Architecture des logiciels et langages de modélisation

Piotr Breitkopf — Gilbert Touzot

Université de technologie de Compiègne

RÉSUMÉ. *Le présent document souligne les difficultés qui apparaissent lors du développement de logiciels de modélisation, puis propose quelques remèdes, certes non définitifs. Après un bref rappel du sens donné ici au mot "modélisation" et des particularités de ce domaine de recherche, nous décrirons les caractéristiques communes aux logiciels de modélisation ; nous détaillerons ensuite les différents problèmes qu'ils posent. Enfin, nous décrirons quelques solutions mises en place dans le cadre de l'architecture générale du logiciel SIC, dont les prototypes ont été mis au point à l'UTC depuis 1985, en collaboration avec quelques industriels et des laboratoires de Marseille, Grenoble, Montpellier, Poitiers.*

ABSTRACT. *The article points out major difficulties emerging in the software developement for computational engineering purposes. Solutions are suggesteed, some of them are not definitive. The common characteritics of existing programs are described ; the problems encountered are detailed. Finally the solutions adopted for the SIC program general architecture are presented. SIC is developed at UTC, since 1985 in cooperation with industry partners and scientific laboratories from Marseille, Grenoble, Montpellier, Poitiers.*

MOTS-CLÉS : *architecture des logiciels, modélisation, éléments finis.*

KEY WORDS : *software architecture, computational engineering, finite elements.*

1. La modélisation

1.1. *La modélisation, mais encore ?*

Nous savons tous que le mot "modélisation" revêt des significations très différentes selon les interlocuteurs ; par exemple :

• pour certains il s'agit de l'*élaboration de relations* entre les variables caractéristiques d'un système ou processus physique donné, capables de bien simuler le comportement de celui-ci dans un contexte donné ; en ce sens de très nombreux scientifiques font de la modélisation ;

• pour d'autres, la modélisation se confond avec la *simulation numérique*, c'est-à-dire avec la résolution d'équations d'évolution de la physique ;

• pour d'autres enfin la modélisation correspond à l'ensemble des activités qui permettent la création et la mise en œuvre sur un ordinateur de *maquettes virtuelles de systèmes physiques.*

Nous retiendrons ici cette dernière définition, en nous restreignant à la modélisation *utile pour l'ingénieur* dans sa démarche de *conception et d'optimisation de produits.* C'est ce que l'on nomme en anglais *"computational engineering"*. En ce sens, la modélisation peut faire référence à toutes sortes de tâches interdisciplinaires bien différentes de la simple simulation numérique, telles que :

• la description informatique des divers aspects de systèmes physiques complexes,
• la génération de familles de modèles d'un même système, de complexité variable, et répondant à des besoins différents,
• la création de nouvelles méthodes et formulations adaptées à la modélisation,
• l'amélioration de l'insertion de la modélisation dans le processus de conception,
• l'utilisation optimale, dans la conception de modèles, de l'expérience accumulée lors d'essais, d'expérimentations ou de modélisations,
• l'automatisation des tâches de préparation, d'exécution et d'exploitation des résultats de la modélisation par des techniques issues de la recherche en intelligence artificielle ou en calcul formel etc.

Ce sont autant de thèmes de recherche qui posent des problèmes de méthodologie, et qui n'ont avec le calcul numérique que des relations de bon voisinage, ou encore de client à fournisseur.

1.2. *La modélisation en sciences de l'ingénieur*

La modélisation consiste à simuler des systèmes physiques variés à l'aide d'un ordinateur, de manière à en étudier le comportement ou à en modifier certaines caractéristiques. Bien qu'utilisée dans de très nombreux secteurs de la recherche et de l'industrie, la modélisation présente des spécificités et une importance particulière en sciences de l'ingénieur. En effet, au cours de la conception et de l'optimisation de produits industriels, l'ingénieur fait de plus en plus souvent appel à la simulation sur ordinateur pour éviter la construction de maquettes et de prototypes, ce qui permet des économies substantielles, fournit des informations auxquelles on ne peut accéder directement par l'expérience, réduit parfois les risques liés à certaines expérimentations, et diminue la durée du cycle de conception des produits.

Les ingénieurs ont besoin de simuler des *systèmes physiques de nature très diverse* (pièces mécaniques, sous-ensembles de véhicules ou de machines, ouvrages de génie civil, circuits intégrés, systèmes de production, procédés de fabrication,

fours à induction, réacteurs biologiques, moteurs électriques...). Les *phénomènes physiques* modélisés sont également très variés : comportements mécanique, thermique, acoustique, électro-magnétique, effets de chocs, écoulements, transport de matière en suspension, propagation de fronts, apparition de discontinuités, ... Très souvent, plusieurs de ces *phénomènes interagissent*, et la prise en compte de leurs différents modes de couplage constitue l'un des défis actuels de la recherche en modélisation. Malgré cette grande diversité, la modélisation en sciences de l'ingénieur présente une *forte unité* ; elle suit une démarche générale unique qui s'adapte à chaque application particulière.

- Analyse de la physique du système étudié, sélection d'un sous-système modélisable et des grandeurs significatives, compte tenu de l'objectif visé ; choix d'hypothèses simplificatrices à partir de la connaissance à priori dont on dispose.
- Définition de ce qui est considéré comme connu et inconnu; élaboration d'un modèle mathématique "bien posé" et capable de simuler le comportement du système étudié.
- Élaboration d'un modèle informatique correspondant au modèle mathématique; choix d'un mode de description du système étudié (aspects géométriques et physiques).
- Implantation puis exécution du modèle informatique sur un ordinateur.
- Validation du modèle par comparaison avec d'autres modèles et avec l'expérience.
- Exploitation des résultats, par exemple sous forme graphique.
- Corrections ou améliorations successives du modèle.
- Etudes paramétriques, analyse de sensibilité, optimisation.

La modélisation en sciences de l'ingénieur est une *activité fédératrice* par nature. Elle fait appel aux diverses sciences de base de l'ingénieur (résistance de matériaux, mécanique des structures, des solides et des fluides, thermique, electricité, ...), ainsi qu'aux mathématiques appliquées et à l'informatique. La recherche en modélisation doit intégrer rapidement les nouveaux concepts et résultats issus de tous ces domaines, les adapter à ses propres besoins, puis les insérer dans des logiciels de plus en plus complexes. Ces logiciels de modélisation concentrent aujourd'hui et le savoir faire des organismes de recherche publique, ainsi que celui des entreprises, pour les rendre facilement utilisables par les concepteurs de produits nouveaux. Ces logiciels sont le plus souvent issus d'une collaboration étroite entre l'industrie et la recherche publique ; par exemple le logiciel NASTRAN est né à la NASA, puis a été développé par Mc Neal Swendler. SAMCEF est né à l'université de Liège et s'est ensuite répandu dans l'industrie aéronautique européenne par l'intermédiaire de la société SAMTECH.

La modélisation revêt aujourd'hui une *importance stratégique* pour de nombreuses entreprises petites et grandes, car elle a une influence directe sur la compétitivité de leurs produits. La recherche publique doit donc s'efforcer d'améliorer, par différents moyens, la qualité et l'efficacité des outils de modélisation dont dispose le pays, de manière à éviter une dépendance trop

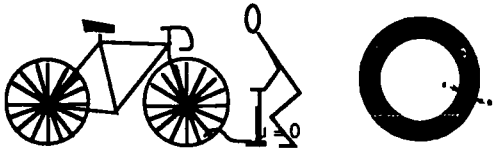
importante de l'industrie nationale vis à vis des logiciels de modélisation étrangers, cette dépendance risquant d'induire rapidement une *dépendance technologique et culturelle*. La recherche doit s'intéresser à la fois aux nouvelles méthodes utilisables en modélisation, à la méthodologie d'utilisation de la modélisation, à ses applications, ainsi qu'aux problèmes posés par la réalisation des logiciels de modélisation, pour en améliorer la fiabilité, la généralité et la facilité d'emploi.

La *mise en commun des efforts* de recherche, des méthodes et des outils informatiques entre les différents secteurs de la recherche en modélisation en sciences de l'ingénieur est très fructueuse : d'une part, chaque secteur bénéficie des percées et résultats obtenus dans les autres secteurs et d'autre part on évite ainsi de reproduire le coûteux développement de nombreux modules informatiques (interaction homme-machine, post-traitement graphique, résolution de grands systèmes d'équations, organisation des données, ...) et ainsi on favorise la construction de modèles couplés.

1.3. *La démarche de modélisation traditionnelle*

Un problème de modélisation est en général caractérisé par la *définition d'un système physique réel* dans lequel certaines grandeurs sont inconnues à priori, et d'autres sont supposées connues (exemple : une chambre à air de bicyclette soumise à une pression interne) et aussi par une ou plusieurs *questions* (par exemple : quelle pression maximale peut-t-on appliquer à cette chambre à air ?).

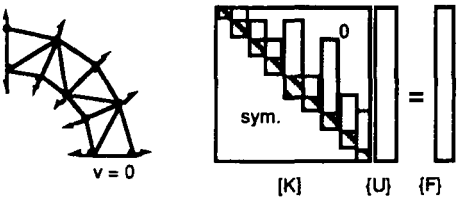
La première étape de la démarche de modélisation consiste à faire une série d'*hypothèses simplificatrices* qui rendent le problème "modélisable" : idéalisation de la géométrie, des conditions aux limites, des sollicitations, hypothèses sur le comportement des matériaux, sur l'amplitude des déplacements et des déformations, isolement d'un sous-système. Dans notre exemple, une première approximation grossière consiste à essayer de ne modéliser que la chambre à air en supposant qu'elle n'interagit pas avec la roue et le pneu.



La seconde étape consiste à *sélectionner les relations* qui régissent le modèle (loi de comportement élastique, relations déplacements-déformations, équations d'équilibre), à *éliminer certaines variables* entre ces relations, à faire des hypothèses simplificatrices (déformations "petites", sans doute fausses), puis à choisir des méthodes de *discrétisation* des équations ainsi obtenues (éléments finis, ...).

La création d'un tel modèle discrétisé utilise de nombreuses *connaissances explicites et implicites* de l'utilisateur : influence des conditions de symétrie du

problème, position d'éventuelles concentrations de contraintes, tailles d'éléments requises, erreurs induites par une finesse de discrétisation donnée, erreurs acceptables sur les différentes variables, etc. On peut ainsi construire un système d'équations algébriques (éventuellement linéaires) qui représente approximativement le comportement du système physique étudié. La résolution de ce système fournit les inconnues, ici les déplacements des nœuds de discrétisation de la chambre à air :



On effectue ensuite des calculs auxiliaires (contraintes, déformations...), puis on procède à une analyse critique des résultats (validité des hypothèses, vraisemblance des solutions obtenues, comparaison avec d'autres résultats analogues, ...) et à une interprétation de ceux-ci : comparaison des contraintes maximales calculées avec les contraintes de rupture du matériau, exploitation du caractère linéaire du problème pour en déduire le niveau de sollicitation admissible etc.

1.4. *Evolution de la modélisation en sciences de l'ingénieur*

La modélisation a en fait débuté par de *simples résolutions d'équations* à dérivées partielles destinées par exemple au calcul de champs de températures (équations de Laplace), de champs de déplacements, de déformations et de contraintes (équations de l'élasticité), de champs de vitesses (équations de Stokes). Les schémas de discrétisation spatiale initiaux étaient le plus souvent basés sur des différences finies.

Les premières évolutions de la modélisation ont porté :

- sur les *techniques de discrétisation spatiales* (formulations variationnelles et éléments finis variés, volumes finis, équations intégrales),
- sur l'augmentation du nombre de *dimensions d'espace* des modèles (1, 2 puis 3 dimensions), les applications tri-dimensionnelles étant très sévèrement limitées par la capacité des ordinateurs,
- sur les *méthodes de résolution* : méthodes itératives variées, méthodes directes, méthodes multi-échelles, méthodes de découpage en sous-domaines.

En parallèle, on a augmenté graduellement la *complexité des modèles physiques* :

- en mécanique des solides, on est passé de l'élasticité à la plasticité, à la visco-plasticité puis à l'endommagement ; on a également pris en compte les grands

déplacements, les grandes déformations, les instabilités ;

• en thermique, après la simple loi de Fourier, on a modélisé la convection, le rayonnement puis les problèmes de transport ;

• en mécanique des fluides, après les équations de Stokes sont venues les équations de Navier Stokes puis des modèles de turbulence de degré de raffinement croissant, puis la prise en compte de phénomènes réactifs.

Le caractère *non-linéaire*, *non stationnaire*, et parfois *non-différentiable* de certains de ces modèles a imposé la mise au point :

• de méthodes variées de résolution de grands systèmes d'équations non-linéaires (variantes de la méthode de Newton-Raphson, BFGS, ...),

• de méthodes de traitement des instabilités, bifurcations, catastrophes,

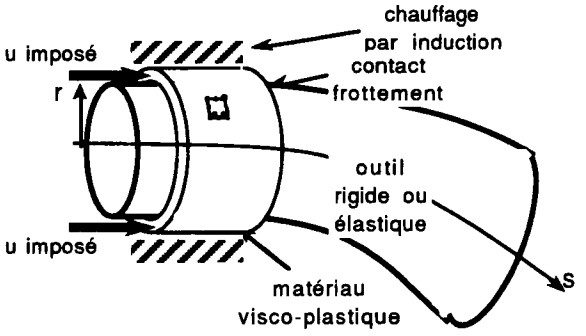
• de méthodes de traitement des relations non différentiables (par exemple, pour le contact et le frottement),

• de méthodes d'intégration de grands systèmes d'équations différentielles du premier et du second ordre (méthodes explicites puis implicites).

Dans la modélisation traditionnelle, on cherche à calculer des champs de variables qui vérifient des équations aux dérivées partielles ou des équations variationnelles équivalentes sur un domaine connu, ainsi que des conditions aux limites données. On a ensuite tenté de *modifier ce qui est considéré comme connu et inconnu* en ajoutant, lorsque c'est nécessaire, des relations additionnelles. Ainsi, pour certains problèmes d'optimisation, la forme du domaine est une inconnue que l'on peut déterminer en exprimant que la masse d'une pièce est minimale, c'est-à-dire que le gradient de la masse par rapport aux paramètres qui définissent la forme est nul. Dans un problème de contact, certaines des conditions aux limites sont inconnues, mais peuvent être déterminées grâce aux relations (inéquations) de contact (réaction normale de contact et distances en corps positives).

Enfin un domaine de recherche très actif est celui qui s'intéresse aux *couplages de modèles* : fluide-thermique, fluide-structures, acoustique-structures, électro-magnétisme-solides, etc. Il peut s'agir de couplages forts qui manipulent deux types d'équations et de variables dans un même modèle, ou de couplages faibles dans lesquels on se contente de traiter alternativement des modèles différents qui échangent de l'information par l'intermédiaire de champs de variables.

Un exemple de modèle couplé est fourni par l'optimisation du procédé de *fabrication de tubes coudés*. L'objectif est la détermination de la forme optimale d'un outil (mandrin), de manière à obtenir la forme de tube coudé souhaitée, à moindre coût. Un tel problème couple mécanique des solides visco-plastiques en grandes déformations, thermique non linéaire , électro-magnétisme (chauffage par induction), contact et frottement, optimisation.



De nombreuses méthodes numériques utilisent des *paramètres dont la valeur optimale dépend de la solution cherchée.* C'est en particulier le cas de la méthode des éléments finis (maillages), et des méthodes d'intégration d'équations différentielles par rapport au temps (taille des pas de temps). Pour fournir les valeurs de ces paramètres, l'utilisateur doit disposer d'une certaine connaissance à priori de ce qu'il cherche, ce qui n'est pas toujours le cas. Sinon il doit procéder par essais successifs. On commence à disposer aujourd'hui de *méthodes auto-adaptatives* qui utilisent les résultats de premiers modèles grossiers pour ajuster les valeurs des paramètres requis, au prix de ré-analyses successives.

2. Les logiciels de modélisation

2.1. *Modélisation et logiciels*

Dès qu'un modèle devient compliqué et dépend de plus de quelques paramètres, on doit faire appel à un ordinateur. Celui-ci se charge alors de l'acquisition des informations supposées connues, de la construction et de la résolution des équations, et de l'affichage des informations déterminées par le modèle. Il est évident que la programmation est d'autant plus longue et coûteuse que le modèle ou les méthodes de construction et de résolution des équations sont complexes. L'évolution de la modélisation, l'ouverture du spectre des problèmes que l'on désire simuler avec le même logiciel pour faciliter les couplages et amortir le coût de développement de celui-ci, ainsi que des considérations commerciales ont provoqué une *augmentation très rapide de la complexité* des logiciels. Ces derniers, n'ayant souvent pas été conçus pour faire face à une telle évolution, deviennent de plus en plus difficiles à développer, à entretenir et à utiliser. Il est donc nécessaire aujourd'hui de *repenser profondément* les processus de *construction, d'amélioration et d'utilisation des logiciels de modélisation.*

2.2. *Connaissance sous forme active*

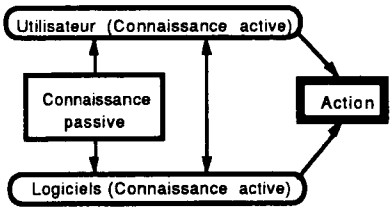
Hier encore la *connaissance logeait dans les esprits et dans les livres.* Que les bibliothèques contiennent de vrais livres ou leurs enregistrements informatiques, la connaissance n'y est disponible que *sous une forme passive* : on ne peut en effet l'utiliser qu'après l'avoir étudiée puis assimilée.

Avec le développement de la modélisation et de quelques autres techniques informatiques, la connaissance s'installe aujourd'hui dans les ordinateurs *sous une forme active* qu'il est possible d'utiliser sans avoir à l'assimiler, ce qui peut conduire à de graves difficultés.

Lorsqu'ils sont exécutés, les logiciels génèrent de la connaissance additionnelle, qui apparaît :

- sous forme passive dans des listes de résultats ou dans des courbes,
- sous forme active dans le cerveau de l'utilisateur qui interprète les résultats du logiciel, et éventuellement dans les modifications et améliorations que l'on peut ensuite apporter aux logiciels.

Les logiciels peuvent ainsi être considérés à la fois comme des *outils de concentration* et d'activation automatique de connaissances passives et comme des *générateurs de connaissances additionnelles.*



Personne ne peut prédire aujourd'hui les limites du champ d'action et de l'autonomie des futures générations de logiciels. En revanche, et sans être exagérément pessimiste, on peut prévoir que la consultation rapide — trop ? — des machines remplacera, progressivement certes, mais presque totalement, l'étude et la réflexion préalables à la résolution des problèmes [d'après B. Nayroles, *Les verrous de la modélisation*, Grenoble, septembre 1990].

2.3. *Variété des logiciels de modélisation*

Il existe de nombreux types de logiciels de modélisation.

- Les *logiciels de recherche spécialisés* sont conçus et utilisés dans les laboratoires de recherche pour simuler et comprendre un phénomène expérimental

particulier, ou pour tester, améliorer, comparer des méthodes, formulations et modèles nouveaux.

• Les *logiciels de recherche généraux* servent de base de travail, de moyen d'archivage et d'outil d'échange dans un laboratoire, dans un ensemble de laboratoires ou dans des réseaux de recherche regroupant laboratoires et industriels.

• Les *logiciels industriels spécialisés* sont destinés à satisfaire un besoin particulier (outils métiers). Ils concentrent de plus en plus la connaissance de l'entreprise.

• Les *logiciels industriels généraux* s'efforcent de répondre à un nombre maximal de besoins, et de s'adapter à des environnement variés ; ils sont de plus en plus développés et diffusés par des sociétés spécialisées.

En fait ces différents types de logiciels ne sont pas isolés les uns des autres. En technologie plus qu'ailleurs, on s'efforce en effet de transférer le plus rapidement possible les résultats de la recherche vers l'industrie. Le *vecteur de transfert* le plus efficace est aujourd'hui le logiciel. En outre, pour diminuer le nombre de logiciels, simplifier leur entretien et garantir leur pérennité, on s'efforce d'insérer les logiciels spécialisés ou les méthodes qu'ils utilisent, dès qu'elles sont stabilisées, dans des logiciels plus généraux.

Les caractéristiques des types de logiciels peuvent être très différentes, qu'il s'agisse de la taille du logiciel, de sa fiabilité, de sa facilité de développement ou de la variété des méthodes qu'il met en œuvre. Il y a en particulier une grande différence, souvent sous-estimée, entre logiciel industriel et logiciel de recherche.

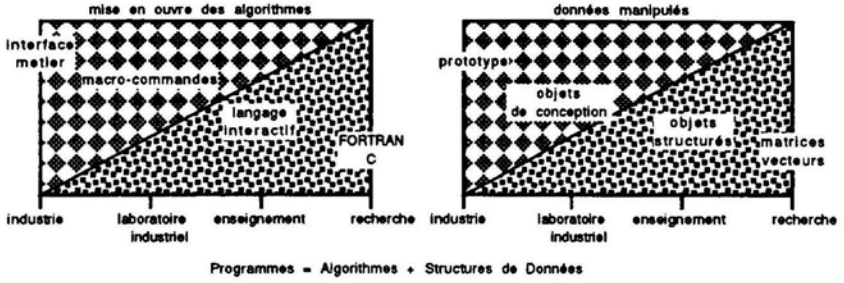
Les logiciels industriels sont (ou devraient être) basés sur des *méthodes éprouvées*, largement testées sur de nombreuses applications. Leur programmation est conçue pour garantir la fiabilité, faciliter la recherche d'erreurs et les développements ultérieurs, au prix d'une éventuelle complexité. De plus il est clair que la qualité de l'interface logiciel-utilisateur et de la documentation sont ici prépondérants. Un logiciel industriel peut inclure des modèles complexes mais son utilisation doit rester simple car il doit être efficace dans un environnement de production. Le logiciel doit contenir des protections contre les utilisations qui conduisent à des résultats faux. La complexité des actions que peut entreprendre l'utilisateur est donc limitée, le logiciel se présentant alors plus ou moins comme une *boîte noire fermée*.

Les logiciels de recherche doivent au contraire offrir un *maximum de flexibilité*, encourager leurs utilisateurs à faire des *modifications*, faciliter au maximum l'*ajout de fonctions* supplémentaires tout en favorisant la réutilisation des modules existants. Les logiciels de recherche sont, selon les cas, des *boîtes à outils* de chercheurs isolés ou d'ensembles de chercheurs, ou des plateformes fédératrices au niveau de laboratoires ou d'ensembles de laboratoires. Ces logiciels utilisent normalement des *méthodes novatrices* qui ne sont pas, par nature, validées sur un large spectre d'applications. Ils n'ont souvent été testés que sur des cas assez particuliers. Enfin la qualité de leur programmation et de leur organisation informatique laisse souvent à désirer, ce qui pose des problèmes lors de la recherche

d'erreurs ou lors de développements ultérieurs. Une autre difficulté provient souvent de la faiblesse de la documentation. Dans le meilleur des cas, ces logiciels peuvent seulement être considérés comme des *prototypes de logiciels industriels.* Vus comme tels, ils sont précieux comme aide à l'écriture des logiciels industriels, mais ne les remplacent pas.

Il faut souligner que dans certaines équipes de recherche de grandes sociétés, la situation est souvent plus proche de celle d'un laboratoire de recherche que de celle d'un bureau d'études. Les problèmes de transferts internes à de telles sociétés sont aussi délicats, si ce n'est plus, qu'entre le monde de la recherche et l'industrie.

Les logiciels utilisables pour l'enseignement se situent entre les logiciels de recherche et les logiciels industriels, bien que plus proches de la recherche. Ils réclament surtout de la simplicité, car le temps dont disposent les étudiants pour assimiler les particularités d'un logiciel est très limité.



Tout en respectant leurs spécificités, une certaine continuité (cohérence, compatibilité) est souhaitable entre les différents types de logiciels pour simplifier les transferts de méthodes et de modules informatiques, depuis les logiciels spécialisés vers les logiciels plus généraux, et depuis les logiciels de recherche vers les logiciels industriels. Cette continuité est difficile à assurer en pratique.

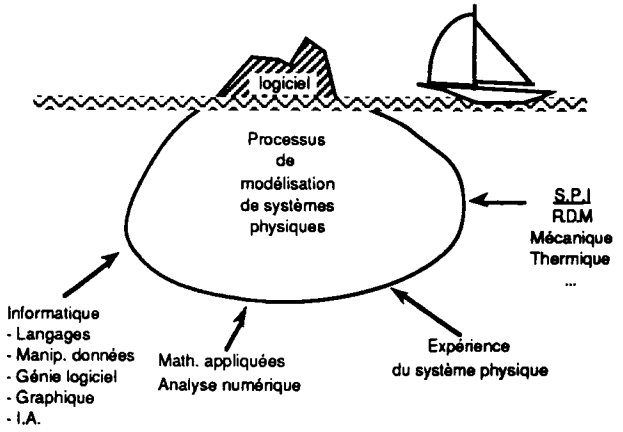
2.4. *Description d'un logiciel de modélisation*

On peut décrire un logiciel de modélisation de différentes manières. L'une d'elles consiste à faire abstraction des caractéristiques internes du logiciel et à le définir uniquement en termes des *données qu'il utilise en entrée et des résultats qu'il produit en sortie*, c'est-à-dire à le considérer comme une *boîte noire.* Cette description s'adapte bien à un logiciel conçu pour traiter un type de problème particulier ; elle s'adapte moins bien aux logiciels généraux pour lesquels il faut lister les très nombreuses configurations d'entrée et de sortie disponibles. Ce mode de description ne permet pas d'évaluer la validité des modèles théoriques sur lesquels sont basés les calculs.

On peut aussi tenter de caractériser la "*complexité*" d'un logiciel sur des critères purement informatiques, tels que le nombre de lignes de programme (souvent de l'ordre de centaines de milliers), le nombre de sous-programmes (souvent de l'ordre de centaines ou de milliers), les possibilités d'exécution sur divers ordinateurs (prestigieux : CRAY ; rapides mais abordables : stations de travail RISC ; très répandus : PC), la richesse des structures de données etc.

La description du logiciel peut aussi porter sur des *aspects purement scientifiques* : on citera alors les caractéristiques des méthodes numériques utilisées, en soulignant l'originalité de l'implantation et pour un logiciel basé sur la méthode des éléments finis on pourra énumérer les types d'éléments disponibles ainsi que les méthodes de résolution ou d'intégration offertes. On peut également s'intéresser au confort de l'utilisateur en décrivant l'interface entre le logiciel et son utilisateur, le langage d'introduction de données, le système de menus, la documentation.

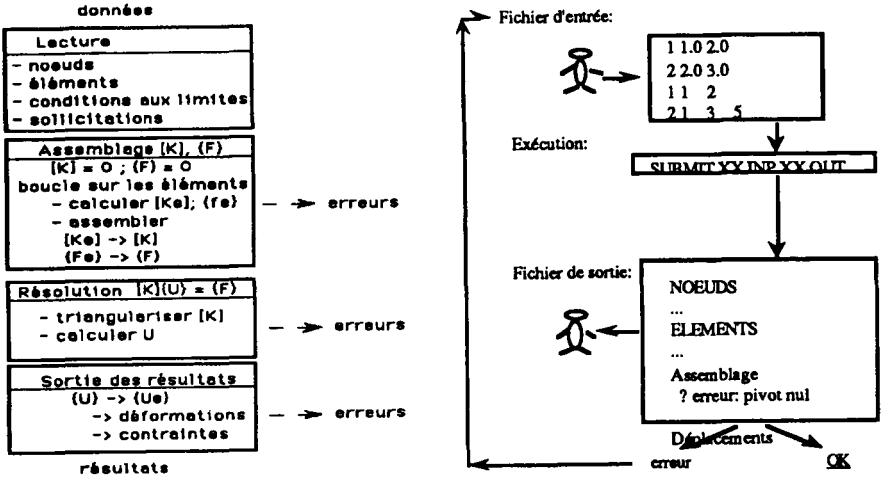
En fait le logiciel de modélisation est une *construction logique complexe et multiforme* difficile à décrire. Il n'est que la partie émergée et visible de l'ensemble beaucoup plus vaste que constitue le *processus de modélisation entier*. La partie invisible contient les connaissances et l'expérience de l'ensemble des personnes qui ont créé le logiciel, ainsi que toutes ses particularités informatiques. Ces informations touchent des domaines aussi variés que l'informatique, les mathématiques appliquées, l'analyse numérique, les sciences pour l'ingénieur. Elles incluent également l'expérience accumulée par les auteurs du logiciel, concernant les systèmes physiques étudiés. Il est clair qu'une description complète du logiciel devrait inclure l'ensemble de ces informations, ce qui est en général impraticable. Le logiciel reste donc *largement inconnu de ses utilisateurs*, et cela d'autant plus qu'il est complexe. Les difficultés liées à l'utilisation d'un logiciel de modélisation sont



de même nature que celles qui apparaissent dans la communication entre deux êtres logiques complexes : mauvaise appréhension des connaissances de l'autre, mauvaise interprétation de son mode de pensée, extrapolations indues. Seule une organisation interne simple du logiciel et une logique d'ensemble facile à décrire permettent de réduire la distance entre le logiciel et l'image que s'en construit son utilisateur.

2.5. *L'évolution des logiciels de modélisation*

Les logiciels de modélisation tels que nous les connaissons aujourd'hui ont subi une longue évolution depuis les premiers programmes du début des années 60. La structure interne d'un logiciel capable de mettre en œuvre la démarche de modélisation traditionnelle décrite au paragraphe 1.3 est relativement simple. De très nombreux logiciels organisés de cette manière ont été écrits au cours des trente dernières années. Il s'en écrit encore dans le cadre des thèses. La majorité d'entre eux fonctionnaient (et fonctionnent encore parfois) dans un environnement de traitement par lots. L'utilisateur prépare un fichier de données d'entrées (autrefois une boîte de cartes perforées), puis soumet celui-ci à un ordinateur. Il obtient, après une période de temps souvent difficile à prévoir, un fichier de résultats (autrefois un simple listing).



Une telle organisation laisse peu de liberté à l'utilisateur, l'enchaînement des opérations étant prédéterminé. En cas de difficultés, celui-ci ne peut que modifier le contenu du fichier de données et exécuter de nouveau complètement le programme. L'information fournie par le logiciel en sortie est elle aussi prédéterminée et souvent mal adaptée aux besoins instantanés de l'utilisateur.

A partir de cette structure de base, les logiciels de modélisation ont évolué sur quatre plans :

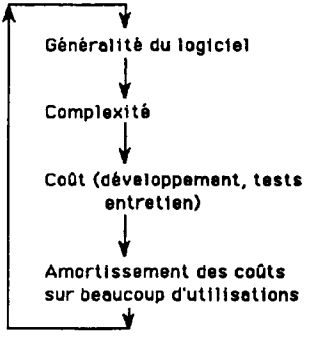
• les logiciels ont dû *suivre l'évolution de la modélisation* mentionnée au paragraphe 1.4, ce qui a multiplié le nombre de boucles et de branchements, rendant de plus en plus difficile la description de l'organisation du programme par des schémas simples ; seuls restent facilement descriptibles les sous-ensembles de plus en plus petits du logiciel ;

• les logiciels généraux se sont efforcés de *regrouper un maximum de méthodes* et de traiter un maximum de classes de problèmes, ce qui conduit à une croissance difficile à maîtriser de leur complexité interne ;

• certains logiciels se sont efforcés de simplifier la tâche de leurs utilisateurs, grâce à des améliorations de l'interface logiciel-utilisateur. Conçus initialement comme de simples *outils d'analyse de systèmes* physiques bien définis, ces logiciels ont ainsi évolué graduellement vers de véritables *outils d'aide à la conception.* En même temps, le développement des logiciels de CAO (manipulation de formes géométriques complexes) a amené les logiciels de modélisation à venir chercher les descriptions géométriques dont ils ont besoin dans les logiciels de CAO. Ceci conduit lentement vers une intégration CAO-modélisation, qui se trouve freinée par des considérations commerciales ;

• les logiciels ont dû s'adapter aux *évolutions rapides de l'informatique* : passage du traitement par lots au traitement interactif, interfaces graphiques multi-fenêtres, stations de travail, réseaux, nouvelles techniques d'organisation des données, traitement distribué et parallèle. En particulier, en ce qui concerne la *"convivialité"*, les logiciels de modélisation sont contraints de s'aligner progressivement sur les traitements de textes, tableurs, etc., utilisés quotidiennement par chaque utilisateur sur sa station de travail ou son micro-ordinateur : entrée assistée des données, visualisation graphique de toute information à tout moment, intervention en cours de calcul, menus déroulants, etc.

Ces évolutions ont augmenté considérablement le *champs d'action de la modélisation*, le *nombre d'enchaînements d'opérations*, de méthodes et de services disponibles dans chaque logiciel. Cette tendance est amplifiée par le *cycle infernal* suivant :

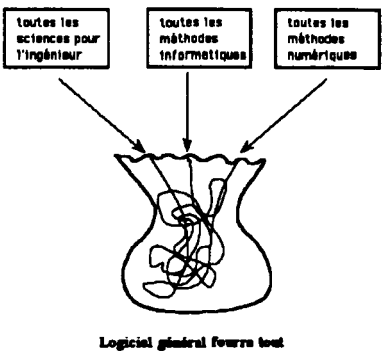


Malheureusement, les possibilités ainsi offertes par les logiciels augmentant, les connaissance requises pour les utiliser augmentent également. On peut affirmer qu'aujourd'hui la limitation la plus critique au développement de la modélisation réside dans le *volume des informations* que doivent assimiler les utilisateurs pour tirer parti des logiciels existants.

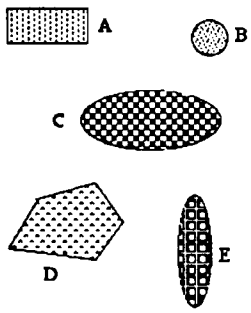
3. Des difficultés, des souhaits, des solutions ?

3.1. *Un logiciel, ... des logiciels*

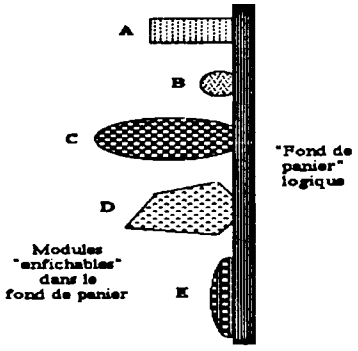
Si un seul logiciel se propose de regrouper l'ensemble des modules nécessaires pour effectuer tous les types de modélisation envisageables, son volume et sa complexité deviennent inacceptables. Cela est déjà vrai dans une discipline donnée telle que la thermique ou la mécanique des solides.



En outre, si chaque nouvelle modélisation donne naissance à un nouveau logiciel, avec ses caractéristiques, son organisation et son interface utilisateur propre, le nombre de logiciels croît de manière totalement incontrôlée. Développer, entretenir et assurer la pérennité de cet ensemble croissant de logiciels très différents les uns des autres devient une tâche surhumaine, même au sein d'une grande entreprise. En pratique seuls quelques rares logiciels survivront à leur phase de test, ou au départ de leur concepteur. D'où un énorme gâchis d'efforts. De plus les couplages entre logiciels seront difficiles ou impossibles, sauf au prix de l'écriture d'un grand nombre d'interfaces ("n" logiciels ... "n^2" interfaces).



Galaxie de logiciels incompatibles



Ensemble de logiciels "coordonnés"

Une solution possible consiste à créer des ensembles de logiciels (ou modules) de taille limitée, qui se partagent un certain nombre de sous-programmes, de structures de données, et qui respectent un nombre limité de normes communes. Deux modules peuvent ainsi communiquer ou même être immergés par la suite dans un module unique pour traiter des problèmes couplés. Selon l'ampleur des similitudes, un utilisateur ou un développeur peut travailler sur l'un ou l'autre des modules avec une période d'adaptation plus ou moins longue. Un tel ensemble de modules "coordonnés" peut être considéré, par analogie avec les ensembles électroniques, comme un *fond de panier logique*, constitué par tout ce que partagent les modules, dans lequel viennent *"s'enficher"* les différentes *cartes logiques* que sont les modules. Il faut bien sûr :

- choisir correctement la définition du fond de panier. S'il est trop contraignant, l'ensemble des logiciels capables de s'y adapter sera limité, et, ce qui est pire, les développeurs ne respecteront pas ses normes. S'il ne l'est pas assez, les interactions possibles entre les logiciels seront limitées ;
- s'assurer que les logiciels (ou modules) sont bien compatibles avec le fond de panier ;
- définir correctement la taille et les fonctions des nouveaux modules. Ceux-ci peuvent être très petits et correspondre à une opération élémentaire (simple modification d'une variable, construction d'un système d'équations, exécution des opérations correspondant à un pas de temps ou à une itération d'équilibre, affichage sélectif d'informations, ...). Au contraire, un module peut être un logiciel de modélisation complet correspondant à une application ou à un domaine donné.

3.2. *Architecture des logiciels de modélisation*

L'architecture d'un logiciel est constituée par l'ensemble des *règles, normes et techniques d'organisation* que doivent respecter tous les constituants présents et futurs du logiciel. Ces règles ont pour but :

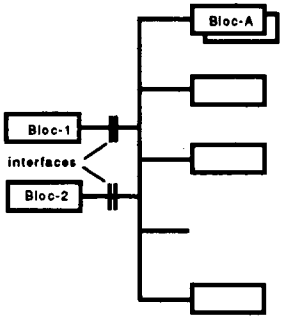
• de simplifier les communications entre les divers sous-ensembles du logiciel ;

• de donner une certaine homogénéité à l'ensemble des modules constitutifs du logiciel, pour en faciliter la compréhension ;

• d'éviter une explosion de la complexité du logiciel, à mesure que celui-ci se développe ;

• d'aider les utilisateurs et développeurs du logiciel à se construire rapidement une représentation mentale correcte du logiciel.

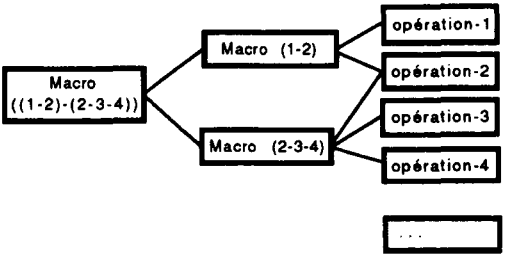
L'architecture d'un logiciel devant être connue en détail par tous les développeurs, et dans ses grandes lignes par les utilisateurs, il importe qu'elle soit simple, limpide, et facile à décrire. Celle-ci est souvent définie par un *découpage logique* du logiciel en blocs fonctionnels, et par des *normes de passage d'informations* entre ces blocs. Ce découpage permet de faire ressortir les grandes fonctions du logiciel, et de modifier ou même remplacer globalement un bloc fonctionnel sans se soucier des autres, pour autant que l'on respecte les normes des interfaces.



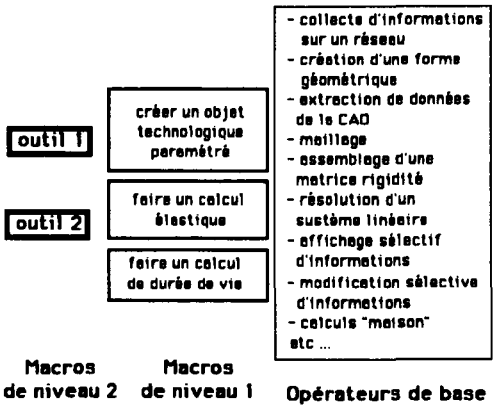
Un second mode de définition d'une architecture, complémentaire du premier, consiste à écrire une première version (de bonne qualité) de chacun des sous-programmes caractéristiques puis à développer le logiciel par duplications et modifications mineures de ces derniers. On peut ainsi diffuser un style de programmation dans l'ensemble du logiciel par *simple mimétisme*.

Il est certain que les règles qui définissent l'architecture d'un logiciel *limitent ou contraignent les évolutions* futures de celui-ci. Comme la durée de vie des logiciels de modélisation est souvent de l'ordre de dix à vingt ans, il importe que les options architecturales ne dépendent pas de techniques informatiques ou de méthodes numériques particulières qui risquent de disparaître bien avant le logiciel lui-même, ou au moins son architecture. Par exemple, s'il est probable que l'on devra, pendant longtemps encore, entrer des données dans un logiciel, puis construire et résoudre des systèmes d'équations, il est possible que la méthode des éléments finis soit un jour remplacée par une méthode plus efficace sur de futurs ordinateurs. De la même manière, il est dangereux de parier sur la pérennité d'un modèle de structure de données particulier (tel qu'une base de données relationnelle). Mais on ne prend pas

trop de risques en affirmant que toutes les opérations qu'exécute un logiciel peuvent être organisés logiquement sous la forme d'un ensemble extensible d'opérateurs de base et de macro-opérateurs représentés par le graphe suivant :



soit, à titre d'exemple :



3.3. *Normalisation*

Depuis quelques années, dans de nombreux domaines de la technologie, les opérations et objets ayant atteint une certaine maturité ont fait l'objet de *normalisations systématiques*. C'est en particulier le cas en informatique (langages, systèmes d'exploitation, bases de données, réseaux, ...). Dans le domaine de la modélisation cette tendance ne s'est pas manifestée. Ce manque de normalisation conduit à la réécriture inlassable de sous-programmes presque identiques, et à d'importantes difficultés pour réutiliser les résultats informatiques d'une recherche dans le cadre d'une recherche ultérieure.

Alors que des langages scientifiques de haut niveau tels que "Mathématica" ou "Mathlab" s'imposent graduellement, rien d'équivalent ne voit le jour en modélisation. Il est intéressant de se demander pourquoi les chercheurs en modélisation acceptent, volontiers semble-t-il, la norme constituée par un langage

assez simple (tel que Fortran), mais ne désirent pas ou n'acceptent pas de normes de plus haut niveau; ils jugent même utile pour la formation que les étudiants écrivent eux-même, de manière souvent maladroite, tous les modules informatiques dont ils ont besoin. Cela limite fortement la complexité des modèles envisageables.

Deux types de normalisation sont souhaitables :

• celle des *opérateurs de base de la modélisation*, certains étant très généraux, d'autres étant liés à des domaines ou applications spécifiques. Ce type de normalisation permet la réutilisation de sous-programmes mais il met également en lumière les similitudes, les différences et les complémentarités des divers domaines d'application de la modélisation ; ceci conduit souvent à de fructueuses fertilisations croisées entre domaines de recherche.

• celle des *structures de données manipulées* par les divers opérateurs de modélisation. Ce type de normalisation permet le couplage des programmes, ainsi que l'écriture de sous-programmes de service généraux qui se chargent d'une part importante de la manipulation des données.

La normalisation favorise la création d'*outils généraux*, utilisables dans de nombreuses circonstances, ce qui diminuent d'autant la taille et le nombre des programmes spécifiques d'une application donnée. La normalisation contribue ainsi à diminuer à la fois la *complexité* et la *taille* des logiciels, à améliorer leur *lisibilité*, et à raccourcir la *durée de formation* des utilisateurs.

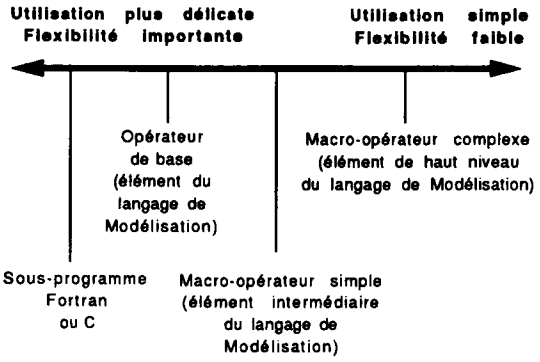
3.4. *Langage de modélisation*

Le processus de modélisation fait appel à de nombreux concepts de haut niveau sémantique concernant la physique, le processus de modélisation par essais et erreurs, des notions d'algorithmique, des connaissances à priori sur le système modélisé. Pourtant l'écriture des logiciels de modélisation se fait à très bas niveau, par l'intermédiaire de langages tels que FORTRAN ou C. Cela est certes justifié par des raisons historiques et par un besoin impératif d'efficacité des logiciels de modélisation. Les langages plus flexibles ou plus puissants tels que LISP, ADA, C++, APL, Mathematica sont en général beaucoup plus gourmands en ressources informatiques que C ou Fortran. De plus, ils ne répondent pas non plus à l'envie des spécialistes de la modélisation de parler leur propre langage et de manier directement des concepts caractéristiques de la modélisation tels que des contraintes, des matrices, des points, des champs de variables, des fissures, des formes, des hypothèses.

Tout en conservant l'*efficacité des langages compilés simples* pour écrire les divers modules d'un logiciel, il est possible de créer un ensemble d'opérateurs de base qui constituent collectivement un véritable langage de modélisation. L'utilisateur peut alors activer ces opérateurs dans un ordre quelconque, par l'intermédiaire d'une syntaxe adaptée à la modélisation, si ce n'est à une application

particulière. La seule condition requise pour activer un opérateur est l'existence et la validité des informations qu'il utilise en entrée ; cette condition peut être testée par chaque module pour éviter des catastrophes informatiques irrémédiables. On associe ainsi l'efficacité des langages compilés et le confort d'utilisation des langages interprétés. Les possibilités ainsi offertes à l'utilisateur averti sont nombreuses : il peut par exemple créer de nouveaux algorithmes, créer des couplages et des enchaînements non prévisibles, afficher et modifier sélectivement toute information en cours de calcul. Cela exige toutefois de la part de l'utilisateur une bonne compréhension du rôle de chaque opérateur et de la logique globale du logiciel qui doit donc être conçu en conséquence.

L'utilisateur moins averti ou seulement intéressé par des opérations plus répétitives pourra faire appel à des enchaînements standards ou macro-opérateurs de plus ou moins haut niveau.



De tels langages de modélisation permettent d'*élever le niveau sémantique de la programmation des applications*, favorisent la réutilisation d'opérateurs de base, et simplifient la tâche des utilisateurs occasionnels. D'autre part il est facile de créer des versions du logiciel adaptées à un besoin particulier en sélectionnant les seuls opérateurs utiles (outil-métier).

3.5. *Différents modes d'interaction avec le logiciel*

L'informatique d'aujourd'hui nous a habitués à l'utilisation de nombreux modes d'interaction avec un ordinateur : langages variés, souris, menus déroulants etc. Chacun peut, à bon droit, avoir ses préférences. Pour s'adapter aux divers types d'utilisation ou d'utilisateur, un logiciel de modélisation idéal doit offrir :

- une entrée clavier traditionnelle, donnant accès à un (ou des) langage(s) de modélisation adaptable(s) au contexte d'utilisation,
- une entrée guidée par une hiérarchie de menus compatibles, si possible, avec le langage de modélisation,

• un système de navigation dans les structures de données du logiciel, permettant l'accès, tant en lecture qu'en écriture, à toutes les informations,

• un système de représentation graphique des entités physiques et géométriques manipulées, incluant la possibilité de sélectionner des entités au moyen d'une souris,

• un système de création/modification de macro-opérateurs à partir d'opérateurs et macro-opérateurs existants.

Une voie attrayante est la commande du logiciel *par simple spécification d'objectifs à atteindre* : construire telle information, modifier telle donnée et recalculer ce qui n'est plus valide, afficher tel résultat lorsqu'il aura été modifié de manière significative. Une possibilité consiste à inclure dans le logiciel une description de lui-même, et à exploiter cette description pour synthétiser un macro-opérateur capable d'atteindre le but spécifié. Cela peut être interprété comme de la programmation automatique dans le langage de modélisation, programmation plus aisée que dans un langage standard compte tenu de la granulométrie plus importante du langage de modélisation. Cette approche peut réduire considérablement les connaissances que doit posséder l'utilisateur sur le logiciel.

4. Exemple de l'architecture SIC

4.1. *Généralités*

Depuis 1985, l'université de technologie de Compiègne a conçu puis testé dans le cadre de plusieurs prototypes successifs, une architecture de logiciel général de modélisation, basée sur les concepts présentés ci-dessus. Cette architecture porte le nom interne "SIC" (Système interactif de conception). Le travail correspondant a été réalisé en étroite collaboration avec plusieurs équipes de recherche, en particulier celles de O. Debordes (IMT, Marseille), de M. Jean (USTL, Montpellier), de B. Nayroles (INPG, Grenoble) ; des industriels, en particulier Renault, Pechiney, Valourec et le CNRS, dans le cadre de l'action de recherche coordonnée du PIRSEM "Thermique et modélisation".

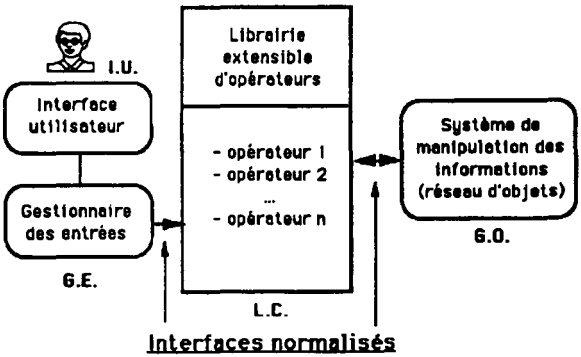
4.2. *Organisation générale*

L'architecture SIC définit trois blocs fonctionnels principaux :

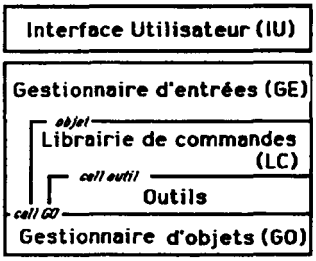
• une *librairie extensible d'opérateurs* (librairie de commandes : **L.C.**) qui regroupe tous les traitements élémentaires que peut exécuter l'utilisateur ;

• une *système de manipulations des informations*, ces dernières étant organisées sous la forme d'un réseau d'objets (gestionnaire d'objets : **G.O.**) ;

• un *système de gestion des entrées*, qui interprète les ordres que fournit l'utilisateur par l'intermédiaire d'un langage de modélisation, de macro-commandes, de menus, etc. (gestionnaire des entrées : **G.E.**).



Une autre représentation de l'architecture SIC, basée sur un modèle en couches, est la suivante :



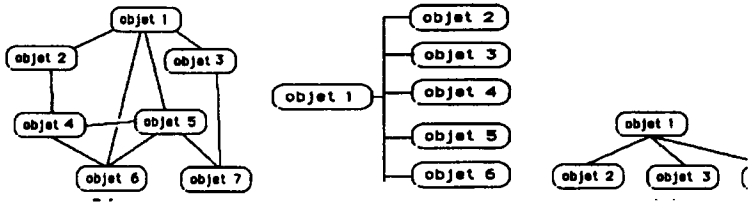
4.3. *Gestionnaire d'objets (G.O.)*

La spécification du *système de manipulation des informations* est l'étape la plus délicate dans la conception d'une architecture de logiciel de modélisation. En effet, elle se heurte à des objectifs contradictoires : tout en manipulant de gros volumes de données, on cherche la rapidité pour les opérations de calcul intensif, mais aussi la flexibilité et la richesse des structures de données pour :

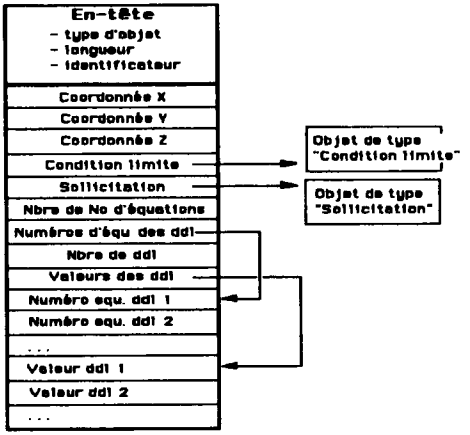
- les interactions homme-machine,
- la représentation paramétrée de systèmes physiques et de formes géométriques complexes,
- la manipulation des différents modèles correspondant à différentes hypothèses,
- l'accueil de structures de données complexes issues des programmes de CAO,
- les opérations basées sur des techniques de manipulation explicite de la connaissance (intelligence artificielle),
- les traitements distribués.

Le modèle général de structure de données retenu est le réseau d'objets ; toutes les informations manipulées par le logiciel sont regroupées en paquets "typés",

nommés ici "objets". Par exemple il existe des objets de type "points", "lignes", "matrices", "vecteurs", "descripteurs de type d'objets", "fenêtres graphiques", "macro-opérateurs", etc. Chaque objet contient un certain nombre d'attributs de longueur fixe ou variable, dont le nombre et le contenu dépend du type d'objet considéré. Chaque objet est identifié de manière unique par un identifieur, qui lui est attribué par le G.O. lors de sa création, ainsi qu'éventuellement par un nom. Chaque attribut d'un objet peut contenir un ou des nombres réels ou entiers, des caractères (noms d'objets), ou des identifieurs d'autres objets. De cette manière, chaque objet peut pointer vers d'autres objets pour former des structures variées : réseaux, listes, arbres, piles...



A titre d'exemple, les attributs d'un objet de type "noeud" sont les suivants :

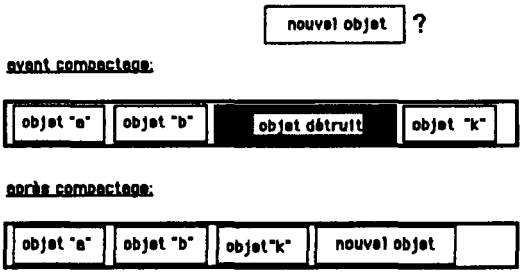


Attributs des objets de type "Noeud"

Les services offerts par le gestionnaires d'objets sont organisés en couches :

• *couche 1 : gestion de l'espace de stockage*

Cette couche gère un espace virtuel en mémoire. Elle permet de créer, déplacer, rallonger, supprimer des objet de type indifférencié. Elle permet également de récupérer l'espace laissé libre par des objets détruits ou raccourcis, grâce à un mécanisme de compactage, dont l'activation, coûteuse, doit être limitée au minimum nécessaire.



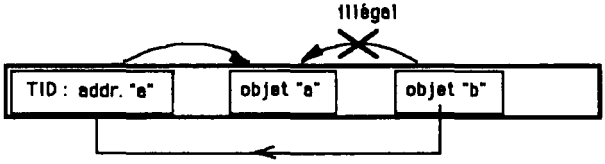
• *couche 2 : gestion interne des objets*

Cette couche permet d'accéder aux attributs des objets, d'en modifier le contenu, éventuellement d'en modifier la longueur. On peut accéder à chaque attribut de chaque type d'objet :

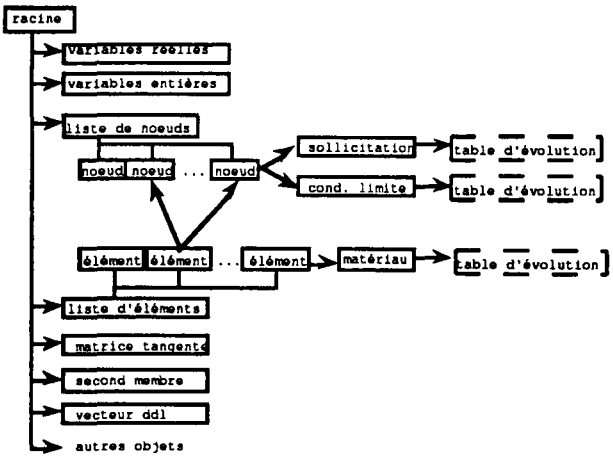
– dans un programme, par un paramètre qui définit la position de l'attribut dans l'objet,
– en interactif, par le nom de l'attribut.

• *couche 3 : gestion d'un réseau d'objets*

Cette couche permet de naviguer dans un réseau d'objets qui pointent les uns sur les autres ; elle gère en particulier les noms des objets, leurs identifieurs, leurs adresses, ainsi que divers mécanismes d'accès aux objets. Un objet ne peut pointer sur un autre directement par son adresse puisque le mécanisme de récupération d'espace de la couche 1 peut être amené à déplacer des objets, c'est-à-dire à changer leurs adresses. On utilise pour ce faire l'identifieur de l'objet qui est relié à son adresse grâce à un objet spécial appelé la "table des identifieurs" (TID).

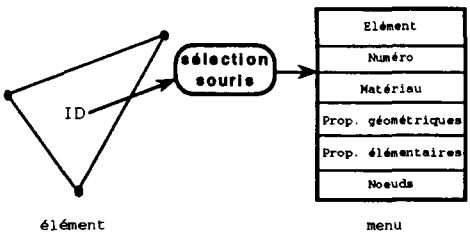


Chaque grande classe d'application (modèles basés sur les éléments finis, optimisation, représentation de formes géométriques) utilise un réseau d'objets particulier adapté à ses besoins. Par exemple, le réseau destiné au calcul par éléments finis est le suivant :

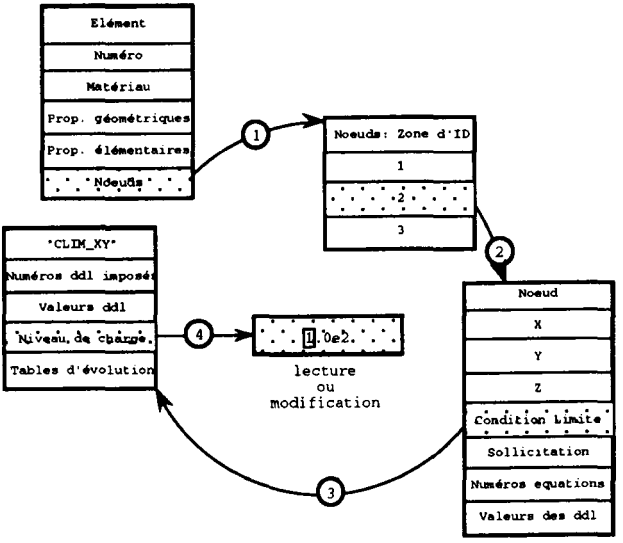


• *couche 4 : navigation dans un réseau d'objets*

Cette couche utilise le mécanisme de désignation d'objets par leurs identifieurs ainsi qu'un système de menus dynamiques. On fait appel à deux modes de représentation graphique d'un même objet : son modèle géométrique et un menu dont les cases correspondent aux attributs de l'objet.

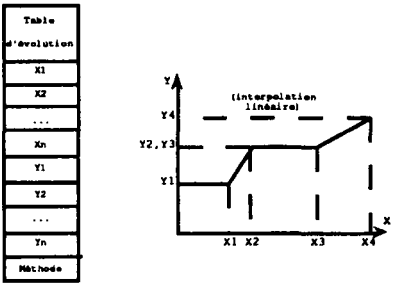


L'identifieur d'un objet peut être obtenu par sélection sur la représentation graphique de l'objet au moyen de la souris. L'objet ainsi sélectionné devient alors l'objet courant du navigateur et son contenu (attributs) apparaît dans un menu sur l'écran. Le comportement d'un tel menu est le suivant : la sélection avec la souris d'un attribut qui contient une valeur (numérique, caractères) permet d'éditer cette valeur ; celle d'un attribut de type "identifieur" selectionne l'objet correspondant et affiche le contenu de cet objet. Ainsi, en partant d'un élément fini on peut accéder à ses nœuds (1, 2), ensuite aux sollicitations ou conditions aux limites affectées à l'un de ces nœuds (3) puis éventuellement modifier le niveau de chargement (4).



• *couche 5 : encapsulation*

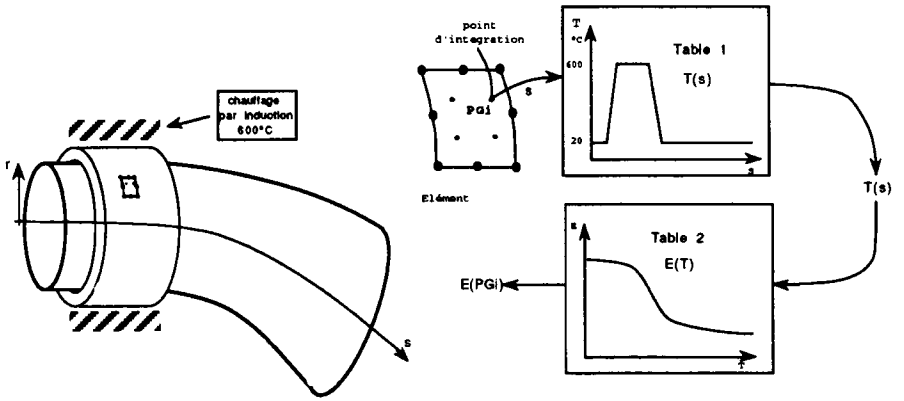
Les différents types d'objets peuvent être manipulés par toutes les opérations standards du GO : création, destruction, modification etc. On peut également définir d'autres opérations (méthodes) spécialisées qui ne s'appliquent qu'à un type d'objet particulier. Une description de type d'objet associée aux opérations spécifiques qui s'y appliquent définit un *service*. Un des services de SIC les plus utilisés est celui de la *"table d'évolution"*. Ce mécanisme est particulièrement puissant car il est utilisable dans de nombreux contextes parfois très éloignés conceptuellement les uns des autres. Une table d'évolution, au sens de SIC, est un mécanisme d'interpolation qui à partir d'un ensemble de points de contrôle et d'un algorithme d'interpolation donné, est capable de rendre une valeur interpolée en chaque point d'un domaine à une dimension. La description de l'objet "table d'évolution" est la suivante :



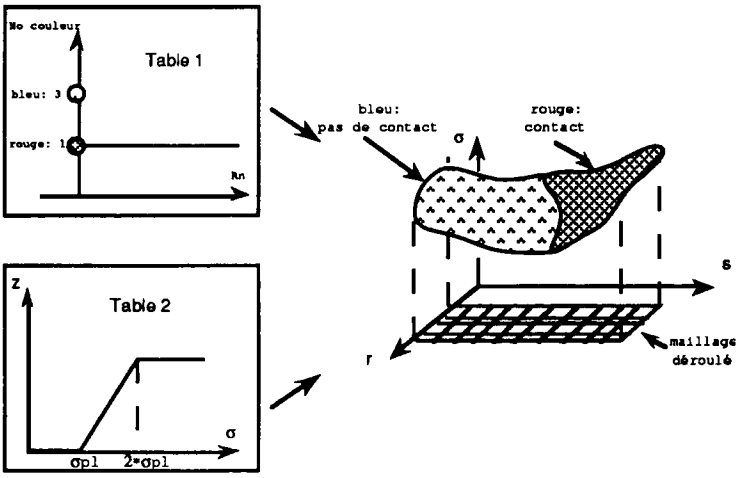
Les zones Xi et Yi contiennent les points de contrôle de la table. L'attribut "méthode", de type nombre entier, selectionne l'algorithme d'interpolation (1 = paliers, 2 = linéaire, 3 = quadratique, etc.). Le calcul est effectué par une fonction spécifique :

y = ytevcv(identifieur_table_evolution, x)

Différents effets peuvent être obtenus, en fonction du type d'objet auquel on affecte une table d'évolution. L'affectation d'une table d'évolution à une "condition à la limite" ou une "sollicitation concentrée" permet le pilotage du niveau des charges. Affecter une table d'évolution à une propriété de matériau permet par exemple de définir sa dépendance vis-à-vis de la température. Dans l'exemple précédent du coudage de tubes, le module de Young du matériau dépend de la température, elle-même variable selon la position axiale "s" vis-à-vis de l'outil. La détermination de la valeur du module de Young en un point d'intégration numérique passe alors par deux "tables d'évolution" qui relient ; l'une la température à la position axiale et l'autre, le module de Young à la température.

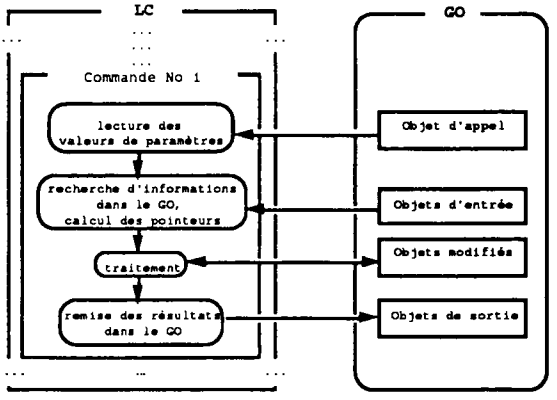


Une "table d'évolution" constituée de paliers peut être affectée à un objet de post-traitement graphique du champ des réactions de contact sur la face interne du tube. Ceci permet le filtrage des iso-couleurs. Les zones en contact (réaction positive) seront tracées en rouge et les zones où la réaction est nulle (non contact) en bleu. Une seconde table, linéaire par morceaux, permettra de définir la cote "z" des iso-surfaces représentant par exemple le niveau de la contrainte de Mises :



4.4. *Librairie de commandes (L.C.)*

Chaque classe d'application de SIC comporte deux parties : un réseau d'objets G.O. qui structure les données, et une collection d'opérateurs (ou commandes) associés qui exécutent des actions. La structure interne du sous-programme qui correspond à une commande est la suivante :



Les applications de la méthode des éléments finis en mécanique des solides sont caractérisées par :

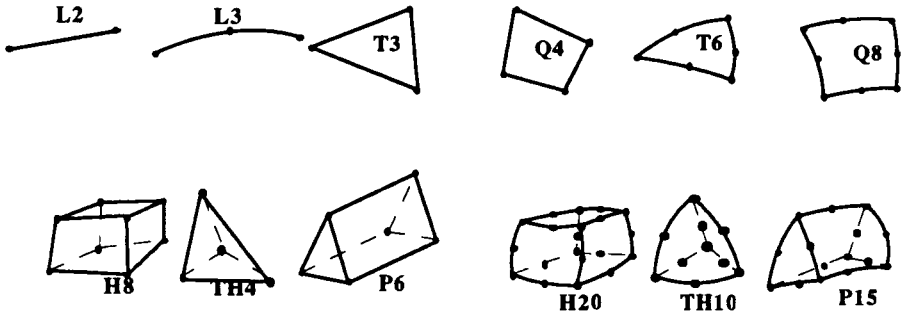
1) les lois constitutives utilisées :

- loi élastique linéaire de HOOK,
- élasto-plasticité avec les différents schémas d'écrouissage,
- élasto-viscoplasticité...

2) les schémas cinématiques retenus :

- grandes/petites déformations,
- hypothése cinématique sur un pas de charge,
- repère dans lequel on écrit la loi de comportement...

3) les types d'approximation spatiale par éléments finis retenus ; actuellement, le système SIC reconnaît les types suivants :

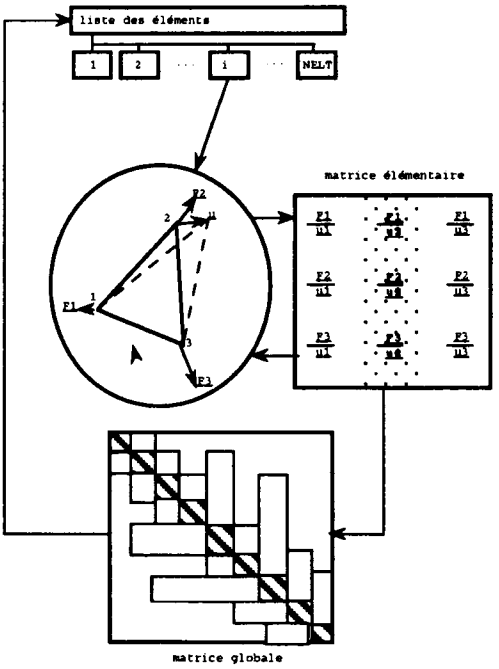


Dans SIC, une famille élémentaire est caractérisée par le triplet (loi de comportement, schéma cinématique, type d'élément).

Dans la version actuelle de SIC, les familles élémentaires suivantes sont disponibles :

type de modélisation	type d'élément
• Elasticité plane	T3,Q4,T6,Q8
• Elasticité tridimensionnelle	H8,H20,TH4,TH10,P6,P15
• Diffusion thermique plane	T3,Q4,T6,Q8
• Diffusion thermique axi	T3,Q4,T6,Q8
• Diffusion thermique tridimensionnelle	H8,H20,TH4,TH10,P6,P15
• Echange thermique plane	L2,L3
• Echange thermique axisymetrique	L2,L3
• Echange thermique tridimensionnelle	T3,Q4,T6,Q8
• Coque mince	T3
• Coque axisymetrique	L2
• Coque non-lineaire	
• Contact bidimensionnel	éléments de liaison
• Plasticité plane	T3,Q4,T6,Q8
• Plasticité axisymetrique	T3,Q4,T6,Q8
• Plasticité tridimensionnelle	H8,H20,TH4,TH10,P6,P15
• Visco-plasticité plane	T3,Q4,T6,Q8
• Visco-plasticité axisymetrique	T3,Q4,T6,Q8
• Visco-plasticité tridimensionnelle	H8,H20,TH4,TH10,P6,P15
• Visco-plasticité g-des déformations plane	T3,Q4,T6,Q8
• Visco-plasticité g-des déformations axi.	T3,Q4,T6,Q8
• Visco-plasticité g-des déformations tridim.	H8,H20,TH4,TH10,P6,P15

Une commande, associée à la description des structures de données qu'elle manipule, constitue un service général, capable de s'appliquer dans des contextes variés. Un exemple de service général de la librairie de commandes est l'*assemblage de la matrice tangente d'un système d'équations non linéaires par perturbation* des degrés de liberté. La commande correspondante est basée sur les objets "liste d'éléments finis" et "forces internes élémentaires" et construit une "matrice globale tangente". Le traitement consiste à parcourir les éléments de la liste, à évaluer par perturbations la matrice tangente de chaque élément, puis à l'assembler.

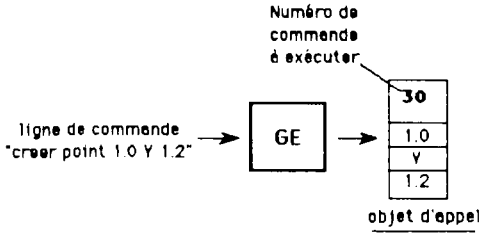


Pour chaque élément, on saisit ses caractéristiques et les valeurs courantes de ses degrés de liberté. Ensuite, la commande standard de SIC "calcul des forces internes" d'un élément fini est utilisée pour le calcul des forces internes dans la configuration courante et dans des configurations perturbées. Les colonnes de la matrice élémentaire sont alors obtenues par une formule de différences finies. Une fois la matrice tangente élémentaire évaluée, celle-ci est assemblée dans la matrice globale, par un appel à l'opérateur d'assemblage standard de SIC. Ainsi le mode stockage de la matrice globale et les particularités des calculs élémentaires n'interviennent que dans les opérateurs standards de SIC. On peut donc utiliser le service "calcul d'une matrice tangente par perturbation" pour n'importe quel mode de stockage global et n'importe quel type d'élément. Les applications de ce service sont nombreuses, tant en mécanique non linéaire qu'en optimisation. Voici les groupes de commandes de la version standard de SIC :

- Création d'informations dans la base d'objets
- Affichages et modifications d'informations variées
- Opérations liées aux macro-commandes
- Opérations de base de la méthode des éléments finis
- Opérations liées aux résolutions non linéaires
- Commandes pour la création d'entités géométriques
- Calculs divers sur la géométrie
- Commandes pour la création des entités constitutives d'un domaine
- Opérateurs divers concernant la gestion des entités topologiques
- Commandes relatives au maillage
- Commandes relatives au raffinement d'un maillage
- Commandes pour le repositionnement barycentrique ou élastique des noeuds par résolution directe
- Commandes pour la définition du problème physique
- Résolution des inégalités dues au contact-frottement
- Commandes d'homogénéisation
- Gestionnaire graphique
- Paramétrage - Optimisation
- Opérations de manipulation de la base d'objets
- Gestion de l'historique des commandes exécutées
- Interface avec d'autres logiciels
- Commandes diverses

4.5. *Gestionnaire des entrées (G.E.)*

Le gestionnaire d'entrées (GE) reçoit les ordres transmis par l'utilisateur au logiciel sous la forme de lignes de commande classiques. Le GE est l'interpréteur de ces lignes de commandes. Chaque fois qu'une ligne de commande est émise par l'utilisateur, le GE vérifie sa syntaxe puis génère un objet de type "objet d'appel de commande" dans la base de données. Cet objet contient, sous une forme compacte (ou "compilée"), l'information nécessaire à l'activation d'une commande.

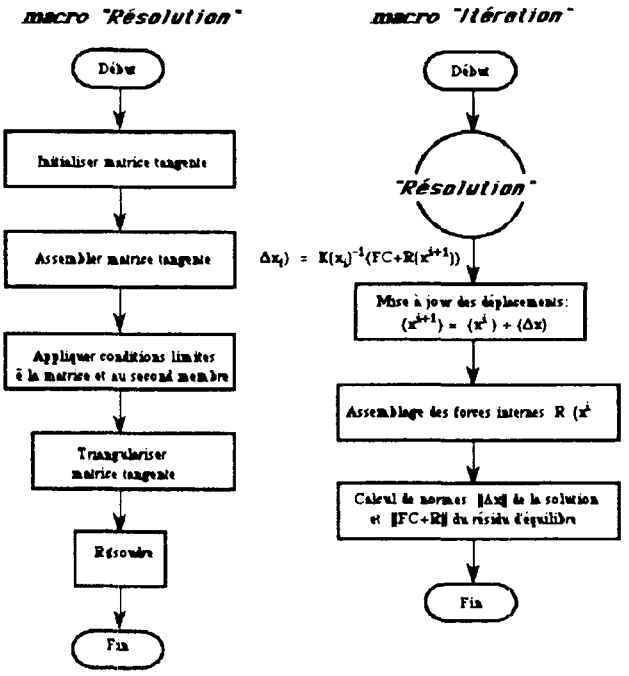


L'objet d'appel obéit aux mêmes principes que tous les autres objets du GO. Il contient un attribut qui *identifie la commande* à exécuter ainsi que des attributs de différents types pour le stockage de valeurs des *paramètres de la commande*. Dès

que le système détecte la présence d'un nouvel "objet d'appel", la commande requise est exécutée en mode synchrone ou asynchrone. Les valeurs des paramètres peuvent être restituées à l'aide des fonctions d'accès aux attributs standards offertes par le GO. Un "objet d'appel" correspond ainsi à une *commande "compilée"*.

Une liste d'"objets d'appel" forme naturellement une macro-commande. Une telle liste peut être placée dans un "objet macro-commande". Le système de macro-commande est hiérarchique en ce sens que les macros peuvent être appelées par d'autres macros. Les services de base de ce système offrent la possibilité de créer, de modifier et de mettre au point pas à pas des macro-commandes.

Le système de macro-commandes hiérarchiques permet de construire progressivement un algorithme de résolution compliqué. Par exemple la macro "Résolution" suivante peut être utilisée pour la résolution du problèmes linéarisé interne à une itération de la méthode de Newton Raphson.



Le dialogue GE suivant permet de définir une telle macro :

```
Sic3> CREER MACRO 'RESOLUTION'
RESOLUTION> INITIALISER MATRICE TANGENTE
RESOLUTION> ASSEMBLER MATRICE TANGENTE
RESOLUTION> APPLIQUER CONDITIONS_LIMITES
RESOLUTION> TRIANGULARISER
```

```
RESOLUTION> RESOUDRE
RESOLUTION> TERMINE
Sic3>
```

L'exécution de cette macro nécessite l'exécution de quelques commandes préalables de création des objets qu'elle utilise en entrée :

```
Sic3> CALCULER NUMEROTATION
Sic3> CALCULER LARGEUR_BANDE
Sic3> CALCULER FONCTIONS_INTERPOLATION
Sic3> INITIALISER VECTEUR DEPLACEMENTS 'DU'
Sic3> INITIALISER VECTEUR UPAS
Sic3> EXECUTER MACRO 'RESOLUTION'
Sic3>
```

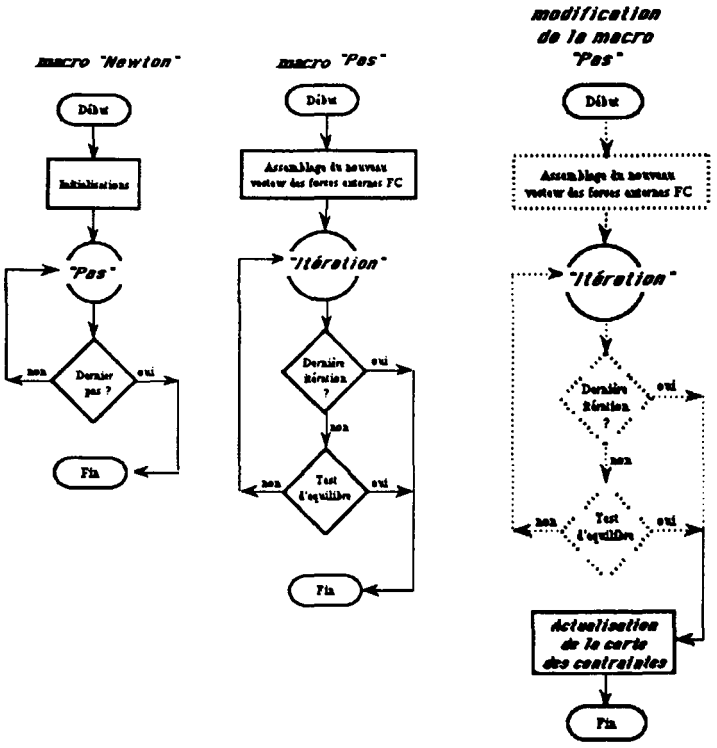
On peut ensuite visualiser l'état des contraintes sur l'écran à l'aide des commandes :

```
Sic3> ACTUALISE DEPLACEMENTS /VECTEUR='DU'
Sic3> POSTRAITE RESULTATS-ELEMENTAIRES SIGMA-MISES
Sic3>
```

Nous pouvons alors réutiliser la macro 'Résolution' dans une nouvelle macro "Itération" qui correspond à une itération de résolution d'un problème non-linéaire :

```
Sic3> CREER MACRO/NOM=ITERATION
ITERATION> CALCULER SOMME/OBJET_1=FC/OBJET_2=R/OBJET_3=R
ITERATION> EXECUTE MACRO 'RESOLUTION'
ITERATION> CALCULER SOMME 'DU' 'UPAS' 'UPAS'/COEFFICIENT-1=BETA
ITERATION> INITIALISER VECTEUR RESIDU /NOM='R'
ITERATION> ASSEMBLER VECTEUR FORCES_INTERNES /VECTEUR='R'
ITERATION> CALCULER NORME/VECTEUR=DU VARIABLE/NOM='NMDU'
ITERATION> CALCULER NORME/VECTEUR=UPAS VARIABLE/NOM='NMU'
ITERATION> DIVISER VARIABLE 'NMDU' 'NMU' 'NMDL'
ITERATION> TERMINER
Sic3>
```

Les schémas suivants illustrent le fonctionnement des macros 'Pas' correspondant à un pas de charge et 'Newton' permettant d'enchaîner un ensemble de pas.




```
Sic3> CREER MACRO 'PAS'
PAS> INITIALISER VECTEUR UPAS
PAS> AUGMENTER VARIABLE 'TMPS' 'DTMP'
PAS> ACTUALISER NIVEAUX
PAS> INITIALISER VECTEUR FORCES_CONCENTREES
PAS> ASSEMBLER VECTEUR FORCES_CONCENTREES/VECTEUR=FC
PAS> INITIALISER VARIABLE 'ITER'
PAS> 200:
PAS> AUGMENTER VARIABLE 'ITER'
PAS> EXECUTER MACRO'ITERATION'
PAS> TESTER/VALEUR=ITER SUPERIEURE/VALEUR=ITMAX ALLER 300
PAS> TESTER/VALEUR=NMDL SUPERIEURE/VALEUR=EPDL ALLER 200
PAS> 300:
PAS> ACTUALISER DEPLACEMENTS/VECTEUR=UPAS
PAS> ACTUALISER VARIABLES_INTERNES
PAS> TERMINER
Sic3>

Sic3> CREER MACRO 'NEWTON'
NEWTON> INITIALISER VARIABLE'MPAS' %1
NEWTON> INITIALISER VARIABLE'EPDL' %2
NEWTON> INITIALISER VARIABLE'DTMP' %3
NEWTON> INITIALISER VARIABLE'ITMAX'%4
NEWTON> IMPRIMER MESSAGE/TEXTE=' >>> Dtmp =  '/VALEUR='DTMP' '<<<'
```

```
NEWTON> INITIALISER VECTEUR RESIDU /NOM=R
NEWTON> 100:
NEWTON> AUGMENTER VARIABLE'IPAS'
NEWTON> EXECUTER MACRO 'PAS'
NEWTON> TESTER/VALEUR=IPAS INFERIEURE/VALEUR=MPAS ALLER 100
NEWTON> TERMINER
Sic3>
```

Dans cette dernière macro-commande le symbole "%i" signifie "paramètre No i de la macro".

Supposons, que l'utilisateur, après avoir exécuté un pas de calcul :

```
Sic3> EXECUTER MACRO 'NEWTON'/PARAMETRE-1=1 /P-2=0.0001 /P-3=0.001 /P-4=10
```

désire modifier ses macros pour suivre l'évolution du champ de contraintes au fil des pas (les parties inchangées sont en pointillé). Il pourra alors apporter des modifications à la macro "Pas" (en ***gras*** dans le texte) à l'aide de l'éditeur de macros, la version finale de la macro étant la suivante :

```
PAS> INITIALISER VECTEUR UPAS
PAS> AUGMENTER VARIABLE 'TMPS' 'DTMP'
PAS> ACTUALISER NIVEAUX
PAS> INITIALISER VECTEUR FORCES_CONCENTREES
PAS> ASSEMBLER VECTEUR FORCES_CONCENTREES/VECTEUR=FC
PAS> INITIALISER VARIABLE 'ITER'
PAS> 200:
PAS> AUGMENTER VARIABLE 'ITER'
PAS> EXECUTER MACRO'ITERATION'
PAS> TESTER/VALEUR=ITER SUPERIEURE/VALEUR=ITMAX ALLER 300
PAS> TESTER/VALEUR=NMDL SUPERIEURE/VALEUR=EPDL ALLER 200
PAS> 300:
PAS> ACTUALISER DEPLACEMENTS/VECTEUR=UPAS
PAS> ACTUALISER VARIABLES_INTERNES
PAS> EFFACE ISOVALEURS
PAS> POSTRAITE RESULTATS-ELEMENTAIRES /DEFORMEE SIGMA-MISES
PAS> ACTUALISE ECRAN REDESSINE
PAS> TESTER/VALEUR=IPAS INFERIEURE/VALEUR=MPAS ALLER 100

Sic3> EXECUTER MACRO 'NEWTON'/P-1=350 0.0001 0.001 10
Sic3>
```

Le GE gère également des variables globales du logiciel, ainsi que des variables locales aux macro-commandes et offre d'autres services tels que :

- la sauvegarde de l'historique (la liste de commandes d'une session),
- la création d'un fichier espion qui permet la reprise d'une session,
- l'exécution de fichiers de commandes permettent d'utiliser le logiciel en mode "traitement par lots".

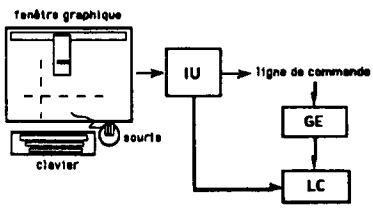
Une autre particularité du GE est la possibilité offerte à l'utilisateur de spécifier, de manière déclarative, la syntaxe et la sémantique du langage de commande. Celles-ci sont en effet définies dans des fichiers de texte modifiables dynamiquement (xxxx.DSC). Elles peuvent être adaptées à une application particulière pour adhérer à la terminologie propre à un métier. Ainsi, en mécanique on appelle "déplacements" les degrés de liberté d'un nœud alors qu'en thermique on parlera de "températures" ; la différence est purement syntaxique et les opérateurs à mettre en œuvre sont identiques. La possibilité d'édition du fichier de définition de la syntaxe du langage de commande facilite également l'internationalisation du logiciel. On peut s'en servir également pour limiter l'accès à certains opérateurs jugés superflus ou dangereux dans un environnement donné, en supprimant la syntaxe correspondante du fichier, le logiciel restant inchangé.

4.6. *Interface Utilisateur (I.U.)*

L'interface utilisateur constitue la couche de gestion de l'interactivité entre l'utilisateur et le logiciel. L'écriture de cette portion du logiciel, variable d'une application à l'autre, constitue la phase ultime du processus d'industrialisation du logiciel. Cette couche permet d'isoler l'utilisateur de la complexité du logiciel général, lorsque cela s'avère utile, et de créer des outils métiers simples adaptés à des classes d'applications limitées. Il y aura donc autant d'IU qu'il y aura d'outils métiers. Pour réaliser des IU efficaces, on prendra en compte les habitudes et les modes de travail de chaque groupe d'utilisateurs industriels.

Du côté "utilisateur" de l'IU, on pourra utiliser toutes les techniques disponibles : des menus bien sûr mais aussi des images vidéo, la reconnaissance de la parole, divers modes de présentation graphique. De nombreux outils, basés sur la norme OSF/MOTIF facilitent la créations d'IU très raffinés.

L'IU est couplé avec le reste du logiciel par l'intermédiaire de lignes de commande qui sont ensuite interprétées par le GE, ou plus directement par l'intermédiaire d'un objet d'appel de commande. Les commandes générées par l'IU activeront plus souvent des macro-commandes que des opérateurs de base. Le passage par le langage de commande isole mieux l'IU du reste du logiciel, et simplifie son écriture : on peut ainsi simuler aisément le fonctionnement de l'interface indépendemment du reste du logiciel.



L'individualisation de l'IU proprement dit facilite les développements, car, adapté à un problème particulier, il présente des spécifications assez simples, et ne doit connaître qu'un nombre réduit de lignes de commande. Il est facile de modifier un IU sans tenir compte de la complexité du logiciel général, et en se concentrant sur la convivialité et l'ergonomie de l'IU. Toute commande issue de l'IU doit s'exécuter sans problème dans le logiciel, la vérification de la cohérence des données étant assurée par l'IU. Contrairement au GE, l'IU n'admet pas de données invalides (en dehors du domaine admissible par le logiciel) et ne permet pas l'exécution d'une commande avant que toutes les données nécessaires ne soient collectées et validées. Cela simplifie la tâche de l'utilisateur inexpérimenté en évitant que le logiciel ne s'arrête sur une erreur, et conduit à une amélioration de la*connivence* entre l'utilisateur et le logiciel et une meilleure *tolérance du programme aux erreurs humaines.*

5. Bibliographie

[FEL 81] C.A. Felippa "Architecture of a distributed analysis network for coputational mechnics", *Computer & structure vol. 13* (1981) p. 405-414

[FEL 80] C.A. Felippa "Database in Scientific Computing Part I", *Computers & Structures vol. 12* (1980) p. 131-146

[AUN 90] S. Aunay, Architecure de Logiciel de modélisation et Traitements Distribués, thèse de doctorat, UTC, France 1990

[DHA 84] G. Dhatt, G. touzot, *Une présentation de la méthode des éléments finis*, 2e éd., Maloine S.A., Paris 1984

[BRE 91] P. Breitkopf, C. knopf-Lenoir, L. Morançay, G. Touzot, R. Younsi, "An Intefrated Software Architecture for Generalized Sensitivity Analysis and Optimization of Physical Systems" *European Conference*, Giens April 1991

[SAM] S.AM.C.E.F., "Système d'Analyse des Milieux Continus par Eléments Finis", SAMTECH, Liège, Belgique

[MAC] "MSC/NASTRAN", The MacNeal-Schwendle Corporation, Los Angeles, California