Chapitre 3

Objectifs de Motor-2

3.1 Un simulateur « idéal »

Compte tenu des problèmes mentionnés dans le chapitre précédent, on peut tenter d'imaginer un environnement de simulation idéal. Cet environnement devrait soutenir toutes les phases d'une étude de système comme nous les avons présentées. L'objectif d'un outil logiciel est d'alléger et rendre plus efficace le travail d'un ingénieur de simulation. Ce programme doit couvrir les deux phases principales, la modélisation et la simulation.

Pour la phase de modélisation, l'environnement doit permettre de développer de nouveaux modèles et de gérer les modèles existants. Les modèles existants peuvent aider à la création de nouveaux modèles de deux manières. D'une part, ils peuvent servir comme patron pour le développement de modèles entièrement neufs. D'autre part, ils peuvent être des points de départ pour les modèles qui ne se distinguent que peu et qui sont dérivés des modèles existants. Sans trop restreindre la liberté de formuler l'algorithme pour le nouveau modèle, l'environnement peut proposer un cadre pour la représentation des modèles. Nous avions déjà mentionné les PROFORMAS qui poursuivent cet objectif. Des outils logiciels vérifient l'adhésion au cadre prescrit et la cohérence interne d'un modèle. À titre d'exemple, ils peuvent tester les unités des variables utilisées dans les équations (voir [7, chap. 2.7]).

Quant à la deuxième phase, la simulation, c'est une application classique de l'ordinateur. Comme nous l'avons vu au chapitre précédent, il y a déjà beaucoup de logiciels qui traitent cette phase.

Un bon environnement de simulation peut guider l'utilisateur dans son choix de modèles pour le problème donné. La description du système étudié doit être à la fois facile, pour garder une bonne vue d'ensemble, et souple, pour pouvoir facilement modifier une configuration en vue de comparaisons. Un mécanisme de contrôle peut vérifier les connexions entre les modules, la cohérence des modèles appliqués et la cohérence entre les modèles et les connexions.

Pour la résolution numérique du système, on peut imaginer un logiciel qui prépare les modules et leurs connexions d'un manière à rendre les calculs qui

suivent les plus efficaces possible. Ces calculs sont effectués pour résoudre le système. L'environnement doit donc proposer un algorithme de résolution qui soit rapide et adapté aux types de modèles. Beaucoup de difficultés numériques peuvent se présenter pendant la résolution. Un environnement idéal doit les reconnaître et aider à les surmonter.

Le temps d'exécution d'une simulation reste toujours un problème ouvert. Un utilisateur veut toujours ses résultats le plus vite possible. Le programme idéal de simulation est portable sur différents types de machines et il est rapide.

Aujourd'hui, les interfaces graphiques sont devenues indispensables. À toute phase d'interaction de l'utilisateur avec l'environnement logiciel, il faut une interface facilement compréhensible, que l'on peut piloter par des actions simples comme « *cliquer* » avec le bouton d'une «*souris* ». Ceci concerne aussi bien le développement de nouveaux modèles, la saisie de la structure du système étudié et des paramètres des modules utilisés que le traitement des résultats comme la présentation des graphes et des courbes.

Il y a quelques années, le projet SYMBOL a été formalisé et mis en chantier dans le groupe GISE. Parmi quelques autres objectifs, on y trouve aussi le développement d'un environnement de simulation destiné aux problèmes thermiques de bâtiment.

3.2 Cadre : le projet SYMBOL

3.2.1 Présentation générale

Les principales applications du projet SYMBOL concernent les systèmes complexes de grande taille, comme les bâtiments. Les objectifs de ces travaux sont premièrement l'amélioration de la connaissance des systèmes thermiques par la modélisation et l'analyse, et deuxièmement l'offre d'un ensemble modulaire et cohérent d'outils logiciels. SYMBOL est un acronyme qui signifie SYnthèse Modale et Boîte à Outils Logiciels et évoque les deux axes principaux du projet.

L'*analyse* et la *synthèse modale* reposent sur le fait qu'un système thermique linéaire peut être représenté avec précision par un modèle modal d'ordre faible. On peut obtenir un tel modèle par diagonalisation et réduction d'un modèle numérique discret détaillé, par réduction directe d'un modèle analytique ou par des méthodes heuristiques ([7, chap. 4], [62]). Si le couplage entre des sous-systèmes est linéaire aussi, la *synthèse modale* permet de trouver directement le modèle modal de l'ensemble en fonction des modèles réduits des sous-systèmes (voir [36]).

Le deuxième axe du projet SYMBOL concerne la « Boîte à Outils Logiciels ». L'apparition des nouveaux concepts logiciels et outils de programmation ont initié cette nouvelle approche de la modélisation dans le projet. Un des objectifs du projet SYMBOL est de mettre à dispositions des ingénieurs thermiciens ces nouveaux moyens. On se place à leur niveau pour la présentation des outils logiciels

comme des codes de calcul et des logiciels qui seront développés dans ce cadre. Ces outils doivent se présenter au thermicien d'une façon naturelle et conforme à son environnement habituel. Leur intérêt est d'alléger la tâche de l'ingénieur en évitant les nécessités habituelles d'un travail sur ordinateur. On veut lui épargner les questions qui se posent normalement au physicien théorique, à l'informaticien, au mathématicien ou au numéricien. C'est la *capitalisation* des connaissances sous forme de programmes et données informatiques.

Ces objectifs généraux du deuxième axe du projets SYMBOL sont abordés à l'aide des idées de la *modularité* et de la *visibilité*. Modularité et visibilité sont appliquées à travers toutes les phases d'une étude systémique mentionnées précédemment (§ 2.1.4).

Modularité

Parmi les conclusions que l'on peut tirer du chapitre précédent, on voit qu'il est souhaitable de pouvoir connecter facilement des morceaux de programmes afin d'en créer de nouveaux. Ces morceaux peuvent provenir de sources différentes, et peuvent être partagés entre plusieurs études. Des conventions relatives à la forme des programmes développés dans le projet SYMBOL ont conduit à une architecture informatique modulaire dans laquelle on peut connecter et substituer des modules écrits pour des applications *a priori* différentes (voir plus loin § 3.2.2). On trouve dans l'environnement SYMBOL d'une part des modules plus ou moins généraux comme par exemples les descriptions de composants d'un système qui sont dans un format précis. Ces descriptions ne sont pas spécifiques à une application particulière. Elles peuvent éventuellement être réemployés pour d'autres applications. D'autre part, il existe dans SYMBOL des modules qui sont spécifiques à certaines tâches, c'est à dire, adaptés à un contexte particulier. Les fichiers décrivant les modes de simulation pour le logiciel Motor-2 en sont un exemple.

Cette modularité encourage fortement une réutilisation des morceaux existants. Cette utilisation fréquente des modules induit des modules plus sûrs, mieux conçus, mieux documentés et surtout validés grâce à un plus important effort initial et continu globalement sensible. [1]

Visibilité limitée

Parmi les objectifs du projet SYMBOL se trouve avant tout la mise à disposition des résultats de recherche appliquée au monde des ingénieurs thermiciens. Pour diffuser les méthodes et algorithmes développés par les chercheurs, il faut souvent les réécrire et les remettre en forme. Au cours des travaux de recherche ceci a souvent déjà été fait, au moins en partie. Notre object est de réutiliser

1. Il faut remarquer que l'amélioration des modèles ne peut pas être garantie par une utilisation plus fréquente. Certaines modifications qui sont une « amélioration » pour un modèle donné, peuvent poser des problèmes dans un autre contexte. La validation et l'amélioration dépendent fortement de l'environnement d'usage.

directement ces résultats déjà existants. La séparation des rôles d'un utilisateur et d'un développeur de méthodes dans la conception du projet SYMBOL et Motor-2 a pour but de découpler les démarches et de faciliter la transmission de connaissances. Un des moyens pour l'atteindre est la notion de *visibilité*. Il existe des règles pour spécifier quelle est la part d'un module qui peut être vue par les autres. C'est une sorte d'interface qui définit d'une manière abstraite le comportement du module, une *spécification de ce qu'il fait*. Seule cette partie est accessible par le reste de l'environnement. L'implémentation de ces spécifications, *comment il fait*, est totalement cachée. Des modifications éventuelles dans cette partie n'ont pas de conséquences (directes) sur le continu des autres modules. Cette séparation entre spécifications et implémentations traduit une visibilité limitée sur les modules. Elle augmente la modularité et permet des échanges de modules sans difficulté. Elle facilite en même temps la maintenance et le développement de modules, et de cette manière la diffusion de ces derniers.

Le concept de visibilité limitée est aussi utilisé dans la nouvelle approche de programmation *orientée-objet* que nous allons traiter plus loin (voir § 4.3). L'idée de séparation entre différentes couches descriptives sera reprise et élargie dans la présentation des niveaux d'abstraction (§ 4.1), mais d'abord nous allons regarder l'architecture globale du projet SYMBOL et ce qui concerne la « Boîte à Outils Logiciels ».

3.2.2 Structure de l'environnement SYMBOL

L'environnement SYMBOL comprend aujourd'hui plusieurs modules logiciels autonomes. La communication entre les programmes passe en général par des fichiers et est assurée par une syntaxe spéciale basée sur les idées de modularité et visibilité[2].

Si on regarde la figure 3.1, on peut constater que tous les programmes s'organisent autour de bibliothèques. Ceci souligne le rôle important que jouent ces bibliothèques dans le projet SYMBOL. La gestion d'un ensemble de modèles (mais éventuellement aussi de module) passe par une « modélothèque », une sorte de bibliothèque de modèles. Ici on stocke les informations relatives aux modèles. Plus particulièrement on spécifie la partie visible d'un modèle, son interface. Différentes implémentations pour une même spécification peuvent être disponibles. Comme dans un jeu de construction on peut choisir les briques pour les connecter et pour construire une représentation du système étudié.

Néanmoins on retrouve dans la conception du projet SYMBOL et dans son architecture (fig. 3.1) un chemin principal qui part d'une description de système, passe par un logiciel d'analyse ou de simulation pour finalement obtenir des résultats. Regardons cette figure plus en détail.

Au milieu se trouvent les bibliothèques pour les différents types d'éléments

2. une description complète de la structure des fichiers SYMBOL se trouve en B.1.

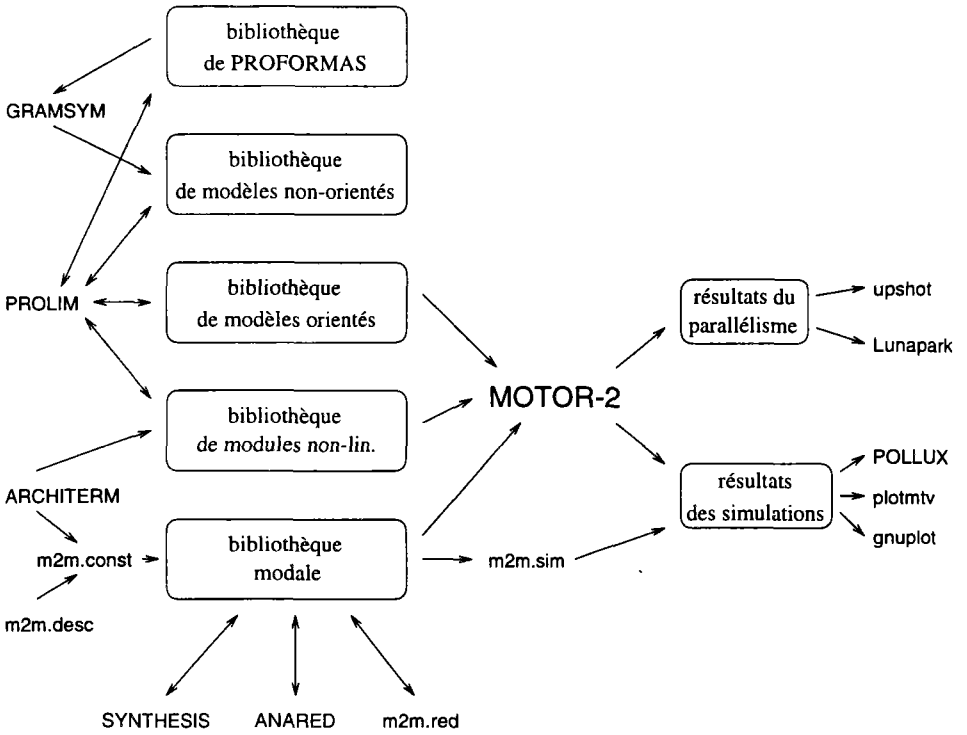

FIG. 3.1 - *La place de Motor-2 dans la structure du projet SYMBOL.*

du projet. Les interactions que l'on peut avoir avec une bibliothèque sont :

- l'interrogation de son contenu avec la possibilité de choisir un élément pour son application,
- des actions de maintenance comme ajouter, modifier ou enlever un élément,
- et finalement préparer et effectuer des transitions entre les bibliothèques. C'est à dire spécifier des paramètres et d'autres informations pour passer d'une bibliothèque générale à une plus spécifique.

Toutes ces actions sont des actions typiques d'une banque de données. Un prototype du gestionnaire de bibliothèques a été déjà développé sous le nom PROLIM (PROforma LIbrary Manager [30]). Une adaptation d'un programme de banque de données comme par exemple ORACLE remplacera ce gestionnaire à moyen terme. Puisque les bibliothèques sont encore très petites et le logiciel gestionnaire n'est pas réellement opérationnel, les actions sur les bibliothèques sont le plus souvent effectuées « à la main ».

Le deuxième grand bloc de modules logiciels concerne la description d'un problème. L'utilisateur saisit son projet en spécifiant les types et paramètres de ses modules. La tâche la plus importante pendant cette phase est la description des liens entre les modules. Nous reviendrons sur ce sujet. Une interface

graphique peut faciliter énormément ce travail de description. Un premier outil a été développé dans cette direction: c'est le logiciel ARCHITHERM. Sur la base d'un tracé en plan de chacun des niveaux d'un bâtiment, ARCHITHERM en produit une description structurée. La composition des composants et les caractéristiques thermophysiques des matériaux sont gérées sous forme de bibliothèques auxquelles ARCHITHERM fait référence. ARCHITHERM crée finalement les fichiers nécessaires pour le code suivant, actuellement m2m, les sorties pour Motor-2 et ALLAN sont en cours d'implémentation.

Beaucoup de logiciels existent déjà pour traiter des modèles modaux. La saisie peut passer par m2m.desc ou ARCHITHERM. m2m est un ensemble de blocs logiciels permettant de modéliser, analyser, réduire et simuler le comportement thermique dynamique d'un bâtiment. m2m.desc construit la représentation modale non-réduite d'un bâtiment décrit dans un fichier d'entrée. D'autres programmes travaillant sur des modèles modaux existent comme SYNTHESIS [36] pour la synthèse modale, m2m.red pour la réduction modale, ou ANARED [62] pour la création automatique d'un modèle modal réduit.

L'application principale pour la simulation au sein du projet SYMBOL est le logiciels Motor-2. C'est le seul moyen existant dans SYMBOL pour faire une simulation déterministe d'un système non-linéaire[3]. Motor-2 prend une description du projet et construit sa propre structure interne du système. Cette description, initialement abstraite, est traduite sous forme d'une représentation exécutable. De cette façon nous obtenons un noyau de la résolution indépendant des modules. Finalement, il exécute une simulation déterministe du comportement du système où chaque module peut avoir sa propre « file d'exécution »[4] (voir § 4.4 pour une discussion du parallélisme dans Motor-2). L'utilisateur peut choisir les variables à observer pendant la simulation. Différents formats existent pour stocker ces résultats.

Les utilitaires gnuplot et plotmtv du domaine public permettent de visualiser les résultats sous différentes représentations comme des courbes simples où des surfaces en 3D et colorées. Certaines configurations 3D peuvent être affichées à l'aide du programme Pollux, un logiciel développé dans le groupe GISE.

Les résultats qui concernent l'exécution de la simulation par plusieurs files indépendantes et parallèles dans Motor-2 peuvent être examinés par deux utilitaires. Lunapark affiche sur l'écran en temps réel les modules actifs. Il permet d'observer directement à tout moment quel module est traité par Motor-2. Comme Lunapark peut piloter la vitesse d'exécution de Motor-2, on ne peut pas examiner les durées relatives des différentes files d'exécution. Toutes les informations nécessaires pour une telle étude peuvent être écrites dans un fichier. L'utilitaire Upshot qui est habituellement utilisé pour les grandes machines massivement parallèles, peut le lire et afficher tous les instants de lancement, suspension, réactivation ou terminaison de toutes les files d'exécution.

3. un modèle linéaire ou linéarisé peut lui, être mis sous forme modale et utiliser le simulateur m2m.sim.

4. dans la littérature anglophone on emploi habituellement les termes *thread* ou *task*.

3.2.3 Contraintes pour Motor-2

Le cadre SYMBOL impose un cadre pour le développement d'un programme de simulation qui veut s'y intégrer. L'environnement de simulation doit soutenir les mêmes objectifs et idées que le projet SYMBOL. Il doit contribuer à un environnement qui se veut facilement maîtrisable pour un ingénieur thermicien ; usage facile, fichiers d'entrée et de configuration compréhensibles. Les concepts de la modularité et d'une visibilité limitée doivent être respectés. Le programme de simulation Motor-2 accepte la représentation standard des modules dans l'environnement SYMBOL. Motor-2 peut effectuer des simulation d'un système en connaissant seulement les spécifications des modules. (bien entendu, les implémentations des modèles utilisés doivent être disponibles.)

À part les contraintes imposées par la conception de l'environnement SYMBOL, il existait aussi des conditions organisatoires. Pour rendre effective la capitalisation des morceaux de programmes, la décision a été prise d'utiliser le langage Ada. Ce langage permet une conception *orientée objet* sur laquelle nous revenons dans le chapitre 4.3. Ada est un langage qui soutient bien le travail en équipe et la réutilisation, car on trouve une traduction directe de la visibilité dans les paquetages Ada.

Au niveau matériel, le laboratoire GISE dispose surtout des stations de travail Sparc. Le développement de Motor-2 a eu lieu sur ces machines et le système d'exploitation Unix. Néanmoins le programme de simulation tourne aussi sur des PC.

Après cette définition du cadre de travail, nous présentons une vue globale de l'environnement Motor-2 avec une initiation aux concepts de base. Ces derniers seront approfondis dans chapitre suivant.

3.3 Le simulateur Motor-2

Le logiciel Motor-2 est un programme de simulation basé sur une structure modulaire. Il est surtout conçu pour des simulations de modules continus dont le comportement dépend du temps.

Le logiciel actuel a été précédé d'une maquette qui a été développée au début du travail. Nous rappelons quelques détails de cette maquette qui pourrons nous être utiles dans ce qui suit.

3.3.1 Historique

Déjà dans la première version, appelée MOTOR (la version actuelle s'appelle Motor-2), nous travaillions avec une description du système qui était basée sur un découpage hiérarchique (voir plus loin et le chapitre 4.2 pour une discussion détaillée de ce concept). Un arbre de décomposition d'un système introduit la modularité dans la description du problème.

Dans le souci de créer une image informatique la plus proche possible de la réalité, chaque objet du système modélisé est représenté par une structure informatique équivalente. Dans le cas de MOTOR, nous utilisions des paquetage Ada pour implanter une visibilité limitée. La partie des spécifications Ada précise les parties visibles du module. MOTOR est un programme spécifique qui prend la description arborescente du système et qui génère des fichiers contenant le code Ada d'un programme qui simule le comportement du système. Pour chaque système, MOTOR crée un code de simulation qui est une image fidèle du système physique avec les données et méthodes nécessaires[5].

La technique numérique utilisée est une méthode de relaxation. Elle est liée à un raccordement entre deux modules qui se base sur l'égalité des températures et compatibilité des flux de chaleur. On envoie des nouvelles valeurs des températures de connexion aux modules connectés. Les modules calculent leur nouvel état et le flux qu'ils reçoivent du raccordement. Le bilan des flux est effectué sur chaque connexion et MOTOR modifie la température jusqu'à ce que le bilan soit nul. La méthode utilisée nécessite que chaque module soit capable de calculer la dérivée du flux par rapport la température.

Limites de la maquette

L'approche de MOTOR était intéressante pour de premières expériences sur l'assemblage des modèles et la génération du code. Mais nous avons trouvé des limitations sévères à l'usage de MOTOR. Premièrement il est lié à un algorithme numérique particulier ; seul des raccordement température - flux de chaleur sous condition de DIRICHLET sont possibles. En plus, la méthode demande aux modèles des capacités spécifiques comme les calculs des dérivées. D'autre part, l'usage de MOTOR est rendu très lourd par la génération de code. Pour chaque composant du système un paquetage de procédure a été créé ce qui cause une multiplication des lignes de code. Une simple modification dans la structure ou un nouveau système que l'on voulait simuler nécessitait une nouvelle génération de code et une nouvelle compilation du programme.

3.3.2 Présentation de Motor-2

Dans la version actuelle, nous voulons également garder des unités informatiques qui soient des traductions directes des modules de notre système du monde technique. L'idée principale est d'avoir une représentation informatique qui est la plus proche possible d'une image de la réalité vue par un ingénieur thermicien. On peut dire que l'objet technique est encapsulé dans une structure informatique.

Comme déjà mentionné, l'analyse du système à simuler passe par une description structurée de celui-ci. On découpe récursivement le système en sous-systèmes jusqu'à ce que les sous-systèmes élémentaires ne soient plus découpables. C'est une approche naturelle et elle est utilisée – même si ce n'est pas

5. C'est l'origine du nom : MOdule generaTOR.

explicite – dans tous les environnements de simulation. Nous considérons chaque composant ainsi obtenu comme une *boîte noire*. Pris séparément, il est libéré des contraintes de son environnement, et n'a plus d'effet en retour sur les autres éléments du système. Nous appliquons une notion de visibilité et de raccordement aux modules. Pour cet objectif, nous utilisons nos définitions des *frontières* en ce qui concerne la communication des module et des *interfaces* en ce qui concerne le raccordement entre les modules.

Le découpage hiérarchique nous fournit un arbre de dépendances qui décrit les liens entre les modules. Afin que la collection de modules représente bien le comportement de l'objet réel dont le système est le modèle, il faut les *raccorder* en imposant les contraintes nécessaires. Pour une description pertinente d'un système il ne nous faut pas seulement des modèles et ses paramètres pour les différents composants, mais nous devons décrire également les types des connexions entre les composants. Une des particularités de Motor-2 est le fait que l'on garde cette structure arborescente dans une représentation interne pendant la simulation. La hiérarchie descriptive du système est gardée et exploitée pendant la résolution. L'algorithme de résolution ne travaille que sur les connexions locales dans un module composé.

Un des objectifs de l'environnement Motor-2 est une interface d'interaction avec l'utilisateur dans laquelle on peut travailler à un niveau technique sans s'occuper de la réalisation informatique interne. Ceci concerne la description du système, ses composants, mais aussi les connexions entre les composants. Dans ce but a été développé une structuration de la démarche pour la description des modules. Pendant la description du système à simuler, l'utilisateur associe un modèle existant à chacun des composants et des connexions.

Cette approche permet une grande modularité de notre environnement de simulation. Nous pouvons créer des bibliothèques de modèles de composant et de modèles de couplage. Ces bibliothèques se remplissent à chaque nouvelle étude. La description de modèle peut faire référence aux modèles existants pour exprimer une similarité et une dérivation entre les modèles.

Non seulement au niveau des modèles, mais aussi au niveau de l'implémentation, nous augmentons l'efficacité par une réutilisation de l'existant. L'approche de la *conception* et de la *programmation orientée objet* s'y prête particulièrement bien. Elle convient bien avec notre approche globale et elle aide à réaliser concrètement ces idées à l'aide d'un ordinateur. Cela nous donne un environnement « ouvert », c'est à dire extensible dans lequel l'utilisateur peut facilement ajouter et modifier les modèles des bibliothèques.

Pour mieux exploiter les ressources en puissance de calculs qui existent dans les réseaux d'ordinateur et dans les machines parallèles émergentes, nous associons à chaque composant du système son propre code exécutable et sa propre progression dans les calculs. On peut ainsi distribuer les calculs sur plusieurs processeurs et exécuter en parallèle des parties de la simulation.

Les quatre concepts fondamentaux qui ne sont que brièvement présentés dans ce paragraphe sont approfondis dans le chapitre suivant. Nous parlons

notamment de la description du système par *découpage hiérarchique* et des *différents niveaux d'abstraction*, de la réalisation sur ordinateur dans le projet Motor-2 par une *conception orientée objet* et son approche du *parallélisme*.