine de composants, chaque composant possède environ de vingt à trente variables d'entrée et de sortie ; l'utilisateur doit donc réaliser les "bonnes" connexions en manipulant entre deux cents et trois cents variables ! L'automatisation des connexions va l'affranchir de cette tâche, l'utilisateur économise :

- le temps utilisé à résoudre la question "quelles sont les variables qu'il faut connecter ?" ;
- le temps consacré à la réalisation graphique des connexions ;
- le temps consacré à résoudre la question "quelles sont les valeurs initiales à affecter à chaque connexion ?".

L'automatisation des connexions peut s'effectuer :

- par le test sur les unités des variables ;
- par la fabrication d'une base de connaissances.

3-2-3-1 Test sur les unités

Les fiches Proforma donnent toutes les informations concernant les unités des variables. L'exploitation de ces données permet d'automatiser dans une certaine mesure l'instanciation des connexions, cette approche consiste à tester les unités des différentes variables et de connecter celles qui possèdent la même unité.

Exemple

Soient les variables d'entrée et de sortie représentées par le tableau suivant :

Variables de sortie	Unité	Variables d'entrée	Unité
As	i	Ae	i
Bs	i	Be	k
Ds	i	Ce	i
Es	k	De	i

Tableau 3-4 : Base de connaissances "Variable-Unités"

Le test sur les unités produira par exemple les connexions suivantes : (As, Ae), (Bs, De), (Ds, Ce), (Es, Be).

Evaluation

Cette configuration n'est pas la plus judicieuse ; en effet par cette méthode il est possible de connecter une température d'eau avec une température d'air. Dans la pratique, cette démarche conduit dans la plupart des cas à une configuration fausse. Par ailleurs, si le moteur d'inférence doit retrouver toutes les solutions, on retrouve le problème de l'explosion combinatoire.

Cependant, dans le cadre d'une connexion manuelle, cette approche permet de réaliser des tests de vérification : on ne peut pas connecter des variables dont les unités sont différentes.

3-2-3-2 Approche systématique

Rappelons qu'une liaison lie dans un panneau d'assemblage deux modèles ; l'un de ces modèles sera appelé "modèle amont", l'autre "modèle aval".

L'approche systématique consiste à répertorier dans une base de connaissances toutes les connexions de variables qu'il est possible de réaliser au sein du code de calcul. Le raisonnement reproduit par le moteur d'inférence (MI) est le suivant :

- la base de faits initiale contient la liste (LF) de toutes les liaisons qui existent au niveau du panneau d'assemblage ;

- le MI choisit la première liaison (LAT) présente dans la liste LF, et recherche une règle applicable ; cette recherche consiste, dans un premier temps, à observer les noms des modèles "amont "et "aval" de la liaison LAT ; dans un deuxième temps, il s'agit pour le MI de parcourir la base de règles, pour déterminer s'il existe une (ou des) règle (s) à déclencher, c'est à dire s'il existe une règle qui contient dans sa prémisse les mêmes modèles "amont" et "avals" que ceux de la liaison LAT ;

- s'il existe une règle applicable (RA), les connexions définies dans la partie *conclusion* de la règle RA sont instanciées ;

- enfin, la base de faits est reinitialisée, la liaison LAT est enlevée de la liste LF.

Les règles produites ont la forme générale suivante :

Si la liaison qui est l'objet du problème a
- **comme modèle amont un objet de nom i ;**
- **comme modèle aval un objet de nom j ;**

Alors connecter

les variables s^i_k e^j_l avec pour valeur initiale vi^{ij}_{kl}

Remarque : dans le cadre du prototype réalisé, les règles se présentent sous la forme suivante.

```
R1 :   ( (si (
       (liaison? '(solar_radiat solar_collec) a-traiter)
                alors
                (
                        ( (i ()) (i ()) 0)
                        ( (id ()) (id ()) 0)
                        ( (it1 ()) (ray-dir ()) 0)
                        ( (teta 1) (teta ()) 0)
                        ( (beta 1) (beta ()) 40)
                )
       )
       ))
```

La règle R1 se lit comme suit : si la liaison à traiter lie un module de nom "solar_radiat" (processeur d'ensoleillement) et un module de nom "solar_collec" (capteur solaire), alors connecter :

- la variable de syntaxe "i" de "solar_radiat" avec la variable de syntaxe "i" de "solar_collec" avec pour valeur initiale 0 ;
- la variable de syntaxe "id" de "solar_radiat" avec la variable de syntaxe "id" de "solar_collec" avec pour valeur initiale 0 ;
- la variable de syntaxe "it1" de "solar_radiat" avec la variable de syntaxe "ray-dir" de "solar_collec" avec pour valeur initiale 0 ;
- la variable de syntaxe "teta1" de "solar_radiat" avec la variable de syntaxe "teta" de "solar_collec" avec pour valeur initiale 0 ;
- la variable de syntaxe "beta1" de "solar_radiat" avec la variable de syntaxe "beta" de "solar_collec" avec pour valeur initiale 40.

Evaluation

Il existe pour la plupart des associations "module-module" de TRNSYS une seule façon de connecter des variables dans une liaison. Ainsi, cette approche systématique permet d'être pratiquement sûr du traitement d'une liaison. Les règles de connexions sont totalement indépendantes les unes des autres. L'inconvénient reste dans la difficulté de répertorier dans une base de connaissances l'ensemble des cas. En particulier, introduire un nouveau modèle nécessiterait l'écriture de plusieurs dizaines de nouvelles règles dans la base de connaissances, de façon à ce que la fonction "Connexions Automatiques" puisse fonctionner (si la base de connaissances en question est l'ensemble des fiches Proforma des modules, il faudrait alors créer une rubrique qui permettrait d'insérer les différentes façons de connecter les variables pour un module).

Une solution consiste à utiliser les propriétés d'héritage des langages orientés objets. Illustrons ces propos par un exemple ; supposons que l'utilisateur crée un nouveau modèle de nom "solar-collec-CSTB", et qu'il l'insère dans la bibliothèque de composants, dans la classe où se trouve l'objet "solar-collec". Si le modèle "solar-collec-CSTB" possède des variables d'entrée de syntaxe "i", "id", "teta", "beta", "ray-dir", les règles concernant la connexion de ces variables avec les variables de sortie de l'objet "solar_radiat" n'ont pas à être réécrites par l'utilisateur.

Les règles produites auraient la forme générale suivante :

Si la liaison qui est l'objet du problème a :

- **comme modèle amont un objet**
  faisant partie de la classe i
  ou
  faisant partie d'une sous-classe de i;

- **comme modèle aval un objet**
  faisant partie de la classe j
  ou
  faisant partie d'une sous-classe de j;

Alors connecter

les variables s^i_k e^j_l avec pour valeur initiale vi^{ij}_{kl}

L'introduction d'un nouveau modèle comme sous-classe d'un modèle existant évite l'écriture d'un trop grand nombre de nouvelles règles de production, du fait de l'héritage ; il n'en demeure pas moins que des règles relatives aux variables spécifiques du nouveau modèle devront être écrites.

3-2-3-3 Approche par les variables

Cette approche consiste à réaliser la fonction "Connexions Automatiques" en appuyant le raisonnement sur la manipulation des variables. Il s'agit, d'une part de répertorier dans un dictionnaire des variables, l'ensemble des familles de variables utilisées au sein du code de calcul (température d'air, température d'eau, flux thermique, énergie, puissance,...), et d'autre part de définir une correspondance entre les familles de variables répertoriées, correspondance qui doit exprimer les possibilités de connexion des variables.

Il faut tout d'abord définir "l'objet variable" ; celui-ci doit posséder les trois champs suivants : le nom abrégé, le nom complet, et le nom type. Le nom abrégé est celui qui apparaît à l'écran, il correspond au nom que les utilisateurs assidus du code de calcul ont l'habitude de manipuler (par exemple, Ti pour température d'entrée). Le nom complet doit mettre en valeur les différentes facettes de la variable. Le nom type désigne la famille de la variable, et doit obligatoirement appartenir au dictionnaire des variables (par exemple, la variable "débit d'eau à la sortie du capteur solaire" appartient à la famille "débit d'eau").

La connexion des variables s'effectue de la façon suivante : les variables qui appartiennent à la même famille ou dont les noms types sont mis en correspondance seront connectées.

Evaluation

L'avantage dans cette approche réside dans le fait que lorsque un utilisateur ajoute un nouveau modèle dans la bibliothèque de composants, il n'a rien à définir pour ce qui concerne la fonction "Connexion Automatique", si ce n'est les noms type des variables. Dans ce cadre, l'interface devra proposer préférentiellement au créateur de nouveaux modèles les noms types, déjà répertoriés au niveau du dictionnaire des variables.

3-2-2-4 Evaluation des approches qui permettent la connexion automatique des variables

Aucune des solutions proposées (cf. § 3-2-3-2, cf. § 3-2-3-3) ne permet de résoudre correctement le problème des connexions automatiques ; si on veut tout automatiser, il faut plus d'informations pour pouvoir reproduire le raisonnement de l'expert lorsqu'il réalise manuellement les connexions.

Supposons qu'il existe plusieurs variables de sortie d'un même module, qui appartiennent à la même famille (plusieurs débits d'eau, plusieurs températures par exemple) ; il est alors impossible à partir d'un dictionnaire des variables, de choisir parmi ces variables de sortie laquelle doit être connectée à une variable d'entrée, appartenant à la même famille que celle des variables de sortie (voir figure 3-5).

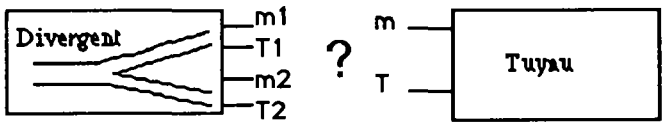


Figure 3-5 : Connexion Diverteur-Tuyau

Les variables "m1" et "m2" du divergent et la variable "m" du tuyau appartiennent à la même famille ("Débit d'eau") ; laquelle des variables "m1" ou de "m2" doit être connectée à "m" ? Seule une règle qui manipulerait une information donnant la position du tuyau par rapport au divergent dans le réseau hydraulique permettrait une connexion automatique.

Les ambiguïtés peuvent être levées grâce à des informations supplémentaires concernant l'ensemble du système et non pas seulement les modèles amonts de la liaison considérée. Les informations pourront effectivement figurer dans une base de faits dès lors que les outils de CAO qui utilisent une sémantique riche [POYET 2-92] seront opérationnels (c'est à dire connectés aux ESI).

L'automatisation complète des connexions fera l'objet de travaux ultérieurs.

3-2-4 Aide au choix de valeurs numériques

Le problème à résoudre dans un outil numérique de simulation reste le grand nombre d'informations quantitatives détaillées à introduire et à manipuler lors des différentes phases de résolution.

L'étape dans le processus de conception d'un système qui consiste à choisir les valeurs numériques est un travail courant dans les bureaux d'études, elle correspond au travail de dimensionnement. C'est une étape délicate : en effet, l'utilisateur pour pouvoir simuler, doit introduire en moyenne jusqu'à cent cinquante valeurs numériques par session. Il paraît difficile pour un utilisateur de mémoriser une telle quantité de valeurs.

Ces valeurs numériques doivent permettre de répondre aux spécifications d'un cahier des charges, exprimées en termes d'objectifs à atteindre et de contraintes techniques. Dans un premier temps, l'utilisateur aboutit à la proposition de plusieurs solutions techniques, qui possèdent le plus de chances de répondre aux exigences imposées par le cahier des charges. Ces solutions sont ensuite optimisées, notamment en manipulant des données numériques ; ces solutions sont ensuite comparées pour choisir parmi l'une d'entre elles, la "meilleure solution". Cette comparaison s'effectue en général sur la base de critères financiers : évaluation des coût initial et coût de fonctionnement (cf. § 3-1). Un ESI doit permettre à l'utilisateur d'effectuer ce travail d'optimisation.

Le choix des valeurs numériques s'apparente à un processus itératif : il s'agit de générer les valeurs numériques de départ, de lancer des simulations, d'analyser les résultats, de remettre en cause les valeurs de certains paramètres, de relancer des simulations, et ainsi de suite jusqu'à trouver la bonne combinaison des valeurs numériques.

Les informations qui permettent à l'utilisateur de remettre en cause certaines valeurs numériques sont soit contenues dans les Fiches Proforma, soit peuvent être déduites en fonction du contexte (si on prend l'hypothèse du monde clos, les températures d'air dans le bâtiment prennent des valeurs comprises dans l'intervalle [-50 °C, 80°C] ; si le résultat d'une simulation donne une température d'air de l'ordre de 200°C, l'anomalie doit être signalée). L'exploitation d'informations telles que les domaines de validité et les valeurs interdites, issues des Fiches Proforma, permet le dialogue entre l'ESI et l'utilisateur (détection d'erreurs éventuelles ou de certaines anomalies).

L'utilisateur peut à chaque fois remettre en cause :

- l'utilisation d'un modèle ;
- la valeur d'un paramètre ;
- la valeur initiale d'une entrée ;
- la valeur d'une carte de contrôle.

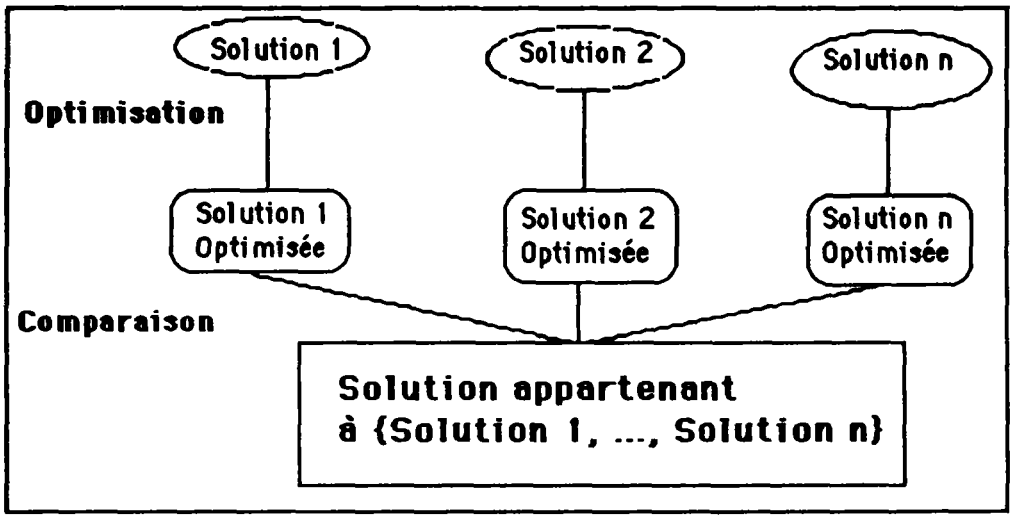


Figure 3-6 : Processus d'optimisation

Pour pouvoir générer les valeurs numériques de départ, la solution consiste à associer à chaque paramètre un nombre réduit de valeurs prédéfinies.

Celles-ci sont issues des Fiches Proforma, ce sont les valeurs par défaut. Elles sont choisies dans des intervalles (domaine de validité) et sont généralement significatives d'un comportement. Elles sont choisies de façon à pouvoir aider l'utilisateur à lever des ambiguïtés dans son assemblage. Pour cela, les valeurs par défaut peuvent correspondre à différents ordres de grandeur. En général, les valeurs par défaut sont liées soit au modèle lui-même, soit au contexte de son usage.

Les valeurs numériques peuvent être aussi issues d'un modèle de données intégré [DUBOIS 92], [SOUBRA 92].

Dans le cadre d'IISIBât, les valeurs des paramètres de chaque module instancié sont, à l'état initial, celles des valeurs par défaut.

Cependant, quelle que soit la méthode utilisée pour engendrer la combinaison des valeurs numériques de départ (Proforma, Modèle de Données Intégré), dans la plupart des cas

celles-ci ne permettent de répondre que partiellement au cahier des charges : elles doivent toujours être affinées, d'où le caractère itératif du processus de conception. Si certaines des valeurs numériques ne sont jamais remises en cause, car faisant partie des contraintes techniques imposées par le cahier des charges (en général, il s'agit de données relatives à la situation géographique, aux propriétés thermo-physiques des matériaux), d'autres par contre constituent des problèmes à résoudre pour le concepteur. Ces données touchent en général aux volumes mis en jeu, et à l'énergie nécessaire au fonctionnement de l'équipement du bâtiment (puissance, rendement, température, énergie utile, puissance maximale de chauffage).

Une manipulation des données numériques classique revient à effectuer un grand nombre d'études : il s'agit de faire varier certains paramètres du système, jugés sensibles soit par l'utilisateur soit par l'ESI, et de lancer des simulations. Prenons un exemple simple ; supposons que le concepteur détermine deux paramètres sensibles pour son projet, P1 et P2. La deuxième étape consiste à discrétiser les intervalles des valeurs possibles pour P1 et P2. Supposons que P1 appartienne à l'intervalle [10, 100], et que P2 appartienne à l'intervalle [18,22] ; supposons d'autre part que l'utilisateur impose pour P1 un pas d'investigation de 10, et pour P2 un pas d'investigation de 1. Pour tester tous les cas possibles, l'utilisateur doit effectuer 50 simulations numériques. On imagine facilement la difficulté de mise en œuvre de ce type d'approche, pour un système complexe.

Il serait possible de diminuer le nombre total de simulations (N) à effectuer en faisant appel à une logique probabiliste [FURBRINGER 92] ; il faudrait pour cela connaître le nombre minimal de simulations à réaliser ($n << N$), afin d'être "pratiquement sûr" (résultat assorti d'un coefficient de certitude de 0.8) de trouver la bonne combinaison des valeurs numériques. Nous n'avons pas approfondi cette approche dans le cadre de ce travail, mais elle nous paraît essentielle à poursuivre. Par ailleurs, pour pouvoir effectuer un travail d'optimisation rapide, une approche fondée sur le raisonnement qualitatif physique peut être une solution.

3-2-4-1 Le raisonnement qualitatif

L'intérêt du raisonnement qualitatif est multiple :

- la perception par l'homme de l'évolution des paramètres régissant un système est plutôt de nature qualitative, même si ces paramètres sont souvent définis quantitativement. Ceci facilite le dialogue entre l'expert (ou le système expert) et l'utilisateur ;

- dans la résolution de problèmes, on se heurte souvent à un manque de données quantitatives. Par ailleurs, le traitement de telles données est parfois couteux, comme la résolution de systèmes d'équations aux dérivées partielles ;

- la construction des modèles quantitatifs complets n'est pas toujours possible. En effet, les relations exactes reliant différents paramètres ne peuvent pas toutes être formulées mathématiquement. La représentation qualitative permet l'expression explicite des influences respectives des paramètres les uns sur les autres. De plus, elle a l'avantage d'être concise.

Les objectifs peuvent être :

- prédire les comportements des systèmes physiques en termes qualitatifs ;

- fournir des modèles de fonctionnement destinés à être intégrés dans des architectures existantes de systèmes experts ;

- diagnostiquer un système, c'est à dire expliquer les comportements anormaux d'un système.

Il existe trois approches pour le raisonnement qualitatif [JOUVENEL 91] :

- l'approche centrée contrainte [KUIPERS 86]. Elle revient à considérer un certain nombre de contraintes sur les paramètres ;

- l'approche centrée composant [DE KLEER et BROWN 84]. Elle s'appuie sur la notion de composant. dans cette approche un système physique est décrit à l'aide de ses composants, chacun d'eux étant soumis à des lois physiques ;

- l'approche centrée processus [FORBUS 84]. Elle consiste à identifier tous les processus susceptibles d'intervenir. Cette approche est construite à partir de la précédente, en modélisant non seulement les composants, mais aussi les processus agissant sur eux.

Dans le cadre de notre étude, nous nous intéresserons plus particulièrement à l'approche centrée composant.

Dans cette approche, le comportement global d'un système est dérivé de la composition des comportements de chaque composant du système. La modélisation se fonde sur la notion d'équation différentielle qualitative (cf. Annexe 6), elle consiste à représenter un système physique par un ensemble d'équations qualitatives. Un système est décrit comme un ensemble de composants physiques disjoints et de connexions entre eux. Il est modélisé :

- par un ensemble de paramètres représentant les grandeurs physiques de ce système ;
- par un ensemble de relations entre ces paramètres.

Chaque paramètre est représenté par son état qualitatif, caractérisé par le couple valeurs, tendance.

Les valeurs d'un paramètre sont prises dans l'espace des quantités du paramètre et sont en nombre fini (valeurs limites).

Par exemple, les valeurs qui ont un intérêt pour caractériser la température d'eau sont celles qui correspondent aux différents changements d'état de ce fluide :

$$\text{espace des quantités} = \{ 0 ; (0 ; 100) ; 100\}$$

La tendance d'un paramètre est soit :

- décroissante ;
- stable ;
- croissante ;
- indéterminée.

3-2-4-2 Intérêt du raisonnement qualitatif dans le cadre du choix des valeurs numériques

Le raisonnement qualitatif va permettre de décrire qualitativement le comportement physique d'un système, on parle alors de simulation qualitative.

La simulation qualitative consiste à partir d'un système à un instant donné, à décrire tous les comportements possibles à partir de cette situation. Les étapes à suivre sont les suivantes :

- on part d'un système à l'équilibre ;
- on perturbe cet équilibre en modifiant la tendance d'un paramètre ;
- on observe l'effet de cette perturbation après que le système ait atteint un nouvel équilibre.

Expliquons cette notion à travers un exemple : le régulateur de débit. Cet exemple est classique du fait de l'importance de la régulation dans les systèmes physiques [HATON 91].

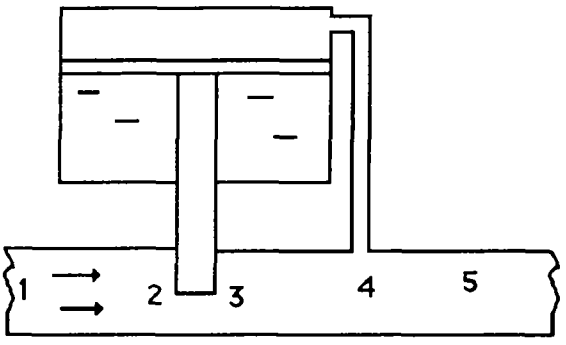


Figure 3-7 : Régulateur de débit

La fonction de ce régulateur est de maintenir constant le débit d'un fluide circulant dans le conduit de 1 vers 5 lorsque les paramètres d'entrée et internes au dispositif varient, en l'occurrence dans ce cas la pression à l'entrée.

Un raisonnement purement qualitatif permet de donner une explication du mode de fonctionnement du dispositif. Une augmentation de la pression d'entrée en 1 fait augmenter la pression au niveau de la vanne en 2. Puisque le flux du liquide varie dans le même sens que la pression, le flux à travers la vanne (2 vers 3) augmente également. Cette augmentation du flux entraîne celle de la pression sur le piston, ce qui provoque la descente du piston, réduisant ainsi la section au niveau de la vanne, ce qui a pour effet de réguler le débit.

S'il est possible de formaliser ce raisonnement qualitatif (dans le cadre d'un système expert à base de connaissances), l'utilisateur non expert pourra, sans effectuer de simulations numériques, avoir une idée du fonctionnement du régulateur, et en particulier comprendre comment varie le débit en fonction de l'évolution de la pression en entrée.

Les règles fondées sur le raisonnement qualitatif peuvent être utilisées dans une stratégie de recherche de solutions.Ces règles permettent :

- de prévoir le comportement du système sans effectuer de simulation numérique ;
- d'orienter la recherche des variables à faire varier (analyse de sensibilité) ;
- d'indiquer à l'utilisateur dans quel sens (augmentation, diminution) il doit faire varier les valeurs de certaines variables, en fonction de ce qu'il veut obtenir.

3-2-4-3 Application possible de la physique qualitative à IISIBât

Considérons l'exemple suivant ; soit le tableau suivant (tableau 3-5) représentant une base de connaissances :

	Variable d'entrée	Variable de sortie
Objet A	Q ↑	P ↓
Objet A	Q ↓	P ↑
Objet A	F ↑	P ↑
Objet A	F ↓	P ↓
Objet B	P ↑	H ↑, K ?
Objet B	P ↓	H ↓, K ?

Tableau 3-5 : Base de connaissances "Qualitative"

Le tableau 3-5 se lit de la façon suivante :

Si la variable Q de l'objet A augmente,
alors la variable P de l'objet A diminue.

Si la variable P de l'objet B augmente,
alors la variable H de B augmente
on ne peut rien dire sur l'évolution de la variable K.

Si les variables P des objets A et B sont connectées (figure 3-8), on peut prédire, en termes qualitatifs, l'influence de l'évolution de la variable Q sur le système constitué par les objets A et B. La tâche du moteur d'inférence manipulant cette base de connaissances consiste à propager une perturbation à tout le système.

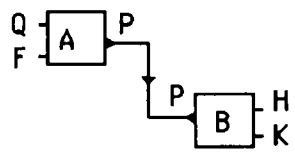


Figure 3-8 : Exemple d'application de la physique qualitative

Supposons que l'utilisateur observe la variable de sortie H de l'objet B, et veut diminuer sa valeur. Si la perturbation consiste à augmenter la valeur de la variable Q de l'objet A, le moteur d'inférence produit les résultats suivants :

- si la variable Q de l'objet A augmente alors :

  la variable P de l'objet A diminue ;
  la variable P de l'objet B diminue ;
  la variable H de l'objet B diminue, objectif recherché par l'utilisateur ;
  on ne peut rien dire sur l'évolution de la variable K.

Si la perturbation consiste à diminuer la valeur de la variable F de l'objet A, le moteur d'inférence produit les résultats suivants :

- si la variable F de l'objet A diminue alors :

  la variable P de l'objet A diminue ;
  la variable P de l'objet B diminue ;
  la variable H de l'objet B diminue, objectif recherché par l'utilisateur ;
  on ne peut rien dire sur l'évolution de la variable K.

Evaluation

Le formalisme utilisé ne tient pas compte des variations importantes de certains paramètres, et de l'influence de ces variations sur le comportement d'un système. Dans l'exemple précédent, il est impossible de dire de quelle manière (diminution faible, sensible, forte) la variable H évolue. De plus, il est impossible de superposer les effets des perturbations : on ne peut rien dire sur l'évolution de la variable H de l'objet B, si l'utilisateur augmente simultanément les variables Q et F de l'objet A. Une solution consisterait à effectuer le raisonnement qualitatif sur les ordres de grandeur.

Des mécanismes de raisonnement fondés sur le raisonnement qualitatif ne pourront avoir lieu que si des informations, du même type que ceux présentés dans le tableau 3-5, sont introduites dans les Fiches Proforma.

La construction de systèmes à base de connaissances à partir d'informations qualitatives reste un objectif du futur. La modélisation qualitative propose un bon support de la connaissance profonde sur laquelle pourront se fonder les systèmes d'intelligence artificielle à venir.

3-2-5 Compromis réalisé

Nous avons vu dans ce chapitre qu'il était possible d'automatiser, au moins partiellement, les différentes étapes nécessaires à la construction d'un assemblage :

- fabrication automatique des liaisons entre modules grâce à la connaissance des modèles amonts et avals d'un module de TRNSYS ;
- fabrication automatique des connexions dans les liaisons grâce à une base de connaissances appropriée ;
- introduction automatique des valeurs numériques dans l'assemblage grâce aux valeurs par défaut ;
- aide à l'interprétation des résultats grâce à l'exploitation d'informations telles que les domaines de validité.

Toutes les connaissances utiles à l'automatisation sont disponibles (ou peuvent l'être) dans les fiches Proforma, qui constituent donc une structure de données intéressante.

Dans la version actuelle des fiches Proforma, il n'existe pas de rubriques relatives à la connexion des variables ; il faut donc la prévoir (si on adopte la solution préconisée au § 3-2-3-2) pour pouvoir utiliser les mécanismes d'aide : il faut pour chaque variable d'entrée d'un modèle définir une liste de variables de sortie faisant partie de la définition d'autres modèles, connectables à cette variable d'entrée.

Cependant, il fallait trouver un compromis face au très grand nombre d'informations manipulées, et face au très grand nombre de cas qu'il est possible de rencontrer. Ce compromis doit tenir compte du fait que la bibliothèque de modèles de TRNSYS est extensible et que par conséquent les connaissances manipulées par l'utilisateur sont sujettes à évolution.

Construire des règles qui tiennent compte de la complexité du raisonnement, qui met en jeu la connaissance sur les modèles et la connaissance des phénomènes physiques, conduit à manipuler des bases de connaissances difficiles à gérer.

Construire des règles basées sur des mécanismes de raisonnement simples, permet d'assurer la maintenance du système expert de résolution.

Il s'agit de trouver un optimum entre une approche complexe, et une approche simplifiée. Un compromis, au stade actuel de la recherche, est schématisé dans la figure 3-9. Ce compromis consiste à manipuler pour chaque modèle :

- ses modèles amonts et avals, en incluant une relation d'ordre ;
- les connexions qu'il est possible de réaliser entre les variables du modèle et les variables des modèles amonts et avals ;
- les valeurs par défaut des paramètres du modèle ;
- le domaine de validité des variables manipulées.

Les rubriques qui contiennent donc les informations pour chaque modèle ci-avant citées sont donc essentielles à remplir par le concepteur de modèles pour pouvoir utiliser les mécanismes d'aide.

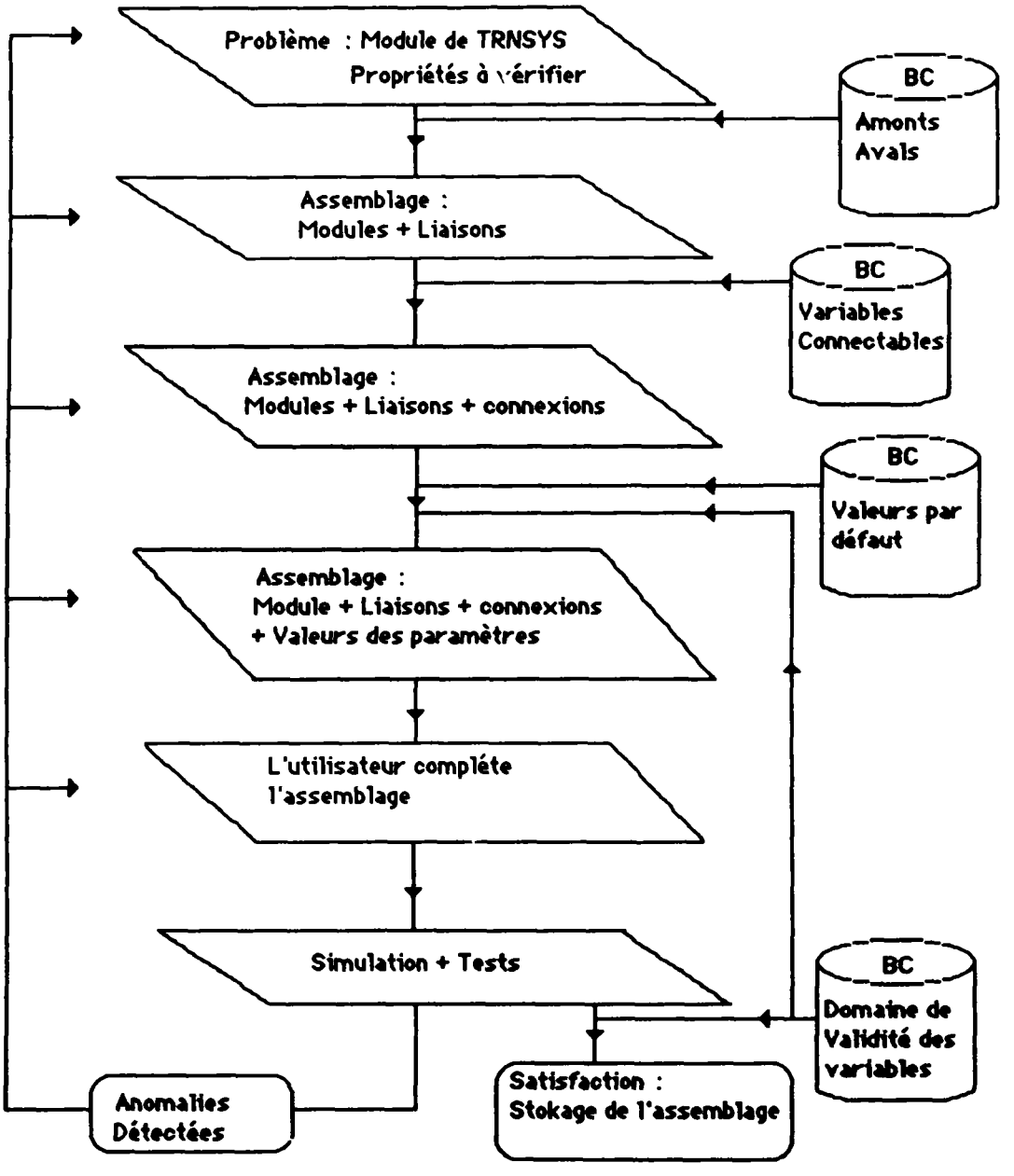


Figure 3-9 : Compromis réalisé

3-3 Autres modèles de raisonnement

Plusieurs modes de raisonnement présentent un grand intérêt :

- le raisonnement analogique ;
- le raisonnement distribué.

La base du raisonnement par analogie consiste en la mise en correspondance de deux situations, dans un univers d'application donnée, de façon à déduire un comportement devant une situation nouvelle.

Le raisonnement distribué se fonde sur la répartition de l'intelligence entre plusieurs acteurs, intelligents ou non.

3-3-1 Raisonnement par analogie ou raisonnement par cas

Le raisonnement par analogie est présent dans la vie quotidienne chaque fois qu'il faut résoudre un problème, en puisant dans le passé, un cas analogue au problème posé. Cette démarche est surtout utilisée lorsqu'il faut raisonner dans un contexte abstrait : on a plutôt tendance à raisonner d'abord sur un cas connu, puis à généraliser à partir du résultat obtenu. Le raisonnement par cas consiste à raisonner à partir d'expériences et de cas déjà rencontrés pour résoudre de nouveaux problèmes.

Le raisonnement par cas permet de compléter les méthodes de raisonnement classiques, notamment dans les domaines où la formalisation est délicate.

Pour y arriver, l'analogie met en œuvre un mécanisme de correspondance entre différentes structures, en fonction d'un problème à résoudre. Cette mise en correspondance doit pouvoir établir des relations entre objets en privilégiant celles qui apparaissent comme prioritaires, l'objectif étant de retrouver les cas les "plus similaires" au problème posé.

Deux types d'approches existent :

- la première approche a pour objectif de résoudre des problèmes, c'est à dire de dériver de nouvelles solutions en se fondant sur d'anciennes ;
- la deuxième approche a pour objectif l'explication, la justification d'une solution.

Il faut donc posséder dans une bibliothèque un certain nombre d'exemples typiques, chaque exemple étant représentatif d'une catégorie dont l'utilisation simplifie le raisonnement. Un système à base de connaissances ne peut devenir véritablement opérationnel que si une part importante des connaissances du domaine d'application est représentée.

Il est aussi nécessaire de disposer d'outils pour rechercher les informations, les mettre à jour et maintenir la cohérence de la base.

Les systèmes à héritage permettent de résoudre ces problèmes, et par conséquent permettent d'automatiser le mécanisme du raisonnement par cas.

3-3-1-1 La résolution de problèmes

Les unités de la base de connaissances sont considérées comme des solutions préétablies et partiellement ordonnées.

Ces unités peuvent être représentées de deux façons :

- représentation par objets ;
- représentation par *frames*.

Un *frame* est une structure composée d'attributs décrivant un concept et ses différentes propriétés. Un attribut est à son tour décrit par des facettes qui spécifient la nature de l'attribut et le comportement qui lui est associé.

La recherche d'une solution revient à établir une association entre l'objet du problème (objet noté X) et une des unités de la base de connaissances (objet noté U). Cette association ne signifie nullement que cette unité représente la solution idéale du problème posé, mais elle permet à l'utilisateur d'avoir une solution initiale à partir de laquelle il pourra constituer sa solution finale.

La comparaison s'appuie sur la recherche des unités partageant certaines propriétés avec X, une propriété pouvant être vue comme un couple (attribut, valeurs).

On pose que X est comparable avec une unité U si un ensemble non vide des propriétés de X est défini dans U. Il existe trois cas possibles :

- soit X est "plus spécifique" que U, alors X est une sous-classe de U ;
- soit X est "moins spécifique" que U, alors U est une sous-classe de X ;
- soit X et U ne sont pas comparables.

En général, trois étapes sont à prévoir :

- il s'agit de transformer les données décrivant le problème en élément de solution ;
- ensuite, il s'agit d'associer ces éléments de solution à une unité de la base ;
- dans un dernier temps, il s'agit de spécialiser l'hypothèse retenue en l'affinant.

3-3-1-2 Réalisation d'un système

Le problème qui se pose à ce niveau reste la façon de décrire les différentes unités de la base de connaissances. Une représentation efficace consiste à prendre en compte la typicalité de chaque unité en lui associant un objet typé comparable à un formulaire comportant des cases à remplir et des cases remplies par défaut.

Les données relatives à un problème posé par l'utilisateur constituent un cas auquel on associe un objet typique générique. L'association entre cet objet et une unité de la base de connaissances consiste à examiner les caractéristiques de cet objet afin de retrouver la classe à laquelle il se rattache.

Dans le cadre d'IISIBât, la base de connaissances est constituée par des objets tels que les projets. Ces objets représentent des solutions testées antérieurement. Le système intelligent, basé sur le raisonnement par cas, s'appuie autour des étapes suivantes :

- l'utilisateur doit d'abord décrire son problème grâce à un dialogue avec la machine. Le problème à traiter est transformé en un objet structuré (au sens Langage Orienté Objet) possédant des attributs. Ceux-ci sont ensuite utilisés dans une base de connaissances pour extraire des caractéristiques spécifiques.
- Les caractéristiques déterminées à partir de l'étape précédente sont utilisées pour rechercher dans la mémoire de cas un cas similaire. Cette recherche nécessite la définition de critères de comparaison.
- Une fois le cas déterminé, il est modifié de façon à tenir compte des spécifications du problème à traiter. Il est ensuite proposé à l'utilisateur.

Exemple

Un utilisateur recherche un assemblage qui s'applique au logement collectif, où l'énergie utilisée est le fioul et qui contient un module de type pompe.

Soit l'objet de type *demande* qui représente le champ d'investigation du moteur d'inférence. Cet objet possède par exemple les attributs suivants :

(contexte, type-des-modèles, énergie, solution)

Les valeurs des attributs de l'objet de type *demande* spécifique à notre problème sont :

contexte = logement-collectif ;
énergie = fioul ;
solution = (pompe) ;
type-des-modèles = tous.

La recherche s'effectue en comparant les attributs de cet objet avec ceux des objets de type *demande* instanciés à partir des bibliothèques de cas. L'utilisateur peut fixer comme critère essentiel l'énergie utilisée.

Supposons que le moteur d'inférence trouve une solution, c'est à dire un objet dans la bibliothèque de cas proche du problème posé, mais qui ne contient pas l'objet pompe. Si le cas est validé par l'utilisateur, l'assemblage proposé par le moteur d'inférence sera l'assemblage correspondant au cas trouvé, et modifié pour pouvoir tenir compte de l'objet pompe. Il reste à vérifier si la modification est possible, par exemple en regardant si un des éléments de la solution peut avoir comme modèle amont ou aval un objet de type pompe.

La démarche globale est illustrée dans la figure 3-10.

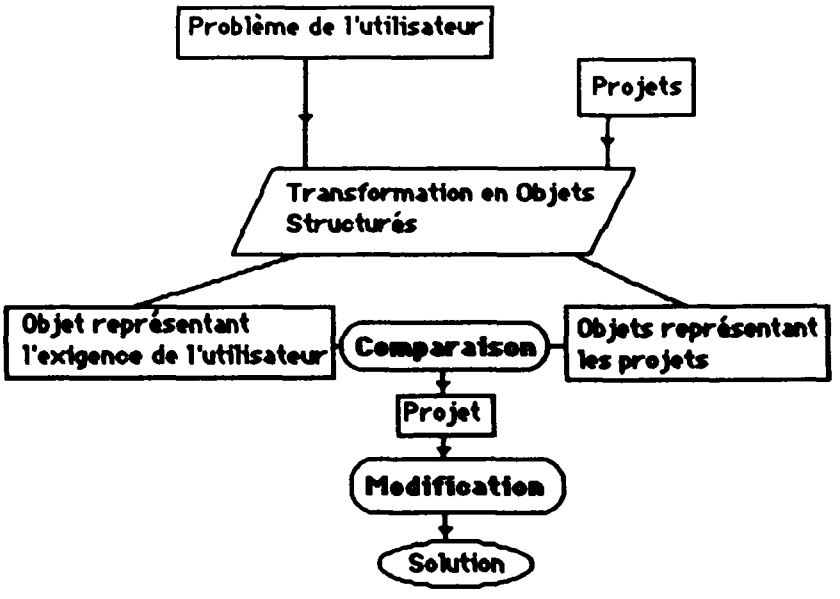


Figure 3-10 : Etapes du raisonnement par cas

L'efficacité de ce type de raisonnement dépend des cas sur lesquels il s'applique. Il s'agit de répertorier les cas de façon "intelligente" : ils doivent être suffisamment généraux pour permettre leur utilisation dans diverses situations, mais ils doivent être suffisamment spécifiques pour éviter les conflits dans la recherche d'un cas similaire et pour couvrir le domaine d'application.

Un cas est une mine d'informations, il synthétise un savoir-faire touchant à plusieurs domaines. Un cas peut s'avérer être l'équivalent de dizaines de règles de production.

Le raisonnement par cas est une approche intéressante, dans la mesure où des outils basés sur l'exploitation des cas permettent de constituer et de faire évoluer une mémoire collective d'une communauté d'individus : les utilisateurs des codes de simulation lèguent en général aux futurs utilisateurs essentiellement des projets réalisés, plutôt que des règles de conception bien formulées.

3-3-2 Les systèmes multi-agents

Le principe des systèmes multi-agents est de partager et distribuer entre plusieurs agents l'ensemble des connaissances et la capacité du raisonnement que possède un système intelligent. Chacun de ces agents est spécialisé dans un sous-domaine, il est capable de résoudre une partie ou la totalité du problème de départ. La collaboration avec les autres agents du système lui permet d'améliorer sa propre participation à la résolution du problème global. On distingue les systèmes à tableau noir et les systèmes d'acteurs.

Dans un système à "tableau noir" ou d'acteurs, les agents sont des modules qui renferment chacun une partie de la connaissance, et qui communiquent par messages. Un agent n'a pas la connaissance du problème complet, il ne réagit qu'aux informations auxquelles il a accès. Ces agents peuvent être conçus indépendamment les uns des autres ce qui assure modularité et évolutivité du système.

La différence entre les systèmes à "tableau noir" et les systèmes d'acteurs réside dans les modes de communication utilisés par les agents pour échanger de l'information.

Dans les systèmes à "tableau noir", les éléments de solution sont mémorisés dans une zone de données accessible à tous les agents : cette zone est appelée "tableau noir".

Dans les systèmes d'acteurs, les éléments de solution sont distribués entre les différents agents qui communiquent entre eux directement par envoi de messages.

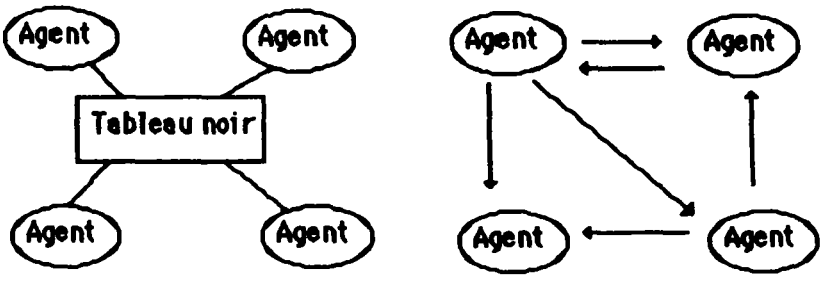


Figure 3-11 : Communication dans les systèmes à tableau noir et d'acteurs

Les avantages de cette architecture de raisonnement sont multiples :

- il est possible d'avoir deux agents dont les expertises se chevauchent, mais dont les points de vue sont différents ; cela permet de résoudre des problèmes complexes ;
- l'architecture informatique est complètement modulaire ;
- l'efficacité est assurée : un agent non opérationnel ne perturbe pas le fonctionnement du système.

Dans le cadre d'IISIBât, les agents peuvent être spécialisés par tâches :

- dialogue avec l'utilisateur ;
- choix des modules ;
- connexions des variables ;
- choix des valeurs numériques ;
- simulation qualitative ;
- choix de cas similaires au problème posé.

Conclusion du chapitre 3

La mise en place d'une couche experte, pour un outil de simulation, passe par la conception de modèles de raisonnement permettant d'appréhender les différentes facettes du processus de modélisation/simulation, dont un aspect important est la rétroactivité

Cette couche experte, après évaluation, est d'un apport précieux pour les utilisateurs. Construite à partir de mécanismes de raisonnement même simples, elle permet de gagner en productivité et assure le développement du produit logiciel. Elle va dans le sens de l'amélioration de la qualité en facilitant l'utilisation de l'interface.

L'utilisation du cadre formel de la logique classique permet l'automatisation partielle de certaines tâches que doit accomplir un utilisateur pour construire sa solution ; dans tous les cas, l'utilisateur et le système doivent collaborer pour trouver ensemble une ou plusieurs solutions.

Des règles qui permettent, d'une part l'aide au choix de modules, et d'autre part la fabrication automatique des liaisons entre modules, peuvent être écrites ; elles sont fondées sur la manipulation des "modèles amonts" et des "modèles avals" des modules TRNSYS (cf. § 3-2-2-1, 3-2-2-2). Ces règles peuvent aussi manipuler d'autres types d'informations, comme l'objectif de la simulation, les données disponibles ou le contexte de la simulation (cf. § 3-2-2-4).

Des règles de connexion peuvent être écrites, pour permettre à la fonction "Connexions Automatiques" de fonctionner. Il faut pour cela définir pour chaque variable d'entrée d'un module, une liste de variables de sortie, faisant partie de la définition d'autres modules, connectables à cette variable d'entrée. Le grand nombre de règles à écrire, lors de l'introduction d'un nouveau modèle dans une bibliothèque, nécessite la conception d'un système d'introduction de règles expertes exploitant les propriétés d'héritage des langages orientés objets (cf. § 3-2-3-2). Une autre approche consiste à fonder le raisonnement sur la manipulation des variables (cf. § 3-2-3-3) ; cette approche permet à l'utilisateur de ne pas écrire de règles de connexion lors de l'introduction d'un nouveau modèle dans une bibliothèque.

L'aide au choix des valeurs numériques de départ dans l'assemblage est possible grâce à la manipulation des valeurs par défaut des variables (cf. § 3-2-4). Dans le cas où il existerait plusieurs variables par défaut possibles pour un paramètre, nous serions amenés à réaliser des règles expertes pour aider l'utilisateur à opérer un choix.

Le raisonnement, fondé sur la modélisation qualitative, peut être intégré dans un environnement expert (cf. § 3-2-4-1). Les règles produites dans ce cadre peuvent être utilisées dans les phases de mise au point des assemblages, notamment en aidant l'utilisateur à choisir les valeurs numériques optimales pour son assemblage.

Enfin, il est possible d'aider l'utilisateur à interpréter les résultats, notamment en exploitant des informations comme les domaines de validité, ou en exploitant l'hypothèse du monde clos (cf. § 3-2-4).

La construction d'un modèle profond de raisonnement, fondé sur un modèle complexe des phénomènes mis en jeu, est difficile à mettre en œuvre dans le formalisme de la logique classique, du fait du problème de la maintenance de la base de connaissances.

Des modèles de raisonnement qui ne sont pas des extensions de la logique existent. Le raisonnement par cas (un cas étant une solution préétablie) est un modèle intéressant (cf. § 3-3-1). D'une part, il permet d'éviter à un concepteur de systèmes experts l'écriture d'un trop grand nombre de règles expertes : en effet, un cas contient une somme de connaissances synthétisées, et peut s'avérer être l'équivalent de dizaines de règles de production. D'autre part, le raisonnement par cas permet d'assurer la continuité d'un travail collectif, puisque les utilisateurs des codes de simulation lèguent essentiellement aux futurs utilisateurs des projets réalisés, sur lesquels justement s'applique le raisonnement par cas ; cet aspect rend donc les anciens projets accessibles aux nouveaux utilisateurs.

Les systèmes multi-agents présentent, quant à eux, une architecture séduisante, ils permettent la structuration du raisonnement (cf. § 3-3-2). Cet aspect est important dans le cadre du bâtiment, du fait de l'hétérogénéité des connaissances manipulées, et des stratégies de résolution variées. Au sein d'une telle architecture, peuvent s'insérer les modèles de raisonnement précités.

Concusion Générale

Concusion Générale

Cette recherche a été motivé par le souci de mettre à la portée d'un grand nombre de professionnels du bâtiment des outils de simulation performants jusqu'ici réservés aux seuls laboratoires de recherche. Pour ce faire, un concept d'ESI (Environnement de Simulation Intelligent) a été développé ; il s'agit de réaliser un environnement de simulation générique, grâce auquel il soit facile de générer des applications dédiées.

Les travaux réalisés dans le cadre de cette recherche consistaient à :

- contribuer au développement du concept d'ESI, en améliorant et en faisant évoluer le modèle abstrait qui supporte ce concept ;
- mettre en œuvre ce concept pour réaliser une application spécifique ; cette application dénommée IISIBât a consisté à développer un environnement de simulation pour le logiciel de calculs thermiques TRNSYS ;
- évaluer l'application développée, et à tirer les conclusions pour d'éventuelles améliorations ;
- proposer des modalités concrètes de mise en œuvre des améliorations envisagées.

Il est apparu que la programmation orientée objets était bien adaptée à l'environnement que nous avons développé. Par ailleurs, la mise en place d'une architecture informatique basée sur les données a permis de séparer totalement les fonctionnalités de l'interface, du reste du système (la structure de données en l'occurence). Cela a donné au logiciel l'extensibilité voulue ; il est assez aisé d'ajouter de nouvelles fonctionnalités aux prototypes réalisés.

Dans le cadre d'un outil de simulation qui fonctionne par assemblage de composants, les données de base sont les modèles. Cependant, une structure de données de base fondée exclusivement sur des objets de type modèles, s'est avérée incomplète. Il fallait donc enrichir cette structure de données, et nous avons créé deux autres types d'objets : les macro-modèles et les projets, qui sont en fait des assemblages de modèles et de macro-modèles. Les macro-modèles permettent à un utililisateur d'enrichir la bibliothèque de modèles, avec de nouveaux "modèles" réalisés par assemblage de modèles existants. Par ailleurs, le macro-modèle constitue éventuellement une étape vers la réalisation d'un projet final. Les projets sont des modélisations de systèmes réels plus ou moins complexes ; ils sont entièrement paramétrés et donc prêts à la simulation.

Par ailleurs, il ne s'agissait pas pour nous de considérer l'application interactive comme une simple application qui génére des résultats numériques. Il fallait prendre en compte des fonctionnalités annexes, telles que le stockage des modèles, et l'échange de modèles entre utilisateurs. A cet effet, nous avons créé deux autres types d'objets : les gestionnaires de bibliothèques, dans lesquels nous pensons que l'utilisation des propriétés d'héritage pourraient constituer un atout majeur, et les gestionnaires de comptes.

Le gestionnaire de comptes a essentiellement pour rôle la décomposition des tâches ; concrètement, il s'agit d'archiver un utilisateur dans une certaine catégorie, et de lui donner l'accès de l'outil de simulation avec certaines fonctionnalités. Pour ce faire, nous avons répertorié trois types d'utilisateurs :

- le groupe des utilisateurs qui développent des modèles (Ingénieurs Concepteurs qui constituent donc les bibliothèques) ;
- le groupe des utilisateurs qui fabriquent des systèmes par assemblage de composants (Ingénieurs Assembleurs) ;

- le groupe des utilisateurs dont le rôle est d'analyser les résultats (Ingénieurs Analystes).

Cette différenciation colle aux tâches à réaliser dans le cadre d'un processus de Modélisation/Simulation/Analyse des résultats. Elle permet d'adapter l'interface au type d'utilisateur, les outils proposés par l'interface correspondant au mieux au travail à effectuer.

Un autre objet essentiel dans le développement d'IISIBât est celui qui permet d'assembler les objets de la structure de données de base (modèles, macro-modèles et projets) : il s'agit du panneau d'assemblage.

Nous avons fondé notre application sur la mise en place d'un environnement informatique basé sur le fenêtrage, et sur la manipulation directe des objets graphiques ; le modèle iconique s'est avéré être un modèle d'interaction adapté.

A ce stade de l'étude, nous avons procédé à une première évaluation : certes, l'outil conçu est simple d'utilisation ; cependant il fallait trouver des solutions pour, d'une part réduire le grand nombre de possibilités offertes par un outil où l'utilisateur reste libre de ses actions (choix de modules, liaisons de modules, connexion de variables, introduction de valeurs numériques -paramètres, valeurs initiales-, introduction des cartes de contrôle), et d'autre part aider l'utilisateur à concevoir des solutions satisfaisantes.

La méthodologie des systèmes à base de connaissances nous a semblé toute indiquée pour répondre à cette attente ; cela nous a permis de réaliser des outils d'aide à la conception, qui font appel à des informations contenues dans des bases de connaissances. L'utilisation du cadre formel de la logique classique a permis d'écrire des règles qui ont apporté, une aide sur le choix des modules (approche fondée sur la connaissance des modèles amonts des modules), une aide sur la connexion des variables, ainsi qu'une aide pour le choix des valeurs numériques (approche fondée sur la connaissance des valeurs par défaut des paramètres, et complétée par des règles faisant appel à la physique qualitative).

Cependant, le formalisme de la logique classique pose des problèmes de maintenance pour des systèmes de grande taille. A quoi s'ajoute le manque de structuration des connaissances exprimées. D'où la nécessité d'élargir ultérieurement cette recherche, en approfondissant certaines propositions qui demandent d'avantage de vérifications. Ces propositions concernent :

- l'architecture du système expert ;
- le stockage des connaissances.

L'architecture présentée par les systèmes multi-agents est séduisante, dans la mesure où elle permet la structuration du raisonnement. Au sein de cette structure, il est possible d'introduire des modèles de raisonnement qui ne sont pas des extensions de la logique classique. Nous pensons au raisonnement par cas, qui est une démarche courante dans le monde de la simulation, notamment dans les domaines où la formalisation de la connaissance est délicate.

Introduire des mécanismes de raisonnement au sein d'un logiciel, nécessite la constitution de bases de connaissances, dans lesquelles seront stockées les règles expertes. Deux approches existent :

- soit utiliser le cadre du Proforma, et considérer qu'un ensemble de fiches Proforma constitue une base de connaissances. Il faudrait alors enrichir les versions actuelles des fiches Proforma ; pour y arriver, il faudrait d'abord réaliser

un modèle profond du concept de Proforma, en tenant compte notamment des fonctionnalités expertes qu'il serait possible d'insérer dans un ESI ;

- soit construire des bases de connaissances complètement autonomes des fiches Proforma ; dans ce cas de figure, la redondance des informations au sein de l'environnement est inévitable. Dans les prototypes réalisés, nous avons opté pour cette solution, puisque les fiches Proforma en l'état actuel ne comportent pas toutes les informations nécessaires.

Nous espérons que ce travail aura contribué à l'évolution dans la bonne direction du concept d'ESI, qu'il aura apporté un certain nombre de réponses pour ce qui concerne le dialogue Homme/Machine et les fonctionnalités à rajouter à l'ESI pour rendre celui-ci véritablement efficace pour toutes les catégories d'utilisateurs, et enfin que les pistes explorées ou proposées pour réaliser certaines de ces fonctionnalités s'avéront les bonnes.

Références Bibliographiques-Bibliographie

Références Bibliographiques

[BAER 92]

A Knowledge-Based Planning System for the Design of Heating-System
K. Baer, F. Schmidt
Forum, Erlagen (Germany), Expert Systems and computer simulation in energy engineering, 1992

[BEAUDOIN 91]

Interfaces homme-machine : vue d'ensemble et perspectives
M. Beaudoin-Lafon
in Génie Logiciel & Systèmes experts, Interfaces homme-machine - Maquettage & Prototypage - Ergonomie, p 4- 16, 1991

[BONNET 85]

Psychologie, Intelligence artificielle et automatique
C. Bonnet, J.M HOC, G. Tiberghien
Livre, Ed. Pierre Mardaga, Bruxelles, 1985

[COMIS 91]

Comis-User Guide
H.E Feustel, A. Rayner-Hooson
Manuel d'utilisation, University of California, Berkeley, 1991

[DE KLEER et BROWN 86]

Theories of Causal Ordering
J. De Kleer, J.S Brown
in Artificial Intelligence, vol. 29, pp. 33-61

[DUBOIS 90]

Eléments de spécification d'un environnement avancé de modélisation et simulation. Application à la thermique du bâtiment
A.M Dubois Courau
Thèse, Nice, Université de Nice Sophia-Antipolis, 1990

[DUBOIS 92]

COMBINE IDM/V3.3
A.M Dubois
Rapport, Valbonne, CSTB, TTA/DPE/92-1218, janvier 1992

[EL HASSAR 90]

Module Ensoleillement : MODSOL

K. El Hassar
Rapport, Valbonne, CSTB, MGL/90-1164/NB, 1990

[ESCUDIER 92]

PROFSYS. Interface Graphique pour la Documentation de Modèles
J.C Escudier
Document de travail, CSTB, MGL/92-1275/JCE, 1992

[FARRENY 87]

Eléments d'intelligence artificielle
H. Farreny, M. Ghallab
Livre, Paris, Hermes, 1987

[FORBUS 84]

Qualitative Process Theory
K.D Forbus
in Artificial Intelligence, vol. 24, pp. 85-167

[FURBRINGER 92]

Evaluation Procedure using sensitivity analysis of model and measurement
J.M Fürbringer
Symposium Budapest, septembre 1992

[GOLDBERG 84]

Smalltalk 80 : The interactive programming environment
A. Goldberg
Addison Wesly Publications, 1984

[HATON 91]

Le raisonnement en intelligence artificielle-Modèles, techniques et architectures pour les systèmes à base de connaissances
J.P Haton, N. Bouzid, F. Charpillet, M.C Haton, B. Lâasri, H. Lâasri, P. Marquis, T. Mondot, A. Napoli
Livre, Paris, InterEditions, 1991

[JOUVENEL 91]

Etude de cadrage sur les applications de la physique qualitative aux domaines du bâtiment
M.N Jouvenel
Rapport de stage, Valbonne, 1991
CSTB DPE/91-1065

[KUIPERS 86]

Qualitative Simulation
B.Kuipers
in Artificial Intelligence, vol. 29, pp 289-338

[LARET 1-89]

Modélisation du comportement thermique de l'immeuble expérimental de la DETN. Rapport Final.
L.Laret
Etude, Valbonne, 1989
CSTB MGL 1075

[LARET 2-89]

Le concept de modèle adapté. Séminaire spécialisé "Outil d'aide à la conception et à la gestion"
L. Laret
Article, Valbonne, CSTB, 1989

[LEBRUN 85]

Modélisation et Désign des systèmes de chauffage, ventilation et conditionnement d'air
J. Lebrun
in Climat 2000, T. 1, Liège, 1985

[MEKOUAR 91]

Recherche et développement de méthodes informatiques innovantes de formation assistée par ordinateur - Application aux systèmes énérgétiques pour le bâtiment
K. Mekouar
Thèse, Université de Nice Sophia-Antipolis, janvier 1991

[MEYER 90]

Conception et Programmation par objets-Pour du logiciel de qualité
B. Meyer
Livre, Paris, InterEditions, 1990

[NATAF-WINKELMANN 91]

Dynamic Simulation of a liquid desiccant cooling system using the energy Kernel System
J.M Nataf, F. Winkelmann
Simulation Research Group, Applied Science Division, Lawrence Berkeley Laboratory
U. of California, Février 1991

[PELLETRET 1-92]

IISIBât : un environnement de simulation intelligent
R. Pelletret, S.Soubra
Document de travail, Valbonne, CSTB, 1992

[POYET 1-92]

Environnements logiciels pour l'ingénierie intégrée
P. Poyet, E. Brisson, P. Debras, A.M Dubois
Article, Valbonne, CSTB, 1992

[POYET 2-92]

Proposition d'organisation des données FARTEC dans le cadre du projet ATLAS
P. Poyet, E. Brisson
DBC / 92-1277 / PP, avril 1992

[ROBIN 91]

Intégration de savoir expert et d'outils de simulation pour la conception thermique des bâtiments et des systèmes
C. Robin
Thèse, Lyon, Institut National des Sciences Appliquées, 1991

[SOUBRA 92]

Concepts et outils pour le développement d'un Environnement de Simulation Intelligent
S. Soubra
Thèse, Ecole Nationale des Ponts et Chaussées, Paris, 1992

[TRNSYS 90]

A transient System Simulation Program
Manual, Solar Energy Laboratoire, Univ. Of Winconsin, USA 1990

BIBLIOGRAPHIE

Ce travail de thèse s'appuie aussi sur les documents suivants, documents qui ne sont pas cités dans le corps du texte.

[ALMETH 87]

Vers une base de connaissances en modélisation thermique du bâtiment, éléments d'analyse d'une modélothèque
L.M Chounet, A.M Dubois, R. Fauconnier, P. Guillemard, A. Lahellec, L. Laret, J. Sornay
ICBEM, Lausanne, septembre 1987

[ASHRAE 89]

ASHRAE Handbook
R.A Parsons
Livre, Atlanta, 1989

[BAILLY 87]

Les langages orientés objets
C.Bailly, J.F Challine, P.Y Gloess
Livre, Toulouse, Cepadues, 1987

[BARR 86]

Le manuel de l'intelligence artificielle
H. Barr, E.A Feigenbaum
Livre, Paris, Eyrolles, 1986

[BOUILLE 77]

Un modèle universel de banques de données, simultanément partageable, portable et répartie
F. Bouillé
Thèse, Paris, Université Pierre et Marie Curie, avril 1977

[BREJON 87]

Logiciels de Thermique
P. Brejon
in CVC, p 15-18, juin/juillet 1987

[CACCAVELLI 87]

Modélisation simplifiée du comportement thermique d'un bâtiment multizone
D. Caccavelli, J.J Roux, J. Brau
in Revue Générale de la Thermique, n° 311, p 585-596, novembre 1987

[CLARKE 85]

Energy Simulation in Building Design
J.A Clarke
Livre, Bristol (Royaume Uni), Adam Hilger, 1985

[COUSIN 88]

Apport de la théorie des catégories à la représentation des connaisances
R. Cousin
Thèse, Paris, Université Pierre et Marie Curie, janvier 1988

[DEMAIZIERE 86]

Enseignement assisté par ordinateur
F. Demaiziere
Livre, Paris, Ophrys, 1986

[DUBOIS 88]

Le projet Proforma-Mythes et réalités
A.M Dubois
Rapport, Valbonne, 1988
CSTB MGL 787

[GAUGE 85]

Modèles Thermiques développés sur ASTEC 3
F. Gauge, A. Burtin
Rapport, EDF DER, HE 112 W2237, 1985

[GOUARDERES 86]

Représentation et manipulation des connaissances dans le dialogue Homme/Machine en enseignement assisté par ordinateur
G.Gouarderes
Thèse, Toulouse, Université P.Sabatier, 1986

[GOURDIN 89]

Méthodes Numériques appliquées
A. Gourdin, M. Boumahrat
Livre, Paris, Techniques et Documentation-Lavoisier, 1989

[GUITTARD 89]

Modélisation des connaissances en thermique du bâtiment-Réalisation d'une maquette au moyen d'un générateur de système multi-expert
C. Guittard
Rapport de stage, Valbonne, 1989
CSTB TTA/MGL 1081

[HATON 89]

Panorama des systèmes multi-agents

J.P Haton
in Haton (ed), EC2, Paris, 1989

[HOLMES 85]

Artificial Intelligence and Simulation
W.M Holmes
SCS, 1985

[ILOG 1-92]

Aida, environnement de développement d'applications
manuel d'utilisation, verion 1.65, mars 1992

[ILOG 91]

Le-lisp de l'INRIA
manuel d'utilsation, version 15.25, novembre 1991

[INRIA 90]

Projet de psychologie ergonomique pour l'informatique
Rapport d'activité 1990, INRIA

[KHEIR 88]

Systems Modeling and Computer Simulation
N.A Kheir
Livre, Dekker, 1988

[LEBEUX 86]

Un système d'acquisition des connaissances pour systèmes experts
P. Lebeux, D. Fontaine
in Techniques et Science Informatiques, vol. 5 n°1, p 7-20, 1986

[LEBRU 81]

Analyse de codes de calcul américains et français appliqués au chauffage de l'habitat par l'énergie solaire (systèmes passifs et actifs)
A. Lebru
Etude, Valbonne, 1981
CSTB TEA S25 AL/P0

[LECLERCQ 89]

La modélisation de l'apprenant
D. Leclercq, C. Luttgens
in Techniques et Science Informatiques, vol. 8 n°1, p 76-81, 1989

[NEVEU 86]

Modèles d'évolution thermique des bâtiments. Conditions pratiques d'identification

A. Neveu, P. Bacot, R. Regas
in Revue Générale de la Thermique, p 423-420, août-septembre 1986

[PELLETRET 1-89]

Un didacticiel sur le thème de la régulation
R.Pelletret
Congrés, Sophia-Antipolis, 1989
CSTB, MGL 1108

[PELLETRET 2-89]

Le didacticiel "Régulation 1"
R.Pelletret
Congrés, Paris, 1989
CSTB DPE 769

[PELLETRET 3-89]

Interfaces Intelligentes pour les progiciels de Modélisation/Simulation
R.Pelletret
Congrés, 1989
CSTB DPE 733

[PELLETRET 2-92]

Model Documentation. The Annex 23 PROFORMA
R. Pelletret
Annex 23 Working document, avril 1992

[RIFFLET 90]

La programmation sous UNIX
J.M Rifflet
Livre, Paris, McGraw-Hill, 1990

[SAIAC 89]

L'informatique appliquée au calcul scientifique
J.H Saïac
Livre, Paris, Dunod Informatique, 1989

[SHORTLIFFE 76]

Computer-Based Medical Consultation : MYCIN
E. Shortliffe
Livre, New-York, Elsevier, 1976

[SIMULATION 91]

Building Simulation '91
Conférence, 20-22 août 1991, Sophia-Antipolis, NICE

[SOUBRA 89]

Cstbat Junior pour Apple Macintosh (version 1.0) - Manuel d'utilisation
S.Soubra
Rapport, 1989
CSTB DPE 678

[SOUBRA 91]

Intelligent Simulation Environnements for Building Modelling and Simulations
S.Soubra
Rapport, Valbonne, 1991
CSTB TTA/DPE/91-1076/SSo

[SOWEL 88]

Dynamic Extension of the Simulation Problem Ananlysis Kernel (SPANK)
Sowel, E.F and W.F Buhl
Conférence, Ostende, Belgique, 1988

[STOECKEL 91]

X_WINDOW Programmation
R. Stoeckel
Livre, Paris, Armand Colin, 1991

[VISENTIN 89]

Modélisation thermique des interactions enveloppes-systèmes dans la cellule DESYS à l'aide du logiciel CSTBAT
I.Visentin
Rapport de stage, 1989
CSTB MGL 1083

ANNEXES

Annexe 1

Principes Généraux de la Modélisation

Un travail de modélisation comporte généralement six grandes étapes [LEBRUN 85] :

- la première étape consiste à analyser l'aspect technique du problème. Cette analyse doit permettre l'étude de la topologie générale du système à étudier ;
- la deuxième étape consiste à analyser le système d'un point physique. A ce niveau de l'étude, on ne doit plus faire apparaître que les flux jugés essentiels dans le cadre du problème étudié ;
- dans une troisième étape, il s'agit de traduire mathématiquement les phénomènes physiques par des équations ;
- dans une quatrième étape, il s'agit de définir des schémas de résolution possibles de ces équations ;
- il s'agit ensuite d'élaborer des algorithmes de calul ;
- il s'agit dans un dernier point d'adapter l'algorithme aux possibilités de la machine.

Les trois premières étapes correspondent à la modélisation, c'est à dire à la représentation abstraite par un modèle d'un comportement d'un objet. Les trois dernières étapes correpondent, quant à elles, au passage d'un modèle mathématique à un modèle numérique approchant au mieux le comportement du modèle mathématique.

Dans le monde de la thermique du bâtiment, la modélisation mathématique des phénomènes repose essentiellement sur des principes de conservation et sur des équations de transferts de deux types :

- transfert des flux d'énergie échangés entre sous-systèmes. On distingue les transferts qui ont lieu sans transfert de fluide, sous la forme de chaleur (modélisation de la conduction) ou de travail, et les transferts qui s'opérent avec transfert de fluide sous forme d'énergie interne, d'énergie cinétique, d'énergie potentielle, etc,...

- transfert des flux de matière (comme les débits d'air, les débits d'eau).

Les flux d'énergie sont régis par les deux premiers principes de la thermodynamique ; par exemple le premier principe se traduit par un bilan tel que :

$$\dot{U} = \Sigma \dot{M}_j h_j + \Sigma \dot{Q} + \Sigma \dot{W}$$

Avec :

$\dot{U}$: énergie interne du volume considéré ;

$\dot{M}_j$: débit massique du fluide j entrant ou sortant du volume de référence ;

h_j : enthalpie massique du fluide j ;

$\dot{Q}$: flux de chaleur apporté ou prélevé au volume de référence ;

$\dot{W}$: flux de travail apporté ou prélevé au volume de référence.

Les équations de conservation du flux de matières sont formulées de la façon suivante :

$$\dot{M}_i = \sum_j \dot{M}_{ij}$$

Avec :

$\dot{M}_i$: masse de la matière contenue dans le volume de référence ;
$\dot{M}_{ij}$: débit de matière entrant ou sortant du volume de référence.

Annexe 2

Exemples de Méthodes Numériques

A2-1 Résolution des systèmes d'équations linéaires

Tous les codes de simulation doivent résoudre des systèmes d'équations linéaires qui modélisent les divers éléments d'un système physique. On s'intéresse à la résolution simultanée de n équations à n inconnues, notée A X = B.

On sépare les problèmes en deux classes suivant les caractéristiques de la matrice A :

- la matrice A est de taille réduite (matrice d'ordre inférieur à 100) et pleine ; par matrice pleine, on entend que A comporte peu d'éléments nuls ;
- la matrice A est creuse (matrice qui comporte peu d'éléments non nuls) et de grande taille (matrice d'ordre égal à plusieurs centaines ou plusieurs milliers).

Suivant la catégorie du problème, on emploie souvent une approche différente. Pour résoudre des systèmes de petite taille et pleins, on préfère utiliser des méthodes *directes*. Pour résoudre les grands systèmes à matrice A creuse, on préfère utiliser des méthodes *itératives*.

Les méthodes directes les plus connues sont :

- la méthode de Gauss, utilisée pour un système à matrice A quelconque ;
- la méthode de Jordan ;
- la méthode de Cholevski, recommandée pour la résolution des systèmes linéaires à matrice symétrique définie positive.

Dans la plupart des problèmes, une pivotation (échange de lignes) est nécessaire ; afin de minimiser l'erreur d'arrondi, on évite le plus possible la division par de petits pivots. En effet, des divisions par des nombres petits entraînent des erreurs de calcul importantes.

Les méthodes itératives consistent à utiliser un vecteur estimé initial du système et de générer une séquence de vecteurs. Les plus connues sont les méthodes de Jacobi ou de Gauss-Seidel. Ces méthodes posent des problèmes de convergence, une simple permutation des lignes peut transformer une convergence en divergence et inversement.

Cette remarque est importante ; en effet, il se peut qu'au cours d'une simulation, il y ait divergence des résultats. Une solution, qui concerne l'utilisation du code de calcul TRNSYS, consiste à changer l'ordre de déclaration des modules dans le fichier DECK, ce qui a pour effet d'opérer une pivotation.

A2-2 Limites des méthodes numériques

Les utilisateurs des outils de simulation ont souvent tendance à négliger l'incapacité pour une machine de représenter correctement les abstractions mathématiques que sont l'infini et le continu. Ils ont tendance aussi à oublier que la quantité d'informations manipulées par un ordinateur est finie (même si elle est énorme), et que le calcul scientifique est dépendant des erreurs d'arrondis qui sont inévitables sur toutes les machines. Ces limites lorsqu'elles sont connues peuvent, d'une part expliquer des résultats de simulation inattendus, et d'autre part peuvent donner lieu à des règles expertes pour aider un utilisateur à résoudre des problèmes.

Pour illustrer ce propos, on se propose de prendre un exemple : la résolution des équations différentielles. On se limitera à la résolution des équations différentielles du premier ordre qui ont la forme suivante :

$$y'(x)=f(x,y(x))$$
$$y(a)=y0$$

pour tout x appartenant à l'intervalle [a,b].

Les problèmes de ce type sont dénommés problèmes à valeur initiale ou problèmes de Cauchy. On fait l'hypothèse que la fonction f est une fonction continue de deux variables vérifiant une condition de Lipschitz par rapport à sa deuxième variable ; c'est à dire que pour tout couple de valeurs y1 et y2 réelles, il existe une constante $L \geq 0$ telle que :

$$|f(x,y1)-f(x,y2)| \leq L\,|y1-y2| \qquad \text{pour tout x dans [a,b].}$$

Sous cette hypothèse, le problème différentiel est un problème "bien posé" : il admet une solution unique y pour toute valeur initiale x0.

A2-2-1 Principe de résolution

La solution mathématique d'une équation différentielle du type précédent est une fonction continue.

Les calculateurs numériques ne connaissent pas le concept de fonction continue, ils ne peuvent fournir (approximativement) que des valeurs prises en des points connus : à la solution mathématique, fonction continue sur un intervalle [a,b], les méthodes numériques calculent une collection de valeurs discrètes en certaines abscisses. Le choix des abscisses résulte en général d'un algorithme de pas adptatifs, augmentant sélectivement le nombre de points dans les intervalles où cela est nécessaire, afin d'optimiser le rapport entre la précision du calcul et le coût.

Le processus de résolution est un processus fini dans lequel on ne calcule la valeur approchée en un point qu'une seule fois. Les erreurs s'accumulent au fur et à mesure que l'on avance.

Tout commence donc par un choix préalable des abscisses xi d'indice i = 0 à N où l'on calculera les valeurs approchées de la solution y dans l'intervalle [a,b]. Ces abscisses xi peuvent être régulièrement espacés d'un pas h = (b-a)/N.

Les méthodes les plus utilisées pour résoudre ce type de problème sont les méthodes de Runge-Kutta. Il s'agit de méthodes explicites à un pas qui ne nécessitent pas le calcul des dérivées de f. Un optimum pratique est la méthode de Runge-Kutta d'ordre 4. Elle s'écrit, pour y_0 donné :

$$\boxed{y_{n+1}=y_n + \frac{1}{6}(k_1+2k_2+2k_3+k_4)}$$

avec :

$$\begin{cases} k_1 = hf(x_n , y_n) \\ k_2 = hf(x_n + \frac{h}{2} , y_n + \frac{k_1}{2}) \\ k_3 = hf(x_n + \frac{h}{2} , y_n + \frac{k_2}{2}) \\ k_4 = hf(x_n + h , y_n + k_3) \end{cases}$$

A2-2-2 Exemple 1

Soit à résoudre :

$y'(x) = - 10 \times y (x)$
$y(0) = 1$
x dans [0 , 2].

$| f (x , y1) - f (x , y2) | = 10 |x| [y2 - y1 | \leq 20 | y2 - y1 |$: la fonction y'(x) est lipschitzienne par rapport à y.

La solution exacte est :

$$y (x) = e^{- 5 x^2}$$

Nous présentons les résultats obtenus par la méthode de Runge-Kutta pour une discrétisation de l'intervalle :

Nombre d'intervalles	Abscisses	Runge-Kutta	Solution exacte
512	0	1	1
512	0.25	0.7316157	0.7316156
512	0.5	0.2865148	0.2865048
512	0.75	0.600546 E-01	0.600546 E-01
512	1.00	0.673794 E-02	0.673794 E-02
512	1.25	0.404645 E-03	0.404645 E-03
512	1.5	0.130073 E-04	0.130073 E-04
512	1.75	0.223803 E-06	0.223803 E-06
512	2.00	0.206115 E-08	0.206115 E-08

On observe l'excellent comportement de la méthode Runge-Kutta, pour cet exemple.

A2-2-3 Exemple 2

Soit à résoudre :

$y' (x) = 5 y (x) - 6 e^{-x}$
$y(0) = 1$
x dans [0 , 10]

La solution exacte est :

$$y(x) = e^{-x}$$

Nombre d'intervalles	Abscisses	Runge-Kutta	Solution exacte
512	0	1	1
512	1.25	0.286505	0.286505
512	2.5	0.823308 E-01	0.820850 E-01
512	3.75	0.1508798	0.235177 E-01
512	5.00	65.98167	0.673794 E-02
512	6.25	34175.74	0.193045 E-02
512	7.5	1.77034 E+07	0.553084 E-03
512	8.75	9.170552 E+09	0.158461 E-03
512	10.00	4.750445 E+12	0.453999 E-04

On observe la dégradation sensible des résultats à partir de $x \geq 3$.

A2-2-4 Explication de l'instabilité

Si on modifie la condition initiale de ß, on modifiera la solution y en une solution yß avec :

$y(0) = 1 + ß$;
$yß(x) = (1+ß)\, e^{-5x^2}$ pour l'exemple 1 ;
$yß(x) = e^{-x} + ß\, e^{5x}$ pour l'exemple 2.

Dans le premier cas, l'écart entre y et yß pour tout x est un infiniment petit du même ordre de grandeur que l'écart ß des valeurs initiales.

Dans le second cas, l'écart entre y et yß croît exponentiellement avec x. Numériquement, cette équation est quasiment impossible à résoudre sur un intervalle [a , b] assez grand. En effet, supposons que l'erreur d'arrondi de la machine est de l'ordre de 10^{-9} sur la valeur initiale, l'écart entre la solution exacte et la solution calculée est estimée par "ecart" :

$$ecart = 10^{-9} \cdot e^{5x}$$

A2-2-5 Incidences sur la conduite de la simulation

Il n'est pas possible de tout résoudre numériquement. Certaines équations instables sont difficiles, voire impossibles à résoudre sur un ordinateur.

La connaissance de certaines notions fondamentales concernant la résolution numérique peut aider un utilisateur à faire des choix dans une bibliothèque de composants. Elle peut aussi l'aider à interpréter des résultats de simulation (cas où ces résultats ont peu de rapports avec la vraissemblance physique).

Ces informations peuvent être regroupées dans une base de connaissances sous forme de règles expertes. Une application directe de ce qui a été présenté précédemment concerne le choix du début et de la fin de la simulation. Si l'utilisateur est en présence d'un cas instable, une solution consiste à réduire le temps simulé.

Une règle peut être la suivante :

Si

une des variables de la simulation n'a aucun rapport avec la réalité physique

alors

permuter les lignes et relancer la simulation
ou
réduire le temps simulé

Annexe 3

Session Type

Objectif de l'utilisateur

L'objectif de l'utilisateur est d'éxaminer la performance d'une installation solaire de production d'eau chaude. Le système est constitué d'un capteur solaire et d'un régulateur qui active une pompe. Le système est fourni en eau à température constante (20°C).

Le capteur solaire est constitué d'un couvercle de surface 6.5 m2, le fluide coloprteur est l'eau (Cp = 4.19 J/kg °C). Lorsque la pompe est en marche, le débit du fluide est de 200 l/s.

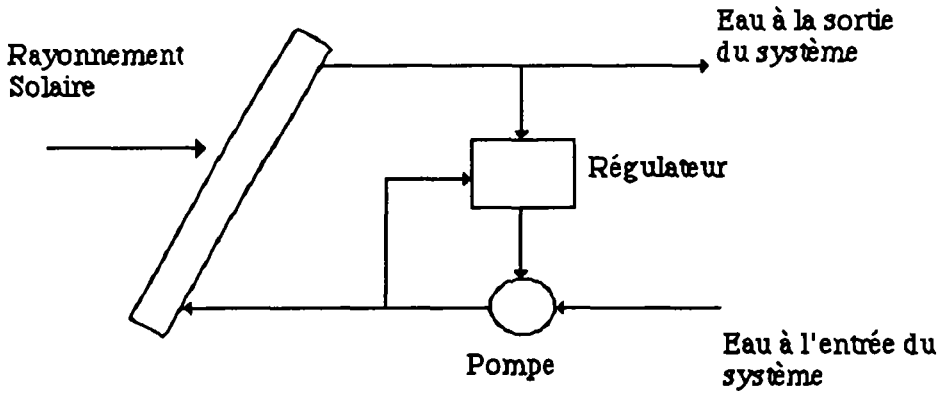
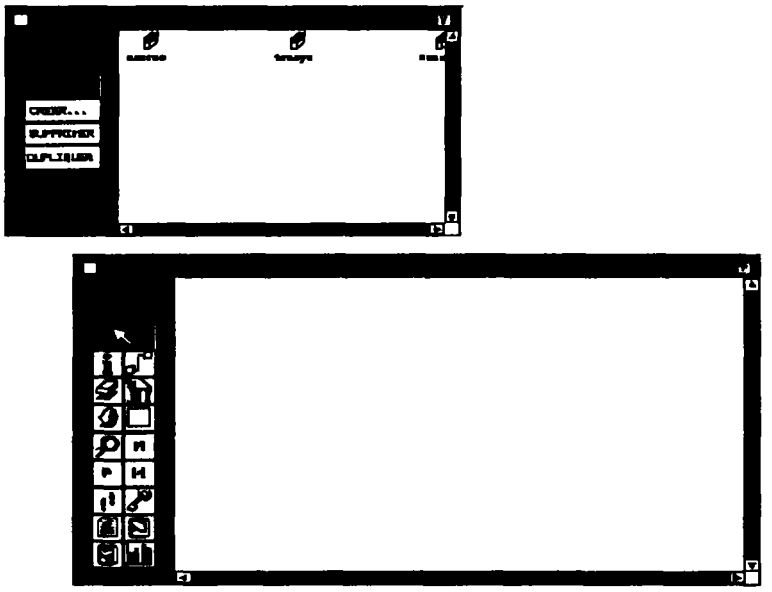


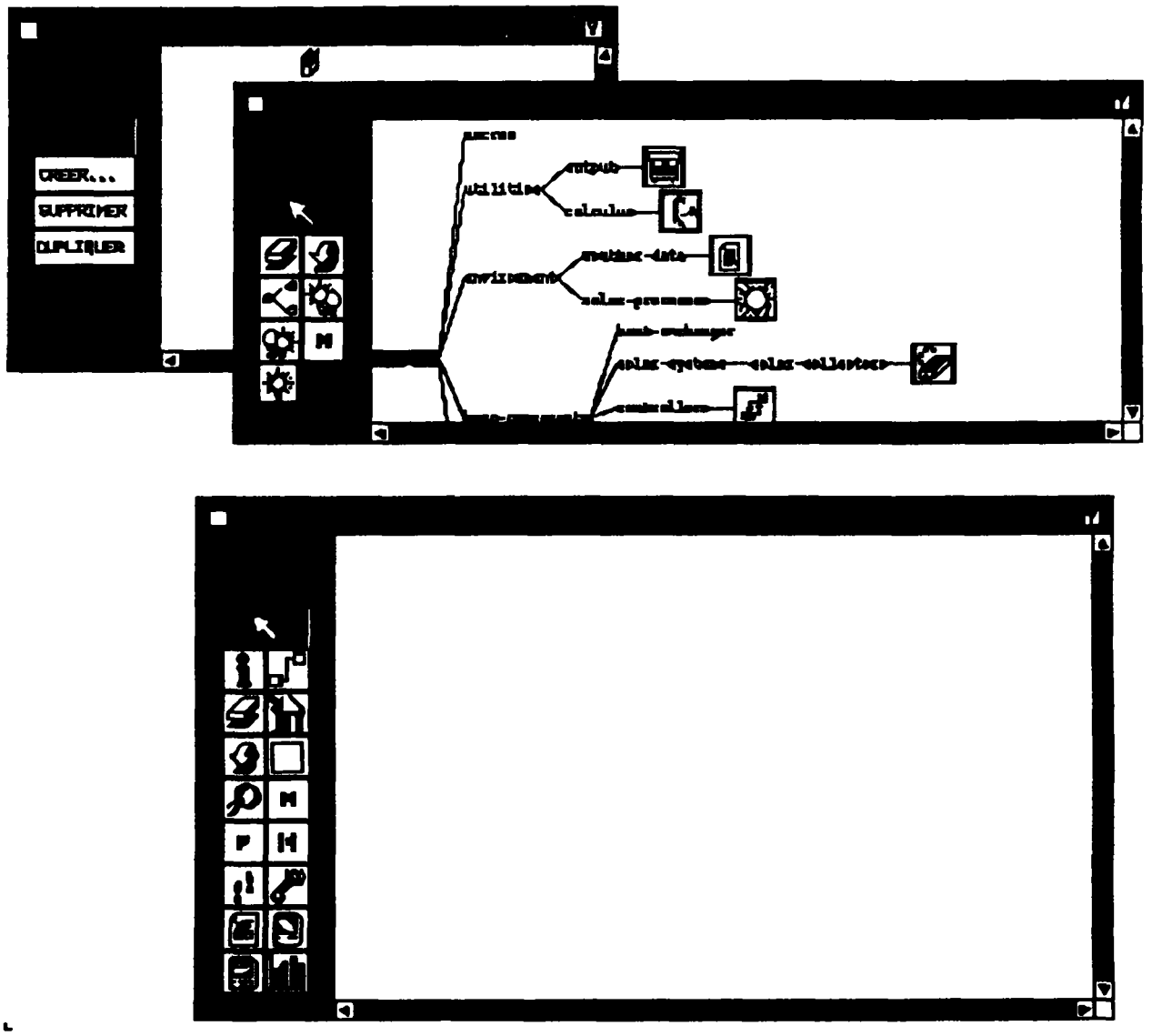
Schéma du système

Début de la session

L'utilisateur dispose de deux fenêtres : un gestionnaire de bibliothèques et un panneau d'assemblage.



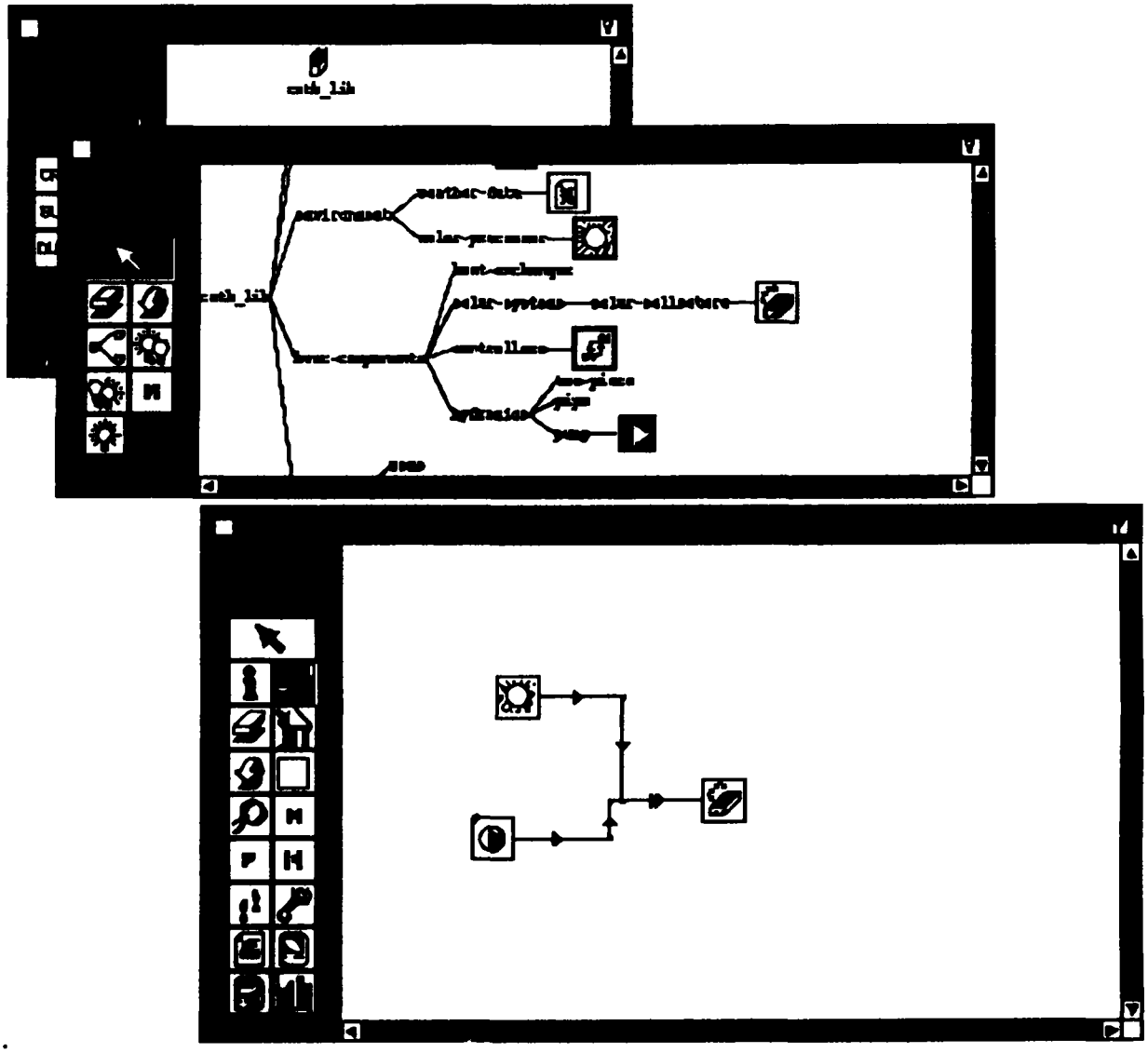
Ouverture d'une bibliothèque



Outil utilisé :

- OUVRIR (outil de la fenêtre "gestionnaire de bibliothèques") ; cet outil permet de visualiser une bibliothèque sous forme arborescente.

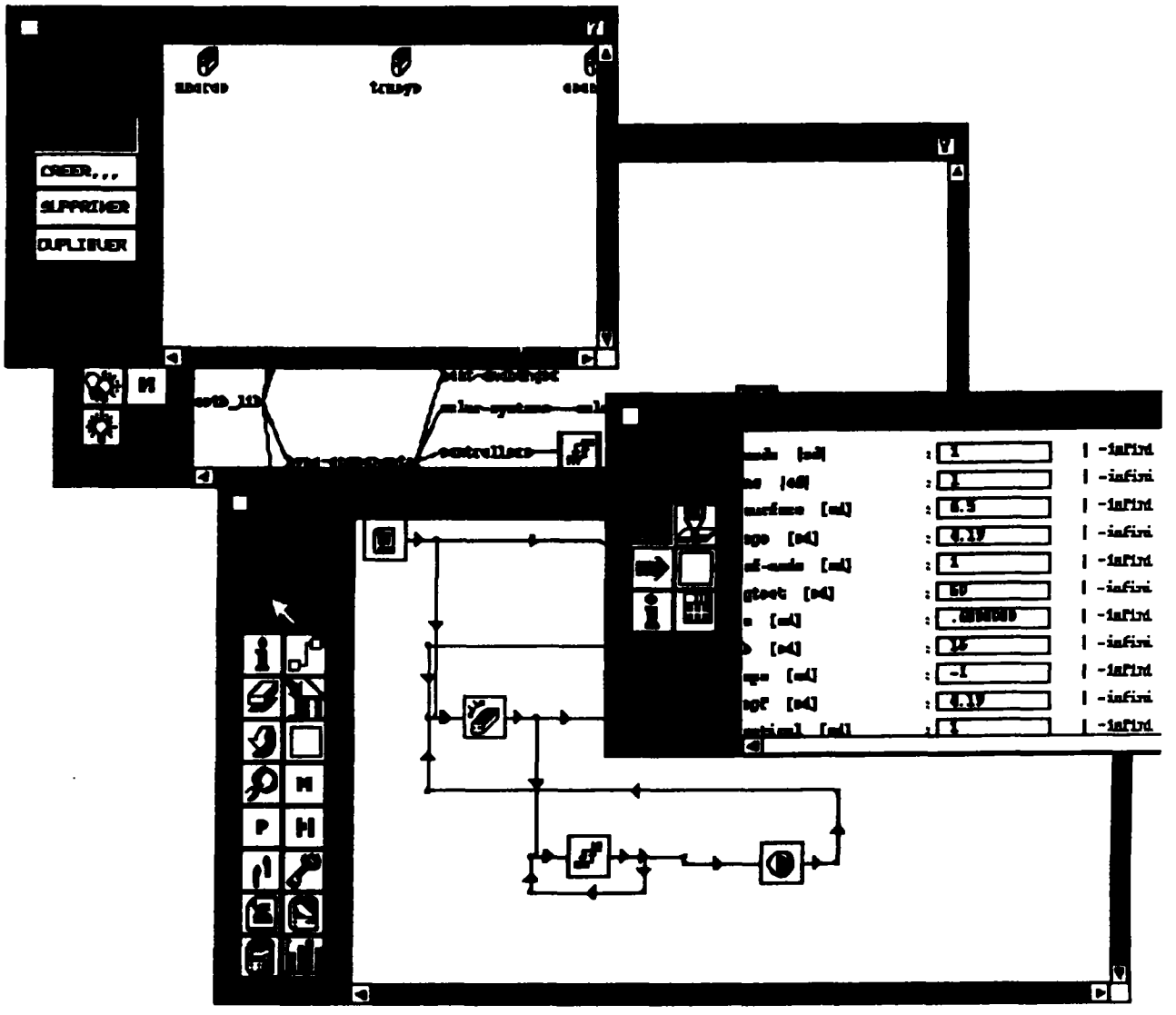
Instanciation des modules dans le panneau d'assemblage et création des liaisons



Outils utilisés

-  outil de nom "Multifonctionnel" du panneau d'assemblage ; cet outil permet d'instancier des modules ;
-  outil de nom "Lier" du panneau d'assemblage ; il permet de lier des modèles (les liaisons entre modèles sont représentées par des segments orientés et contenant des points activables).

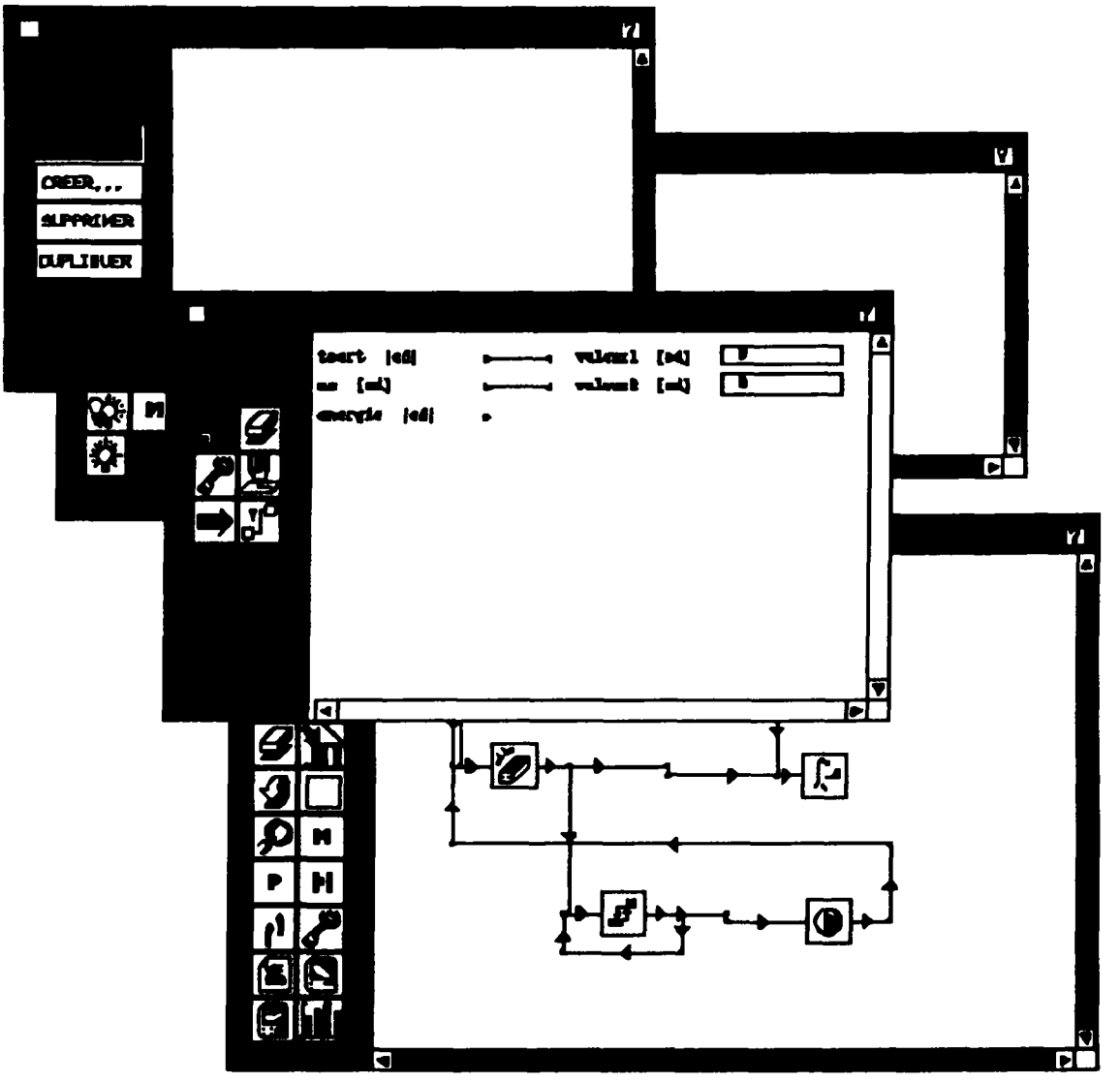
Introduction des paramètres



Outils utilisés

-  outil de nom "Multifonctionnel" du panneau d'assemblage ; cet outil permet d'ouvrir l'application spécialisée d'introduction des valeurs pour les paramètres ;
-  outil de nom "Sauver/Quitter" de la fenêtre spécialisée ; cet outil permet d'enregistrer les valeurs introduites par l'utilisateur.

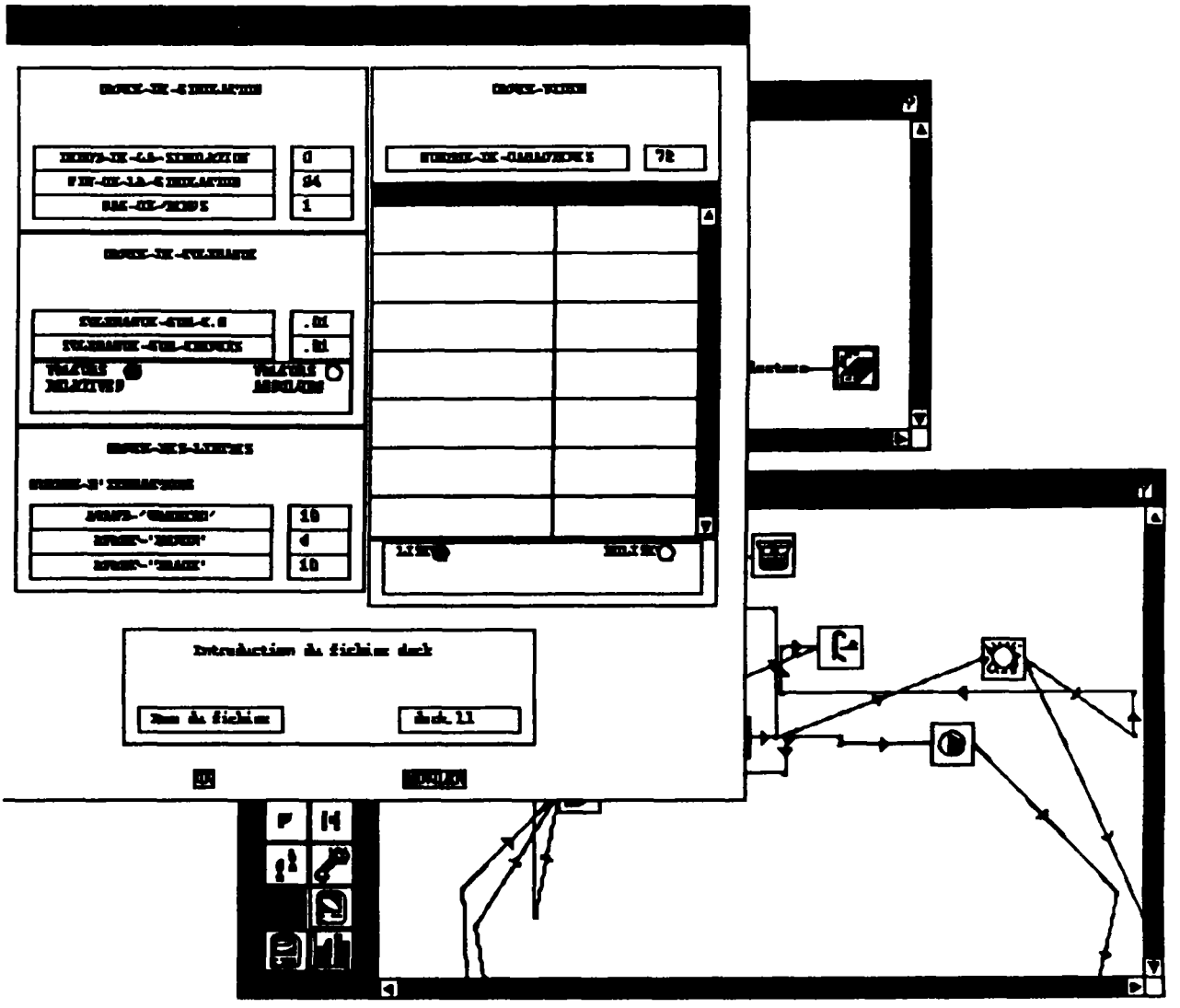
Connexion des variables



Outils utilisés

-  outil de nom "Lier" du panneau d'assemblage ; cet outil permet d'ouvrir l'application spécialisée qui permet de réaliser les connexions de variables ;
-  outil de nom "Connecter" de l'application spécialisée , cet outil permet de connecter les variables entre elles ;
-  outil de nom "Sauver/Quitter" de la fenêtre spécialisée ; cet outil permet d'enregistrer les valeurs introduites par l'utilisateur.

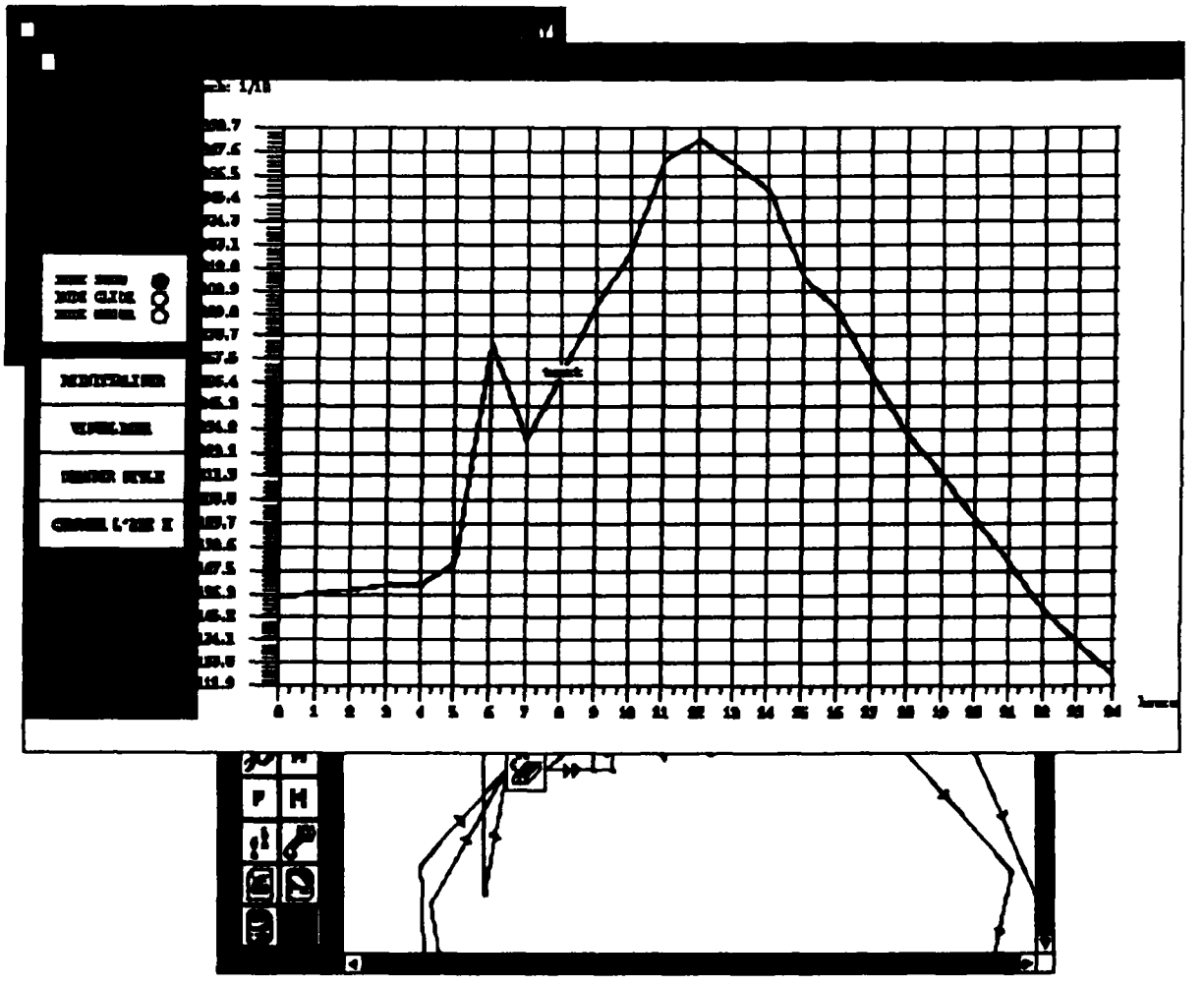
Introduction des cartes de contrôle



Outils utilisés

-  outil de nom "DECK" du panneau d'assemblage ; il permet de générer le fichier DECK relatif à l'assemblage présent dans le panneau d'assemblage (par simple sélection de cet outil).

Visualisation des résultats



Outils utilisés

-  outil de nom "Courbe" du panneau d'assemblage ; il permet de visualiser les résultats de la simulation ; pour cela l'utilisateur doit cliquer sur un utilitaire de type "imprimante".

Annexe 4

Approche générale du raisonnement approximatif

Approche générale du raisonnement approximatif

Cette démarche résulte des travaux concernant le système MYCIN [SHORTLIFFE 76] dans lequel a été mis en place un mécanisme empirique de gestion de *facteurs de certitude.*

A chaque règle est attribuée un coefficient numérique *cr* ; chaque élément d'une prémisse *condi* est connu avec un coefficient *cci*. La confiance accordée à la conclusion est calculée en combinant les coefficients *cr* et *cci*.

Les coefficients peuvent évoluer dans l'intervalle [-1 ; 1] :

- -1 correspond à sûr que faux ;
- 1 correspond à sûr que vrai ;
- 0 correspond à l'inconnu.

La première étape consiste à affecter à chaque prémisses un facteur de certitude *cp*, calculé en fonction des coefficients *cci*, et dépendant de la nature des connecteurs.

En général, les cas de prémisses complexes et de disjonctions ou de conjonctions de conclusions sont traités à partir des opérateurs *min* et *max*. On définit le calcul de *cp* par les formules suivantes :

- fc (P ou Q) ≥ max (fc (P), fc (Q))
- fc (P et Q) ≤ min (fc (P), fc (Q))

La fonction fc exprime une mesure de la probabilité.

Il faut ensuite décider si la règle est candidate ou non à l'application : les prémisses sont considérées comme suffisamment sûres à l'application lorsque *cp* dépasse un certain seuil compris entre 0 et 1 (par exemple 0.8).

Enfin, si la règle est sélectionnée, son application va conduire à l'affectation à la conclusion d'un coefficient *c*, dépendant à la fois de *cp* et de *cr*.

La partie conclusion va alors introduire un fait nouveau dans la base de faits.

Deux possibilités peuvent se présenter :

- ou bien le fait est nouveau et il apparaît dans la base de faits avec le coefficient *c* ;
- ou bien il était déjà présent avec un coefficient *c'* (résultant de l'application antérieure d'une règle) et l'application de la règle substitue à *c'* un coefficient *c''* dépendant à la fois de *c* et de *c'*.

Exemple de calcul de l'incertitude

Soit la règle suivante :

Si cond1 et (cond2 ou cond3) alors conclusion (0.8)
avec cc1 = 0.85 ; cc2 = 0.9 ; cc3 = 0.7 et cr = 0.8

Le coefficient cp est calculé par application de la fonction max pour les disjonctions et min pour les conjonctions :

cp = (min 0.85, (max 0.9, 0.7)) = 0.85 > 0.8

A l'application de la règle, cp et cr sont combinés pour donner :

c = cp * cr = 0.8 * 0.85 = 0.68. C'est la confiance accordée à la conclusion. Si celle-ci est un intermédiaire dans le raisonnement, ce coefficient va se propager vers les nouvelles conclusions qui en dépendent.

Supposons que la conclusion en question était déjà connue avec le coefficient c' = 0.75. Par deux voies de raisonnement, on a deux tendances qui vont dans le sens de la confirmation de la conclusion. Les deux valeurs positives sont combinées de façon à renforcer la confiance accordée à la conclusion par la formule :

c" = c + c' - c * c' = 0.92.

Les formules générales de calcul sont :

- c" = c + c' - c * c' si c et c' sont tous deux positifs ;
- c" = c + c' + c * c' si c et c' sont tous deux négatifs ;
- c" = (c + c') / (1 - min (|c|, |c'|)) sinon.

Evaluation

Cette technique de calcul assure la propagation de l'incertitude, et permet de "peser le pour et le contre " dans la prise de décision. Cependant, la difficulté à résoudre consiste à définir les coefficients pour chaque règle.

Il convient aussi de soulever le problème de l'indépendance des voies de raisonnement aboutissant à une même conclusion ; en effet il n'est pas toujours souhaitable de renforcer une conclusion, obtenue à partir de plusieurs voies interdépendantes.

Annexe 5

Un exemple d'interface non graphique

A5-1 Fonction

Le logiciel RC est une interface écrite en Fortran 77. Selon une description formelle littérale, cette interface permet de décrire un réseau résistances-capacités (en principe thermique) et de former les matrices d'états correspondant à une description qui est utilisable par un module de simulation du logiciel CSTBât (TYPE 50).

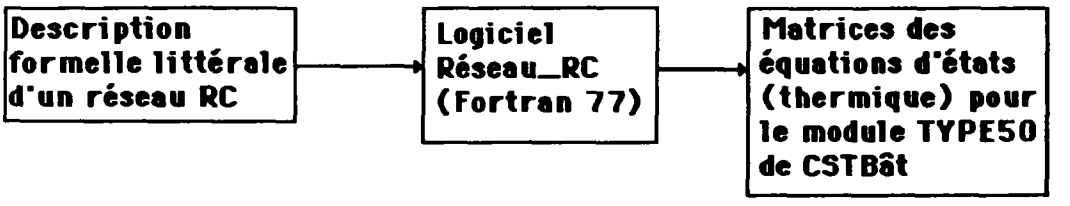


Figure A5-1

A5-2 La modélisation analogique

Décrire un système thermique par son réseau analogue, comportant des capacités, des conductances, des sources de courant ou de tension, est une démarche très courante dans le monde de la modélisation. La modélisation analogique se fonde sur la discrétisation de l'espace, elle a pour hypothèse fondamentale la décomposition de tout système physique en un nombre fini d'éléments de volume, supposés isothermes.

La figure A5-2 représente un exemple de discrétisation de l'espace pour une paroi opaque : le mur est divisé en n tranches de même épaisseur Δx.

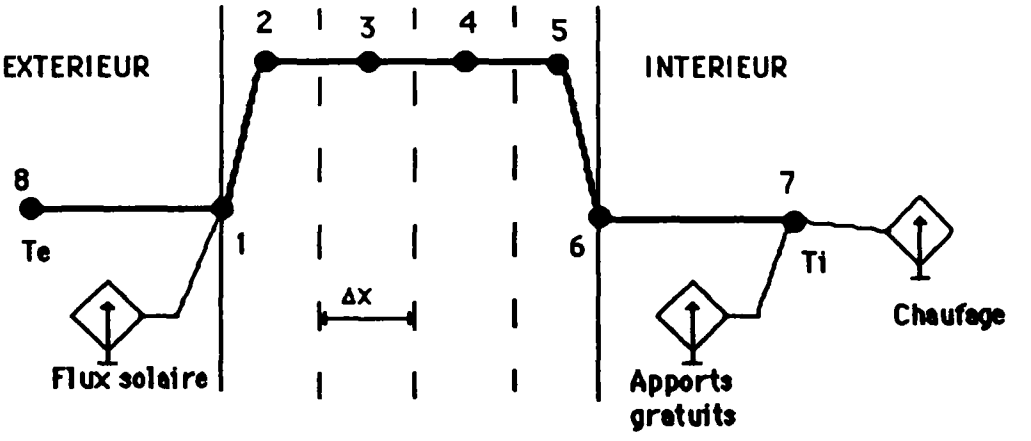


Figure A5-2 : Modèle analogique d'une paroi

Chaque tranche est symbolisée par un noeud (2 à 5 sur la figure 1-2), caractérisé par une température (celle supposée uniforme de la tranche considérée) et par une capacité calorifique.

Le noeud 7 représente l'ambiance intérieure, la capacité affectée à ce noeud rend compte de l'inertie intérieure du local (mobilier).

Les noeuds surfaciques 1 et 6, ainsi que le noeud 8 représentatif de l'ambiance extérieure, ont une capacité nulle.

Certains noeuds peuvent recevoir directement un flux de chaleur, non lié au transferts thermiques à travers le mur : les noeuds surfaciques extérieurs absorbent un flux solaire, ainsi que les noeuds surfaciques intérieurs (prise en compte des vitrages).

L'air du local reçoit un flux d'apports dits gratuits ainsi que le flux de chaleur provenant de l'installation de chauffage.

Modèle mathématique

Le système d'équations régissant ce mur, traduisant la conservation de l'énergie à un noeud i de la paroi, se ramène à un système d'équations de type :

$$Ci \frac{dTi}{dt} = \sum_{j.ne\,i}^{n} Gij\,(Tj - Ti) + Qi$$

Avec :

- T, la température du noeud ;
- Gij est la conductance entre les noeuds i et j ;
- $\sum_{j.ne\,i}^{n} Gij\,(Tj - Ti)$, représente le flux sortant ou entrant du noeud j ;
- Ci est la capacité calorifique du noeud i ;
- Ci dTi représente l'augmentation de l'énergie emmagasinée par le noeud i ;
- Qi représente les apports directs sur le noeud i.

Si l'on suppose que toutes les parois extérieures du local (au nombre de NPAROIS) sont découpées en n tranches, on obtient à un système d'équations de taille [NPAROIS * (n + 2) +1], à [NPAROIS * (n + 2) +1] inconnues. Celles-ci sont les températures des noeuds autres que les noeuds aux limites :

- températures des surfaces ;
- températures des noeuds capacitifs modélisant les parois ;
- température intérieure du local.

Résolution numérique

Pour résoudre ce système, suivant la manière de discrétiser le temps, on obtient trois types de méthodes de résolution numérique :

- méthode explicite ; les températures au pas de temps t + Δt se calculent directement à partir des températures au temps t ;
- méthode implicite ; la température d'un noeud au pas de temps t + Δt se calcule en fonction de sa température en fonction de sa température au temps t, mais aussi en

fonction des températures des noeuds voisins au temps t + Δt ; il faut dans ce cas que les conductances soient indépendantes de la température dans l'intervalle de temps considéré ;

- méthode semi-explicite ; la température d'un noeud à t + Δt est fonction des températures de celui-ci au pas de temps précédent, et des noeuds voisins aux temps aux pas de temps t et t + Δt.

La méthode explicite offre l'avantage de ne pas nécessiter de résolution de systèmes d'équations couplées, d'où une place mémoire exigée relativement faible ; elle présente l'inconvénient de ne pas être absolument convergente. Les deux autres méthodes imposent de résoudre un système d'équations couplées (en général par inversion matricielle), et par conséquent occupent une place mémoire plus élevée. Elles sont, quant à elles, inconditionnellement convergentes.

A5-3 Principe de l'interface

Celle-ci est constituée de deux rubriques principales obligatoires indiquées par les mots-clés suivants, à savoir **NOEUDS** et **CONDUC**, et d'une rubrique facultative indiquée par le mot-clé **VENTIL**. Tout mot-clé doit être précédé d'un point en première colonne. La description du réseau RC s'effectue grâce à l'introduction de constantes numériques séparées par des blancs. Des lignes de commentaires peuvent être introduites en plaçant une astérisque en première colonne. A noter que l'interface ne tolére pas de lignes blanches.

La rubrique .NOEUDS

Il s'agit d'introduire au sein de cette rubrique sur une ligne de 80 colonnes le nom du noeud (différent de la constante 0), son type, les valeurs de la capacité (J/K) et de la condition initiale (°C) s'il s'agit d'un noeud capacitif, ceci n fois (n étant le nombre de noeuds du réseau à étudier).

Les noms des noeuds sont des constantes entières comprises entre 1 et 70. A noter que seules les quatre premières valeurs se trouvant sur une ligne sont prises en compte.

Le type de noeud est indiqué par les constantes 0, 1 et 2 :

- 0 désigne un noeud représentant une condition limite ;
- 1 désigne un noeud sans inertie ;
- 2 désigne un noeud capacitif.

Il n'est pas nécessaire d'introduire les constantes numériques nulles.

La rubrique .CONDUC

Il s'agit d'introduire sur une ligne de 80 colonnes le nom de la conductance (différent de la constante 0), les noms des deux noeuds liés par cette conductance et la valeur de cette conductance (W/K).

Les noms des conductances sont des constantes entières comprises entre 1 et 70.

La rubrique .VENTIL

Celle-ci permet d'introduire les boucles de ventilation. Il s'agit tout d'abord d'introduire le nom de la boucle (obligatoire) en utilisant les lettres majuscules de l'alphabet, les chiffres et le symbole " _ ". Ensuite, il s'agit d'introduire le débit de ventilation (m3/hr) et les noms des noeuds constituant la boucle de ventilation (maximum 40). Il n'y a pas de limitation quant au nombre de boucles.

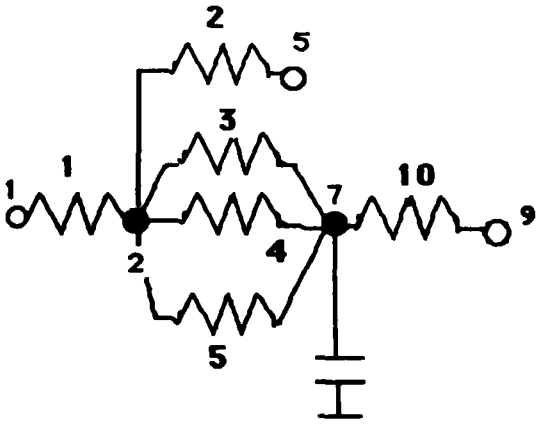
Détection des erreurs

Les erreurs détectées sont les suivantes :

- symbole inconnu (autre que constantes numériques, blancs et points) ;
- existence d'une ligne blanche ;
- une des rubriques obligatoires non introduite ;
- noeud non défini (noeud existant dans .CONDUC ou dans .VENTIL et non défini dans.NOEUDS) ;
- un type de noeud non défini ;
- même nom à plusieurs noeuds ou à plusieurs conductances ;
- nom inapproprié ;
- conductance liant le même noeud ;
- conductance liant deux noeuds conditions aux limites ;
- noeud isolé : noeud existant dans .NOEUDS et non dans .CONDUC ;
- discontinuité du réseau ;
- régles de cohérence pour les boucles de ventilation non respectées.

A5-4 Exemple

La figure A5-3 montre un réseau RC décrit littéralement grâce à l'interface.




```
.NOEUDS
1
9
5
2 1
7 2 120000 15
.CONDUC
1 2 1 61.47
2 2 5 42.9
3 2 7 474.
4 2 7 8.5
5 2 7 55.25
10 7 9 53.3
```

Figure A5-3

Mise en route

Le programme demande à l'utilisateur le nom du fichier (32 caractères) dans lequel sont stockées les données relatives à la description du réseau, le nom du fichier (32 caractères) dans lequel il veut récupérer les résultats et s'il désire des valeurs stationnaires équivalentes au cours de la simulation (l'utilisateur doit répondre oui ou non).

Le "fichier résultat" contient les matrices nécessaires à l'utilisation du type 50 du code de simulation CSTBât.

Annexe 5 : Un exemple d'interface non graphique

Annexe 6

Formalisme de De Kleer & Brown

Formalisme de De Kleer & Brown [HATON 91]

Le formalisme utilisé pour décrire un système est celui qui consiste à écrire des équations différentielles qualitatives. Expliquons cette notion à travers l'exemple du régulateur de débit (cf. § 3-2-4-2).

Considérons tout d'abord un système élémentaire constitué d'un tuyau. Un modèle simple du comportement d'un fluide s'écoulant dans ce tuyau peut être donné par l'équation :

$$Q = C.A.\sqrt{\frac{(2.P)}{Ro}}$$

avec :

Q : flux à travers le tuyau ;
P : différence de pression entre les extrémités du tuyau ;
A : section du tuyau ;
C : coefficient de pertes de charge ;
Ro : masse volumique du fluide.

Des déductions simples telles que " si P augmente Q augmente " ou " Q augmente si A augmente " sont évidentes. L'équation qualitative régissant ce modèle s'écrit de la façon suivante :

$$[\,dP\,] + [\,dA\,] - [\,dQ\,] \approx 0 \qquad (1)$$

Les équations qualitatives régissant le régulateur peuvent s'écrire :

$$[\,dP1\,] - [\,dP2\,] - [\,dQ\,] \approx 0 \qquad (2)$$
$$[\,dP2\,] - [\,dP3\,] - [\,dQ\,] + [\,dA\,] \approx 0 \qquad (3)$$
$$[\,dP3\,] - [\,dP4\,] - [\,dQ\,] \approx 0 \qquad (4)$$
$$[\,dP4\,] - [\,dP5\,] - [\,dQ\,] \approx 0 \qquad (5)$$
$$[\,dP4\,] + [\,dA\,] \approx 0 \qquad (6)$$

La résolution de ce système qualitatif revient à résoudre un système de la forme A X = B où :

- X et B sont des vecteurs qualitatifs,
- A est une matrice qualitative.

La résolution qualitative stipule que l'on peut additionner ou soustraire deux équations et éliminer une variable pourvu que l'on n'en élimine qu'une seule. Si l'on applique cette technique au système d'équations relatif au régulateur, on obtient l'équation suivante liant les pressions aux extrémités et la pression au niveau de la vanne :

$$[\,dP1\,] - [\,dP2\,] + [\,dP5\,] \approx 0 \qquad (7)$$

Il s'agit maintenant d'interpréter ces résultats :

on voit que si P1 augmente alors P2 augmente. On remarque aussi que si P1 et P5 augmente alors P2 augmente. Ce résultat n'est pas une surprise. En effet, l'explication vient du fait que la pression en 2 augmente via le conduit (1, 2) du fait de l'augmentation

de la pression en 1. De plus, l'augmentation de P5 a tendance à accroître P2 via les conduits connectés (2, 3), (3, 4) et (4, 5). Par ailleurs, l'augmentation de P5 a tendance à abaisser la soupape, ce qui a encore tendance à augmenter la pression en P2.

Ceci démontre qu'il est possible de déduire de nouveaux liens entre certains paramètres, seulement à partir de la résolution d'un système d'équations qualitatives.