Implémentation informatique.

Résumé

La *conception d'un logiciel, et à plus forte raison d'un système informatique, n'est pas chose facile. Ce chapitre a pour objectif de présenter les grandes lignes de la réalisation du système Sicam2. Après une présentation synthétique des spécificités du Génie Logiciel, nous abordons l'aspect pratique de la réalisation avec l'étude de l'architecture logicielle. Cette architecture est étudiée sous l'angle des flux d'informations internes au logiciel, puis des communications hommes/machines. Ce dernier point amène une réflexion sur l'aspect ergonomique et sur l'utilisation du logiciel qui sont traités dans la dernière partie de ce chapitre. Nous terminons par quelques commentaires relatifs au logiciel*[1].

Contenu du chapitre

1. Dans un souci de clarté, ce chapitre comporte volontairement quelques rappels d'éléments (tels que des illustrations) présentés précédemment.

6.1 Introduction.

Au même titre que tout produit manufacturé complexe, le logiciel ne peut prétendre à une qualité totale [GMSB96]. On aimerait cependant disposer d'indicateurs de qualité, mais à défaut, on doit souvent se contenter du niveau de satisfaction des utilisateurs. Ceux-là-mêmes se sont en majorité résignés à supporter les bogues et autres désagréments d'utilisation qui ne sont pourtant pas une fatalité. La non-conformité d'un logiciel au cahier des charges initial est un tout autre problème et doit être considérée comme intolérable, surtout lorsqu'elle a des conséquences catastrophiques en termes économiques, ou pire, de vies humaines [Jau90]. M. Barthe relève un décalage entre l'efficacité d'utilisation et les fonctionnalités du logiciel. Il avance quelques causes de la sous-exploitation des capacités du logiciel telles que des fonctions sans intérêt, une mise en œuvre trop compliquée des fonctions ou bien un temps d'apprentissage trop long [Bar95]. Les conséquences les plus connues sont une démotivation des utilisateurs ainsi qu'une hausse des coûts cachés, et vraisemblablement une désaffection pour l'informatique en général. Une partie de ces problèmes est à porter au crédit du manque d'ergonomie, sujet que nous traitons dans le paragraphe 6.3.

Le logiciel, vu comme un produit industriel, est soumis à un processus de production engageant généralement de nombreuses ressources humaines, mais sa production est unitaire (la diffusion en nombre ne requiert qu'une simple opération de copie, sans aucune logistique lourde). Il se caractérise cependant par une tendance marquée au dépassement de délai dans les phases de conception (les exemples sont légion, surtout du côté des logiciels «millésimés»). La méthode de développement doit s'adapter au contexte de création et tenir compte du fait qu'un logiciel reste un produit *invisible* durant son développement. Une solution consiste à procéder à un maquettage du logiciel (ou prototypage). On distingue deux catégories de maquettes : les maquettes exploratoires —l'utilisateur précise ses attentes à l'aune de ce que lui présente le concepteur— et la maquette expérimentale —où le concepteur teste des solutions concurrentes—. Dans les deux cas, la spécification initiale des caractéristiques est essentielle à la définition du produit. La création d'un logiciel scientifique expérimental *ne respecte pas la même logique que la production d'un logiciel commercial*. Nous nous sommes cependant inspirés de certaines pratiques du génie logiciel afin de rationaliser nos efforts. L'une d'entre-elles est la modularité du logiciel, qui facilite sa maintenance, la réutilisation de ses composants et surtout la conception concourante. Elle nécessite des choix importants d'architecture, de langages, que nous détaillons dans les sections suivantes.

6.2 Architecture logicielle.

L'architecture des logiciels scientifiques et expérimentaux nécessite de pouvoir mettre au point et modifier aisément le code. Les systèmes informatiques, particulièrement dans le domaine de la CAO, ont une durée de vie beaucoup plus courte que les produits qu'ils ont aidés à concevoir ou gérer [Loy91]. Il importe donc que le logiciel soit évolutif, et portable sur de nouveaux matériels. Cela se traduira en termes de choix d'architecture logicielle et de langages. D'autres paramètres environnants sont à prendre en compte lors de la conception d'un logiciel. Considérons les acteurs du projet. La conception d'un logiciel de CAO dans le cadre d'un laboratoire de Génie Mécanique fait appel à des mécaniciens (!) qui ne sont pas toujours au faîte des techniques du Génie Logiciel. Nous avons donc composé avec ces compétences. Cela se traduit en partie par des choix de langages que nous justifions en suivant.

6.2.1 Choix des langages de programmation.

Le choix du langage de programmation est un élément essentiel de la réussite d'un projet informatique. Le langage doit être bien défini syntaxiquement et sémantiquement et adapté au domaine d'application [GMSB96].

Le choix du langage C pour l'écriture des modules de calcul s'est imposé par sa pratique très répandue, ainsi que son aptitude au calcul numérique. Il n'est plus nécessaire de présenter le langage C [KR84], né dans les années 70, intronisé depuis par une norme ANSI, qui bénéficie actuellement d'une imposante collection de routines, bibliothèques de fonctions ou outils de développement, souvent gratuits grâce à l'effort de la communauté libre des informaticiens. Ce langage de programmation constitue une référence (bon nombre de systèmes d'exploitation et de langages actuels sont écrits en C !). Le C facilite également l'écriture modulaire de logiciels. Les algorithmes de calculs ont été étudiés et encodés à des dates différentes, ce qui nous a obligé à en faire des modules, dans un premier temps indépendant, que nous avons pu tester et valider individuellement.

Le langage C++ [Bro01] a été imposé par l'utilisation des A.P.I. [2] d'Open I-DEAS®. L'utilisation d'un langage objet était rendu obligatoire par l'architecture CORBA qu'utilise OPEN I-DEAS®. Ce choix nous a également permis de nous orienter vers une librairie mathématique orientée objet, OptSolve+, pour la phase de synthèse coordonée [Cor02].

Afin de faire le lien entre tous ces éléments, et compte-tenu de l'hétérogénéité des langages employés, nous nous sommes tournés vers un langage script. Ce type de langage, présenté comme la révolution informatique du XXI$^{\text{ème}}$ siècle, est souvent assimilé au ciment qui lie les briques d'une construction. Quelles sont les raisons qui nous ont poussé à choisir un langage script ? Les principales que nous avons retenues sont :

- l'application doit intégrer un ensemble de composants ou d'applications existantes ;
- l'application doit intégrer une GUI [3] ;
- l'application doit évoluer rapidement et facilement.

Ajoutons que ces langages sont dotés d'une portabilité hors du commun, qu'ils sont très doués pour le traitement de chaînes de caractères et qu'ils manipulent une grande variété de composants différents [SCR98]. Quatre langages scripts se partagent actuellement le plus gros du marché : Perl, Tcl, Python et PHP. Le Tcl [4], déjà largement employé par ailleurs, a retenu notre attention ([Ous94] et [Ous98]). Sans rentrer dans le détail des petites querelles qui enrichissent les forums, nous avons choisi Tcl pour des raisons (apparentes) de simplicité d'apprentissage, pour l'extension graphique Tk [5] et sa gratuité. Tcl est né en 1988 du besoin d'un langage de commande a raccordé à des logiciels de ... CAO ! Il s'est vu adjoindre une couche graphique en 89 nommée Tk. Ils sont tous les deux codés en C et donc facilement utilisables conjointement (Tcl peut utiliser des fonctions C, ou bien un programme C peut utiliser des bibliothèques de fonctions Tcl via un interpréteur). Tcl est un langage interprété, il n'a donc pas besoin d'être recompilé à chaque modification, ce qui le rend idéal pour l'élaboration de prototype de logiciel. La section suivante traite de l'utilisation de ce langage pour intégrer les divers composants logiciels du système.

2. Application Programming Interface.
3. Graphical User Interface, soit une interface graphique utilisateur.
4. Tool Command Language.
5. Tool Kit, l'interface graphique associée à Tcl.

6.2.2 Liaison des modules.

Nous avons adopté une structure modulaire pour la création du système. Bien que cette méthode présente de nombreux avantages, elle n'est pas naturelle pour un ordinateur qui doit partager sa mémoire entre les données et les divers programmes. Il convient donc de bien opérer le partage des opérations et c'est à l'aide d'un découpage fonctionnel du système et une approche concourante de la construction du système que nous avons architecturé le projet Sicam2. Ce projet est un travail de longue haleine qui ne peut s'inscrire raisonnablement que dans la durée. La version actuelle du logiciel comporte plus de trois mille lignes de codes réparties dans une quarantaine de fichiers, et deux-cent-vingt-quatre modèles volumiques CAO. L'étalement des travaux de développement logiciel a conduit à leur confinement en modules autonomes. Cette modularité a permis la validation partielle, dans un système non finalisé, et une maintenance facilitée du système (et son évolution puisque c'est le principal souci du logiciel scientifique) ; rien n'est plus simple que d'isoler un module pour l'améliorer et de le réinsérer ensuite dans le système. Afin de mener à bien cette modularité, nous avons dû penser aux principaux liens qui devaient exister entre les différents modules : les échanges de données et les mécanismes d'activation. Concernant l'échange de données entre les divers modules, nous avons mis en place un double mécanisme de fichiers de données et de variables globales. L'emploi des fichiers de données en entrée et sortie de modules nous a permis de tester indépendamment les modules en bénéficiant d'une lisibilité parfaite. Ces fichiers sont rédigés en ASCII et font figurer, pour chaque donnée, une ligne descriptive de ladite donnée. Leur nom est composé selon une règle préétablie qui les distingue selon le nom du projet, le type de module concerné ainsi que le type de données contenues. Ces manipulations d'informations textuelles ont été facilitées par l'emploi d'un langage script. Le passage d'arguments par valeur entre module est une des possibilités offertes par le Tcl. Les variables sont déclarées dans des « namespace » locaux ou globaux et sont accessibles par tous les modules qui ont été lancés dans le même interpréteur (ce qui est le cas des modules annexes de Sicam2). L'accès direct, en mémoire, aux variables mises à jour par les menus du module principal est accéléré par rapport à la lecture des fichiers cahiers des charges ou résultats. Cette technique a pour but de réduire les temps d'exécution.

L'ensemble du transit des données de conception est représenté sur la figure 6.1. Celle-ci mérite quelques commentaires. La première source d'informations est constituée par le *cahier des charges principal*. Il rassemble toutes les informations du niveau de définition supérieur, le niveau T3 (cf. §1.6.2.3), dont les positions des liaisons, les dentures d'engrenage... Le module principal produit des cahiers des charges à destination des modules de calcul de chaque liaison. Pour une simplification de la représentation, nous avons englobé dans les modules de calcul les différents algorithmes de dimensionnement propres à chaque DTLM. A titre d'exemple, le module calcul arbre/moyeu comprend les programmes de calcul des liaisons par clavette, par cannelures et par frettage. Chaque module de calcul est donc en relation avec un fichier *cahier des charges,* un fichier *résultats* ainsi qu'un ou plusieurs fichiers *base de données*. C'est dans ces bases de données que l'algorithme de dimensionnement recherche les caractéristiques des composants standards, les propriétés matériaux, etc. On peut constater que le module calcul de la liaison pivot engendre deux fichiers résultats dont un, le fichier nommé résultats statiques, contient les efforts dans les liaisons. Ces données sont *indispensables* au calcul de toutes les liaisons car elles déterminent les sollicitations dans l'arbre. Leur estimation est réalisée dans le module de calcul pivot une fois la décomposition de la liaison pivot connue. Si la liaison pivot est réalisée par une rotule et une linéaire annulaire, les éléments de réduction des torseurs sont alors implicitement connus, et l'on pourrait éventuellement passer au dimensionnement des liaisons arbre/moyeu sans même connaître les DTLM réalisant la liaison pivot. Par contre, l'utilisation de roulements à

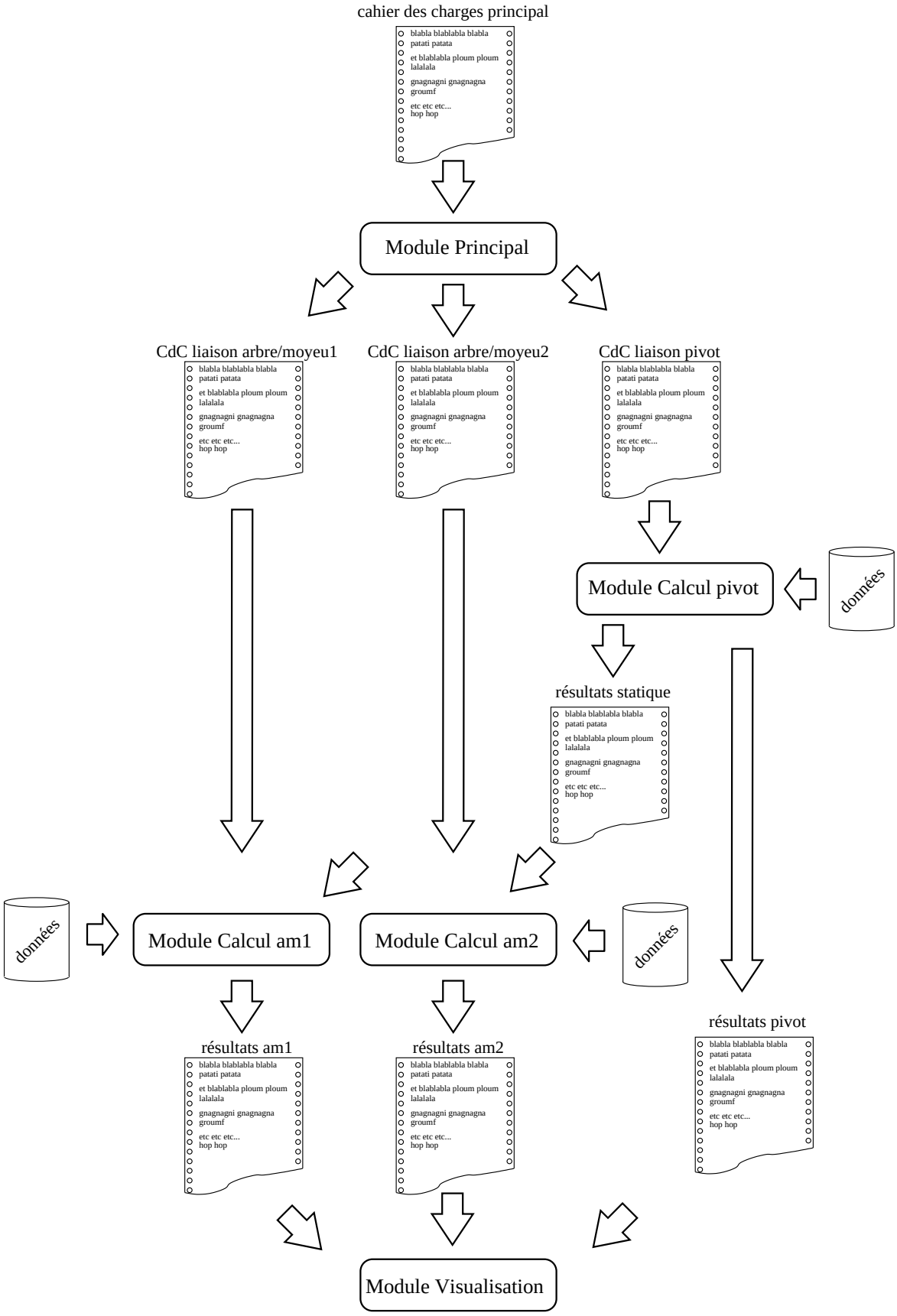

FIG. 6.1 – *Datagramme du système.*

contact oblique ne permet pas de déterminer les torseurs d'efforts tant qu'ils n'ont pas été choisis. Afin d'harmoniser le déroulement du processus de conception, le calcul des DTLM pivots et la détermination des efforts sont couplés dans un même module fonctionnel. Il est donc impératif de commencer la détermination des DTLM par ceux de la liaison pivot. A l'issue des calculs de DTLM, le système dispose de fichiers résultats comportant tous les éléments dimensionnels nécessaires aux modèles 3D. Le module visualisation se charge alors de transmettre ces éléments au modeleur volumique.

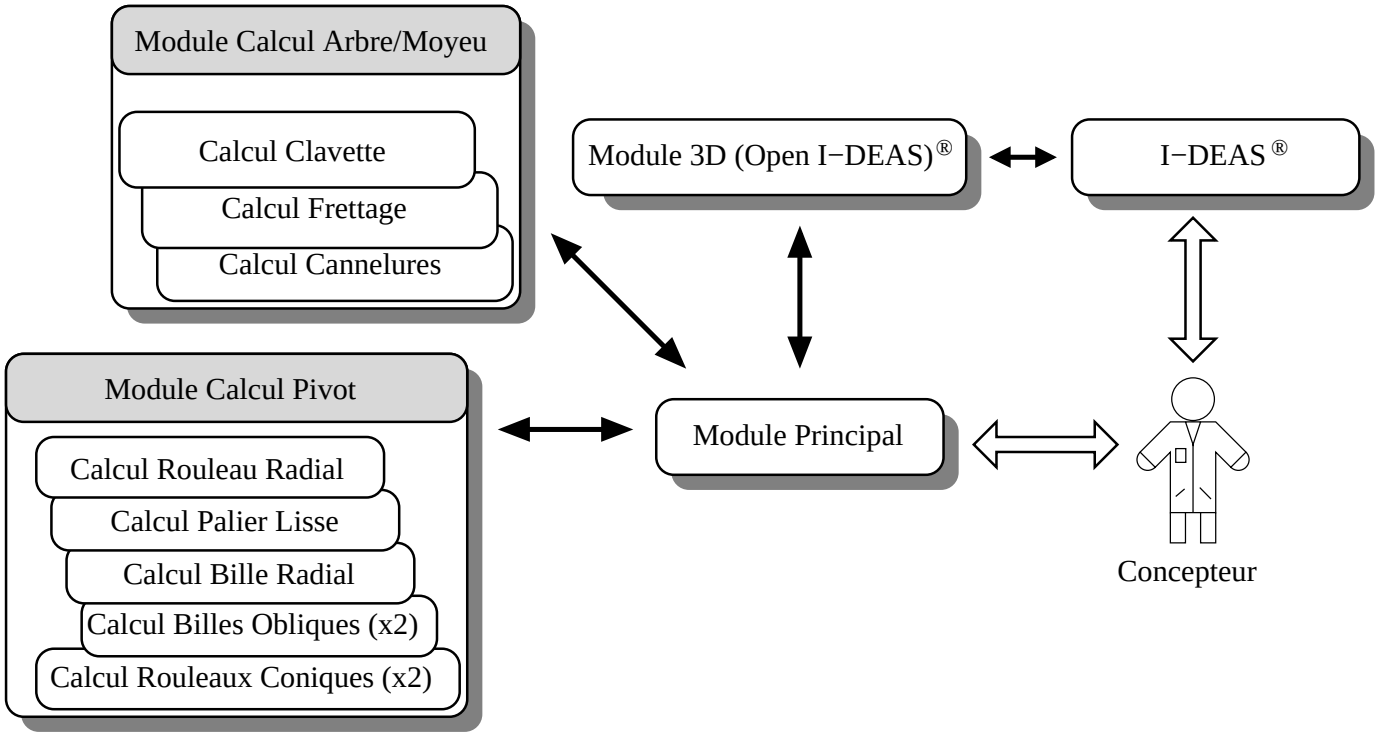

FIG. 6.2 – *Actigramme du système.*

L'activation des différents modules est représentée sur la figure 6.2. Le module principal est au cœur du processus de conception. Il lance les modules de calcul, le module de coordination ainsi que le module de visualisation à la demande de l'utilisateur. L'utilisateur n'a d'interactions avec le système que par le biais du module principal (donc les menus) et I-DEAS®. Le module principal est toujours actif, y compris lors du lancement des modules annexes et conserve ainsi la main en cas de problèmes rencontrés sur ceux-ci. Sans aller trop en avant dans la description d'une séquence de conception avec Sicam2, ceci faisant l'objet des chapitres 3, 4 et 5, nous pouvons en rappeler les grandes lignes suivantes. Une certaine cohérence dans la synchronisation des actions est impérative. Le concepteur doit commencer par remplir le cahier des charges principal, puis s'intéresser à la définition de la liaison pivot qui conditionne les calculs de statiques. Alors il peut, à loisir, définir le reste des liaisons dans l'ordre qu'il désire. L'étape de coordination n'intervient qu'en dernier lieu pour trouver une solution globale (c'est-à-dire impliquant tous les DTLM). Pour bien comprendre l'intérêt d'une définition en deux temps des DTLM, nous renvoyons le lecteur aux chapitres précédents.

6.2.3 L'interface homme/machine.

Les multiples tentatives menées dans le but d'élaborer un système de conception entièrement automatique ont échoué. Ne nous leurrons pas, notre connaissance actuelle du processus cognitif de conception ne nous permet pas de telles prétentions ! Plus modestement, un système de conception assistant efficacement le concepteur est un objectif louable et sage qui nous semble réalisable. Sicam2 a été conçu dans cet esprit. Aussi fait-il appel au concepteur à chaque fois qu'une étape de choix survient dans sa construction d'une solution. Deux raisons à cela :

- Le logiciel n'entend pas se substituer au concepteur dans les choix où son expertise peut (doit) faire la différence. La mécanique regorge de nombreux cas de figure où les constructions, ayant prouvé leur efficacité, n'étaient pas forcément les plus «académiques». En règle générale, le concepteur se raccroche à des solutions éprouvées et connues de tous, d'une part pour se faciliter la tâche, et d'autre part pour s'assurer de leur compréhension (qui vaudra agrément). L'originalité n'est alors pas de mise, et l'on peut passer à côté d'une architecture qui, à défaut d'être révolutionnaire, se révélerait d'une efficience supérieure. en rendant la liberté d'investigation au concepteur, on fait la part belle à la *créativité intelligente* [6].

- Le système informatique doit, et c'est notre intime conviction, se doter de vertus pédagogiques. Les systèmes actuels sont trop souvent basés sur le modèle boite-noire et presse-boutons. L'utilisateur développe une certaine méfiance vis-à-vis de ce genre de logiciels qui n'explicitent jamais les situations d'échecs. Ce constat est d'autant plus criant lorsqu'une part des connaissances est intégrée (et donc cachée) dans le code informatique. Les systèmes experts ont longtemps fait partie de cette catégorie. Ils sont en train d'évoluer vers plus de transparence, notamment en rendant accessible le déroulement du processus, les règles déclenchées, etc. Sicam2 est un outil de construction du savoir. Le concepteur l'utilise comme « calculatrice » sophistiquée afin de dimensionner les DTLM qu'il intègre au mécanisme. Quel élément a penché en faveur de tel ou tel DTLM ? Est-ce la meilleure solution ? Ce DTLM constituerait-il une bonne solution dans un autre contexte ou bien est-il propre à cette configuration de mécanismes ? Quel concepteur ne s'est donc jamais posé ce genre de questions ? En ayant la possibilité de tester rapidement différentes combinaisons de DTLM dans un contexte de mécanisme donné, Sicam2 contribue à enrichir la connaissance du concepteur. Cette démarche est un des principes d'action du groupement d'universités et d'écoles d'ingénieurs PRIMÉCA [7] qui a pour objet de promouvoir le développement des filières de formation d'ingénieurs et de maîtres des universités à la CMAO [Tol00]. Notre souhait est que le logiciel, au sens large du terme, soit un élément d'assistance autant que de construction du savoir. La rapidité de l'outil l'informatique permet d'augmenter considérablement le nombre de cas étudiés et, par conséquent, l'expertise de l'utilisateur.

Puisque l'homme doit dialoguer avec la machine, il est primordial qu'ils se comprennent sans ambiguïté. L'interface homme/machine devra donc être conçue dans le but unique d'échanger les informations entre le système et son opérateur avec une intégrité totale. L'échange bilatéral suppose des mécanismes d'envoi et d'accueil d'informations de la part du système. On distingue deux phases d'échange : la phase de renseignement du système (en amont du dimensionnement du DTLM) et la phase de retour du système (proposition dimensionnelle du DTLM). A chacune

6. Le Petit Robert[RDR02] définit l'intelligence comme la « faculté de connaître, de comprendre ».
7. Pôle de Ressources Informatique pour la MÉCAnique.

de ces phases est dévolue un type d'interface : menus boutons (Tk) et graphisme 3D (I-DEAS®). Les paragraphes suivants détaillent les aspects techniques des menus et de la visualisation 3D. L'aspect ergonomique sera traité en §6.3.

6.2.3.1 Les menus de renseignement du système.

Ils sont écrits en Tk et sont appelés par le module principal dès lors que l'opérateur en exprime le souhait. Ils font une large place aux éléments graphiques afin de visualiser rapidement l'information. Les variables modifiées par ces menus sont globales, c'est-à-dire qu'elles sont accessibles depuis n'importe quelle fonction. Il n'est donc pas rare de voir des variables figurer sous différentes formes, comme les positions des liaisons. La modification d'une de ces valeurs est alors immédiatement répercutée sur l'ensemble du système grâce aux capacités de traitement événementiel [8] du Tk [Wel97]. Cela signifie que Tk intercepte tous les évènements de la couche graphique du système d'exploitation —cela va du simple passage du curseur dans une zone prédéfinie à la commande de fermeture d'une fenêtre— et exécute les commandes par défaut associées à ces évènements. Nous avons alors la possibilité de reprogrammer ces événements pour, par exemple, commander l'ouverture d'un fichier par un double clic sur son nom, ou bien interdire la fermeture d'une fenêtre par l'utilisateur, comme cela est le cas dans notre système. Bien entendu, ces possibilités ne doivent pas desservir le concepteur, en lui imposant un mode de fonctionnement strict. Elles doivent plutôt être vues comme une occasion de mieux contrôler la réponse du logiciel aux actions de l'opérateur, où parfois de prévenir les erreurs. Ces considérations relèvent du domaine de l'ergonomie, et nous renvoyons le lecteur en §6.3 pour plus de précisions sur l'aspect graphique de ces menus.

6.2.3.2 Visualisation graphique 3D des DTLM.

Depuis quelques années les modeleurs volumiques s'ouvrent à l'utilisateur. Dans des mesures différentes, les éditeurs de logiciel optent pour des codes informatiques en partie ou totalement réutilisables par l'utilisateur. La finalité de ces propositions est de fournir un logiciel adaptable aux besoins de chacun. Les éditeurs de logiciel ont bien compris que la demande, malgré quelques éléments communs tels que le modeleur volumique, divergeaient sur les applications métier. En proposant un logiciel dont les fonctions internes sont accessibles, l'éditeur donne la possibilité à l'utilisateur de l'ajuster à ses besoins spécifiques. C'est par ailleurs l'occasion de prendre plus durablement des parts de marché dans un contexte compétitif et très versatile (cf. §1.7.1) avec un investissement moindre dans le développement. Parmi ces éditeurs, nous trouvons MATRA (OPEN-CASCADE), AUTODESK (AUTOCAD®), PTC (Pro-ENGINEER®) et SDRC (I-DEAS®). Au moment où la décision fût prise de retenir I-DEAS®pour la modélisation 3D des DTLM, OPEN-CASCADE n'était pas encore disponible en sources libres. A l'heure actuelle, compte tenu de la plus grande liberté de conception liée à la distribution du code source, cet environnement retiendrait notre attention. L'utilisation de l'extension OPEN I-DEAS®, nous offre la possibilité d'utiliser la majorité des fonctions internes du logiciel dont celles de l'application modeleur volumique (Master Modeler). Celui-ci propose un ensemble d'A.P.I. donnant accès aux principales fonctions de manipulation des entités assemblages et pièces du logiciel grâce à CORBA.

En quelques mots, CORBA a été créé par l'OMG [9] dont l'objectif est de promouvoir la théorie et la pratique de la technologie objet pour les applications distribuées [Gro01]. CORBA est un acronyme pour « Common Object Request Broker Architecture » et désigne une structure

8. Event-driven control.

9. Object Management Group.

d'échange d'objets par requêtes universelles. CORBA définit une méthodologie d'intégration d'applications distribuées en leur autorisant une communication bilatérale, quelles que soient leur localisation ou leur implémentation, grâce à un système de requêtes basées sur le modèle client/serveur. On l'aura compris, la technologie objet est au cœur du processus d'échange. Cette méthodologie est intégrée dans la version 7 d'I-DEAS®par le produit ORBIX®, codé en langage C++, sous forme d'un serveur lancé en même temps que le logiciel. La figure 6.3 représente les interfaces et caractéristiques des composants de l'architecture CORBA [Vin97]. On y distingue

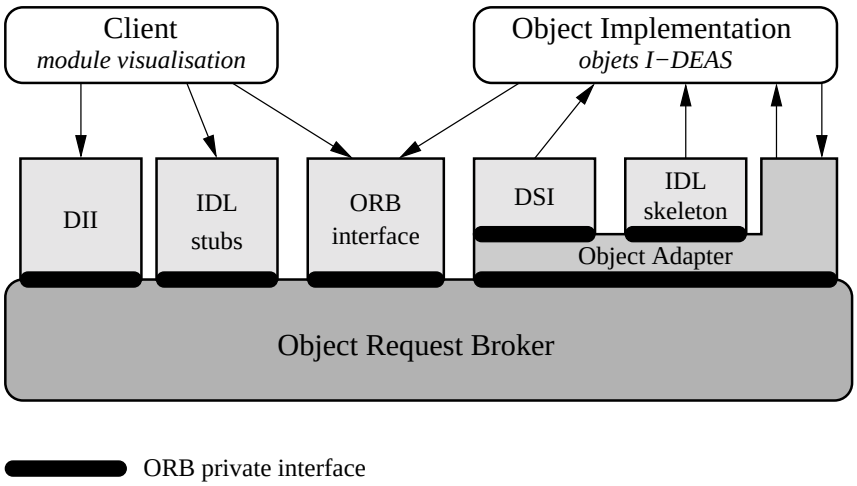

FIG. 6.3 – *Architecture simplifiée de* CORBA.

le client (notre application par l'entremise du module visualisation), le serveur (l'application OPEN I-DEAS®), l'agent d'échange (ORB) et les différentes interfaces. Les « Interface Definition Language» sont compilées en «IDL stubs» et «IDL skeleton» qui forment une sorte de traducteur vers les définitions IDL de CORBA respectivement pour le client et le serveur. Les IDL sont définies de façon tellement rigoureuses que la compatibilité est totale dans les échanges entre « IDL stubs » et « IDL skeleton », même s'ils n'ont pas été écrits dans les mêmes langages. Les « Dynamic Invocation Interface » et « Dynamic Skeleton Interface » permettent à l'utilisateur d'attaquer directement les requêtes au niveau de l'agent d'échange sans passer par les définitions IDL. Les requêtes transitent donc depuis le client vers le serveur par l'intermédiaire de l'agent d'échange. Que sont nos objets I-DEAS®? Ce sont toutes les entités manipulables du logiciel (surfaces, pièces, assemblages, dimensions, etc.) dont on trouve une liste exhaustive dans les IDL d'OPEN I-DEAS®. Les méthodes associées à ces objets sont les manipulations courantes telles que renommer, changer les attributs, déplacer, etc.

L'interface graphique du logiciel I-DEAS®est d'un format relativement classique. Elle se compose de quatre fenêtres (cf. figure 6.4) dont la principale occupe les trois-quarts de l'écran. Elles ont pour fonction l'impression de messages textuels à destination de l'utilisateur, la saisie de données alphanumériques, la saisie de commandes par menu d'icônes et enfin l'affichage graphique des objets modélisés. Nous avons délibérément choisi de ne pas modifier l'interface du logiciel pour des raisons d'ergonomie qui seront exposées dans la section suivante. En conséquence, l'utilisateur ne peut distinguer une version d'I-DEAS®classique d'une version couplée à Sicam2. Au fur et à mesure que le concepteur définit les DTLM à l'aide de Sicam2, il peut les modéliser dans I-DEAS®. La figure 6.5 représente un de ces *assemblages racine*s que nous avons défini au chapitre 3.

L'algorithme que nous avons appliqué (fig. 6.6) implémente des classes d'objets DTLM auxquels nous avons incorporés les méthodes nécessaires à leur traitement. De ce fait, il s'applique

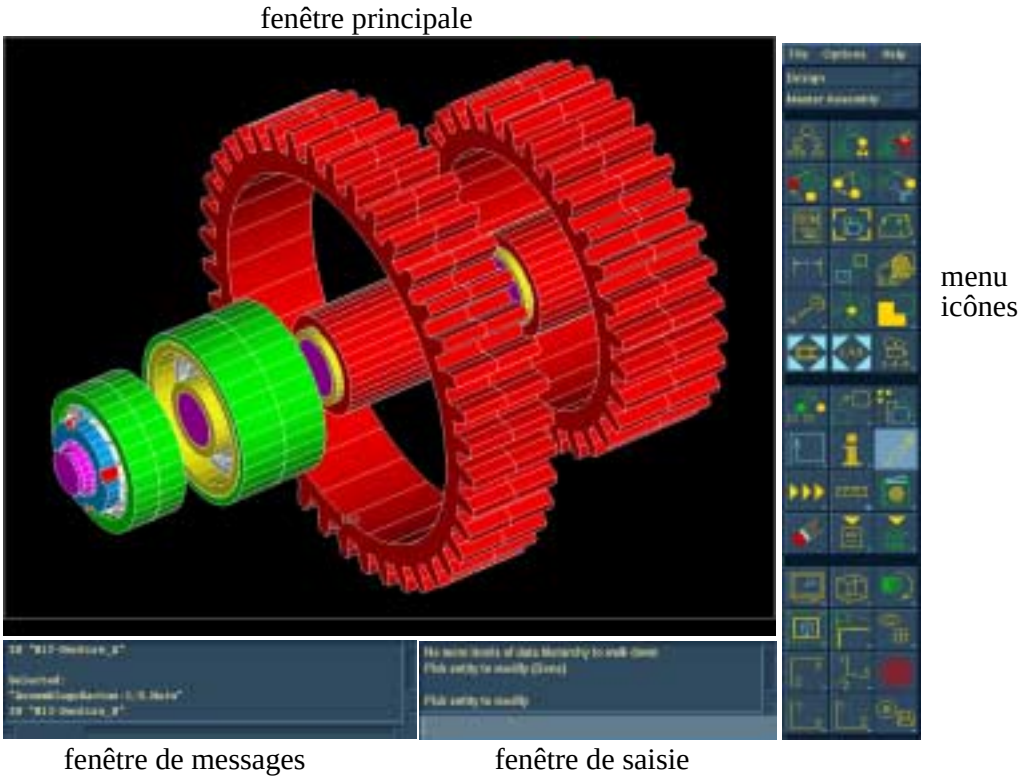

FIG. 6.4 – *Interface graphique du logiciel* I-DEAS®.

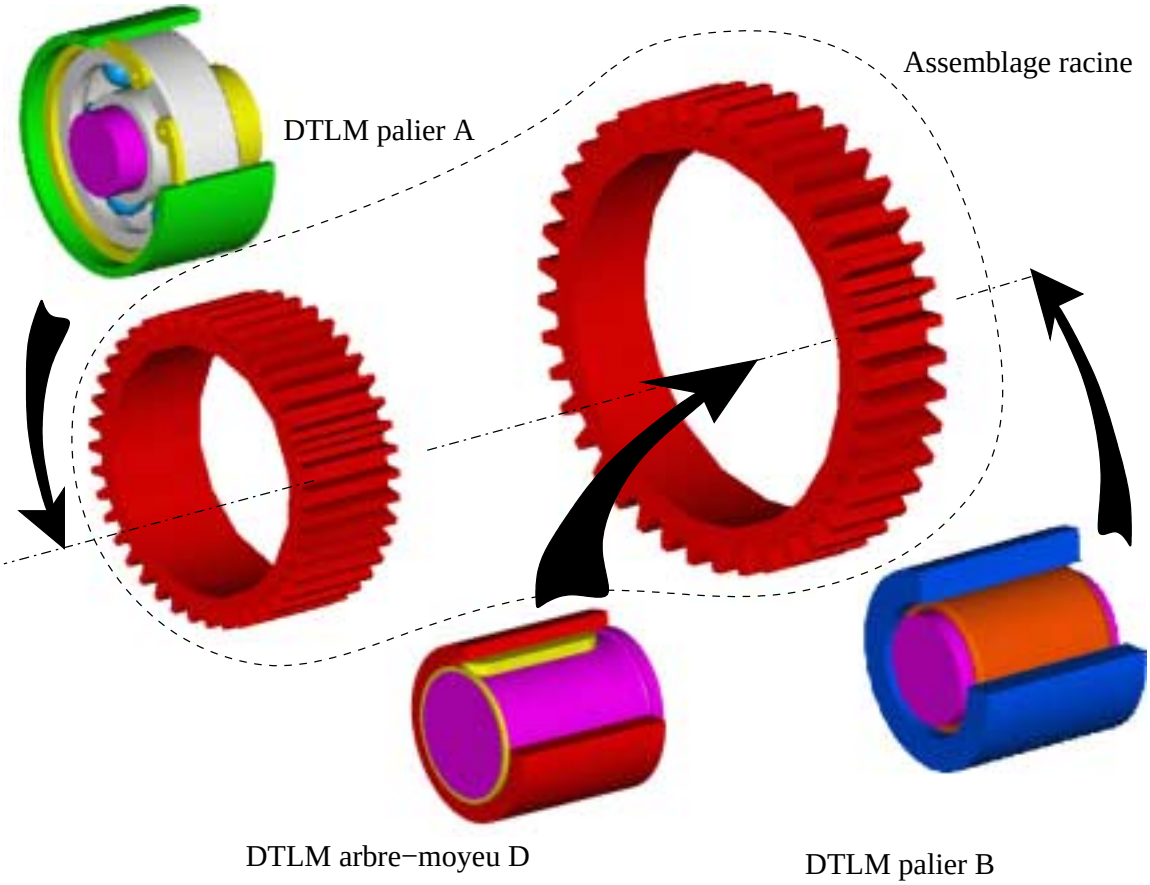

FIG. 6.5 – *Composition d'un assemblage racine.*

à toutes les liaisons, et donc à tous les DTLM. Il a été encodé en langage C++ puisqu'il doit utiliser les IDL d'OPEN I-DEAS®. L'algorithme présenté est extrêmement simplifié, mais il recense les principales étapes de la visualisation d'un DTLM. L'algorithme débute tout d'abord

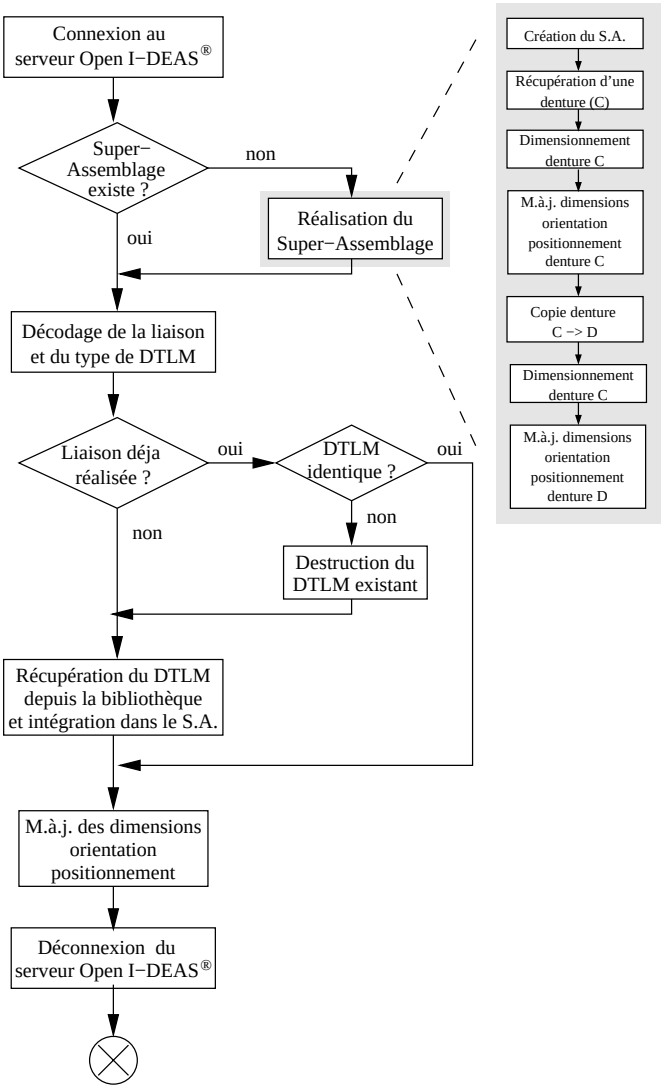

FIG. 6.6 – *Algorithme primaire du module de visualisation.*

par une connexion au serveur OPEN I-DEAS®. Il vérifie la présence de l'assemblage racine dans l'espace de travail et le crée si besoin est, en y intégrant les dentures d'engrenages. Il décode ensuite les arguments reçus du module principal qui lui indiquent quel type de liaison doit être modélisée et à l'aide de quel DTLM. Il vérifie alors que la liaison n'est pas déjà réalisée par un DTLM identique à celui demandé, auquel cas il s'agirait simplement d'un redimensionnement, et procède à l'élimination de l'éventuel prédécesseur. Il reste alors à récupérer le nouveau DTLM en bibliothèque, lire ses paramètres dimensionnels dans le fichier résultat de la liaison concernée, et redimensionner le DTLM. Les modèles volumiques des DTLM sont repérés par un codage de leurs éléments principaux. Ce code unique à chaque DTLM (figure 6.7) autorise l'identification de celui-ci dans toutes les parties du logiciel. L'orientation et le positionnement terminent la mise en place du DTLM dans le mécanisme étudié.

Le paramétrage des modèles 3D des DTLM nous a posé quelques problèmes. Nous avons détecté une lenteur certaine dans le processus de visualisation 3D (entre 1 et 3 minutes suivant

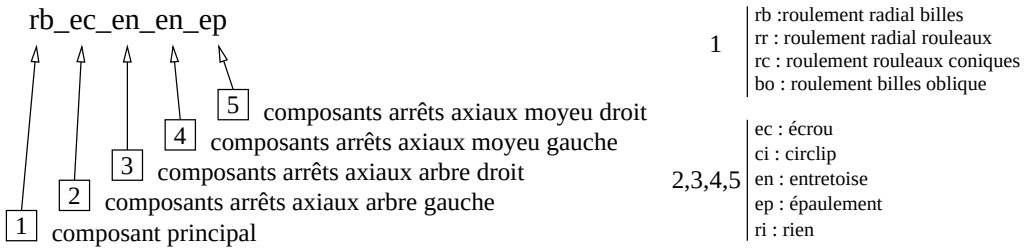

FIG. 6.7 – *Exemple de codage de* DTLM *de liaison rotule.*

le cheminement algorithmique et la complexité des modèles 3D). Cette lenteur, quasi-générale dans le module OPEN I-DEAS®, nous semblait handicapante d'un point de vue ergonomique, car de nature à générer de l'impatience chez l'opérateur. Bien que les temps d'exécution soient restés infiniment moindres que dans le cas d'un traitement manuel, nous avons mené des investigations dans le but de découvrir quel était le maillon faible de la chaîne (i.e. l'opération la plus lente) et le motif de cette lenteur. Compte tenu de la répartition des temps d'exécution du module de visualisation 3D, nous nous sommes focalisés sur la procédure de modification des dimensions et de mise à jour [10] des DTLM, ces deux actions formant le redimensionnement (figure 6.8). Les

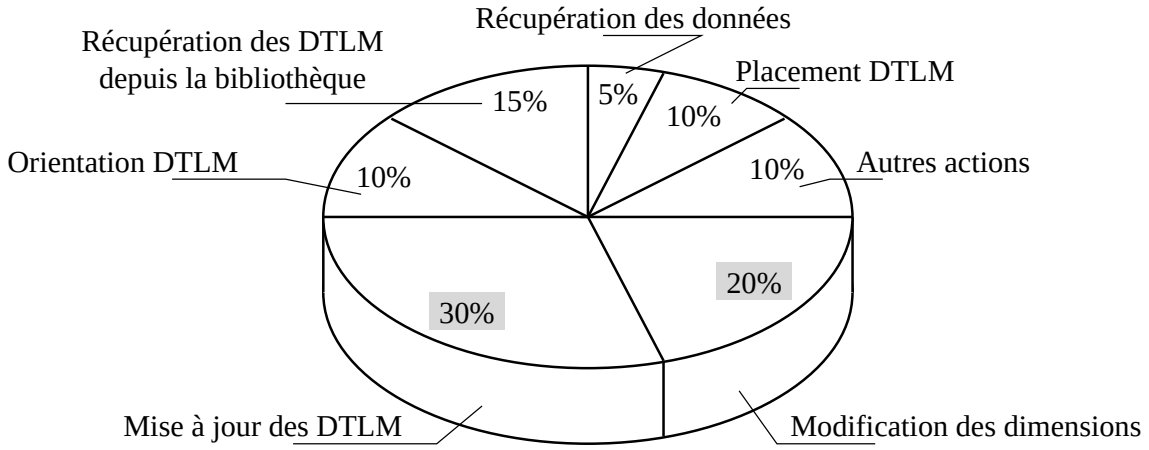

FIG. 6.8 – *Répartition moyenne des temps d'actions du module de visualisation 3D.*

pourcentages affichés reflètent une moyenne obtenue pour la visualisation d'un DTLM dans le cas d'un S.A. vierge (pas de liaison précédemment définie, donc pas de DTLM à effacer). Ils sont globalement indépendants de la complexité du DTLM. La plupart des opérations présentent des temps de traitement incompressibles, à l'image de la récupération des DTLM en bibliothèque (sous réserve d'une modification intégrale de la conception des DTLM pour diminuer le nombre de pièces dès l'assemblage). L'action que nous avons menée concerne le paramétrage des DTLM. Comme il a été précisé au chapitre précédent, le paramétrage du modèle 3D doit être concordant avec le paramétrage mathématique du problème d'optimisation. Ceci n'est pas toujours réalisable, car les méthodes de construction du modèle 3D peuvent répondre à une logique différente. Pour des raisons de robustesse du modèle, on peut préférer, par exemple, une construction filaire — donc à bases de rayons— qui diffère du modèle mathématique utilisant les diamètres. Il convient donc de mettre en place des *règles de conversion* qui, à partir des données mathématiques du problème, produiront toutes les dimensions du modèle 3D paramétré. Prenons l'exemple de la

10. La mise à jour des composants, sous I-DEAS®, est l'opération qui produit le nouveau composant en résolvant, grâce au solveur interne du logiciel, les contraintes géométriques et dimensionnelles affectant celui-ci.

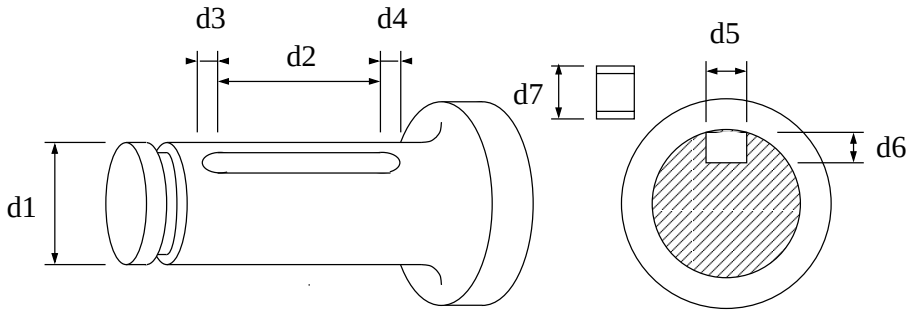

FIG. 6.9 – *Paramétrage simplifié du modèle 3D de l'arbre claveté épaulé.*

liaison arbre/moyeu par clavette dont le paramétrage du modèle 3D est donné figure 6.9. Les deux cas extrêmes sont les suivants :

1. Le module de calcul détermine toutes les dimensions nécessaires au paramétrage du modèle 3D. Le module de visualisation doit alors transférer toutes ces valeurs à OPEN I-DEAS® puis modifier toutes les dimensions du modèle, et enfin mettre à jour le modèle (fig. 6.10).

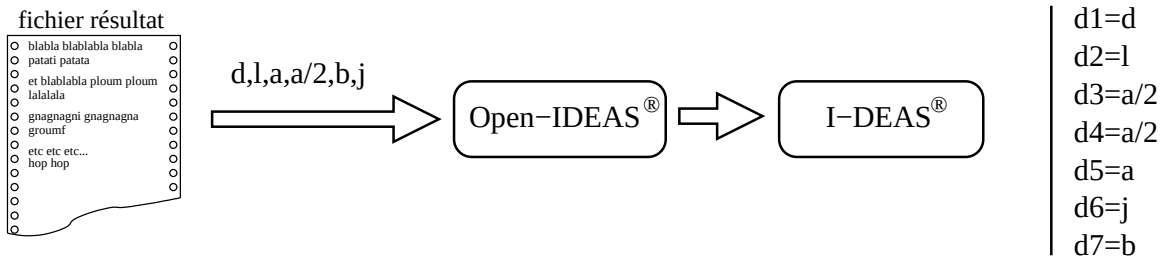

FIG. 6.10 – *Transfert maximal des paramètres dimensionnels.*

2. Le module de calcul détermine uniquement les paramètres de conception du DTLM. Ils sont transmis à OPEN I-DEAS® qui, lorsque l'on procède à la modification des dimensions, propage les dimensions dérivées des valeurs de base (fig. 6.11).

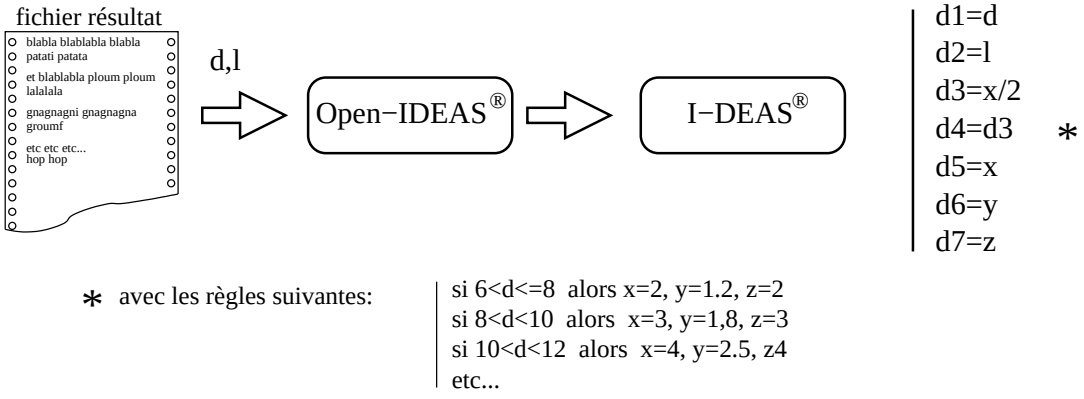

FIG. 6.11 – *Transfert minimal des paramètres dimensionnels.*

Quelles sont les conséquences prévisibles du choix de ces deux configurations ? Il semble logique que dans le premier cas, les temps de transfert de données soient plus important. Nous faisons

l'hypothèse que, toutes choses égales par ailleurs, les calculs numériques sont plus rapides lorsqu'ils sont effectués avec le langage C compilé (le module de calcul), plutôt qu'en script interprété (I-DEAS®). Selon cette hypothèse, nous devrions gagner du temps de calcul dans le premier cas. Reste à savoir si le gain de temps est suffisant en comparaison du surplus engagé pour le transfert des données. Nous avons effectué une campagne d'essais pour mesurer les gains de temps éventuels entre chaque solution extrême. Pour cela, nous avons réalisé deux modélisations sur deux DTLM (de complexité différente), puis nous avons mesuré les temps d'exécutions complets des boucles de visualisation. Les résultats sont les suivants. Le gain de temps est faible, mais toujours en faveur du transfert minimal des données. La différence de gain entre un modèle 3D complexe (faisant intervenir de nombreuses pièces et dimensions) et un modèle simple est négligeable. Les explications à ces résultats semblent être les suivantes :

- le temps de transfert de données via OPEN I-DEAS®est proportionnel au nombre de valeurs à transférer ;
- toutes les dimensions du modèle 3D doivent être modifiées, que ce soit par une affectation simple (cas 1) ou par l'application de règles internes (cas 2). On ne peut donc espérer d'amélioration de ce côté.

En résumé, il faudrait engager de gros efforts de réflexion pour obtenir de faibles gains de temps. De plus, le logiciel est un produit commercial dont l'analyse de la structure nous est totalement impossible. Les mécanismes de modification de dimensions et de gestion de mémoire pourraient nous apporter des informations complémentaires, utiles dans notre démarche. Enfin, l'intégration de toutes les règles de dimensionnement alourdit le modèle 3D et pose dans certains cas un problème de stabilité du système. L'utilisation *intégrale* d'un modèle objet couplé à un modeleur 3D « libre » supprimerait ces problèmes de transfert et de conversion.

En conclusion, nous constatons que, de la structure d'un système informatique (logiciel + matériel), dépendent fortement ses performances. L'utilisation de codes libres où l'écriture complète du logiciel nous libérerait d'un certain nombre de contraintes mais au prix d'un engagement informatique qui n'est vraisemblablement pas à la portée de spécialistes de la mécanique.

6.3 Interface graphique et ergonomie du système.

Dans le milieu des années 80, est apparu un accessoire informatique qui est rapidement devenu indispensable : la souris. Cet accessoire est un des éléments clés de l'environnement graphique WIMP [11] (pour la postérité, c'est le Macintosh® qui restera comme le précurseur en ce domaine). La démocratisation de l'ordinateur est, en grande partie, due à l'avènement de ces nouvelles interfaces graphiques. Les fenêtres se sont progressivement garnies d'icônes, de règles à curseur, de combo-box, etc. que l'on nomme *widgets*. Cette population de petits symboles graphiques grossit régulièrement ses rangs en intégrant des représentations similaires à des dispositifs de la vie courante : potentiomètres, boutons poussoirs, interrupteurs multipositions. L'idée sous-tendue est de simuler le comportement réel de ces dispositifs. Beaucoup de *réflexes opératoires* sont associés aux représentations des widgets, et l'utilisateur s'approprie d'autant plus facilement le logiciel que l'interface lui semble familière. Par exemple, la représentation du menu déroulant déclenche quasi-instantanément chez l'utilisateur le réflexe de choix d'élément dans une liste finie de propositions. A l'inverse, une zone de capture de texte lui suggère une donnée

11. Windows, Icon, Mouse and Pull-down menu (les éléments incontournables des environnements graphiques actuels).

de valeur non prédéfinie. Cette habitude de manipulation, nous avons voulu la mettre à profit dans notre interface graphique. Notre tâche a été grandement facilitée par l'emploi de widgets propres au système sur lequel l'opérateur a l'habitude de travailler, et ce grâce à Tk qui reprend intégralement les widget du système sur lequel il est implanté. On évite ainsi l'apprentissage de nouvelles représentations à l'utilisateur et l'on favorise l'*intégration cohérente* d'outils distincts sur un même poste de travail.

L'ergonomie d'un logiciel ne saurait se réduire à l'apparat de son interface graphique (on a d'ailleurs trop souvent tendance à confondre *interface homme/machine* et *interface graphique*). L'interface graphique n'est qu'un outil du dialogue. Son élaboration n'est donc pas aussi simple et le résultat conditionne grandement la facilité de prise en main du logiciel. M. Barthe, dans [Bar95], pense que « l'ergonomie est un problème de conception et non de réalisation du logiciel ». Il dispense donc un certain nombre de conseils à suivre lors de la conception du logiciel. En premier lieu, il préconise un « tri pertinent de l'essentiel et de l'accessoire [qui] suppose une parfaite connaissance du travail des opérateurs ». Cette opinion est unanimement partagée par les acteurs du Génie Logiciel. Les besoins, et surtout la prise en compte du fonctionnement intellectuel de l'utilisateur importe plus que l'apparence de l'écran. En d'autres termes, le concepteur de logiciel doit épouser les attentes du futur utilisateur afin de lui proposer un outil avec lequel il pourra travailler en parfaite *synergie*. On déplore aussi le manque de considération des erreurs possibles de l'opérateur. On doit tenter de mettre en place des mécanismes visant à éviter les erreurs et une méthodologie de traitement de ces erreurs. Enfin, comme nous l'avons vu dans les paragraphes précédents, l'ordonnancement des opérations est aussi un point important de l'ergonomie. Le logiciel ne doit pas être à l'initiative du dialogue, mais il doit être attentif aux désirs de communication de l'opérateur. Cela se traduit par un accès facilité aux éléments de dialogue là où le concepteur peut en éprouver le besoin. A l'inverse, de nombreux logiciels proposent des structures de menu qui ne sont qu'une reproduction de la structure interne du logiciel ; on est alors dans le cas où la *logique d'utilisation* n'est pas toujours identique à la *logique de fonctionnement*.

6.4 Commentaires sur l'interface proposée.

Plusieurs commentaires s'imposent à l'issue des paragraphes précédents. Le système que nous avons mis en place tente de respecter le mode de travail du concepteur dans la phase de détermination des DTLM. L'interface graphique, qui ne représente que la part visible du logiciel, s'adapte à ce mode de travail. On retrouve, au travers de la hiérarchie des menus, une partie de la structure de données orientée objet discutée au chapitre 2. La hiérarchisation des données se traduit, par exemple, par une chronologie de renseignement du système et une différenciation des fenêtres (comme l'objectif d'optimisation qui se situe dans la fenêtre précédent celle du choix des arrêts axiaux de la liaison arbre/moyeu). Néanmoins, nous avons tenu à limiter l'explosion du nombre de fenêtres afin d'améliorer la lisibilité et le repérage du concepteur ; c'est pourquoi l'arborescence des menus ne présente pas plus de deux niveaux. Rappelons que Sicam2 ne présente pas de point de blocage, car le concepteur peut toujours remonter dans la hiérarchie de définition des DTLM afin de tester de nouvelles solutions. Tous ces efforts ont pour but l'amélioration de l'ergonomie du logiciel, d'un point de vue fonctionnel et esthétique. Cet objectif peut sembler secondaire dans le cadre de travaux de recherche, mais nous pensons qu'une étude de faisabilité doit montrer les possibilités d'avancement dans tous les domaines concernés. Le dialogue homme/machine est à la base du logiciel de CAO, et s'avère parfois être son talon d'Achille. De plus, dans le cas d'une campagne d'évaluation (test en grandeur réelle),

une interface homme/machine réussie constitue un élément perturbateur de moins au travail d'analyse.

Suivant ce principe, nous n'avons pas effectué de modification de l'interface d'I-DEAS®, bien que cela eut été aisément réalisable, pour ne pas perturber l'utilisateur dans son utilisation quotidienne du logiciel. Il doit pouvoir poursuivre sa conception à l'aide des éléments de mécanisme obtenus grâce à Sicam2, les DTLM, de façon transparente. Dans le même esprit de transparence, signalons l'homogénéité de la signalétique employée dans Sicam2, ainsi que l'utilisation des widgets propres au système, par le biais de Tk.

Il serait cependant prétentieux de dire que l'interface homme/machine de Sicam2 est parfaite. Nous déplorons une lenteur certaine dans la phase de visualisation 3D qui est de nature à générer un handicap ergonomique. On doit envisager une implémentation totale en C.O.O. afin de limiter les nombreux transferts entre les modules. La phase de redimensionnement des modèles 3D est particulièrement longue, mais elle reste compétitive par rapport à ce que pourrait obtenir manuellement un opérateur (5 à 10 fois plus rapide suivant le niveau d'expérience).

Le traitement événementiel constitue une avancée considérable dans l'amélioration des relations entre l'opérateur et la machine. Il permet notamment une gestion *active* et *préventive* des erreurs en neutralisant par exemple, en temps réel, les options contraires à un contexte créé dynamiquement par l'opérateur. Nul doute que ce genre de traitement de l'information en temps réel puisse apporter une meilleure convivialité aux logiciels de CAO. Enfin, il est légitime de se questionner sur le devenir de nos observations dans le cas où les modèles employés seraient sensiblement complexifiés. A cela, nous répondrons en deux points :

- La croissance de la puissance de calcul des ordinateurs (loi de Moore) est telle que l'on ne peut plus invoquer la complexité des modèles comme frein au développement d'assistances informatiques. Certaines méthodes de programmation que l'on délaissait voilà 15 ans, comme l'énumération, sont de nouveaux exploitées, avec les avantages qu'elles seules procurent. Il faut savoir adapter la complexité des modèles aux méthodes informatiques disponibles, sachant que les règles de conception du Génie Logiciel évoluent rapidement.

- Nous nous situons dans un contexte d'avant-projet, pour lequel de nombreuses données sont soumises à variations. L'emploi de modèles agrégés simples représente une nécessité. Ces modèles dirigent le concepteur vers un domaine de solutions qui doit volontairement rester large. Imaginons un instant que les modèles mis en place soient rigoureux au point de ne posséder qu'une marge d'erreur infinitésimale. On pourrait donc définir au plus juste tous les composants du mécanisme avec le risque que la moindre modification d'un des paramètres remettent en cause l'intégralité du mécanisme. En quelque sorte, les « marges d'erreur » des modèles fournissent une certaine pérennité aux solutions proposées.

L'idée de prôner l'utilisation de modèles simplificateurs, et non «simplistes», est liée à la synthèse progressive d'un mécanisme. De même que le concepteur progresse dans la structure du mécanisme, les modèles doivent gagner en précision ; cette précision doit toutefois rester à l'échelle des approximations faites sur le fonctionnement du mécanisme. La définition des DTLM n'est pas l'étape ultime de ce processus, et l'introduction à ce niveau, par exemple, de modélisations fines des accidents de formes n'aurait aucun sens compte tenu de l'imprécision du calcul des efforts dans le système.

Le prochain chapitre présente les conclusions de notre travail et dégage des perspectives de recherche dans les différents domaines que nous avons abordés.