Conception et développement de systèmes d'aide à la compréhension de données biologiques et moléculaires; application à la modélisation des récepteurs couplés aux protéines G

Fabien Campagne

G.F.D. Chimie et Physico-chimie moléculaire et théorique
U.F.R. Sciences et Techniques de la Matière et des Procédés

Conception et développement de systèmes d'aide à la compréhension de données biologiques et moléculaires

Application à la modélisation des Récepteurs Couplés aux Protéines G

THÈSE

présentée et soutenue publiquement le 28 septembre 1998

pour l'obtention du

Doctorat de l'Université Henri Poincaré – Nancy 1

(spécialité Chimie informatique et théorique)

par

Fabien Campagne

Composition du jury

Rapporteurs :	G. Deléage
	P. Rodriguez-Tomé
Membres :	J.-L. Rivail
	G. Vriend
	B. Maigret (directeur de thèse)
	A. Napoli (co-directeur de thèse)
Invités :	M.-P. Lefranc
	J.-M. Bernassau

Équipe de Biochimie théorique, UMR 7565 CNRS/UHP Nancy I

Mis en page avec la classe thloria.

Remerciements

J'aimerais remercier ici toutes les personnes qui m'ont aidé, directement ou indirectement, pendant la réalisation de ce travail. Je remercie vivement le Pr. G. Deléage et le Dr. P. Rodriguez-Tomé d'avoir accepté d'évaluer ce mémoire et de rédiger un rapport détaillé.

Je remercie Amedeo Napoli de m'avoir fait découvrir certains aspects de l'informatique que je ne considérais pas avec assez d'intérêt (la branche IA). Si ce mémoire réussit à expliquer, c'est pour une grande part grâce à Amedeo: depuis la rédaction de mon mémoire de DEA, jusqu'à celui-ci, ses commentaires et ses critiques pertinentes m'ont accompagné pendant mes tâches de rédaction,... maintenant même en son absence.

Je remercie Bernard Maigret de m'avoir proposé « ... un petit truc à faire. Tu en as pour deux semaines...» au début de mon DEA. Le *petit truc* s'appelle maintenant Viseur. Bernard sait faire confiance et donner les meilleurs outils de travail aux membres de son équipe. Je le remercie de sa confiance et des facilités dont j'ai bénéficié.

Je voudrais remercier Gert Vriend pour avoir accepté de faire partie de mon jury de thèse, et surtout pour ses invitations renouvelées à Heidelberg, où en quelques jours j'apprenais beaucoup alors que je participais, dans la mesure de mes moyens, à GPCRDB. Je ne suis pas toujours d'accord avec Gert sur les moyens d'obtenir un résultat, mais je sais que nos discussions ont participé à ma compréhension des problèmes pratiques de la bioinformatique. Je remercie aussi les membres de l'équipe de Gert: Florence Horn, Johnaton Weare, Rob Hooft qui ont su rendre mes séjours encore plus instructifs.

Je voudrais remercier le Pr. Marie-Paule Lefranc pour l'intérêt qu'elle porte à ce travail et pour avoir accepté de faire partie de ce jury de thèse.

Je remercie aussi le Pr. Jean-Louis Rivail d'avoir accepté de représenter l'Université Nancy I parmi les membres de ce jury et de m'avoir accueilli dans le DEA Chimie informatique et théorique à Nancy, il y a maintenant quatre ans. Je sais que ses critiques seront avisées et pertinentes alors même que les centres d'intérêts et les méthodes de la bioinformatique sont assez éloignés de ceux de la Chimie théorique.

Je remercie Øyvind Edvardsen pour sa part de notre collaboration à distance. C'était un plaisir que de le voir à Heidelberg au mois de mars 1997, après trois ans d'échanges électroniques.

J'aimerais remercier ici tous les utilisateurs de Viseur, qui par leurs commentaires et encouragements ont contribués au projet. En particulier, et dans le désordre, je remercie Michiel van Rhee, Jean-Louis Reversat, Paul De Cointet, Gennady Poda, Ted Shieh, Michel Seigneuret, Brian Osborne.

Je remercie Sanofi Recherche pour le contrat numéro 061064300, qui a permit de financer une partie des investissements de développement du projet Viseur, entre le DEA et ma thèse.

Je remercie le Centre Charles Hermite pour le financement des travaux réalisés pendant la durée de ma thèse. De grandes parties du travail — les développements des deux dernières années sur le projet Viseur, les deuxièmes et troisièmes parties du mémoire — n'existeraient pas sans le soutien du CCH.

Je remercie finalement l'équipe du secrétariat, Mesdames Grillot et Devienne, qui ont accepté de relire pour partie ce mémoire et contribuent ensemble à une ambiance de travail agréable au laboratoire.

Je tiens enfin à remercier tous ceux qui ont su rendre mon séjour au laboratoire de Chimie théorique plus sympathique: les plus jeunes, Hervé Minoux, Nathalie Reuter, Nathalie Million, Yannick Jeanvoin, Gérald Monard, Philippe Aplincourt; et les moins jeunes, Serge Antonczak, Christophe Chipot, Daniel Rinaldi, Marilia Martin-Costa, Claude Millot, Manuel Ruiz-Lopez.

Table des matières

Table des figures

Avant-propos

Le document que nous présentons ici, comme tout écrit de son espèce, veut condenser les connaissances et les pensées de son auteur en un lieu du temps et de l'espace particulierement réduits : ces quelques feuillets, au moment où je me suis assis et ai décidé d'écrire. Une idée particulièrement répandue, qui à ma connaissance ne s'appuie sur aucune vérification méthodique, consiste à croire qu'il est possible de traduire des connaissances et des pensées sous une forme écrite, de telle façon que l'information passe sans erreur de l'auteur aux lecteurs. Je n'adhère pas à cette croyance pour ma part, et au contraire, j'estime que l'erreur intrinsèque à toute tentative de communication est une source de richesse qui permet aux idées d'évoluer alors qu'elles sont transmises d'un individu à un autre — peut-être ma compréhension des mécanismes de l'évolution est-elle trop abstraite?

Ceci étant posé, rien ne m'empêche, comme beaucoup d'autres avant moi, de relever le défi et de tenter de vous faire découvrir quelques idées et connaissances, qui même si vous les transformez pendant votre lecture pour les adapter à votre compréhension de ces sujets, pourraient vous être utiles sous leur nouvelle forme.

Introduction

1 La bioinformatique à la croisée des chemins

La bioinformatique est une discipline scientifique qui s'est formée sous l'impulsion d'un besoin. Les figures 1 et 2 illustrent l'accumulation des données expérimentales dans deux banques de données. Graduellement, alors que la quantité d'information acquise par l'expérience s'accumulait dans les laboratoires, il est devenu évident que l'utilisation des moyens informatiques était indispensable au stockage et à une utilisation raisonnée des ressources en données biologiques. La question que soulève la bioinformatique est celle du « comment ? » : comment concevoir et réaliser les outils informatiques qui permettront de tirer le meilleur profit, à un certain moment, des données que génèrent les disciplines des sciences du vivant.

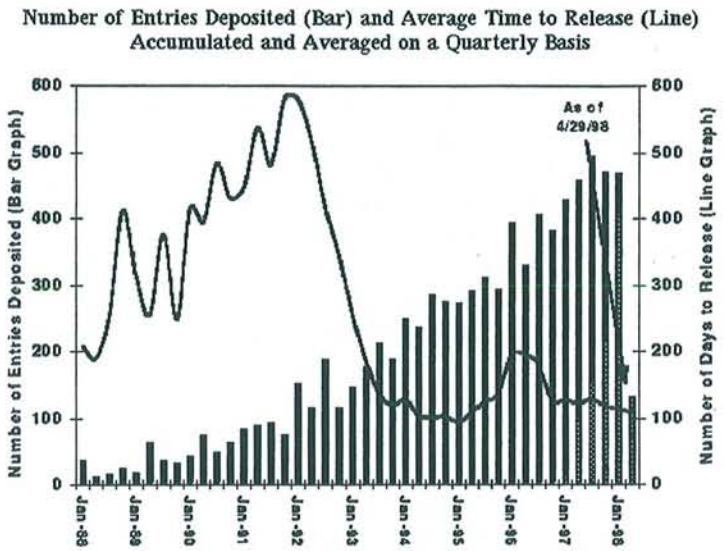


FIG. 1 – *Évolution du volume de données de la Protein Data Bank*
Nombre d'entrées déposées.

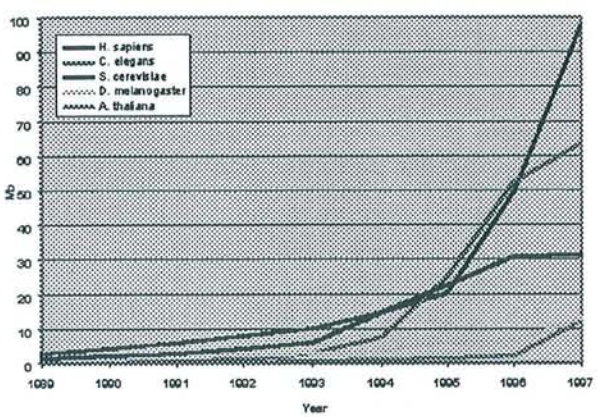


FIG. 2 – *Évolution du volume de données dans la base de nucléotide EMBL*
Nombre de paires de bases total des séquences de plus de 1000 paires de bases.

À la date de l'écriture de ce document, on peut distinguer plusieurs sous-activités de la bioinforma-

tique, par leur rapport aux informations qu'elles permettent de manipuler :

- acquisition, stockage et accès (y compris communication entre bases[1])
- annotation,
- calcul de données dérivées (y compris annotation automatique),
- visualisation et présentation.

Examinons de façon plus détaillée ces quelques activités.

1.1 Stockage et accès : bases de données ou bases de ressources ?

Les bases de données biologiques se transforment. Lors de leur création, elles devaient simplement répondre aux besoins d'archivage et de mise à disponibilité des données. Par exemple, la base de données PDB[2] était constituée, dans les premières années de son existence, d'un ensemble de fichiers décrivant la structure d'une protéine. On observe aujourd'hui une évolution généralisée vers des systèmes qui proposent non seulement des données, mais aussi des données secondaires (dérivées des données brutes), des visualisations adaptées, des traitements particuliers (la construction de données secondaire est dynamique, lorsqu'elle est demandée par l'utilisateur). Les systèmes que l'on appelle *bases de données* évoluent très clairement vers des systèmes que l'on pourrait qualifier de *bases de ressources*. La base de données PDB propose maintenant un service WWW qui permet de chercher l'ensemble des structures par mot clé ou par sous-séquence, des visualisations tri-dimensionnelles des structures de la base, etc.

1.2 Annotation

L'annotation consiste à documenter les données expérimentales en les complétant d'informations dérivées qui aident à les situer dans l'ensemble des autres données. Par exemple, à la vue de la séquence protéique suivante :

```
MELVPSARAE LQSSPLVNLS DAFPSAFPSA GANASGSPGA RSASSLALAI AITALYSAVC AVGLLGNVLV MFGIVRYTKL
KTATNIYIFN LALADALATS TLPFQSAKYL METWPFGELL CKAVLSIDYY NMFTSIFTLT MMSVDRYIAV CHPVKALDFR
TPAKAKLINI CIWVLASGVG VPIMVMAVTQ PRDGAVVCML QFPSPSWYWD TVTKICVFLF AFVVPILIIT VCYGLMLLRL
RSVRLLSGSK EKDRSLRRIT RMVLVVVGAF VVCWAPIHIF VIVWTLVDIN RRDPLVVAAL HLCIALGYAN SSLNPVLYAF
LDENFKRCFR QLCRTPCGRQ EPGSLRRPRQ ATTRERVTAC TPSDGPGGGA AA
```

peu de personnes sont capables de dire quelle est la fonction de la protéine correspondante. La figure 3 illustre l'activité d'annotation. Le résultat est une information plus parlante : l'annotation met en valeur les caractéristiques d'une donnée jugées pertinentes par les personnes qui mettent en place la procédure d'annotation. À la différence du calcul de données dérivées, que nous abordons ensuite, la procédure d'annotation est une opération supervisée par un expert. Celui-ci peut utiliser des données calculées automatiquement à partir des données expérimentales pour s'aider dans sa tâche et pour présenter le résultat de son travail.

1.3 Calcul de données dérivées

Le calcul de données dérivées consiste à construire de nouvelles données à partir des données expérimentales. Les méthodes de calcul varient mais reposent sur un ensemble de modèles biologiques qui fournissent un cadre d'analyse. De ce fait, et parce que le calcul, automatique, ne garantit pas que les données sont parfaitement compatibles avec les modèles pour lesquels il a été conçu, les données dérivées sont moins fiables que les données expérimentales originales ou que des annotations.

1. Le transfert d'information entre bases de données biologiques est indispensable au maintien de la cohérence globale des bases et au calcul des références croisées qui permettent aux utilisateurs de mettre des informations d'origines distinctes en correspondance.

2. Brookheaven Protein DataBank, la base de structures tri-dimensionnelles de protéines.

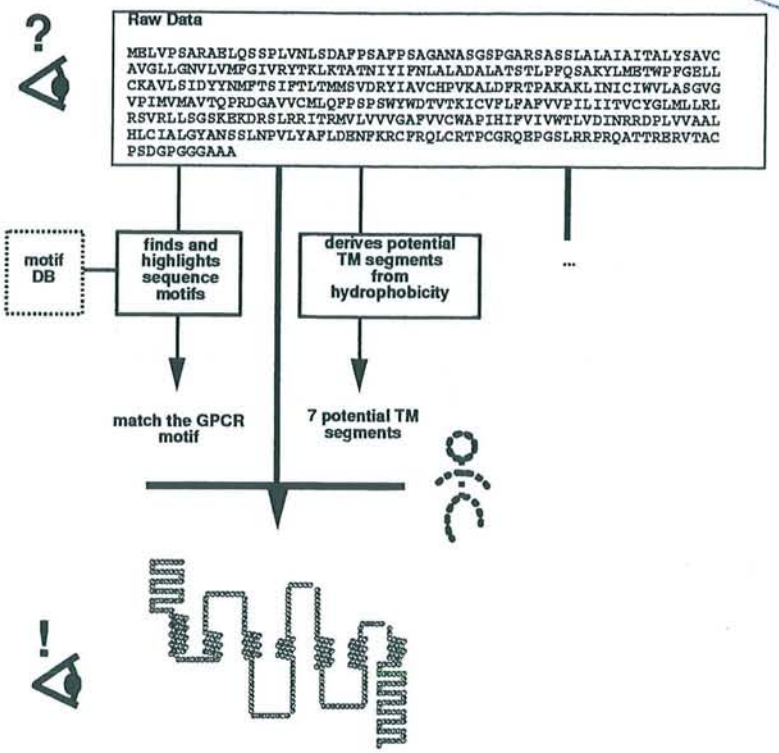


FIG. 3 – *La procédure d'annotation*

La procédure d'annotation consiste à mettre en valeur certaines caractéristiques des données brutes de manière à faciliter leur appréhension. L'expert en annotation utilise des outils de calcul de données dérivées pour mener sa tâche à bien.

2 Thèmes de recherche actuels

Nous citons brièvement ici quelques-uns des thèmes de recherche actuels en bioinformatique de façon à situer la position de ce travail parmi ces thèmes. Cette énumération rapide n'est bien sûr ni exhaustive, ni totalement objective, mais s'inspire en partie de notes prises pendant la session "Future of Bioinformatics" du congrès [Dis96] :

- intégration et/ou mise en correspondance de données hétérogènes,
- visualisations adaptées aux données biologiques (données primaires + annotations + données dérivées) et aux nouveaux moyens de communication,
- définition de standards pour la communication entre Bases de Données Biologiques,
- aide à l'interprétation des données.

3 Présentation du plan

L'organisation inhabituelle de ce document mérite d'être présentée et expliquée. Notre recherche d'un environnement d'aide à l'analyse de données plus efficace n'est pas la première initiative de ce type. Nous décrirons des travaux précédents en troisième partie. Notre démarche se différencie des initiatives précédentes par la manière dont nous *(i)* mettons en évidence et *(ii)* expliquons les besoins de l'analyste (le chercheur confronté à la tâche d'analyse de données). Ceci apparaît dans la première partie du manuscrit, consacrée à la description d'un outil d'aide à l'analyse de données dédié aux Récepteurs Couplés aux Protéines G (GPCRs), que nous développons et maintenons en accord avec les besoins des utilisateurs. Nous ne décrirons donc pas un outil d'analyse de données hypothétique,

mais un outil opérationnel. A la lumière de cette expérience, nous décrirons quels sont les points forts de cet outil, mais aussi ses points faibles. Cette phase de l'analyse est particulièrement importante, puisque la conception que nous décrirons ultérieurement repose partiellement sur ses conclusions.

La deuxième partie nous amènera à découvrir les méthodes de la réutilisation *raisonnée* de codes [3] qui sont indispensables au développement rapide de logiciels fiables. Nous aborderons ces méthodes à travers l'exemple du développement d'un outil de recherche conformationnelle pour petites molécules.

En début de troisième partie nous utiliserons l'expérience acquise pendant la réalisation de nos deux premiers projets lorsque nous chercherons à modéliser l'activité d'analyse de données biologiques (ADB). Ce modèle nous permettra de définir les éléments importants d'un système d'aide à l'analyse de données adaptés à cette activité. Enfin, nous montrerons que des développements informatiques récents permettent de concevoir un système d'aide à l'ADB bien adapté aux contraintes que nous aurons décrites et présenterons le prototype d'un système fait pour faciliter un sous-ensemble des activités d'ADB.

3. Nous employons le mot code pour décrire un logiciel qui peut être utilisé par la programmation (pas seulement du point de vue de l'utilisateur) et qui n'est pas obligatoirement disponible sous forme de sources.

Première partie

Viseur : un projet d'aide à l'analyse de données dédié aux Récepteurs Couplés aux Protéines G

1

Les Récepteurs Couplés aux Protéines G

1.1 Intérêt de l'étude des GPCRs

Les protéines membranaires sont des protéines enchâssées en partie dans la membrane cellulaire. Cette localisation dans un milieu aux propriétés physico-chimiques particulières complique leur étude expérimentale ou théorique. Par exemple, les méthodes de détermination de structure se heurtent à la difficulté de cristalliser ces récepteurs dans leur environnement membranaire. Or, les Récepteurs Couplés aux Protéines G[4] sont particulièrement importants, à l'intérieur de l'ensemble des protéines membranaires, parce qu'ils sont l'élément crucial d'un système de communication et de régulation cellulaire. Depuis l'infection par les rétro-virus à HIV, en passant par la régulation de la pression sanguine, par la reconnaissance des goûts, odeurs et couleurs, ces récepteurs transmettent à l'intérieur de la cellule le signal porté par une hormone, une molécule ou un photon. Ces quelques exemples ne sont qu'un petit échantillon des nombreux systèmes de communication dans lesquels sont impliqués les GPCRs : on connaît aujourd'hui plusieurs milliers de séquences classées comme membre de cette famille de récepteurs et chaque séquence est sélective d'une ou plusieurs molécules.

Parce que les GPCRs entrent en jeu dans la plupart des systèmes de régulations biologiques, les récepteurs de cette famille suscitent beaucoup d'intérêt tant auprès des expérimentateurs, qu'auprès des théoriciens.

1.2 Quelques points de structure

Au niveau structural, la plupart des GPCRs présentent une extrémité extra-cellulaire N-terminale, puis un faisceau de sept hélices transmembranaires qui délimite des boucles intra et extra-cellulaires, et enfin une extrémité C-terminale intra-cellulaire [WLMB89] [DBB+87]. Nous donnons, pour support visuel, une visualisation très schématique d'un GPCR sur la figure 1.1.

Quand on rapproche cette première ressemblance structurelle des analyses phylogénétiques [DT93] des séquences des GPCRs (qui s'accordent au moins sur le point que les séquences appartiennent à des organismes qui ont évolué à partir d'un ancêtre commun) on peut suspecter une homologie structurale entre les membres de la famille des GPCRs.

Bien entendu, la seule évocation de la possibilité d'une homologie structurale suffit à renforcer encore l'intérêt porté aux GPCRs [Kob92]. En effet, si des progrès significatifs étaient réalisés sur quelques récepteurs de la famille on s'attendrait à ce que les connaissances acquises pendant l'étude de ces récepteurs modèles soient transférables, avec plus ou moins d'adaptations, aux autres récepteurs de la famille.

4. G-Protein Coupled Receptors, abrégé par GPCR en anglais, notation que nous utiliserons dans ce document.

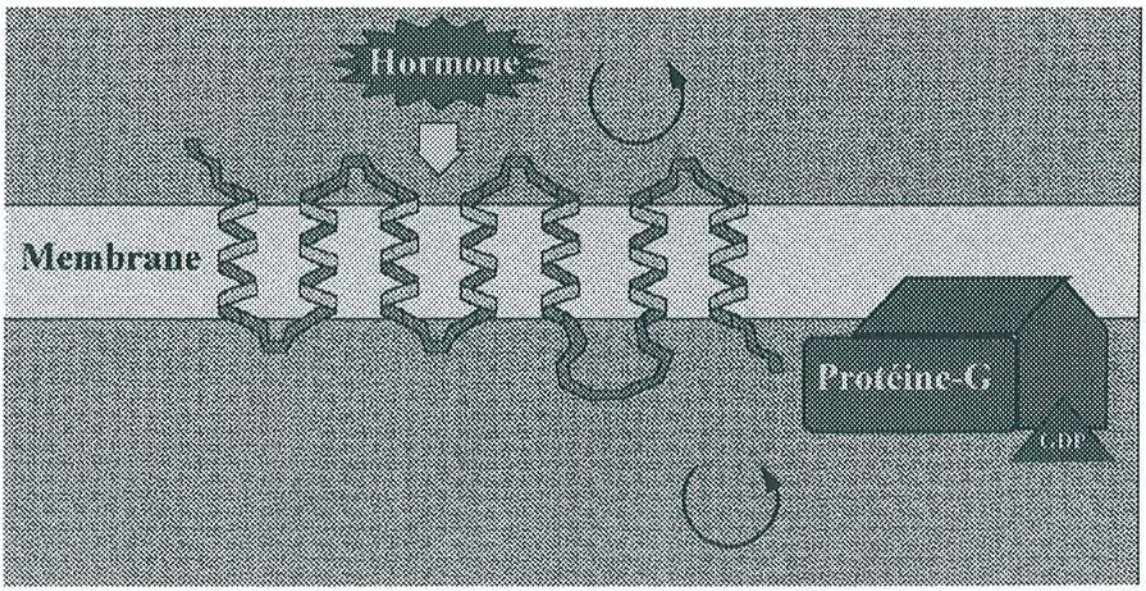


FIG. 1.1 – *Visualisation schématique d'un GPCR enchâssé dans la membrane cellulaire*
La membrane cellulaire délimite l'intérieur de la cellule (en bas) de l'extérieur (en haut). À noter : les boucles intra et extra-cellulaires, le faisceau des hélices à l'intérieur de la membrane. Cette représentation déroule une surface qui passerait par les axes idéalisés des hélices trans-membranaires. Voir la figure 2.4 pour une représentation tri-dimensionnelle des hélices (notées par des cylindres gris).

L'étude d'un GPCR passe par la collecte de données expérimentales dont les types les plus fréquents sont les suivants.

- Séquences protéiques des récepteurs dérivées des séquençages des gènes correspondants.
- Résultats des expériences de *binding*[5] par les agonistes ou les antagonistes, ou mesure de la transduction du signal dans la cellule, après activation par un agoniste.
- Résultats des expériences de mutagenèses dirigées.
- Niveau d'expression du gène dans les membranes.

Puisque des structures moléculaires expérimentales à haute résolution ne sont pas disponibles, la construction de modèles moléculaires de GPCRs est une aide importante à l'avancée des connaissances sur ces systèmes.

D'autres types d'approches existent, comme par exemple l'analyse des séquences apparentées et de leurs relations (analyse de type Correlated Mutation Analysis [Wel90] ou autres [Rei95]). Ces approches tentent de déterminer des résidus importants pour l'interaction avec le récepteur par la seule analyse des séquences, sans hypothèse forte sur le mécanisme de l'interaction ligands/récepteur. Les approches QSAR[6] partent de la structure de plusieurs ligands connus, calculent les valeurs de descripteurs pour plusieurs propriétés, pour chaque ligand, puis tentent de prédire les meilleurs ligands, sur la base des modèles empiriques (de forme mathématique) obtenus par l'analyse.

Malgré leurs qualités, ces dernières approches sont moins satisfaisantes qu'un modèle moléculaire parce qu'elle ne permettent pas de prédictions fines de propriétés, il s'en suit que la modélisation des GPCRs par homologie est la méthode la plus couramment utilisée pour tenter de comprendre ces systèmes.

Toutes ces raisons nous ont amené à développer un ensemble d'outils pour l'aide à la modélisation des GPCRs. Ces outils, dont l'élément principal est le programme Viseur, sont décrits dans la suite de cette partie.

5. Le terme français, liaison spécifique, est si peu utilisé qu'il est difficile à comprendre.
6. Quantitative Structure Activity Relationship

2

Le projet Viseur

2.1 Qu'est-ce qu'un modèle?

Puisqu'il est ici question d'un programme d'aide à la modélisation, il est sans doute à propos de décrire ce que nous entendons par le mot modèle, dans les différents contextes où nous l'emploierons dans cette partie. Nous serons amenés à utiliser plusieurs sens du mot modèle tout au long de ce manuscrit. Nous distinguerons ces différents sens par l'usage d'expressions composées (par exemple "modèle moléculaire", "modèle de données", etc.), à chaque fois que ce sera possible. Dans cette partie, nous comprenons un modèle comme un outil — qui représente une réalité inacessible —, dont l'utilisation permet de faire des prédictions sur la réalité.

Les deux sous-sections suivantes présentent les modèles moléculaires et biologiques et des éléments pour expliquer pourquoi un modèle sera préféré à un autre en fonction de la tâche à entreprendre.

2.1.1 Les modèles de la modélisation moléculaire

Dans le contexte de la modélisation moléculaire, un modèle désigne souvent un modèle moléculaire, c'est-à-dire la description de la conformation d'une molécule dans le cadre du modèle atomiste. Pour parler en termes d'information un modèle moléculaire est la donnée des positions des atomes dans l'espace, de leur type, des liaisons entre ces atomes et souvent des charges atomiques, à un instant donné. Mais c'est aussi l'ensemble des connaissances que recouvre la théorie atomique, qui fixent des contraintes sur les données. Pour prendre un exemple simpliste, deux atomes ne peuvent pas occuper le même espace au même moment, donc leurs positions doivent être différentes (d'autres types de connaissances permettent de préciser la probabilité de rencontrer deux atomes de certains types à des distances non nulles.[7], etc.)

Le modèle moléculaire est aussi une représentation en trois dimensions dans laquelle on dispose des notions de distance et d'orientation entre des parties d'une même molécule ou entre des molécules différentes.

Étant donné qu'une partie de l'enseignement dispensé en Chimie essaie d'apprendre aux étudiants à jauger la force de l'interaction de deux groupements d'atomes en fonction de leur distance et de leur orientation,[8] on comprend que le modèle moléculaire soit la représentation que le chimiste préfère pour "sentir" la force de l'interaction entre deux molécules.

2.1.2 Les modèles du biologiste moléculaire

Ce sont des représentations bien adaptées aux connaissances biologiques. Il est question d'interaction entre macro-molécules à l'intérieur d'une cellule, l'échelle n'est plus la même et on ne s'étonnera

7. Les chimistes ou les physiciens préféreront penser en terme d'énergies d'interactions non liantes entre paires d'atomes...
8. C'est certainement une part du "sens chimique" que les enseignants aimeraient voir leurs étudiants acquérir.

pas que le biologiste pense ces interactions en terme de structures secondaires[9] et de position dans la cellule. Lorsque le biologiste moléculaire s'intéresse à une protéine membranaire, il veut par exemple "voir" du premier coup d'œil dans quelle partie de la cellule se trouve un résidu donné : à l'intérieur de la cellule, à l'extérieur, dans la membrane, etc.. Cette demande est légitime parce que certains résidus peuvent être impliqués dans des interactions avec d'autres composantes de la cellule dont la nature différera radicalement suivant la localisation du résidu dans l'un ou l'autre des environnements.

Ce type de propriété n'est pas d'un accès direct dans un modèle moléculaire, comme l'illustrent les figures 2.1 et 2.2, puisque en fonction de l'angle de vue certains résidus occuperont presque la même position alors qu'ils appartiennent à trois milieux différents. Si l'on introduit le sens de la profondeur à l'aide de la vision stéréographique, il reste difficile d'associer distance entre résidus et position dans la cellule.

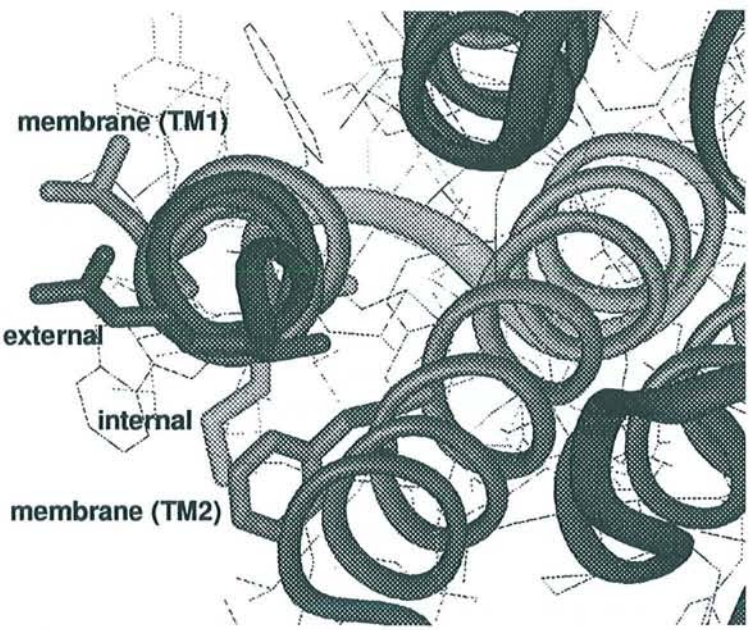


FIG. 2.1 – *Mauvaise adaptation des modèles moléculaires aux attentes du biologiste (i)*
Les modèles moléculaires ne permettent pas d'accéder directement au type d'information qui intéresse le biologiste moléculaire : des résidus qui appartiennent à des environnements très différents occupent des positions voisines dans ce type de représentation. Par exemple, les re'sidus note's `external` (milieu extra-cellulaire) et `internal` (milieu intra-cellulaire)

2.2 Le programme Viseur

Nous avons conçu et développé le programme Viseur, qui est un programme d'aide à la modélisation des GPCRs. Ce programme est décrit en détail dans la première publication, inclue en annexe A. Cette publication insiste sur les aspects techniques et applicatifs du programme que nous ne reprendrons pas ici. Notre objectif est maintenant de donner un aperçu au lecteur des relations qui lient données, utilisateur et outil (programme informatique dans le cadre de ce document). Nous nous plaçons donc dans la position du concepteur d'un programme d'aide à l'analyse de données et décrivons les ingrédients présents dans le programme Viseur, qui selon nous, facilitent la tâche d'analyse. Les descriptions de ces caractéristiques seront utiles à la réflexion dans la troisième partie de ce manuscrit.

9. Groupement d'atomes correspondant à plusieurs amino-acides dans le cas d'une protéine (plus de 100 atomes environ).

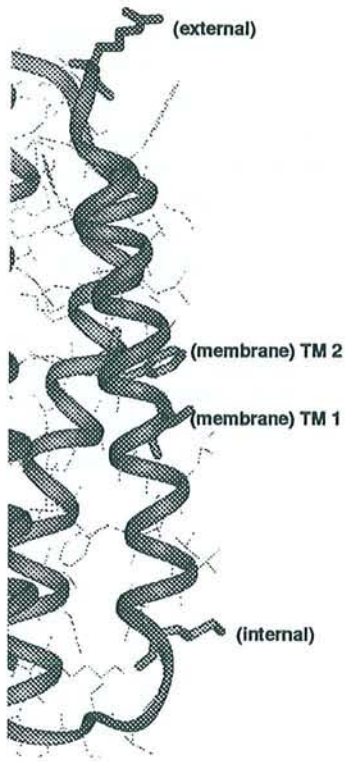


FIG. 2.2 – *Mauvaise adaptation des modèles moléculaires aux attentes du biologiste (ii)*
Le modèle moléculaire de la figure 2.1, vu sous une orientation différente, code l'appartenance au milieu cellulaire par la position mais pas l'appartenance à une sous-unité: Les résidus étiquetés `(membrane) TM1` et `(membrane) TM2` occupent des positions voisines bien qu'ils appartienent à deux domaines transmembranaires distincts.

2.2.1 Comment intégrer des informations hétérogènes?

Le manque de structure de GPCRs à haute résolution conduit à apporter une grande importance à toutes les informations indirectes disponibles sur le système d'intérêt. La tâche est assez difficile parce que ces données sont de natures très diverses et disponibles en quantité parfois importante (séquences des récepteurs, mutants, affinités pour les ligands).

Comment présenter ces données pour faciliter la tâche de modélisation? Une première solution consiste à proposer des visualisations adaptées à chaque type particulier de donnée. C'est la visualisation directe de la donnée, illustrée pour la donnée d'une séquence et de ses limites transmembranaires par la figure 2.3. Cette représentation permet de se familiariser avec les données et leurs particularités mais ne facilite pas leur comparaison, la découverte de relations entre données semblables ou non, etc. Or ces tâches sont centrales à l'activité de modélisation. Le modèle cherché ne doit-il pas être à la fois une synthèse de toutes les informations disponibles, et une explication de leurs relations [10] ?

Dans le cadre d'un programme d'aide à la modélisation, il nous a semblé important d'offrir, en plus des représentations directes, des outils de visualisation qui encapsulent des connaissances du domaine.

Ce point mérite quelques explications. Pour prendre un exemple, comparer des séquences biologiques entre elles, ne peut pas être fait de la même façon que la comparaison des séquences de caractères de deux mots d'un langage occidental (comparaison lexicographique). On peut comprendre qu'une personne qui n'a aucune connaissance des mécanismes de l'évolution des séquences biologiques applique la méthode lexicographique pour comparer des séquences biologiques (dans le but de les classer par exemple). Mais à partir du moment où cette personne apprend l'existence de mé-

10. L'explication étant valable dans la limite de la validité des hypothèses du modèle.

canismes d'insertion et de délétion de bases dans l'ADN, elle est obligée de corriger sa procédure de comparaison pour en tenir compte. Elle devra aussi faire évoluer ses outils de présentation pour qu'ils mettent en évidence la similarité de deux séquences proches (introduction de *gaps*?).

Ceci pour rappeler que la comparaison de deux données dépend très fortement de la connaissance acquise sur des données du même type préalablement à la comparaison. Si notre connaissance des relations entre séquences biologiques était modifiée, les outils que nous utilisons actuellement pour comparer ces séquences devraient évoluer. Par exemple, si les événements de *cross-over* étaient plus fréquents que nous ne le pensons actuellement [Sne93], il nous faudrait envisager des outils d'alignements qui prennent en compte ces nouvelles connaissances, et encore d'autres outils pour détecter ces situations.

Pour résumer ce paragraphe, certains outils permettent de visualiser des données, en soi, alors que d'autres offrent la visualisation d'une donnée dans le cadre de certaines connaissances du domaine. Ces connaissances sont incluses dans l'outil au moment de sa conception. Le programme que nous avons développé contient ces deux types de représentations.

2.2.2 Visualisations adaptées

Visualisation directe. Les modes de visualisation sont adaptés aux données — mais ne mettent pas en valeur leurs relations.

Injection dans une représentation. On représente une donnée dans le cadre de plusieurs modèles, chacun adapté à une façon de comprendre certains aspects du système.

Nous avons choisi de proposer ces deux types de stratégies, en insistant sur l'injection dans une représentation puisque nous cherchons à favoriser l'intégration des données et le développement de modèles.

Visualisation directe

Le programme Viseur propose quelques modes de visualisation directe. La figure 2.3 présente une visualisation directe de la séquence d'un GPCR et des limites des domaines transmembranaires. Les autres annotations de la séquence restent accessibles à travers une vue texte du fichier de séquence. On est loin d'une situation idéale puisque l'utilisateur n'a aucun contrôle sur le choix des données qui lui sont présentées. Néanmoins, la visualisation disponible dans le programme (en haut sur la figure) présente les relations entre domaine transmembranaire et séquence. L'utilisateur appréciera cette représentation lorsqu'il devra ajuster la position des limites transmembranaires en fonction de la composition locale de la séquence.

Injection dans une représentation

Le programme Viseur propose plusieurs types de représentations, associées chacune à un ensemble de connaissances différent, mais complémentaire des autres. Les représentations — donc les modèles qui permettront d'analyser les données — que nous avons introduites dans le programme sont les suivantes.

Diagramme en serpent (snake-like). Représentation adaptée à la connaissance biologique des GPCRs (connaissance de la topologie de ces protéines : protéines à hélices transmembranaires).

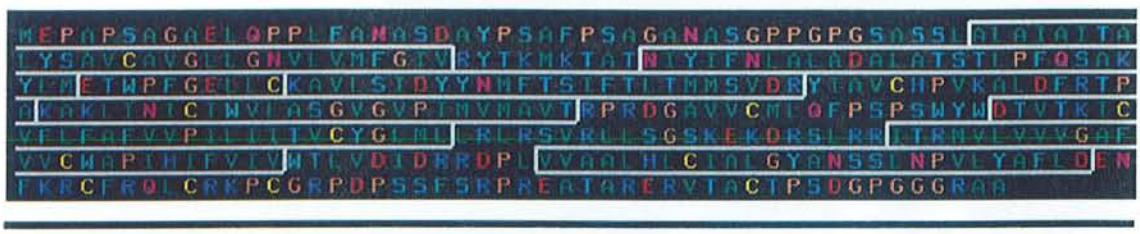
Alignement de séquence. Représentation adaptée à la comparaison de séquences (connaissances des modèles d'évolution des séquences biologiques).

Modèle moléculaire. Représentation adaptée à l'examen des interactions moléculaires (connaissances de la théorie atomique).

Ces représentations offrent d'une part une représentation directe de certaines données (séquences, alignements, structure) mais permettent aussi l'examen des données de mutagenèse dirigée, de *binding* et de structure des ligands, au moyen de l'injection.

A titre d'exemple, en coloriant de la même couleur les résidus d'un GPCR pour lesquels des expériences de mutagenèse montrent un effet sur le *binding* d'un ligand, on obtient une représentation moléculaire de la zone d'interaction ligand/récepteur. L'utilisateur peut combiner les informations géométriques présentes dans cette représentation (distance entre les résidus impliqués dans le *binding*) avec sa connaissance de la structure du ligand et des interactions moléculaires (en général), pour postuler les interactions de *ce ligand* avec *ce récepteur*.

En théorie, il y a de multiples façons de coder une information à l'intérieur d'une représentation (à l'aide de modalités comme les codes couleurs, des formes différentes, des liens entre des objets, etc...). En pratique, pour que la méthode soit utilisée à son plein potentiel, il faut que l'injection d'une propriété dans une représentation soit aussi simple que possible. C'est-à-dire, essentiellement que *(i)* l'outil soit capable de changer les modalités de sa représentation à partir d'instructions, dynamiquement et *(ii)* que l'utilisateur dispose d'un outil qui convertit ses requêtes (exprimées dans un langage proche de ses préoccupations) vers les instructions nécessaires au changement de la représentation.



```
FT   TRANSMEM     46     75        1 (POTENTIAL).
FT   TRANSMEM     85    102        2 (POTENTIAL).
FT   TRANSMEM    125    144        3 (POTENTIAL).
FT   TRANSMEM    175    190        4 (POTENTIAL).
FT   TRANSMEM    216    238        5 (POTENTIAL).
FT   TRANSMEM    262    284        6 (POTENTIAL).
FT   TRANSMEM    294    310        7 (POTENTIAL).

     MEPAPSAGAE LQPPLFANAS DAYPSAFPSA GANASGPPGP GSASSLALAI AITALYSAVC
     AVGLLGNVLV MFGIVRYTKM KTATNIYIFN LALADALATS TLPFQSAKYL METWPFGELL
     CKAVLSIDYY NMFTSIFTLT MMSVDRYIAV CHPVKALDFR TPAKAKLINI CIWVLASGVG
     VPIMVMAVTR PRDGAVVCML QFPSPSWYWD TVTKICVFLF AFVVPILIIT VCYGLMLLRL
     RSVRLLSGSK EKDRSLRRIT RMVLVVVGAF VVCWAPIHIF VIVWTLVDID RRDPLVVAAL
     HLCIALGYAN SSLNPVLYAF LDENFKRCFR QLCRKPCGRP DPSSFSRPRE ATARERVTAC
     TPSDGPGGGR AA
```

FIG. 2.3 – *Visualisation directe de la séquence et des limites transmembranaires*
Visualisation directe de la séquence (en haut et en bas) et des limites du domaine transmembranaire (à l'intérieur des rectangles blanc, en haut, sous forme de texte, en bas).

Représentations dynamiques

Le programme Viseur propose des visualisations de données avec lesquelles l'utilisateur peut interagir. L'interaction est basée sur l'association représentation/modèle. Nous offrons aux utilisateurs la possibilité de changer les paramètres d'une représentation associés à certains degrés de liberté du modèle.

- Les diagrammes en serpent réagissent à la modification des limites transmembranaires d'une séquence : l'utilisateur a la possibilité de visualiser tous les modèles schématiques que la variation de ces paramètres permet d'atteindre.

- L'éditeur d'alignement permet de décaler les séquences pour favoriser leur comparaison dans le cadre des modèles de l'évolution (insertion/délétion de *gaps*, visualisation de familles de résidus par l'usage de codes couleurs appropriés).

- Le modèle moléculaire permet de translater et tourner les hélices autour de leurs axes, de façon complètement interactive (illustration figure 2.4). Ceci en accord avec la précision des modèles de GPCRs, pour adapter le modèle de travail aux données de mutagenèse.

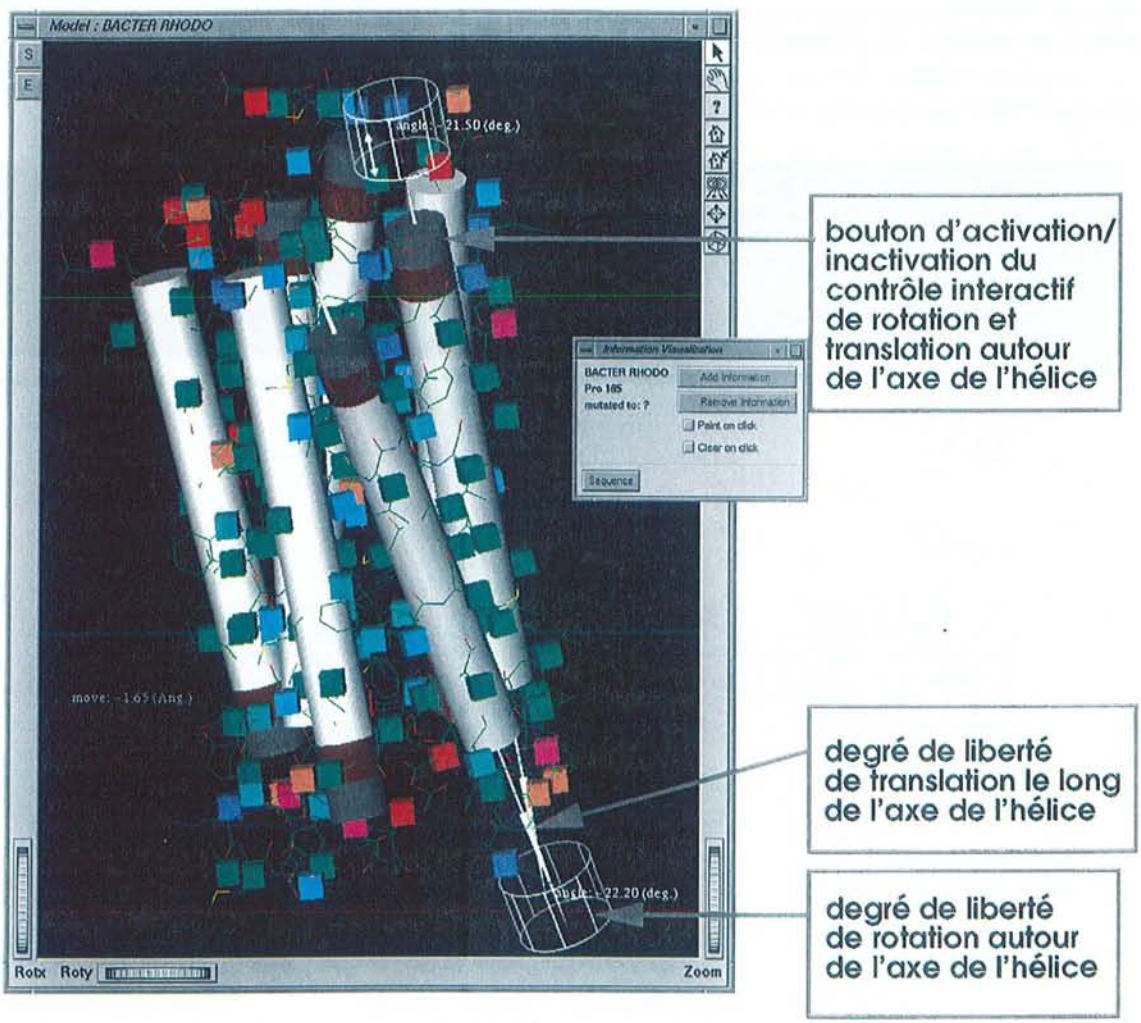


FIG. 2.4 – *Représentation interactive d'un modèle moléculaire*
Cette figure présente des développements récents de l'éditeur de modèle moléculaire qui ne sont pas décrits dans la publication en annexe A. Ces développements permettent de parcourir l'espace des conformations d'un récepteur, de façon schématique, en modifiant la position des hélices les unes par rapport aux autres. Les degrés de liberté implantés dans le programme pour permettre cette exploration comprennent *(i)* les degrés de liberté de rotation autour de l'axe de l'hélice (axe du cylindre de couleur grise), *(ii)* les degrés de translation autour de ces axes et *(iii)* un outil permettant de faire tourner une hélice autour d'un point de l'espace (non représenté sur cette figure).

2.2.3 Utilisation de sources de données distantes

Avec la multiplication des sources de données bioinformatiques, il devient très difficile de maintenir des copies locales, *à jour*, de toutes les données utiles à une tâche donnée. D'autres facteurs non techniques entrent en jeu, qui vont dans le même sens : les institutions qui ont investi leurs efforts dans la nécessaire création et le maintien d'une collection de données peuvent préférer interdire la duplication de la totalité des données qu'elles proposent.

Nous avons géré ces problèmes en proposant dans le programme Viseur un accès aux données de mutagenèse dirigée du serveur TinyGRAP [KDE96], à travers internet, de façon relativement transparente pour l'utilisateur.[11]

Il nous semble important de remarquer que la possibilité d'utiliser des ressources d'informations distantes enrichit les outils bioinformatiques qui la propose, et donc est de plus en plus souhaitable, du point de vue de l'utilisateur. Quand on rapproche ce besoin de la difficulté de programmer des accès à des données ou services distants (c'est l'informatique client-serveur, absente de la formation dispensée aux biologistes ou aux chimistes) on arrive à la conclusion qu'il manque un maillon à la chaîne d'information (une chaîne dont une extrémité est la base de données et l'autre extrémité est l'utilisateur de l'information, donc le spécialiste du domaine).

2.2.4 Bilan

Point de vue applicatif

Le développement du programme Viseur constitue la partie la plus importante du projet Viseur. Nous avons réalisé un outil d'aide à la modélisation des Récepteurs Couplés aux Protéines G qui permet à de nombreux utilisateurs de parcourir les données disponibles sur leur système d'intérêt et de les visualiser, pour mieux les comprendre, en accord avec les connaissances biologiques actuelles.

Pour résumer rapidement l'aspect applicatif de ce programme, il faut encore écrire qu'il

- offre un environnement d'aide à la découverte de données adapté aux spécificités des GPCRs (recherche et enseignement),
- automatise la représentation de diagrammes en serpent pour les protéines à hélices transmembranaires,
- fournit des représentations adaptées aux réseaux — internet ou intranet — pour la diffusion et l'échange de connaissances.

Examinons maintenant les apports méthodologiques de ce développement.

Point de vue méthodologique

Le programme Viseur illustre comment intégrer des informations hétérogènes dans le cadre de modèles modérément prédéterminés. C'est parce que l'on sait que les données appartiennent au système des Récepteurs Couplés aux Protéines G que leur analyse à l'aide des outils fournis par le programme a un sens.

Beaucoup d'utilisateurs du programme sont maintenant convaincus que le développement d'outils spécialisés peut être préférable à l'utilisation de programmes dits généralistes. Nous préférons penser que les outils informatiques doivent intégrer des connaissances du domaine d'application (de façon à rendre simple les manipulations qui ont un sens dans le cadre des modèles, mais pas les autres[12]). Une conséquence importante est que les outils informatiques devraient évoluer en même temps que le domaine d'application, et même si possible, en fonction des besoins ponctuels des utilisateurs du domaine.

Cette considération peut sembler utopique quand on se souvient que les personnes capables de développer des programmes pour la biologie ou la chimie sont en nombre beaucoup plus faible que

11. L'utilisateur sélectionne les séquences pour lesquelles il veut obtenir les mutants décrits dans TinyGRAP

12. Les outils informatiques sont avant tout des outils. Considérez comment la clé à molette facilite le vissage et le dévissage d'écrous, alors qu'elle n'est pas adaptée à la manipulation de vis.

les utilisateurs de ces programmes. Nous verrons en deuxième et en troisième partie comment il est possible de limiter ce problème.

Avant cela, nous vous invitons à découvrir quelques avantages et problèmes liés à la mise à disponibilité d'outils et données via le réseau mondial internet. La présentation d'information sur le WWW, parce qu'elle peut potentiellement atteindre un large public à peu de frais, occupe une place importante des développements bioinformatiques actuels.

2.3 Les services WWW Viseur

2.3.1 Objectifs

L'objectif principal d'un serveur WWW est de fournir un service à des utilisateurs distants. Dans la grande majorité des cas, ce service peut être :

- la fourniture d'informations ou
- la possibilité d'utiliser un programme, ou
- toute combinaison des deux premiers services.

Dans le cas du projet Viseur, il est apparu assez vite que le fait que le programme ne soit disponible que sur une seule architecture était un frein à son utilisation par un public plus large. Un autre frein est commun à beaucoup de programmes du type de Viseur : certains utilisateurs potentiels ne savent pas installer un programme de ce type sur leur système, ou n'ont pas le temps de l'installer au moment où ils auraient besoin *d'utiliser* ce programme, où n'ont pas envie d'investir du temps dans l'installation des mises à jour du logiciel, etc..

Tous ces points ont leur importance dans la détermination du niveau d'utilisation d'un programme, donc dans son utilité globale. La solution qui consiste à offrir les mêmes services qu'un programme à travers internet (ou un intranet), vise à éliminer ces problèmes en les déplaçant du côté du serveur WWW, donc à un endroit où les compétences nécessaires à la résolution des problèmes d'installation ou de mises à jour sont disponibles.

Du point de vue du développeur, offrir un accès distant à un programme, permet de détecter et donc de corriger plus facilement les défauts du logiciel. En effet, reproduire l'erreur signalée par un utilisateur est souvent la partie la plus difficile du processus de correction de défaut.

Dans le cadre de ces objectifs, nous avons donc réalisé deux services WWW :

- visualisation *snake-like* d'une séquence Swiss-Prot fournie par l'utilisateur, limite transmembranaires paramétrables;
- conversion d'un alignement MSF en alignement HTML [CM98] (voire publication B, [CJR+98] et [CBM97]).

2.3.2 Méthodes

La figure 2.5 illustre l'architecture d'un service WWW construit au dessus d'un programme. Nous avons suivi cette architecture dans le cas des deux services que nous mentionnons, en utilisant pour l'implantation des scripts CGI [CGI98] écrits dans le langage Perl [Chr98]. Cette implantation a été choisie parce qu'elle était la mieux documentée, sur le WWW, à l'époque de cette réalisation.

Puisque le programme Viseur réalise des traductions, adaptées au réseau, des représentations qu'il propose, l'adaptation des données au WWW est immédiate. Les méthodes de représentations HTML d'un alignement multiples sont décrites dans la deuxième publication (en annexe) et sous la forme d'une présentation électronique [CM97].

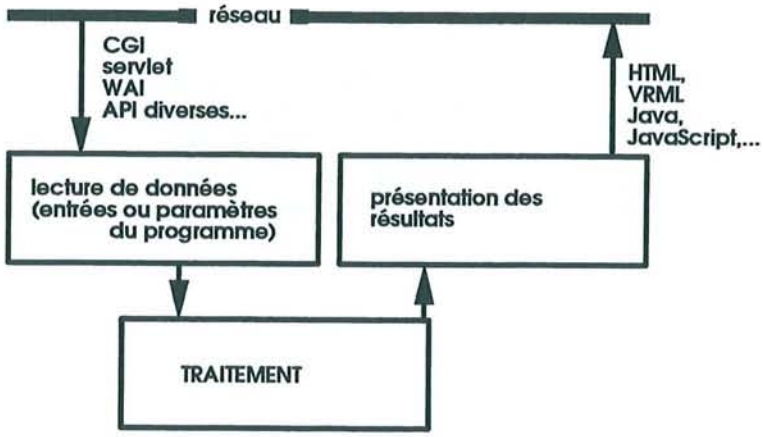


FIG. 2.5 – *Architecture schématique d'un service WWW dynamique*
Ce type de service WWW propose les services d'un programme : les requêtes de l'utilisateur sont traduites sous une forme utilisable par le programme, le traitement est exécuté, le résultat est traduit sous une forme compatible avec les protocoles supportés par le réseau, le résultat est envoyé à l'utilisateur.

2.3.3 Bilan

Point de vue applicatif

Les services WWW que nous avons mis en place dans le cadre du projet Viseur offrent deux ressources en accès libre aux chercheurs qui voudraient utiliser le programme Viseur depuis n'importe quel périphérique d'accès au réseau internet. Les visualisations HTML et PostScript obtenues peuvent être utilisées pour présenter des résultats de recherche, ou discuter pendant les étapes intermédiaires d'un travail collaboratif, sur papier ou à travers le réseau.

Point de vue méthodologique

La réalisation du service de conversion d'alignement multiple (de protéines) en HTML a été rendu possible par la mise au point de méthodes de représentation des alignements multiples. Ces méthodes permettent de réaliser des présentations en couleur, avec liens hypertextes, accessibles à travers les réseaux, qui sont potentiellement très utiles à la présentation de pages d'informations contenant des fragments d'alignement multiple avec d'autres données.

Ces méthodes sont de mise en œuvre assez simple par la programmation mais souffrent du problème qu'il faut déterminer quel type de représentation est adapté au navigateur WWW de l'utilisateur afin de lui présenter une page d'aspect attendu. [13] C'est exactement le même type de problème que celui que rencontraient les développeurs d'applications plus classiques (plusieurs plateformes de développement, plusieurs bibliothèques de calcul ou d'affichage, plusieurs taille de périphériques de présentation, etc...), et qui semblait devoir être éliminé par le développement de HTML et du WWW.

Il sera souhaitable de se souvenir de la récurrence de ces problèmes d'adaptation à un environnement changeant (pendant la courte histoire de l'informatique) lorsque nous aborderons la troisième partie de ce document.

2.4 Viseur et GPCRDB

13. C'est le meilleur moyen d'éviter de recevoir des courriers électroniques clairs, pertinents et précis, que l'on peut résumer par "It does not work (for me)!"

2.4.1 Objectifs

GPCRDB [HWB+98] est une base de donnée dédiée aux Récepteurs Couplés aux Protéines G. Cette base est le résultat d'une collaboration internationale dont l'objectif est de proposer à tous les chercheurs intéressés par les GPCRs une source de données dédiée, fiable, à jour, pratique.

Nous avons rejoint les groupes de recherche associés au développement de GPCRDB pour enrichir cette base des représentations schématiques que le programme Viseur permet de réaliser automatiquement.

2.4.2 Méthodes

Types de diagrammes

Il existe potentiellement plusieurs façons d'utiliser le programme Viseur pour construire des diagrammes de GPCRs. Nous avons retenu les représentations suivantes.

- Les résidus de la séquence sont coloriés en fonction de leur nature, ou en blanc lorsqu'une mutation est disponible dans TinyGRAP. Les résidus mutés sont liés à l'entrée correspondante dans TinyGRAP. Illustration figure 2.6.
- La séquence de consensus d'une famille est uniformément grise, sauf pour les résidus appartenant à un réseau calculé par CMA, qui partagent un même code couleur. Illustration figure 2.7.
- La séquence de consensus d'une famille est uniformément grise sauf pour les positions de l'alignement de la famille où une mutation au moins est documentée dans TinyGRAP (la couleur est blanche dans ce cas). A la date de rédaction de ce document, cette représentation n'est pas encore intégrée à la version de GPCRDB disponible au public.

Mise en place dans GPCRDB

Tous les diagrammes en serpent de GPCRDB sont construits une fois pour toutes, à l'avance, et stockés comme des pages WWW statiques. En ce sens, l'approche est différente des services dynamiques décrits dans la section précédente (du point de vue utilisateur, les données sont mises à jour à chaque nouvelle diffusion de GPCRDB, au contraire d'un site dynamique où les données sont constamment à jour).

La méthode de construction des diagrammes Viseur pour les pages de GPCRDB que nous avons mis en place sur le site de l'EMBL (Heidelberg) peut se résumer comme suit. Pour préparer la mise à jour des données, l'utilisateur responsable de la base de donnée active un script qui automatise l'ensemble des tâches à effectuer pour mettre à jour les pages correspondant à un type de diagramme (parmi les trois que nous avons décrits).

Prenons l'exemple des diagrammes qui relient une séquence de GPCRDB aux données de mutagenèse présentes dans TinyGRAP. Pour chaque séquence de GPCRDB, le script

1. détermine les résidus documentés dans TinyGRAP,
2. construit un fichier de commande pour le programme Viseur (qui colorie chaque résidu muté en blanc et lui associe une URL[14]),
3. exécute Viseur avec le fichier de commande (création de la page).

La figure 2.8 résume les données et ressources utilisés pendant l'exécution du script.

2.4.3 Bilan

Les diagrammes en serpent du programme Viseur enrichissent considérablement l'ergonomie d'accès au contenu de GPCRDB. D'autres utilisations de Viseur pour présenter les données de GPCRDB sont à l'étude, comme la construction de représentation VRML [SGI97] des modèles moléculaires de GPCRDB, liés vers les informations de TinyGRAP ou coloriés pour présenter les réseaux de résidus obtenus par CMA.

14. Une URL, pour *Uniform Ressource Locator* est l'adresse d'une page WWW sur internet.

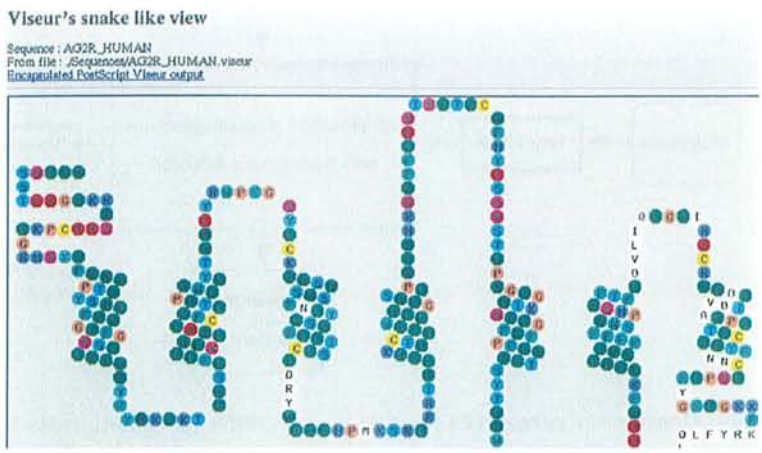


FIG. 2.6 – *Visualisation du récepteur* AG2R_HUMAN *et de ses mutants*
La documentation disponible dans TinyGRAP est indiquée par la couleur blanche d'un résidu. L'utilisateur peut clicker sur les résidus blancs, ce qui dirige son navigateur vers la page d'information de TinyGRAP à propos de ce résidu.

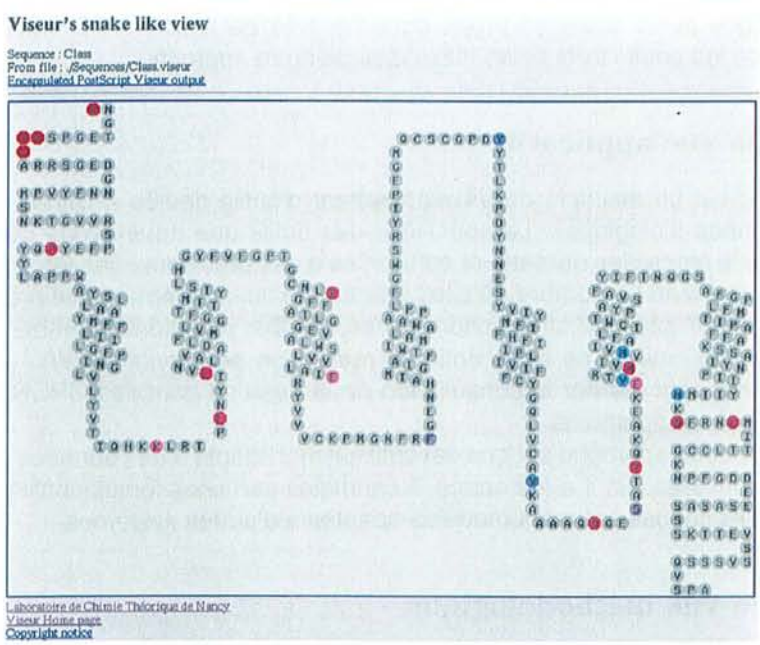


FIG. 2.7 – *Visualisation des réseaux de résidus calculés par Analyse de Mutation Corrélées (CMA) en considérant les séquences de la famille des (Rhod)opsines*

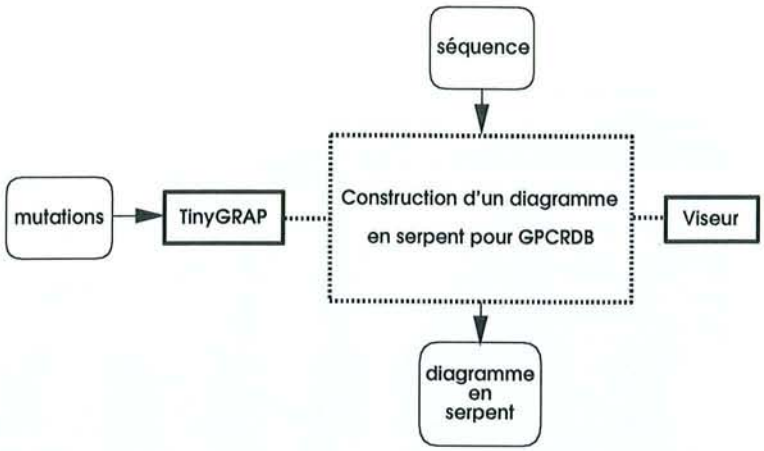


FIG. 2.8 – *Données et ressources pour la construction de diagrammes en serpent*
Les données et ressources mises en œuvre par le script de construction de diagramme en serpent pour GPCRDB. En pointillé : le traitement, rectangles : ressources, boîtes arrondies : données.

sont à l'étude, comme la construction de représentation VRML [SGI97] des modèles moléculaires de GPCRDB, liés vers les informations de TinyGRAP ou coloriés pour présenter les réseaux de résidus obtenus par CMA.

2.5 Le point sur ce projet

Il n'est pas ici question de répéter, par juxtaposition, les bilans intermédiaires des sections précédentes. Nous voulons plutôt situer ce projet dans l'activité de recherche bioinformatique actuelle en mettant en évidence les points forts et les faiblesses de notre approche.

2.5.1 Point de vue applicatif

Le projet Viseur est un exemple de développement d'outils dédiés à l'étude d'un sous-ensemble important des données biologiques. La spécificité des outils que nous avons développés permet de répondre aux besoins précis des utilisateurs confrontés à des problèmes que les outils plus généralistes ne couvrent pas facilement. Le nombre de sites utilisateurs qui ont demandé le programme Viseur pour une utilisation locale : 57 sites (47 sites académiques, 10 sites industriels) semble indiquer qu'il existait un réel besoin pour un outil de ce type. Enfin, la réalisation de services WWW et surtout l'utilisation du programme Viseur pour faciliter la consultation de la base de données GPCRDB confère une plus grande utilité à ces développements.

Le point faible de cette approche est que cet outil est mal adapté à des données qui ne se conforment pas aux modèles pour lesquels il a été conçu. Néanmoins certaines fonctionnalités seraient utiles, associés à des modules conçus pour des modèles adaptés à d'autres systèmes.

2.5.2 Point de vue méthodologique

Le projet Viseur nous a donné l'occasion de tester plusieurs méthodes de tracé de diagrammes, d'aide à l'analyse de donnée et à la modélisation.

- Tracé automatique de diagrammes en serpent pour protéines à hélices transmembranaires.
- Représentations adaptées au réseau, en couleur, avec liens, d'alignement multiples de séquence.
- Documentation de modèles moléculaires par liens hyper-texte sur certains résidus.

notre expérience du développement de ce projet, il semble très difficile de continuer à développer des outils de la même façon que nous le faisions jusqu'à présent.

Plusieurs raisons à cela.

1. La production de données biologiques issues des programmes de séquençages de génomes entiers (changement méthodologique par rapport au séquençage ponctuel de gènes) nous oblige à chercher les méthodes qui permettront, à ressources humaines et financières quasi-constantes, de continuer à produire de la connaissance utile.
2. Les moyens de communication évoluent vers l'utilisation intensive des réseaux. Doit-on continuer à écrire des programmes qu'il faudra ensuite interfacer (nouveau travail) avec les réseaux ?
3. GPCRDB illustre le développement d'une ressource bioinformatique par des groupes de recherche géographiquement éloignés. C'est une contrainte commune à beaucoup d'initiatives bioinformatiques. Les méthodes de développement actuelles sont mal adaptées à ces conditions.
4. Les méthodes de développement utilisées dans ce projet (et dans les projets bioinformatiques en général) facilitent peu la réutilisation des efforts de développement précédents.

De toutes ces raisons, la dernière est sans doute la moins évidente, *a priori*. Qu'est-ce que la réutilisation ? Pourquoi est-il souhaitable de la favoriser ? La deuxème partie de ce document présente quelques éléments de réponse, dans le cadre du développement d'un outil de recherche conformationnelle.

A

Publication Viseur

F. Campagne, R. Jestin, J.L Reversat, J.M. Bernassau, and B. Maigret. Visualisation and Integration of G Protein-Coupled Receptor Related Information Help the Modelling: Description and Applications of the Viseur program. *J. Of Computer Aided Molecular Design*, 1998. in press.
Cette publication est disponible sous forme électronique sous `http://www.lctn.u-nancy.fr/Etudiants/FC/thes`

Visualisation and Integration of G Protein-Coupled Receptor Related Information Help the Modelling: Description and Applications of the Viseur program

Campagne† F., Jestin‡ R. Reversat‡ J.L., Bernassau‡ J.-M. Maigret†⋆ B.

⋆ to whom correspondence should be addressed.

† Laboratoire de Chimie théorique
Centre Charles Hermite
Unité Mixte de Recherche 7565
CNRS - Université Henri
Poincaré Nancy I
BP 239
54506 Vandoeuvre-les-Nancy
France

‡ Sanofi Recherche
Service de Chimie Structurale
371, Rue du Pr. Blayac
34184 Montpellier cedex
France

keywords: GPCR INTEGRAL MEMBRANE PROTEINS
COMPUTER PROGRAM TOOL MUTAGENESIS
ALIGNMENT HTML VRML

September 16, 1998

Abstract

G Protein-Coupled Receptors (GPCRs) constitute a superfamily of receptors that forms an important therapeutic target. The number of known GPCR sequences and related information increases rapidly. For these reasons, we are developing the Viseur program to integrate the available information related to GPCRs. The Viseur program allows one to interactively visualise and/or modify the sequences, transmembrane areas, alignments, models and results of mutagenesis experiments in an integrated environment. This integration increases the ease of modelling GPCRs: visualisation and manipulation improvements enable easier databank interrogation and interpretation. Unique program features include: (i) automatic construction of "Snake-like" diagrams or hyperlinked GPCR molecular models to HTML or VRML and (ii) automatic access to a mutagenesis data server through the internet. The novel algorithms or methods involved are presented followed by the overall complementary features of the program. Finally, we present two applications of the program: (i) an automatic construction of GPCR snake-like diagrams for the GPCRDB WWW server, and (ii) a preparation of the modelling of the 5HT receptor subtypes. The interest of the direct access to mutagenesis results through an alignment and a molecular model are discussed. The Viseur program which runs on SGI workstations, is freely available and can be used for preparing the modelling of integral membrane proteins or as an alignment editor tool.

1 Introduction

G Protein-Coupled Receptors (GPCRs) are membrane proteins involved in one of the major cellular signaling system: from HIV infection and blood pressure regulation to taste, smell and colour recognition. These receptors transmit into the cell the signal brought from the extracellular medium by a hormone, molecule, or photon. Since GPCRs are involved in most biological regulation systems, an increasing activity is devoted to this receptor super-family and GPCRs are an interesting target for the discovery of new drugs of pharmaceutical importance. Structurally speaking, all GPCRs share an extracellular N-terminal part, a trans-membrane seven helix bundle and a C-terminal intracellular part. Such properties suggest a structural homology between these receptors that, again, increases their interest.

The most frequent data obtained from experiments on these membrane proteins are

- receptor sequences derived from gene cloning,
- agonist and/or antagonist ligand binding and/or transduction properties obtained from bioassays and
- mutagenesis information obtained from site-directed experiments.

The last two sets of data show how the receptor expression or the binding or activation by a ligand are modified when one changes the receptor sequence. These data are becoming more and more useful as their number increases rapidly. A review of GPCR mutations is given in [1]. Important sources for GPCR mutations are provided in [2] and [3].

Due to the lack of precise tridimensional experimental data for this receptor family (the only GPCR for which preliminary structural information is available is rhodopsin [4]), GPCRs models are usually obtained by homology modelling, using the only available tridimensional structure, of an acceptable resolution, currently available for a protein showing a seven-helix bundle. The structure of bacteriorhodopsin (which is not a GPCR, but could be a far common ancestor of the family members) is known at the atomic level of detail with a 3.5 Å resolution [5] and is usually used as a tridimensional template for modelling. In addition to the tridimensional structure of the template, homology modelling also requires the establishment of a sequence alignment between the sequence template and the protein to be built. Such an alignment between bacteriorhodopsin and any GPCR is not straightforward because the sequence homology between them is poor (less than 20 per cent). This means that usual sequence homology based alignment methods cannot be employed. The alignment methods used are, therefore, very specific to the GPCR family [6]. Once an alignment is obtained between the receptor of interest and the template, a tridimensional structure for the transmembrane part of the studied receptor can be built. Extra- and intra-cellular loops connecting the transmembrane helices can be added to the incomplete structure at this step or at a latter stage in the modelling process by a special procedure, as their treatment cannot rely on alignment techniques. Next, the helices can be rotated or moved along their axes to maximise the agreement between the structure and the experimental results known at that time.

To understand why one can, and should, adapt the model structure in this way, one has to consider the following points.

1. The template resolution is weak.
2. We know that the template is only indicative: bacteriorhodopsin is not a GPCR and its electron density projection map differs significantly from that of rhodopsin, so that the transmembrane helix axes are only roughly indicative of the relative position of these fragments inside the membrane bilayer.

3. The proline residues known to break the helical arrangement of residues are distributed among the transmembrane segments in different distributions for bacteriorhodopsin and GPCRs [7]. According to these elements, some models of rhodopsin were constructed in order to be used as templates for the modelling of GPCRs, according to various experimental data. The latter arguments to helices rotation around their axes apply also to this models.

4. The molecular modelling methods presently applicable to such molecular systems, namely minimisation, molecular dynamics or simulated annealing, can only approach the energy of the isolated receptor without any explicit consideration of the effects of the surrounding environement (i.e. phospholipids, water and ions are not included). However, some experimental studies support the hypothesis that interactions between the helices are more important to the structure of the bundle than interactions between helices and the surrounding environement. This is in agreement with the observed stability of modelled bundles using molecular dynamics in a vacuum in which the helices remain packed during the whole simulation, and with experiments in which truncated receptor moieties fold together. But, even according to this hypothesis, the simulation time for these systems, which consume a reasonable amount of computer time (one or two weeks), is currently limited to only one or two nanoseconds. This precludes the bundle from undergoing important rearrangements during the simulations. The helices will remain more or less in their original relative orientations. In the case of molecular dynamics, the system size limits the simulation to a very small period of simulated time, far away from the time scale necessary to obtain important rearrangements of the receptor. It seems reasonable, therefore, to bias the conformational search according to the current experimental knowledge in order to approach, as closely as possible, the interesting conformational sub-space where the agreement with experiment is satisfactory.

This introduction to GPCR modelling is far from complete, since some attempts are already proposed to obtain tridimensional GPCR models from automatic procedures. Such possibilities to retrieve tridimensional GPCR models are available on the WEB from the EMBL [3] or Swiss Model [8] sites. The associated procedures oblige one to work blind, as no control on the process itself is possible by the end-user. More sophisticated procedures, associating molecular dynamics refinements of the raw tridimensional GPCR structure obtained from homology modelling have been proposed [9]. Most of the above approaches, however, do not take directly and explicitly into account, in the simulation process, the many available experimental data about the receptors and their ligands.

An attempt to introduce such a control from experimental data into the modelling process was introduced recently [10] but, in the proposed algorithm, the user has little possibilities to influence the different steps of the process. Furthermore, we have to mention that the GPCR modelling is not a sequential process. It may happen that looking at the receptor alignment and the mutant information, one might notice inconsistencies that will have to be fixed. This confrontation of information could happen at different stages during the modelling process: (i) during the alignment procedure while looking at a snake-like diagram of the sequences, based on assumptions on the nature of side chains related to their environment and (ii) when building the model by checking its consistency with mutational information.

It appears, therefore, that GPCR modelling is a complex task which implies taking care of a lot of heterogeneous data that are highly inter-related. For this reason, we are developing the Viseur program which is designed to store, manage and visualise the GPCR-related data and to produce a raw tridimensional model. It enables one to build and use an in-house GPCR data bank containing available

subsequence	begins at	ends at	representation
N-terminal	1	TM1b-1	EXTREMITY -1
TM1	TM1b	TM1e	TM 1
IL1	TM1e+1	TM2b-1	LOOP 1
TM2	TM2b	TM2e	TM -1
EL1	TM2e+1	TM3b-1	LOOP -1
TM3	TM3b	TM3e	TM 1
IL2	TM3e+1	TM4e+1	LOOP 1
TM4	TM4b	TM4e	TM -1
...	...	...	...
IL(n/2+1)	TM(n-1)e+1	TMnb -1	LOOP $(-1)^{\frac{n}{2}+1}$
TMn	TMnb	TMne	TM $(-1)^{n+1}$
EL(n/2)	TMne+1	TM(n+1)b-1	LOOP $(-1)^{\frac{n}{2}}$
...	...	...	...
C-terminal	TM(nmax)e+1	seq length	EXTREMITY 1

Table 1: Limits and directions of drawing of the subsequence decomposition of a GPCR for the snake-like algorithm.

information and personal notes on a subset of GPCRs (we prefer to use the expression "data bank", as opposed to "database", to avoid confusion because the latter have a very precise meaning in the computer field).

In the first part of this paper, we describe the novel algorithms and methods implemented in the Viseur program. The second part describes the overall program features the use of which are illustrated in the third part by the creation of the GPCRDB snake-like diagrams, a preparation to the modelling of the serotoninergic receptors (5HT) and a reflection about the interest of browsing mutant data through an alignment.

2 Methods

2.1 Automatic Snake-like diagram construction

A snake-like diagram is a schematic representation for a transmembrane protein sequence. The data needed to construct this representation are: the protein sequence, the number of TM segments and their transmembrane limits. These limits will be notated as TMnb for the position of the first residue in the transmembrane region n and TMne for the last residue of the same TM. According to this notation, the sequence could be divided into subsequences, each one associated with one of the `EXTREMITY`, `TM` or `LOOP` graphical representations. The extra integer number, which follows this representation mode, distinguishes between the direction of the drawing: -1 means that the sequence is drawn from top to bottom (at least for the first part in loops), while 1 means the opposite direction. Subsequence division based on the TM limits are given in table 1.

The algorithm uses the following functions:

- `draw_residue(position,residue)`: draw a residue at a given position with the given colour.
- `residue_radius`: is the radius of the residue. When the residues are drawn as circles, this is the `radius` of the circle. If the residues are drawn as squares, this is the diagonal of the square.
- `sequence_length(sequence)`: gives the length of the parameter sequence.

- `get_TM_number(protein)`: the number of transmembrane segments to present.
- `TM_SPACE` is the horizontal space separating two transmembrane segments.
- `reverse(sequence)` denotes the sequence read in the reverse order.
- `TM_start(loop)` (`TM_end(loop)`) gives the number of the TM segment before (respectively, after) the loop.
- `get_extremity_size(subsequence)` is the size of the drawing of the extremity.
- `shorter(subsequence)` is a subsequence truncated by one residue at each end.
- `direction(transmembrane_segment)` is 1 or -1 according to the direction of the TM (1 means top to bottom, see table 1).

In addition, the following notations are used: if `p` denotes a position, then `p.x` (`p.y`) is the `x` co-ordinate of the position (respectively, `y`). If `tm` denotes a transmembrane segment, then `tm.top_left` is the position where the first residue of the TM subsequence was drawn. Using the same convention, `tm.bottom_right`, `tm.top_right` and `tm.top_left` are defined as the positions where the residue at the bottom right, top right and top left of theTM drawing were drawn, respectively.

The algorithm details are presented in Annexe D. The most difficult technical problem, drawing such pictures, is to determine, *a priori*, the dimensions of the drawing. This is needed for a large number of output media, from the computer screen to the PostScript description for a page. Our implementation computes the maximum dimensions of the drawing using the same algorithm than for drawing, but using a `draw_residue` function that draw nothing but updates the drawing limits, according to the position parameter.

2.2 Automatic access to a mutagenesis data server, through the internet

The viseur program was modified to take advantage of the GPCR mutant retrieval system at Trömso University [2]. The idea was that it would be more useful to visualise and access the mutant data using the dynamic visualisation tools provided by the Viseur program than from a WWW static interface. We intend to describe here the method we use to retrieve the mutant information through the internet. The following features needed to be implemented.

1. A protocol describing precisely the interactions between the client program (Viseur) and the server (TinyGRAP).
2. A client module, that follows the protocol, to access the data through internet.
3. A server module to serve the data according to the protocol. The first and second features are described in this section. The third one was realised by Ø. Edvardsen and al. at Tromsö adding features to the GRAP data bank (constructed on the WWW CGI protocol [11]).

2.2.1 Interaction protocol

This protocol is the result of discussions that lead to an agreement between the GRAP team and us. The protocol says that mutant data will be delivered on request, from the GRAP WWW server, to a client connecting to a special URL,

through an HTML document. This URL may change in the future so we will not give it explicitly here, but just mention that it enables the encoding of the receptor for which the client module requests mutant data. The key is the SWISS-PROT accession number that perfectly identifies a receptor for Viseur and GRAP. The key is sent as a parameter through a GET CGI-BIN query. The HTML file generated by the GRAP server contains a list of items that describes the mutations, written in a strict format to simplify the parsing. For each sequence for which mutant data is requested, the server sends a list of mutants identified by the residue position, the residue type in the wild type and the residue type in the mutant. Because multiple mutations are difficult to interpret and to visualise, they are not sent by the server.

2.2.2 Client Module

The client module is responsible for connecting to the GRAP server, sending the querie, getting the reply from the server, and storing the result on the client machine for latter use by the Viseur program. According to the WWW protocol choice described in "Interaction protocol", and because we wanted the client module to be independent and portable to a wide set of architectures, we decided to implement it using the Java language and associated libraries. These libraries propose an easy way to connect to a WWW server and to read a document from an URL. Finally, for facilitating the re-use of the data obtained from the server by the Viseur program, the client module (FetchTinyGRAP) translates the data to Viseur script files.

2.2.3 Integration of Mutant data to the Viseur program

Two aspects are important here: *(i)* we have to let the user know by some way that mutant data is available for one residue and *(ii)* we have to provide a way for the user to access the data as easily as possible, directly from the Viseur program.

We achieved point *(i)* by colouring residues on request, according to the following: if a mutant data exists for the residue, we colour that residue with a colour named "mutation". Point *(ii)* was obtained via an asynchronous communication with the WWW browser: the Viseur program instructs the browser to update its Viseur reserved view to the URL that describes the mutant data at Tromsö when the user requests information on one residue for which mutant data exists. We used Netscape as the browser for this implementation, as it is available by default on SGI systems, but all browsers that provide commands to redirect a running instance of themselves to a URL are usable. Redirecting the user to the original WWW server data, rather than providing a copy locally, presents the advantages that the user always accesses the real, up-to-date information.

2.3 VRML molecular models hyperlinked on residues

The Viseur program can create OpenInventor [12] files that can be translated into the VRML language. The conversion is done by the ivToVRML [13] tool. A Viseur script command may be used to insert OpenInventor or VRML objects, into the model. This way, ribbons generated from a PDB file as VRML by PDB2VRML [14] can be inserted in the GPCR Viseur Model usual view. This use of the OpenInventor standard to render pieces of a model with different programs and integrate the result in one view seems a first way of improving tridimensional molecular representation reuse. Another application would be the extension of hyperlinked VRML output to non GPCR proteins.

Constructing a VRML model hyperlinked on residues requires that the VRML scene is constructed hierarchically with a WWWAnchor separator object as the top node of the drawing subgraph for each residue. This means that each VRML node

that draws a part of a residue should be a son of the residue WWWAnchor node: this makes the picking work as expected. A URL is then added to the VRML anchor for each residue to link. From VRML2.0, it is possible to add parameters to the URL that redirects the URL to another frame. This feature is really useful to present the data: the tridimensional model and the information data are displayed both at the same time, enabling one to change the point of view while consulting the information. The Viseur implementation makes it easier to pick residues by adding cubes on the C_β. This atomic position was chosen as optimal for GPCR models because the direction between the helix axis and the C_β helps visualise the orientation of the side chains, but for general proteins the C_α position may be preferable, as it enables to highlight Glycine residues as well.

From the application point of view, these VRML representations, that allow one to browse residue information in a tridimensional molecular representation, may allow the user to look at and evaluate a model without any specialised tool besides a VRML aware WWW browser. Whereas such browsers are becoming more and more frequent, regardless of the platform, one could expect a growing interest for this type of representation. We could mention the following limitation of such representations: the tridimensional representation of the protein is static (there is no way to change a few atoms to ball and stick after the representation has been constructed for instance). This may be a sensible limitation for certain uses but the actual WWW technology does not allow one to build, from the client side, tridimensional models on the fly that will be rendered with correct performance. The current development of client side API for 3D may enable the development of macromolecular hyperlinked model browsers in the future.

3 Results

The novel features presented are implemented into the Viseur program, which proposes an integrated solution for storing, visualising and manipulating GPCR sequences, alignments or molecular models. This section will describe the features of the implementation at the time of writing.

3.1 Receptors

The program allows the user to read protein sequences. The following file formats are recognised.

- SWISS-PROT (original [15, GCG modification, and N.I.H. variant)
- SQIDE3 (CAS) Registry System, [16]
- MSF (from an alignment)
- VISEUR (defined in the appendix)

The transmembrane domain limits are read from the file when it is SWISS-PROT but not SQIDE3 or MSF. The sequence view enables the user to view the receptor sequence and the associated information (clicking on one amino acid in this view with the right mouse button displays or updates the information view for this sequence). As illustrated in figure 1, the transmembrane limits are displayed on the top of the sequence with rectangular areas. The user can move a TM area dragging the TM rectangle. He can also resize a TM area by dragging the left or right side of the rectangle.

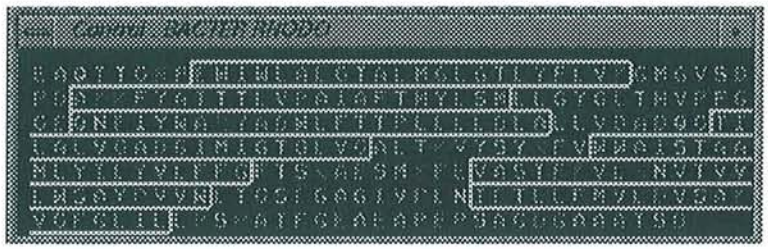


Figure 1: Viseur's program Sequence view for the bacteriorhodopsin.

3.2 Information

An information view is updated when the user selects one amino acid from one view, whatever this is. The information view displays the following information.

- Receptor name,
- residue position,
- residue type,
- information files attached to this receptor (clicking on these buttons brings up an editor). Currently, these files contain: the sequence itself when it is imported and a Viseur script that describes the mutations attached to the receptor;
- mutations: what was the residue mutated into? If there exists a mutation on the selected residue, then the Web browser is requested to display the URL associated with that mutation.

The information files are ASCII text files organised in directories. Directories are named according to the sequence the information belongs to. One file contains the original sequence file that was imported into Viseur so that no information is lost when sequences are imported. Information about amino acids is stored in files in that special directory. Such a way to store data makes possible the use of different file formats. For example one can think about using HTML files to describe the information attached to an amino acid.

3.3 Colouring

By default, the residues are coloured according to their types and to a user defined association colour table. For instance, it is possible to colour residues according to their chemical properties, or according to given crystallographic conventions. It appeared from a preliminary use of the program that being able to colour residues individually on the basis of their properties is a powerful feature. Viseur enables the user to colour individual amino acids manually or via the script language and to switch between a default colour visualisation mode and a user-defined colour visualisation mode (mode Point out information off/on). Colouring amino acids is available on every view: sequence, snake-like, alignment and model. The colouring can be saved or restored on request, leading to a set of coloured annotations: the user can use this feature to highlight similarities between sequences in an alignment, to point out data from mutagenesis databases, or at his convenience. We describe in Appendix C the file record format for the colouring of amino acids and the example of a script that can be useful to someone who wants to build such files with a program.

3.4 Snake-like View

The snake-like view, shown on the picture part of figure 2 allows the user to visualise the receptor in a schematic way. The sequence is drawn according to the TM limits contained in the receptor object. A snake-like view is updated every time the helix positions are changed through the associated sequence view.

This view can be printed as Encapsulated PostScript or written as a HTML file for Web publishing. The HTML output is shown in figure 2. The image is clickable. Using the script language one can easily build an HTML page from a SWISS-PROT file and add URL and different colour to the interesting amino acids in the sequence. This presentation is a particularly attractive and clear way of publishing GPCR mutational information on the World Wide Web [17]. Such an application is presented in this article.

3.5 Alignments

The alignments can be read from MSF files or from Viseur alignment files. A Viseur alignment consists of a set of receptors and it describes the gaps that should be added inside the alignment to have the optimum number of matches between amino acids in all sequences. Viseur was designed to separate the receptor and the alignment concepts. An alignment should contain only the gap information and references to receptors because the receptor information is redundant when also included in an alignment object. This is a common engineering practice that simplifies the management of data, but it is usually neglected by alignment software designers (e.g. the Multiple Sequence Files: MSF, a widely used file type). This information partition enables users to access and modify the information attached to a sequence from any alignment that contains the sequence, without breaking the information consistency.

The Viseur alignment module (figure 3) was designed with the idea that no perfect automatic alignment program is yet available. Instead, alignments should be tuned manually to take into account all the available information. This module is a tool which makes it easier to align a lot of receptor sequences manually. The main features of the editor are:

- unlimited receptor sequences,
- receptor sequences ordering by block of sequences,
- fixed areas (column areas are constrained aligned),
- multiple gap insertion/removal,
- multiple sequences destination for gap insertion/removal,
- visualisation of transmembrane areas (full light level),
- high quality postscript output for publications,
- amino-acid deletion/insertion,
- amino-acid painting: to highlight information on the top of an alignment.

Moreover, the integration of the alignment module as a Viseur object enables the automatic update of amino-acid colour and transmembrane area positions, direct access to the information module and construction of clikable molecular models for immediate inspection.

3.6 Models

The models are read or written in a PDB format [18]. The connectivity information is assumed to be implicitly present in PDB files: it can be derived from atom types. A model, in the Viseur program, is a GPCR tridimensional structure. It contains atomic positions and connectivity. It is constructed from an original model (experimental or modelled data) or from an original model and an alignment. We decided to include a molecular model visualisation module into the program to take advantage of comparisons between sequences, alignment derived properties and a tridimensional representation.

The program allows one to build a model from an alignment in two ways.

- The first way creates, given just the alignment of a receptor sequence with the template sequence, a Sybyl (Tripos Inc.) command file. This Sybyl SPL file can be applied (with Sybyl) to the template structure to get the new protein with the skeleton from the template and the side chains rotated to remove poor contacts.

- The second way creates a schematic model internally using the following procedure.

    1. Read a template structure from a PDB file;
    2. Clone the template structure;
    3. Remove the loops (defined according to the template receptor transmembrane areas position);
    4. For each aligned amino acid available in the template: replace the atoms of the side chain on the cloned structure by new side chain atoms. This results in a template skeleton structure carrying the side chains of the transmembrane regions of the target receptor.
    5. The side chains are rotated to put the new C_β at the template C_β position. No more optimisation/changes are done.
    6. The new structure is written in a PDB formatted file.

This simplistic model building method will create raw models. The main interest of such crude models is that they can be used advantageously to combine all the information available on a system or as a starting tridimensional arrangement for testing hypotheses and confronting with sequence oriented data.

Ribbons are drawn on the model view thanks to the PDB2VRML tool. The tridimensional model visualisation displays different representations according to the state of the Point Out information mode. When the mode is off, this view adds cubes to any C_β, coloured with the default residue colour (described previously). When the mode is on, only those residues that were painted are highlighted by a C_β cube, coloured with the painting colour for each residue. This highlights information on the model.

3.7 Dynamic behaviour of objects

Dynamic behaviour is an important feature of the Viseur program. The user may change the attributes of objects and see the consequences immediately in each view related to that parameter. The preferences object was designed as an interface between the user and the parameters that affect Viseurs behaviour. If a parameter is changed by the user through the preferences object, it affects immediately the related Viseur behaviour. The preferences object gives a consistent interface to the Viseur static attributes (as opposed to functions): its use reduces the time

spent programing the interface. This object also concentrates all the configurable attributes in one location, with a uniform interface, so that users who need to change the parameters of the program only need to look in one place. The main problems with this object are that *(i)* it is restricted on the type of the parameters it can handle and *(ii)* it cannot be used to apply multiparameter actions to the Viseur objects.

4 Applications

4.1 Hyperlinked snake-like HTML pages on the GPCRDB WWW server

The snake-like HTML output was used as an access point to mutational information on the Web for the GPCRDB server. This type of presentation is very attractive because it highlights immediately the nature of mutations: is the mutated amino acid located in the transmembrane part of the receptor or in the loops? Is it near the extracellular side or near the intracellular side? Is the amino acid in a highly lipophilic region or is it surrounded by hydrophilic residues? Following this quick inspection, the user can go into the details of a mutation by a click of the mouse on the amino-acid drawing (classical hyperlink behaviour on images). The Viseur script file needed to build a snake-like HTML page from a Web server side is presented in appendix B.

4.2 Application to a modelling study of 5HT receptors

We present here a preparation of the modelling of receptors from the 5HT family. This family was chosen because the modelling of these receptors was already well studied by independent laboratories, and that the results, if not the methodology, gave rise to a consensus. Our goal here is not to propose a new model of these receptors, but rather to highlight how the concepts introduced in the design of the program are useful for obtaining such a model.

4.2.1 Obtaining the data

Modelling of a 5HT receptor is most efficiently achieved considering the entire family. The following receptor sequences were, thus, retrieved from SWISS-PROT:

```
5HT1A_(HUMAN, RAT),
5HT1D_(CANFA, HUMAN, RABBIT, RAT)
5HT1B_(CRIGR, DIDMA, HUMAN, MOUSE, RABBIT, RAT),
5HT1E_HUMAN,
5HT1F_(HUMAN, MOUSE, RAT),
5HT2A_(CAVPO, CRIGR, HUMAN, MACMU, MOUSE, PIG, RAT),
5HT2B_(HUMAN, MOUSE, RAT),
5HT2C_(HUMAN, MOUSE, RAT),
5HT5A_(HUMAN, MOUSE, RAT),
5HT5B_(MOUSE, RAT),
5TH6_(HUMAN, RAT),
5HT7_(CAVPO, HUMAN, MOUSE, RAT),
5HT3_(HUMAN, MOUSE, RAT),
```

This constitutes a set of 43 sequences. The 5HT1 and 5HT2 sequences were loaded into the Viseur program and an automatic mutant search was run to retrieve mutagenesis information from the TinyGRAP data bank (algorithm described previ-

ously). This resulted in the integration of mutant information in the 5HT focused data bank maintained by the program during our modelling.

4.3 Aligning 5HT1 and 5HT2 receptor sequences

Aligning the sequences was easier than with classical alignment tools because the TM limits described in the SWISS-PROT files (`FT TRANSMEMBRANE`), even though indicated "POTENTIAL" are a good starting point for finding the TM subsequences. Their quality could be controlled on a few sequences by using the Snake-like view before continuing. The idea is to roughly align sequences on the basis of the highlighted subsequences (TM), reordering the sequences after each TM, if necessary, to group sequences with higher sequence homology. This way we noticed that the 5HT1A TM1 subsequences are really different from other TM1 sequences, although they are reasonably homologous for other TM subsequences. This may indicate that a recombination has occurred at some stage during evolution, and this information should be taken into account for the modelling because the importance of the functional role of TM1 in the whole family is now questionable : if the function was retained after a mutation as important as the replacement of one TM (maybe followed by point mutations at a usual rate), a simple hypothesis would be that this TM contribution to the function was, and is, negligible. An other hypothesis is that the recombination resulted in a small change to the receptor pharmacology, resulting in the emergence of a new subtype: 5HT1A.

The alignment was refined, looking for conserved residues among the TM. At this stage, one may choose a colour residue association that groups chemical properties of residues: a colour for aromatic residues, another for lipophilic residues, etc... Viseur is able to paint residues, on a set of sequences, when they appear mutated (from the project data bank). This feature was used intensively here to browse the mutant information while refining the alignment.

The obtained alignment was compared to published alignments [1] [3] [19] [20] [21] [22]. All were found to be in good agreement (for TM1, TM2, TM4, TM6, TM7), except for TM3, as, according to the authors, one or two helicies turn difference appear for the alignment of bR with the 5HT sequences. The alignment we retained between bR, `5HT1A_HUMAN`, `5HT2A_HUMAN` was printed as PostScript with Viseur and is presented on figure 4.

4.4 Evaluating raw molecular models

At this stage, a few raw models were constructed for `5HT1A_HUMAN` and `5HT2B_HUMAN`, to check the agreement between the alignment and mutant data on the tridimensional model. There are two ways to change the model to fit the mutant data. One could rotate the helices or shift the alignment by a few residues. According to the low homology between bR and 5HT sequences, the two methods can be applied equally here, unless some special method is used for an automatic alignment of the sequences based on calculated properties. On the contrary, if the alignment is straightforward, the rotation of the helices is the only rational degree of freedom in the model. These considerations lead to the following changes to the model.

- TM5: For the receptor `5HT1A_HUMAN`, Ser199 and Thr200 are found by mutagenesis studies to be important to the binding. Here the alignment was shifted by two residues.

- TM7: For the receptor `5HT2A_HUMAN`, N376 (TM7) seems to interact with D120 (TM2) [23], which implies that these two residues should be narrow in space. The TM7 was rotated to accommodate this constrain.

As rotating the helices is not yet possible from Viseur Model view (this feature is planned to be added in a latter release of the program), this part of the modelling was done with Sybyl, using the Sybyl command file generated by Viseur to construct models from the bR PDB file. The modelling was pursued using molecular modelling methods.

À la date de rédaction de cette thèse, cette fonctionnalité est implantée.

Finally, Viseur was used to check the agreement of the resulting models with the mutant data. We may use it in the future (after updates to the mutation server) to check the agreement of refined models with new mutants.

4.5 Accessing mutagenesis data through the alignment, application to the IN/OUT concept

Alignments may be used to transfer a special type of information: structural information from one amino acid (taken from the templates) to one aligned amino acid of the receptor of interest. With the construction of the exact determination of the set of aligned residues, this is the basis of the homology modelling. Here, we suggest that in certain cases, where homology between proteins is high, the information transfer could be extended to mutational data. The cases were this transfer is not possible, because of contradictions, are important to notice because the homology hypotheses are certainly not fulfilled for the parts implied in the alignment. This criterion will become more and more useful as the mutational data volume increases.

If in an alignment, we define $(A)_{i,j}$ as the residue of sequence i at the position j in the alignment (j includes the gaps: $\mathtt{A_1B_2{-}{-}C_5}$), $(A)_{I,c}$, as the residue for the sequence of the receptor I being modelled at the position c in one alignment, and $(A)_{R,c}$ the residue from receptor R at the same position c in the alignment, then, according to these notations, we say: if there is no mutational experimental result on $(A)_{I,c}$, but mutational experimental results exists for $(A)_{R,c}$, then one can derive new information on $(A)_{I,c}$. When only one residue in a column is present, for which we have information, the transfer is $Information((A)_{I,c}) \leftarrow Information((A)_{R,c})$. Because there is only one information in the column, we cannot confrontate it with others, so we consider its quality to be inferior to the following cases. If an ensemble of residues exists in a column

$$Sc = \{(A)_{Rk,c}, (A)_{Rl,c} \ldots\}$$

and

$$card(Sc) > 1$$

and

$$\forall R \; Information((A)_{R,c}) = consensus$$

then it is nearly reasonable to say Information$((A)_{I,c}) \leftarrow consensus$.

The last case is when a set of residues exists in a column

$$Sc = \{(A)_{Rk,c}, (A)_{Rl,c}, \ldots\}$$

and

$$card(Sc) > 1$$

and

$$\forall R \; Information((A)R, c) = I_R$$

Assigning Information$((A)_{I,c}$) becomes a subjective task. It seems to us that this situation should be detected because it can highlight some regions of the alignment where the homology hypotheses are not well fulfilled. Browsing mutant data directly on the alignment is a way to facilitate the detection of such situations.

When mutational information is not available on the receptor of interest (or the information quantity is low) the above procedure can be used when confronting the IN/OUT problem. The problem is essentially *(i)* to know which amino acids are pointing towards the core of the receptor (IN amino acid) and which are pointing towards the lipids (OUT amino acid) and *(ii)* to correct the helix arrangement according to the information from *(i)*.

The Viseur program can be useful during step *(i)*. IN/OUT information can be derived from mutagenesis experiment results by the user with the tools provided by the program. If a receptor amino acid is mutated and the new receptor has quite the same activity as the original one, the amino acid can be considered anywhere on the model (OUT or IN but not interacting). If the activity changes (agonist binding change) then the receptor could be considered IN though the possibility of indirect effects makes this assignment unsafe.

IN/OUT reduced information is determined for each amino acid of the alignment for which some mutagenesis information is available, for instance colouring residues according to an (IN,OUT) convention. The procedure described above may be used, manually, to transfer that data to the receptor of interest (i.e. colouring the receptor residues IN or OUT). Considering the significant uncertainty about these derived data, it is advisable to consider an average over adjacent residues when tuning the helix arrangement.

The Viseur alignment editor can be used to display the IN/OUT information by a given colour on the alignment. The Model view can be of valuable help during step *(ii)*. The IN/OUT amino acid can be visualised easily on the tridimensional Model. Please see figure 5: this view shows the OPRD_HUMAN receptor model (from [24]) IN/OUT painted according to mutagenesis experimental results (OUT means that the residue should not interact with the ligand).

5 Conclusions and perspectives

In this article, we have described an algorithm to draw snake-like diagrams. This algorithm allows the presentation of GPCR sequence and mutant data in a schematic way, which GPCR scientists are used to. A method to present GPCR models on the WWW, hyperlinked per residue to information, is also presented and its interest discussed. We have described the Viseur program which implements these algorithms, as well as some applications.

The Viseur program was originally developed for GPCR modelling but may be used for modelling any integral transmembrane proteins since the helix number is configurable. From a practical point of view, the program proposes interconnected visualisation modes that can be used to integrate a lot of the information that is available for a set of GPCRs, and so, to facilitate the modelling. But this program also allows one to test and evaluate the relevance of new working methods: access to mutant information through a sequence multi-alignment editor or a molecular model. It seems to us, according to our experience on GPCR systems, that these methods are of great help during the modelling process, and that their implementation for modelling different protein systems would be advisable.

Finally, the Viseur program visualisation features allow one, considering the HTML and VRML output features, to publish a GPCR annotated Model on the Web. The latter features may improve information exchanges on GPCR between the experimental and the modelling scientific communities.

For up to date information about the Viseur program, including distribution access, and recent developments, please visit http:/www.lctn.u-nancy.fr/viseur/-viseur.html.

6 Acknowledgements

The main part of this work was realised in collaboration with "Sanofi Recherche", according to the grant number 061064300. The latest developments are realised thanks to the grant accorded to F.C. by the high performance computing center "Centre Charles Hermite" and the "Université Henri Poincaré, Nancy I". F. C. would like to thanks G. Vriend, G. Poda for interesting discussions and H. Vollhardt for the PDB2VRML tool. Last but not least, we would like to thank Christophe Chipot and Steven Spencer for carefully reading the manuscript.

7 APPENDIX

7.1 APPENDIX A — VISEUR input/output file syntax and grammar

Quite often, when a new program is written, a new data format appears. The fact is regrettable but seems inevitable. The reason for this fact is that a new program usually adds new attributes to the data it transforms which cannot be stored in a standard way using existing file formats. The situation is worse when the programmers "adapt" an existing standard to accommodate the own needs of their programs without describing explicitly the "adaptations". This way one can find at least three SWISS-PROT dialects differing only by the format of one part of one line. A consequence is that a program designed to read SWISS-PROT should be aware of the three dialects to be able to read any SWISS-PROT file available at this time. This increases program development time and complexity, which is not really compatible with research development. When designing Viseur we wanted a file format with some concept oriented properties and a good potential to change without becoming incompatible with older releases of the program. We describe here the syntax and grammar used widely by Viseur files (receptor, alignment, colouring and script files). The files are a "list of thing" like this (optional elements are surrounded by brackets):

```
identifier [ (tag1) [(tag2)] ... ] {

 [list of thing]
}
```

A more formal grammar description is given here : `list_of_commands` : `command` $\texttt{list\_of\_commands}_{opt}$

`command` :
`identifier` $\texttt{list\_of\_tags}_{opt}$ $\texttt{command\_block}_{opt}$

`list_of_tags` : `tag` $\texttt{list\_of\_tags}_{opt}$

`command_block` :
{ `list_of_commands` }

A command is first composed of an identifier ([^]+) followed by any number of tags (anything enclosed by two matching parenthesis) then optionally another list of command enclosed by " and ". Comments

```
#[^\n]*
```

can appear anywhere on a line, they begin at a '#' (not inside a tag) and finish at the end of the line. This syntax enables concept/object oriented description of the data if used in the following way.

```
RECEPTOR {

  NAME (Vasoactive intestinal polypeptide receptor human)
  LEN (457)
  ...
}
```

The identifier should name the concept. For example, Viseur uses the RECEPTOR identifier to group attributes for a GPCR: the concepts enclosed in the braces will refer to the GPCR. RECEPTOR defines a GPCR scope. A GPCR has a name: the NAME identifier inside the GPCR scope is the name of the GPCR. The tag of the NAME attribute gives the value of the NAME attribute for the GPCR.

8 APPENDIX B — snake-like HTML output script

We present here the Viseur script file needed to build a snake-like HTML page from a Web server side.

```
# viseur command file (2.42 compatible).
# Creates a snake-like html output in ./Export.
# Needs further configuration of the HTML attributes depending
# on your web http daemon and file organisation.

load-sequence (OPRD_HUMAN.sw) (s1) # load the sequence OPRD_HUMAN
                                   # with the attached identifier s1

viseur-color {                     # define a colour with a legend.
        RGB (255) (255) (255)
        text (info here)
}

view-painting (on)                 # switch on AA painting display.

sequence (s1) {                    # put the scope on sequence s1
        uri (sequence.html)        # attach a URI/URL to it.
        paint-AA (2) (info here) # paint amino acid 2 with the
                                          # colour "info here".
        paint-AA (32) (info here)
        aa (2) {                   # put the scope on amino acid 2 (in s1).
                uri (2.html)       # attach a URI/URL to it.
        }
        aa (32) { uri (32.html)}
}
il-work-around-kludge

2dview (s1) (2d1) {      # create a Snake-like view from a sequence (s1).
                         # give it an identifier (2d1)
#       show             # show Snake-like view (open the window).

        html {           # puts the scope on HTML attributes.
```

```
                        # please take a look at
                        # http://www.lctn.u-nancy.fr/
                        # viseur/using/documentation/CommandFiles.html
                        # for detailed info on these attributes.

                base-name (OPRD_HUMAN)
                image-map-tool (http://htbin/htimage)
                default-uri (def.html)

                server-type (W3C) # W3C or NCSA depending on your server.
        }
        export-html (./Export)  # put the snake-like html output
                                # in the directory ./Export.
                                # the directory is created if
                                # it didn't exist..
}
```

9 APPENDIX C — Colouring script example

```
for-sequence (OPRD_MOUSE) {

   paint-AA (95) (agonist binding change)
   paint-AA (107) (no effect)
}
for-sequence (OPRK_MOUSE) {

   paint-AA (105) (agonist binding change)
}
```

The for-sequence node puts the scope on the receptor sequence. The name of the receptor is given by the first tag. The `paint-AA` nodes in this scope will change the colour of the amino acid whose number is given by the first tag to the colour associated with the legend (second tag). The legend is a text associated with a colour. This way the colouring files are more readable and the legend could be used on screen or in a PostScript output. The colours are attached to legends using a colour set file. An in depth (and up to date) description is available in the machine inline Viseur documentation.

10 APPENDIX D — Algorithm details

```
begin draw_snake_like(prot is a protein,
seq is a sequence,
start is a position)

p = start;

EE is a subsequence initialized to seq[1-(TM1b - 1)]
IE is a subsequence initialized to seq [(TMne+1)-
get_sequence_length(seq)]

upper_left is a position = (RADIUS_AA, RADIUS_AA)
bottom_right is a position =
```

```
  get_extrimity_size(EE) +  upper_left
bottom_EE is a position = (0, bottom_right.y)

draw_extremity(TM1.upper-left,
upper_left, bottom_right,
-1,
reverse(EE))

for each TM tm:
TMi is a subsequence initialized to sequence[TMib-TMie]
pos.x= pos.x + TM_SPACE
pos.y = bottom_EE.y
 direction = direction(tm)

draw_tm(pos,TMi, direction)

endforeach

upper_left is a position =
(TM(get_TM_number(protein))).bottom_right
bottom_right is a position = get_extremity_size(IE)
start is a position =

(TM(get_TM_number(protein))).bottom_right

draw_extremity( start,
upper_left, bottom_right,
1,
IE)

for each LOOP l
  tmb is an integer = TM(originTM(l))b
 tme is an integer = TM(endTM(l))e
  LOOPi is a subsequence = sequence[tmb-tme]

if tmb modulo 2 equal 1 then  # IL

start is a position = originTM(l).bottom_left
 end is a position = endTM(l).bottom_right
direction = -1

else # EL

start is a position = originTM(l).top_left
 end is a position = endTM(l).top_right
direction = 1
endif

  draw_loop(start, end, LOOPi, direction)
end draw_snake_like

begin draw_extremity (start is a position,
upper_left_limit is a position,
bottom_right_limit is a position,
```

```
direction is an integer,
subseq is a subsequence)

start.y = start.y + 2*residue_radius * direction
 start.x = start.x + 2*residue_radius
# new position
# for drawing is just beside the start position, according
# to the direction

horizontal_direction is an integer = -1 # to the left
 vertical is an integer = 0

while ( exist (r is the next residue of subseq) )

if vertical equal 0 then
if horizontal_direction < 0 then
if start.x > upper_left_limit.x then
start.x = start.x - 2 * residue_radius*direction
else
vertical = 1
start.y = start.y + 2 * residue_radius*direction
endif start.x
else
if start.x < bottom_right.x then
start.x = start.x + 2 * residue_radius*direction
else
vertical = 1
start.y = start.y + 2 * residue_radius*direction
endif start.x
endif horizontal_direction
else
start.y = start.y + 2 * residue_radius*direction
vertical = 0
horizontal_direction = horizontal_direction * -1
endif vertical

draw_residue(start, r)
endwhile
end draw_extremity

begin draw_loop(start is a position,
end is a position,
loop a subsequence,
direction is an integer)

start.y = start.y + 2*residue_radius * direction
end.y = end.y + 2*residue_radius * direction

len is an integer = sequence_length(loop)

if len < 0 end draw_loop

if len * 2*residue_radius <
(end.x - start.x + 2*residue_position) then
```

```
# everything on one line

start.x = start.x + residue_radius
while (exist (r is the next residue of loop)
draw_residue(start, r)
start.x = start.x + residue_radius
else
draw_residue(first(loop), start)
draw_residue(last(loop), end)
draw_loop(start,end,shorter(loop),direction)
endif len
end draw_loop

begin draw_tm(start is a position,
tm is a subsequence,
direction is an integer)

is_for is an integer = 1
while (exist (r is the next residue of tm))
... we will not describe this part : it simply draws the
subsequence corresponding to the TM in a rectangular area,
according to the direction indicated as a parameter.
endwhile
end draw_tm
```

11 APPENDIX E — Technical information

The first releases of the Viseur program were written with the C language, but new functionalities are now added with the C++ language. The interface was written for the X Window system and Motif toolkit. The program itself is available for SGI workstations only. A few parts of the code should be changed to allow portability but developing, maintaining and packaging releases for more than one architecture would consume too much of our resources. The Model view is written with the OpenInventor 2.1.1 API. The program was tested under IRIX 5.3, 6.1, 6.2, 6.3, 6.4.

12 REFERENCES

References

[1] M. Van Rhee K. A. Jacobson, Drug Development Research., 37, (1996) 1- 38.

[2] K. Kristiansen, S. G. Dahl, Ø. Edvardsen, *A Database of Mutants and effects of Site-Directed Mutagenesis Experiments on G Protein-Coupled Receptors. Proteins: Structure, Function, and Genetics*, 26 (1996) 81-94. URL: http://www-grap.fagmed.uit.no/GRAP/homepage.html.

[3] URL: http ://www.sander.embl-heidelberg.de/7tm

[4] G.F.X. Schertler, A.C.M. Villa, R. Henderson, *Projection structure of rhodopsin*, Nature 362, (1993) 770-772.

[5] N. Griegorieff, TA Ceska, KH Downing, JM Baldwin, R. Henderson, *Electron-crystallographic refinement of the structure of bacteriorhodopsin*, J. Mol. Biol. (1996), 259(3) 393-421.

[6] M. Cserzö, J.-M. Bernassau, I. Simon, *New alignment strategy for transmembrane proteins*, J. Mol. Biol., 243 (1994) 388-396.

[7] H. Weinstein, BIOSCI International Newsgroups for Molecular Biology, bionet.molbio.proteins.7tms_r, 11 Apr. 97.

[8] URL: http ://expasy.hcuge.ch/cgi-bin/ProMod-GPCR.pl

[9] M.-P. Joseph, B. Maigret, J.-C. Bonnafous, and H. A. Scheraga,*A computer Modeling Postulated Mechanism for Angiotensin II Receptor Activation*, J. of Protein Chemistry, Vol. 14 5 (1994) 381-398.

[10] P. Herzyk and R. E. Hubbard,*Automated Method for modeling Seven-Helix Transmembrane Receptors from Experimental Data*, Biophys J., 69 (1995) 2419-2442.

[11] CGI-BIN protocol, specifications 1.1, URL: http://hoohoo.ncsa.uiuc.edu/cgi.

[12] Open Inventor Architecture Group, 1994, *The Inventor Mentor : Programming Object-Oriented 3D Graphics with Open Inventor*, Addison-Wesley Publishing Company, ISBN 0-201-62495-8.

[13] URL: http://vrml.sgi.com/intro.html, URL: http://webspace.sgi.com/Tools

[14] H. Vollhardt, and J. Brickmann, Proceedings of the Pacific Symposium on Biocomputing '96, L. Hunter, T.E. Klein, Eds., World Scientific Publishing, Singapore, (1995) 663-673.

[15] A. Bairoch and R. Apweiler, *The SWISS-PROT protein sequence data bank and its new supplement TrEMBL.* Nucleic Acids Res., 24 (1996) 21-25

[16] Chemical Abstracts Service, 2540 Olentangy River Road, P. O. Box 3012,Columbus, OH 43210-0012 USA.

[17] T. Berners-Lee, R. Cailliau, J. Groff, and B. Pollerman, World Wide Web : the Information Universe, Electronic Networking : Research, Applications and Policy, Vol. 1. (1992).

[18] PDB Contents Guide, URL: http://www.pdb.bnl.gov

[19] M.F. Hibert, S. Trumpp-Kallmeyer, A. Bruinvels, and J. Hoflack, Mol. Pharmacology, (1991), 40: 8-15.

[20] S. Trumpp-Kallmeyer J. Hoflack, A.Bruinvels, and M.Hibert, Modeling of G-Protein-Coupled Receptors: Application to Dopamine, Adrenaline, Serotonin, and Mammalian Opsin Receptors, J. of Med. Chem., (1992), Vol 34, 19: 3448-3462.

[21] R.A. Glennon, R.B. Westkaemper, 5HT1D Receptors: A Serotonin Receptor Population for the 1990s. Drug News Perspectives, (1993) 6: 390-405

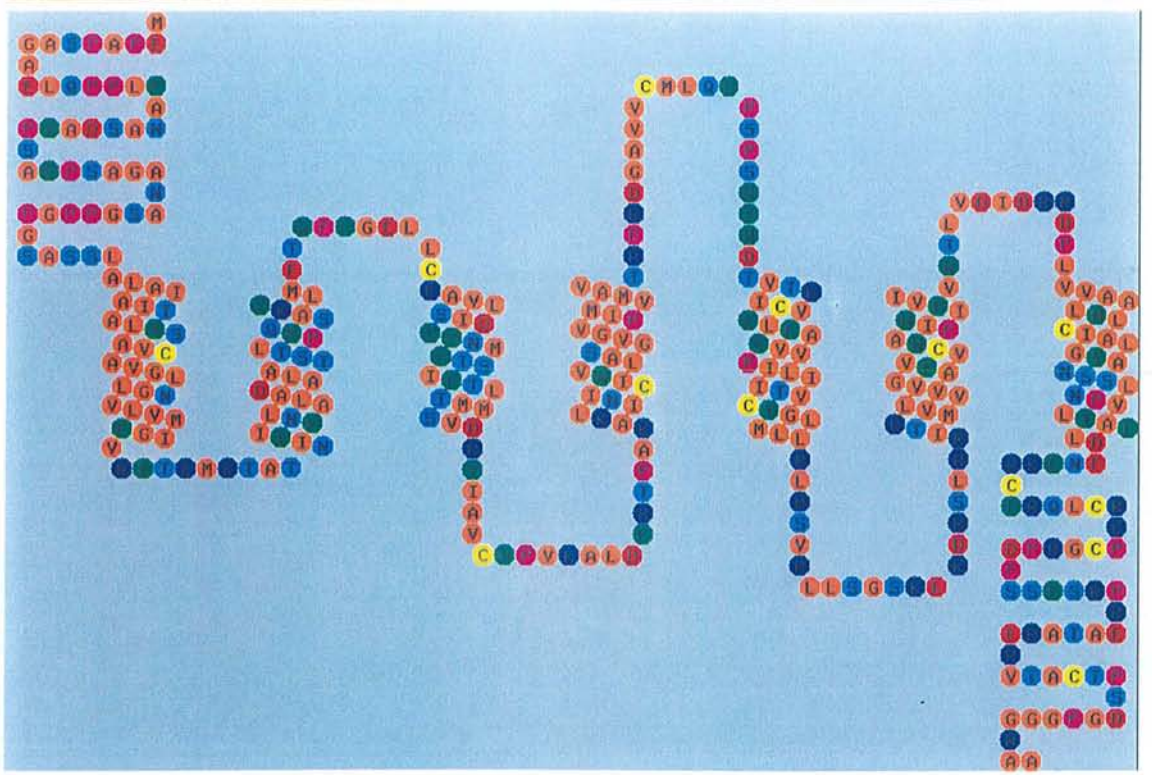
[22] R.A. Glennon, M. Dukat, R.B. Westkaemper, A.M. Ismaiel, The binding of propanolol at 5-hydroxyptamine 1Db T355N Mutant Receceptors may involve formation of two hydrogen bonds to Asparagine, J. of Mol. Pharmacology, (1996) 49: 198-206.

[23] S.C. Seafon, L. Chi, B. J. Ebersole, V. Rodic, D. Zhang, J. A. Ballesteros, and H. Weinstein, Related Contribution of Specific Helix 2 and 7 Residues to Conformational Activation of Serotonin 5-HT2A Receptor, J. of Biological Chemistry, (1995), Vol 270, 28: 16683-16688.

[24] G. Poda, B. Maigret, Postulated activation mechanism and signal transduction by d opioid receptor and its interaction with d-selective enkephalin analog DPDPE. In preparation.

Viseur's snake like view

Sequence : OPRD_HUMAN
From file : ./Sequences/OPRD_HUMAN.viseur
Encapsulated PostScript Viseur output



Laboratoire de Chimie Théorique de Nancy
Viseur Home page
Copyright notice

Figure 2: Viseur's Snake-like View, HTML output for the OPRD_HUMAN receptor.

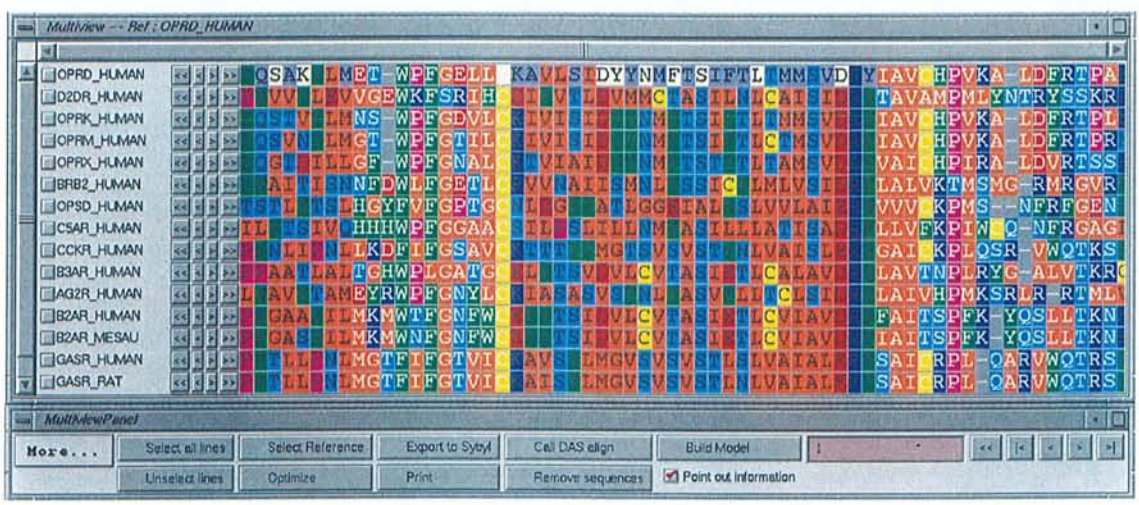


Figure 3: Viseur's Alignment editor, with residues painted according to user defined convention for OPRD_HUMAN sequence.

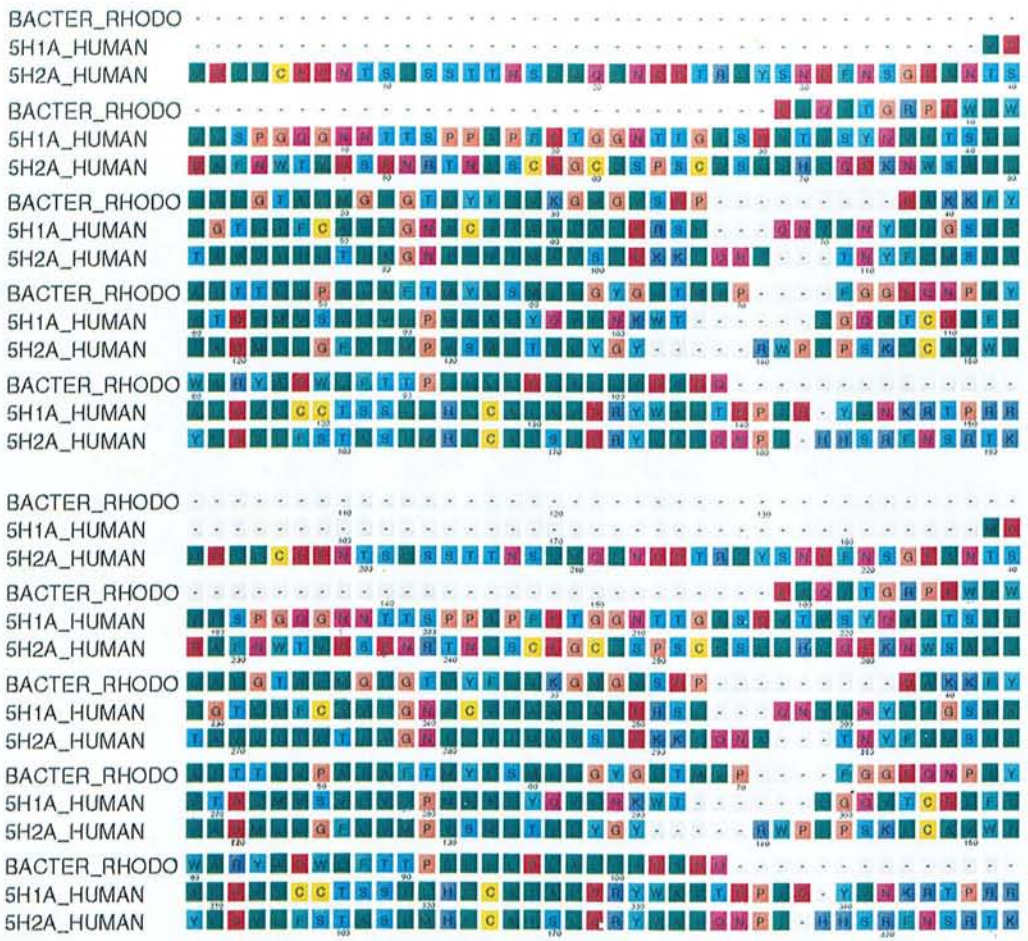


Figure 4: Sequence alignment of bR, 5HT1A_HUMAN, 5HT2A_HUMAN, (configurable PostScript output generated with the Viseur program).

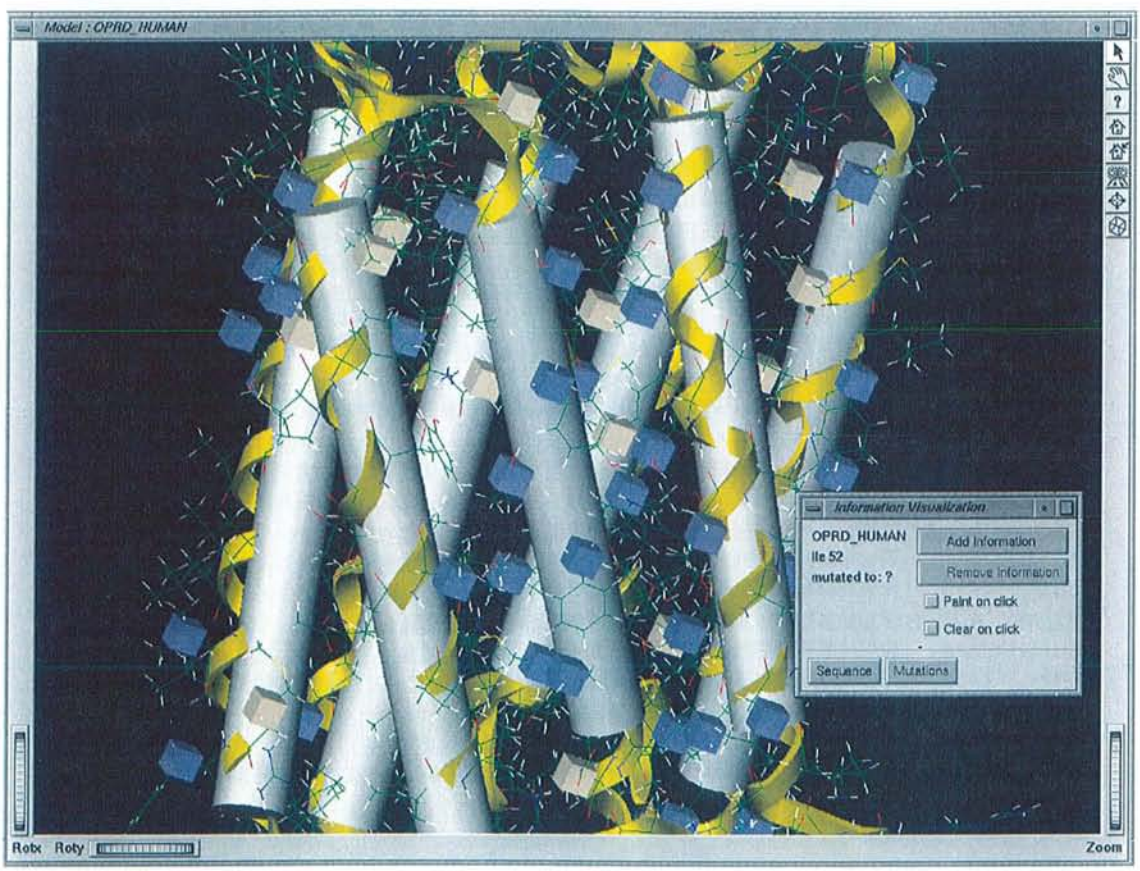


Figure 5: Viseur's Model View of the OPRD_HUMAN receptor, IN/OUT painted. White couloured residues are condidered IN, blue residues are OUT or IN but without contact with the ligands.

B

Publication Alignements Multiples en HTML

F. Campagne and B. Maigret. Multiple Sequence Alignment in HTML: colored, possibly hyperlinked, compact representations. *J. Of Molec. Graph. & Mod.*, 1998. in press.

Multiple Sequence Alignment in HTML: colored, possibly hyper-linked, compact representations*

Campagne F. and Maigret* B.

* To whom correspondence should be addressed.
E-mails: campagne@lctn.u-nancy.fr, maigret@lctn.u-nancy.fr

Laboratoire de Chimie théorique,
Université Henri Poincaré Nancy I,
UMR CNRS 7565,
BP 239,
54506 Vandoeuvre-les-Nancy
France

keywords:
Multiple Sequence Alignment, HTML representation

Abstract

Protein sequence alignments are widely used in protein structure prediction, protein engineering, modeling of proteins, etc. This representation is useful at different stages of the scientific activity: looking at previous results, working on a research project, and presenting the results. There is a need for making it available through a network (intranet or WWW), in a way that allows biologists, chemists, and a non-computer specialist, to look at the data to carry on his research – possibly a collaborative research. Previous methods (text-based, java-based) are reported and their advantages are discussed. We developed two novel approaches to represent the alignments as colored, hyper-linked HTML pages. The first method creates an HTML page that uses efficiently the image cache mechanism of a WWW browser, thereby allowing the user to browse different alignments without waiting for the images to be loaded through the network, but only for the first viewed alignment. The generated pages can be browsed with any HTML2.0 compliant browser. The second method that we propose, uses W3C-CSS1 style sheets to render alignments. This new method generates pages that require recent browsers to be viewed. We implemented these methods in the Viseur program and

*Paper submitted to Electronic Conference of the Molecular Graphics and Modelling Society, MGM EC-2, October 97

made a WWW service available that allows a user to convert an MSF alignment file in HTML for WWW publishing. The latter service is available at http://www.lctn.u-nancy.fr/viseur/services.html

1 Introduction

1.1 General interest of sequence alignment representations

Multiple sequence alignments are a useful tool to compare, work with, and represent related (bio)polymer sequences. They help highlighting similarities and differences between regions of the sequences, that would not come into sight considering sequences individually. Program tools [1] were developed to present and display *Multiple Sequence Alignments* (MSA) for purposes of MSA result publication, help thinking, etc..., on paper media. We are presenting here methods to achieve these results with network media: intranet and WWW.

1.2 Interest of MSA network representations

There is a growing interest in accessing all kinds of research results through a network. Protein sequence oriented data are not an exception to this trend: from database content, to research results, people would benefit from MSA being integrated with other materials. We observed several examples of this while working on the Viseur project [2], and collaborating on GPCRDB [3].

Visualization of MSA is also important for collaborative research, and people involved in collaborations know how much improvements in efficiency can be obtained using network tools. In such cases, people need to view, comment partial results from collaborators, or create representations for their own results. As an MSA makes easier to communicate key ideas about a set of sequences, it should not be neglected by the information exchange collaborative system used to carry on the collaborative task.

This explains the need of an appropriate representation of MSA data for use through a network. Our intention is to restrict to HTTP technology [4], the common foundation to WWW and intranet technologies. With respect to our background, working with MSA and HTTP, an appropriate representation should be:

- portable: the representation should be available to the broadest audience. This means that you have to create documents that will be displayed the same way by different browsers. At least, you have to be sure that differences in rendering the document will not alter the information you publish,
- colored: to highlight class or individual properties of residues,
- possible to hyper-link at the residue level (in the Viseur program we found really valuable to have some residues hyper-linked to mutation data) and
- transferable to the user local working area (local copy of MSA on his working machine or network for fast access or data compilation).

From a server-side point of view, it requires that MSA is represented using one of the following ways:

- HTML pages: text, images,...
- Helper applications other document types: proprietary file formats,
- Java applets.

In the last part of this introduction, we describe how these features were used, or not, to represent MSA on the WWW, and how they achieve, or not, our criteria. The method section of this contribution describes two novel methods we propose to generate MSA representations that fit the constraints we described.

1.3 Current representation of alignment data on intranet or WWW

1.3.1 Text-based HTML representation

This is currently the most common representation. Alignments are included in an HTML document as simple text, using a fixed-pitch font[1]. Vertical and horizontal alignments are found, but the user cannot switch according to his preference: the choice was made by the creator of the page (sometimes, the author proposes both orientations). For example, to represent an alignment of two peptidic sequences, the piece of HTML illustrated on figure 1 can be used.

The second row of the previous table displays the representation, as rendered by your browser. To this representation, it is possible to add colors: The Alignment Color Viewer [5] is a WWW tool to create HTML alignment pages from two sequences (with a few options to change the coloring of the residues according to four sets of residues: Blue, Red, Green, Yellow). Figure 2 is an example of this service output:

Using a similar text-based HTML, it is also possible to hyper-link some residues with information that refer to them. Consider the HTML fragment shown on figure 3.

You will notice that colors and hyper-links cannot be used independently: our intention was to color one block of amino acids red, but we had (at least on our browsers) two red blocks (P/I, GFT/GFS), separated by one block the color of which depends on your browser configuration i.e. GG/GG. Hyper-links may have a higher priority than colors for your browser, so you will not see the GG pair colored red, or it could be red, until you select the link. Then its color will change to the "selected link color". In fact, there is no standard convention that describes what happens to text colors when the text is also hyper-linked. It would be bad a design to rely on a particular browser behavior for representing local properties of MSA. Thus, we find that this representation can be used to represent sequence alignments with hyper-links and without colors, or with colors but without hyper-links.

1.3.2 External viewer representation

This method was not used to represent sequence alignment. It would require an alignment viewer program to be installed on each client machine. The installation of the alignment viewer may be the source of problems because:

- installation of an external viewer, using MIME (Multipurpose Internet Mail Extensions) conventions, though not difficult, requires experience that the end-user may not have,
- an alignment viewer would have to be available, for every platform, reading the same file format. At present time, it is not clear whether such a collection of tools exists, and if it exists, will it be maintained on each platform ?

These considerations are less important for intranet publishing: in these situations, both the server and the client technology are under control. But for both WWW and intranet, external viewer usage would make difficult the integration of alignment data with other materials, or to hyper-link residues of the alignments to information[2]. For these reasons, we believe it is not an interesting MSA representation technique.

[1] Each graphical representation of characters in fixed-pitch fonts spreads over the same width on the page, whatever the character is. For example, a `fixed-pitch font`.

[2] PostScript MSA representation, for example [6], suffers from these limitations.

1.3.3 Java applet representation

CINEMA[7][8] is an applet which was developed for visualization and edition of multiple alignments. To write this visualization tool, its authors chose the Java^{tm} language , that ensures portability and broad availability for such a software, at least in theory. Colored alignments can be displayed and changed graphically. Residue hyper-linking is not supported by the last release in our possession (2.02). Currently, only a few ways exist to transmit the alignment to the applet, that require the alignment to be present on the WWW server machine in a specific format. CINEMA is widely, and freely, available to the community to create new local alignment servers.

Alignment files are not difficult to create, so that CINEMA could constitute a good MSA viewer when the following criteria are met:

- portability could be limited to java-enabled browers (this limits the audience to whom the information is provided),
- annotations are not required on the page where the alignment is (CINEMA creates a new window for each alignment where annotations cannot be included),
- complex page layout are not required (vertical alignments or small piece of alignment comparisons may require complex page layout),
- hyper-links on residues are not needed.

We were not ready to accept these limitations and decided to develop methods to render MSA using HTML[3]. These methods are presented in the remainder of this article.

2 Methods

Although the methods are illustrated for protein MSA, they are applicable to the whole MSA representations.

2.1 Single image

A simple method to generate hyper-linked HTML alignments would be to create images like on figure 4.

The DRY column is hyper-linked to information, via a standard map (client and server-side image maps). Unfortunately, when the alignment becomes larger, time transfer for images makes that method no longer useful in practice. Our illustration contains 265 positions (residues and gaps) for a size of 341406 bytes (average: 118 bytes per position).

2.2 Small set of small images

To address this performance problem, we decided to take advantage of the fact that proteins are constituted of repeated units. We generated sequence alignment HTML pages made of small colored images. The HTML fragment shown on figure 5, is constructed from this idea.

[3]This should not be taken as a criticism of CINEMA: this package was clearly designed as a highly interactive MSA *editor* and this constraint makes it somewhat unadapted to the use we present here (i.e. as an MSA *view*).

```
<PRE>
APGGGFTLLIA first sequence
IIGGGFSIILA second sequence
</PRE>
```

```
APGGGFTLLIA first sequence
IIGGGFSIILA second sequence
```

Figure 1: no colors, no hyper-links simple text alignment.

```
(1) AITALYSAVC.AVGLLGNVLV.MFGIVRYTKL.KTATNIYIFN.LALADALATS
(2) AITALYSAVC.AVGLLGNVLV.MFGIVRYTKM.KTATNIYIFN.LALADALATS
```

Figure 2: sample alignment from EBI text alignment converter.

```
<PRE>
A<FONT COLOR="red">P<A HREF="#SelectGG">GG</A>GFT</FONT>LLIA first sequence
I<FONT COLOR="red">I<A HREF="#SelectGG">GG</A>GFS</FONT>IILA second sequence
</PRE>
```

```
APGGGFTLLIA first sequence
IIGGGFSIILA second sequence
```

Figure 3: colors and hyper-links conflict together for text-based MSA representations.

Figure 4: MSA as a single GIF color image.

```
<IMG SRC="images/AA_.gif"><IMG SRC="images/AAM.gif"><IMG SRC="images/AAL.gif"><IMG
 SRC="images/AAT.gif"> first sequence<BR>
<IMG SRC="images/AAP.gif"><IMG SRC="images/AAN.gif"><IMG SRC="images/AAF.gif"><IMG
 SRC="images/AAT.gif"> second sequence<BR>
```

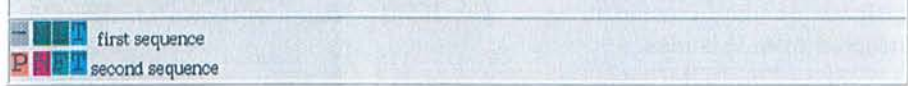


Figure 5: small set of small images MSA representation. As usually, the top part of the figure displays HTML source that is rendered in the bottom part.

Each residue is assigned a picture, that has to be accessible, from the name "images/AA?.gif". The question mark encodes a residue (?={A, C, D, E, F, G, H, I, K, L, M, N. P, Q, R, S, T, V, Y}) or a gap (?={-}).

This type of representation, to be viewed in good conditions, requires the user to have a graphical browser that renders efficiently many identical small images from its cache.

A real example is shown, which illustrates that performances are better than with a single image. Our example contains 6050 small images, which are loaded as 21000 bytes (20 residues, one gap, less than 1000 bytes each) for the images, plus the size for the HTML document itself (179033 bytes). The total is 185083 bytes (average: 31 bytes per position).

Using the simple image generation method would require a 760 kb image size: approximately four times larger.

2.2.1 Introducing hyper-links

Hyper-links can be added to this representation, in a way that is portable across browsers. To solve the portability problem due to browsers over lining a hyper-linked image with a border, one can use the BORDER tag, conjugated with explicit WIDTH and HEIGHT tags. A few examples will help understand these precautions: consider figure 6.

The border was set to 10 pixels, in order to make it clearly visible. This shows how a few hyper-linked residues can disturb the overall alignment if no precaution is taken: the border simply enlarges the residue position, and this offsets following positions by twice the border width. To correct this behavior in a portable way, it is not enough to reduce the size of hyper-linked images, by twice the border, in both dimensions. It is also necessary to use the BORDER tag, to force the browser to use the exact border value we used for the reduction.

Example on figure 7 shows an appropriate piece of HTML alignment, including hyper-links (first line). Last, this graphical representation can be proposed together with different set of small residue images to provide for user customization (images of different sizes, different color schemes, etc...). One way of doing this consist in using different image directories and to create the HTML representation on the fly, according to user's preferences.

2.3 Cascade Style sheets

Recently, Cascade Style Sheets (CSS), were partially implemented in WWW graphical browsers. CSS1 recommendations [4] are not entirely supported by any browser, at the present time. Fortunately, a subset of CSS1 is enough to represent a multiple sequence alignment, and this subset is supported by browsers mostly used by experimentalists or modeling people: InternetExplorer^{tm} 3.0+, Netscape^{tm} 4.0+.

For more information on using CSS, the interested reader is encouraged to consult the Cascading Style Sheets page [9], at W3C. Here, we are going to focus on CSS1 elements that are essential to the representation we propose. To represent the residue colors, we use the ability to change the background color of characters with the SPAN tag. This eliminates the need for colored images, which, surprisingly, are not efficiently rendered by style sheet compliant browsers.

Into the HTML HEADER, new classes are added to the SPAN HTML tag, which encode the style for each residue. The style contains:

- a restriction to mono-spaced character fonts. To ensure that each residue will be rendered the same width, to conserve the alignment,
- encoding of the foreground color,

- encoding of the background color,

You may notice definition of SPAN classes are enclosed into comments. This is a common trick to avoid non CSS1 compliant browsers to misinterpret the style definitions. In the body of the document, each residue code is preceded by the SPAN tag the class of which corresponds to the residue. Like common tags, the SPAN tag is closed when its effect is no longer required (`</SPAN>`). Please note that in the case where a few residue classes are to be distinguished, it may become interesting to factorize the `SPAN` tags, per residue class. For example: `<SPAN CLASS=A>ALILLL</SPAN>` colors contiguous aliphatic residues using the aliphatic class.

2.4 Introducing hyper-links

Figure 8 illustrates this method used with hyper-links. Here, each residue is encoded as HTML text. To hyper-link text, browsers sometimes underline the text, and always change its foreground color to a user defined color. Because we used the background to encode the property color, and because underlining, when it occurs, does not change the bounding box of characters, we are able to add hyper-links without any other precautions.

3 Results

Because the methods are near impossible to follow by hand, for real cases of alignments, we implemented three methods in the Viseur program: (i) text-based, (ii) graphical method, and (iii) style-sheet method. Because this program was developed to help the modeling of integral membrane proteins (GPCR originally), it is limited to handle MSA of protein sequences. These implementations enable us to provide a WWW service for translating MSF sequence alignments to HTML pages. This service is available free of charge, through the WWW, from our server. Here are illustrations of this service output, for the same MSA, using per method HTML alignments produced by

- text-based method,
- graphical method,
- style-sheet method.

The current release of this service does not enable to generate alignments with hyper-linked residues. But the program itself does. Should you need this feature, we suggest that you get the Viseur program. People interested in using the conversion methods intensively, or who prefer not to submit confidential alignments through the WWW, are encouraged to get and install the Viseur program, for their private use on their own machine. More information on this program, including the description of our distribution policy is available from the Viseur Home Page [2].

4 Discussion

4.1 Compression

The structure of the HTML pages produced with the methods proposed here, is highly redundant. This may be important, mostly, for three reasons.

```
<A HREF="#Image">
<IMG SRC="image.gif"
BORDER="10">
</A>
<IMG SRC="image.gif">
```

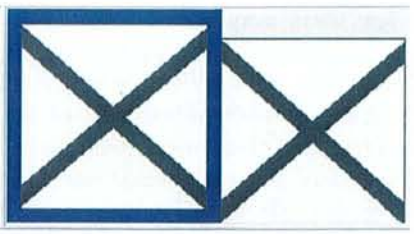


Figure 6: introducing hyper-links disturbs image alignments.

```
<IMG SRC="images/AA_.gif">
<A HREF="#EX_M" ><IMG SRC="images/AAM.gif" BORDER=2 WIDTH=11 HEIGHT=17></A>
<A HREF="#EX_L" ><IMG SRC="images/AAL.gif" BORDER=2 WIDTH=11 HEIGHT=17></A>
<IMG SRC="images/AAT.gif"> first sequence<BR>
<IMG SRC="images/AAP.gif">
<IMG SRC="images/AAN.gif">
<IMG SRC="images/AAF.gif">
<IMG SRC="images/AAT.gif"> second sequence<BR>
```

first sequence
second sequence

Figure 7: colored, hyper-linked HTML MSA.

```
<HTML>
 <HEADER>
  <TITLE>style-sheet alignment representation</TITLE>
  STYLE TYPE="text/css"> <!--
  SPAN { font-family: moonscape; color: #1c1c1c; background: #11c633 }
  SPANNER { font-family: moonscape; color: #1c1c1c; background: #4e6dff }
  SPAN { font-family: moonscape; color: #1c1c1c;  background: #11c633 }

 -->
 </HEADER><PRE>

SPAN CLASS=A><A HREF="methods.html#first_alanine">A</A></SPAN><SPAN CLASS=A>A</SPAN><BR>
SPAN CLASS=R>R</SPAN><SPAN CLASS=R>R</SPAN><BR>
SPAN CLASS=I>I</SPAN><SPAN CLASS=I>I</SPAN>

</PRE></HTML>
```

(you will not see any color in the background of residues if your browser have no support for CSS1, but we preserved alignment with <PRE></PRE> tags)

Figure 8: Cascade Style Sheet MSA representation with hyper-links.

First, if you manage a databank that stores alignment data and serves HTML representations, you may want to compress that data, in order to reduce the storage capacity needed, then to uncompress the data on request, as you send it.

Second, a few users may access your data through a modem. This hardware usually proposes compression of the data on the fly.

Third, in the future, network technology (including hardware and software) may evolve to propose compression of transmitted data, to take higher profit of the actual bandwith. We agree that this consideration is purely hypothetical, but we note that, if this evolution occurs, our representations will take advantage of it, transparently.

Because there are only about twenty residues, plus one code for gaps, choosing one, or n bytes to encode a position in an alignment will not increase the compressed size of the representation n times . Instead, according to the way compression algorithms work ([10] reviews data compression algorithms), the size will increase by, approximately, $21 \times n$.

To illustrate this, we used the Lempel-Ziv algorithm [11], as implemented in gzip (gzip is a compacter available under the terms of the GNU software license) and obtained a 93-96 % compression. This is typical of our data, and much higher that what is observed, for example, for english natural text (60-70 %). Table 1 presents the amount of data needed for each representation.

Should server-side compression of HTML pages be supported by every browser, we would be able to send compacted files through the network, and let the browser uncompact the file before it displays the content. Currently, this works solely with Netscape^{tm} browsers under UNIX. For illustration of this principle, an example is available as a compressed HTML alignment page, to be viewed on UNIX systems, only, (compressed size: 6979 bytes, average: 1.06 byte per position). For this to work, the WWW server should encode files ending with `.gz` as MIME `x-gzip` documents. Transfer time through the network becomes negligible versus decompression and rendering time (browser and machine dependent) and it is not clear whether there is a great benefit to use it in this case, except to reduce the bandwidth consumed between the server and the client.

4.2 Advantages and drawbacks

We summarize, in table 2, advantages and drawbacks of the methods we described in this article.

As suggested by this table, these two methods are partially orthogonal, with respect to the public they target. Graphical alignments are viewed best with HTML2 browsers, which still constitute, until each browser of this type is updated to a more recent release, an important part of the audience. Style sheet alignments are targeting a public that uses recent browsers. Researchers who would like to publish MSA on the WWW should propose both alternatives to visitors on their sites. Moreover, they should clearly describe what page the user should access, according to his browser.

5 Conclusion

In this article, we described two novel methods for the visualization of Multiple Sequence Alignment (MSA) using HTML. These methods construct colored alignments, possibly hyper-linked. They differ from the audience they can reach. The

[1] 31 bytes are needed to represent colors for which no symbolic name is defined. This number of bytes is required for expressions like `<FONT COLOR="#501000">A</FONT>`, that represent one colored position.

[2] 20 bytes for links to images are a minimum. Hyper-link inclusion increases this minimal value. Twenty bytes are obtained for `<IMG SRC="i/_.gif">`.

first method uses HTML2.0 language. This ensures that it produces HTML pages that will be rendered the same way by every HTML2.0 graphical browser. The second method we propose rely heavily on Cascade Style Sheets, as specified in the W3C CSS-1 recommendation. Thus, it is intended to be used with recent and future browsers. At the present time, according to the type of browsers available to users, both methods are complementary. We expect this situation to evolve gradually, until a time where the style sheet method could be used alone. Finally, we implemented these novel methods, restrained to protein sequences, in the Viseur program and provide a WWW conversion service, from protein MSA (from MSF file format) to HTML (colored text, graphic, style-sheet). The service is available from the URL http://www.lctn.u-nancy.fr/viseur/services.html.

6 Acknowledgments

Fabien Campagne would like to acknowledge the supercomputing center Centre Charles Hermite for his PhD grant, and Christophe Chipot for proofreading this manuscript.

References

[1] Rodriguez-Tome, P. The BioCatalog Alignment Editing and display. European Bioinformatic Institute, WWW: http://www.ebi.ac.uk/biocat-/biocat_ebi.html, Oct. 1993–1997, includes references to publications.

[2] Campagne, F., Bernassau, J.-M.; Maigret, B. The Viseur program. Laboratoire de Chimie théorique de Nancy, WWW: http://www.lctn.u-nancy.fr/viseur/viseur.html, 1995–1997, Viseur project home page.

[3] Horn, F., Weare, J., Beukers, M.W., Hörsch, S., Bairoch, A., Chen, W., Edvardsen, Ø., Campagne, F.; Vriend, G., GPCRDB: an information system for G protein-coupled receptors, *Nucleic Acids Res.* 1998, **1**, 275–279.

[4] Numerous authors. HTTP Specifications and Drafts. W3C, WWW: http://www.w3.org/Protocols/Specs.html, Nov. 1997.

[5] di Tommaso, Matteo. The Alignment Colour Viewer. European Bioinformatic Institute, WWW: http://www.ebi.ac.uk/htbin/visalign.pl, 1995.

[6] Barton, G.J., ALSCRIPT – A Tool to format multiple sequence alignments, *Protein Engineering* 1993, **6**, 37–40. WWW: http://barton.ebi.ac.uk/barton/servers/amas_server.html.

[7] Attwood, T.K., Payne, A.W.R., Michie, A.D.; Parry-Smith, D.J., A Colour INteractive Editor for Multiple Alignments - CINEMA, *EMBnet.news, WWW: http://ben.vub.ac.be/embnet.news/vol3_3/software.html* 1997, **3**.

[8] Attwood, T.K., Payne, A.W.R., Michie, A.D.; Parry-Smith, D.J. A Colour INteractive Editor for Multiple Alignments - CINEMA. UCL's Bioinformatics server, WWW: http://www.biochem.ucl.ac.uk/bsm/dbbrowser/CINEMA2.02/, Nov. 1997.

[9] Håkon Wium, Lie and Bos, Bert . Cascading Style Sheets page. W3C, WWW: http://www.w3.org/Style/css/, 1995-1997.

[10] Lelewer, Debra A.; Hirschberg, Daniel S. Data Compression. Information and Computer Science, University of California, Irvine, WWW: http://www.ics.uci.edu/ dan/pubs/DataCompression.html.

[11] Ziv, J.; Lempel, A. A Universal Algorithm for Sequential Data Compression. In *IEEE Trans. Inform. Theory* (1977), pp. 337–343.

	text (MSF)	colored text	simple image	colored graphics	style sheets
not compressed (bytes/position)	1	26 *or* > 31[1]	118	$>= 20$ [2]	$>= 24$
gziped (bytes/position)	1.06	2.03	118	1.06	1.18

Table 1: amount of data needed by each representation, per residue in an MSA

	graphical alignment	CSS1 alignments
can be viewed with graphical HTML2.0 browsers (Netscape^{tm} 3-, InternetExplorer^{tm} 2-, etc...)	yes	no
can be viewed with minimally compliant CSS1 browsers	yes (may be slow)	yes (display should be efficient)
colors per residue class	yes	yes
specific color on some residues	yes	yes
hyper-link support	yes	yes
special graphical symbols	yes	no

Table 2: comparison of graphical and CCS1 MSA representations

C

Publication GPCRDB

F. Horn, J. Weare, M.W. Beukers, S. Hörsch, A. Bairoch, W. Chen, Ø. Edvardsen, F. Campagne, and G. Vriend. GPCRDB: an information system for G protein-coupled receptors. *Nucleic Acids Res.*, 1:275–279, January 1998. see also `http://www.gpcr.org/7tm`.

Nucleic Acids Research, 1998, Vol. 26, No. 1 275–279

GPCRDB: an information system for G protein-coupled receptors

F. Horn, J. Weare, M. W. Beukers[1], S. Hörsch[2], A. Bairoch[3], W. Chen[4], Ø. Edvardsen[5], F. Campagne[6] and G. Vriend*

BIOcomputing, EMBL, Heidelberg, Germany, [1]LACDR, Farmacochemie, LEIDEN, The Netherlands, [2]EBI, Hinxton, Cambridge, UK, [3]Department of Medical Biochemistry, University of Geneva, Geneva, Switzerland, [4]GMD, Darmstadt, Germany, [5]Institute of Medical Biology, University of Tromsø, Norway and [6]Laboratoire de Chimie Théorique, Nancy, France

Received September 22, 1997; Revised and Accepted October 24, 1997

ABSTRACT

The GPCRDB is a G protein-coupled receptor (GPCR) database system aimed at the collection and dissemination of GPCR related data. It holds sequences, mutant data and ligand binding constants as primary (experimental) data. Computationally derived data such as multiple sequence alignments, three dimensional models, phylogenetic trees and two dimensional visualization tools are added to enhance the database's usefulness. The GPCRDB is an EU sponsored project aimed at building a generic molecular class specific database capable of dealing with highly heterogeneous data. GPCRs were chosen as test molecules because of their enormous importance for medical sciences and due to the availability of so much highly heterogeneous data. The GPCRDB is available via the WWW at http://www.gpcr.org/7tm

INTRODUCTION

G protein-coupled receptors (GPCRs) consist of a single protein chain that crosses the membrane seven times. These, presumably α-helical, transmembrane regions are probably arranged with similarity (1) to bacteriorhodopsin. Except for low resolution electron diffraction studies of frog and bovine rhodopsins (2,3) and NMR structures of fragments of loops of the bovine rhodopsin receptor (4–6) not much solid structural information is available. GPCRs are of enormous importance for the pharmaceutical industry because 52% of all existing medicines act on a GPCR (7). This explains why so many three dimensional models of GPCRs have been built. Most models are based on the atomic coordinates of the bacteriorhodopsin structure. In all of these modeling studies (bio)chemical and pharmacological data had to be used to refine the models. Clearly, mutation data and structure–affinity relationships (SARs) are essential for modeling studies aimed at the analysis of receptor–ligand interactions.

It is also possible to arrive at detailed structural knowledge using sequence analysis techniques combined with extensive data mining rather than via the path of building models. Correlated mutation studies, for example, have shown great potential for the determination of inter molecular contacts between GPCRs and their G proteins or ligands (9,10), for drug design purposes, for analyzing the role of olfactory receptors in the organization of the olfactorial nerve system (8) etc. None of these correlated mutation studies could have been performed without the availability of large amounts of experimental data.

Due to the lack of high resolution structural data, theoretical research on GPCRs relies on the availability and easy accessibility of all available data in an information system that allows for the four basic data dissemination functions: browsing, retrieval, querying and inferencing. Additionally, this information system should provide tools for data gathering, data input, data validation and data annotation. We will first discuss the data and then the four dissemination facilities (see also Fig. 1). In all cases we will discuss the present situation and the plans for the near future.

DATA

Data types introduction

The GPCRDB holds three kinds of experimental data: sequences, mutation data and ligand binding data. All three data types have their own specific problems. Sequences can be wrong, truncated, or can occur in several alternative translations. The interpretation of mutation studies and ligand bindings studies depend strongly on experimental conditions such as cell type, stable/transient transfection, density of e.g. receptors, second messenger assay, and the kinds of ligands used.

Additional data (types) can easily be incorporated in the GPCRDB. We will, however, only enter new data types if a certain degree of completeness can be reached, and if a mechanism for updating exists. We would like to add data about, for example, the preferred G protein, receptor localization, second messenger, etc., but at present it does not seem likely that a high degree of completeness can be reached for these kinds of data.

The number of experimental data types is limited, but there are no limits to the number of data types that can be derived computationally. It is therefore important to think about the questions that the GPCRDB should help answer when adding

*To whom correspondence should be addressed at: Meyerhofstrasse 1, 69117 Heidelberg, Germany. Tel: +49 6221 387473; Fax: +49 6221 387517; Email: vriend@embl-heidelberg.de

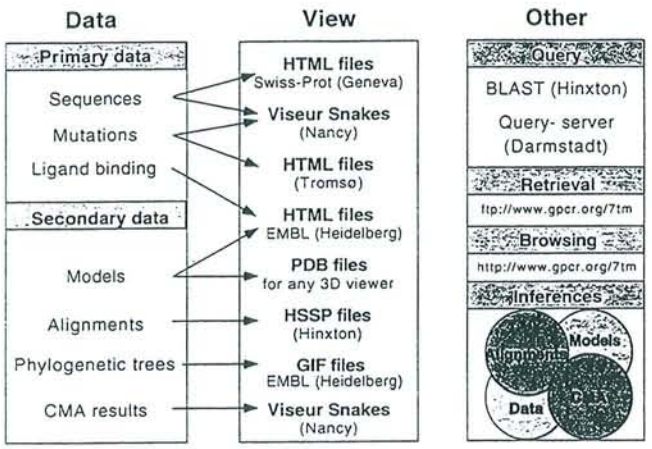


Figure 1. Schematic representation of the GPCRDB organisation. Data is shown at the left, the viewing methods in the middle and the database facilities at the right. Cities in brackets indicate the original location of the data. Arrows indicate the viewing methods available for each data type. The coloured circles represent the interplay between the data and the methods that can lead to discoveries.

more and more computational data. At present the central computational data type is multiple sequence alignments (for which a method was designed that is dedicated for GPCRs; 11). Phylogenetic trees and correlated mutation analyses (CMA; 11) are derived from these multiple sequence alignments.

Visualization of data is important for the human users of the system. In the GPCR field two dimensional representations by so-called snake like diagrams (or snakes for short) are commonly used to visually combine a sequence with other types of information such as three dimensional localization, mutation results, ligand binding or biochemical studies. In the GPCRDB there is a snake for every sequence, and these diagrams are hyperlinked to the most important related data types.

It is not a very big job to keep the computationally derived data complete, but the extraction of the underlying experimental data from literature is such an overwhelming task that in the long run it cannot be done without the help of the experimental scientists around the world. Convincing the community that there is mutual benefit for individuals to infrequently invest small amounts of time entering data might be a more daunting task than all other aspects of the GPCRDB project. Moreover, the direct input of data by the experimentalists allows the incorporation of non-published data in the GPCRDB. The inclusion of such data will prevent researchers from reinventing the wheel.

Primary data

Sequences. At present, all sequence data is extracted from the SWISS-PROT database (12). This has the advantage that we get well annotated sequences from expert database curators. The disadvantage is that the GPCRDB sequence data at present is not yet as complete and as up to date as possible. SWISS-PROT maintains lists of sequences for certain classes for which niche databases such as the GPCRDB exist. In the future the SWISS-PROT data will be augmented with the results of sequence searches in other sequence databases. The SWISS-PROT sequence files will, however, stay at the heart of the GPCRDB sequence efforts because of their high quality.

In the September 1997 release of the GPCRDB, more than 800 GPCR sequences are available, divided into five major and 151 minor sub-families.

Sequence fragments are deleterious for most computational purposes but they do hold useful information. At present (September 1997) we have stored 74 sequence fragments, extracted from SWISS-PROT. In the near future, putative GPCR ESTs (Expressed Sequence Tags) will be added.

Mutation data. The GRAP database (13) and its derivative, TinyGRAP, holds information for about 3900 point mutations. Some data on chimeric, deletion and insertion mutations is also available in the GRAP database and the planning is to include more of such mutations in the near future. This information is accessible directly (14), from the Viseur site (15) and from the GPCRDB. In the GPCRDB this data is available in several forms, the most prominent of which is hyperlinking from the snakes. Presently, GRAP focuses on class A receptors but an input form to add mutant data for all classes is available, and by the time this article goes to press, the first 100 class B mutations will have been entered. We strongly encourage the experimental scientists in the GPCR field to enter their mutation data into the GRAP database via the WWW input form available from the GPCRDB pages.

Ligand binding data. Ligand binding data was obtained from P.Seeman (16). This very impressive collection of drug dissociation constants was manually extracted from the literature. Seeman collected data for neuroreceptors and transporters. About 12 000 dissociation constants are available for 10 GPCR families. The heterogeneity of the experimental conditions make it hard to use this data for query purposes because it is not possible to determine the value of the numbers without human interpretation. However, the data is very useful for browsing purposes. The data was originally stored in tables that were sorted by receptor type, but in the near future the data will be stored internally in a data structure that will allow for alternative visualization methods such as sorting by ligand, and for hyperlinking to facilitate more natural browsing behavior by the users.

Secondary data

Multiple sequence alignments. Multiple sequence alignments are performed with WHAT IF (17) as described by Oliveira *et al.* (11). These alignments are made for whole families, but also for sub-families, sub-sub-families, etc. The alignments are presented as HSSP files (18) and as MSF files (19) so that the user can choose between two standard output formats one of which displays the sequences horizontally (MSF), and one vertically (HSSP). Although seemingly trivial, interactions with users made clear that this freedom of choice is rather important.

Throughout the GPCRDB we have used the residue numbering scheme suggested by Oliveira *et al.* (11). In this numbering scheme the residues are numbered such that 100s digits indicates the helix number, and the most conserved residue in every helix has a round number.

Alternative alignments will be incorporated if submitted to the GPCRDB.

Phylogenetic trees. Phylogenetic trees are a good visualization tool for relationships between sequences in a family. This

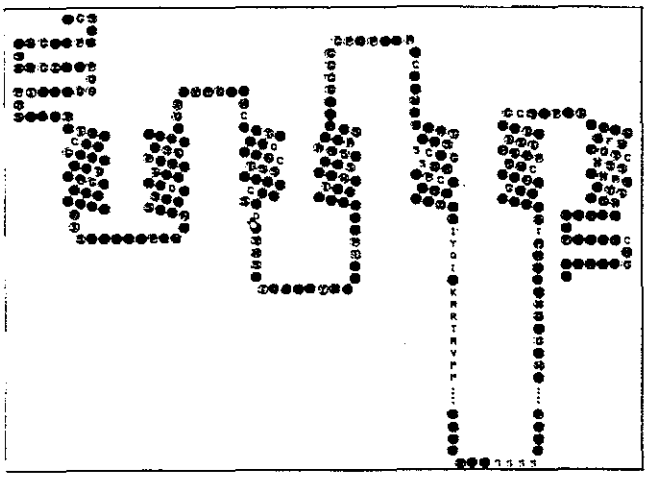


Figure 2. Snake like diagram of an adrenergic receptor. Residues are coloured based on their biochemical nature except white-coloured positions which are hyperlinked to mutant data.

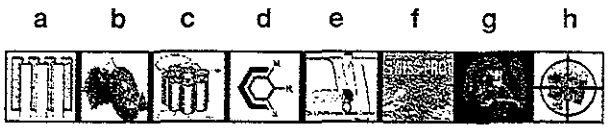


Figure 3. Logos representing the informal collaboration of GPCR databases. (a) The GPCRDB maintained at the EMBL. (b) The GRAP mutant database contains data on mutants of family A GPCRs maintained at the University of Tromsø by Ø.Edvardsen and K.Kristiansen. (c) The ORDB database holds sequences of olfactory receptors proteins. It contains public and private sections which provide tools for investigators to analyse the functions of this very large gene family of GPCRs. It is maintained at the Yale School of Medicine. (d) The Molecular Recognition Section. One of their projects is a database of GPCR related information including alignments and mutation analysis by A.M.van Rhee, NIH, Bethesda. (e) The GCRDb is maintained by F.Kolakowski at the University of Texas, Health Science Center at San Antonio. (f) The SWISS-PROT database: the annotated protein sequence database maintained by A.Bairoch at the University of Geneva, Geneva, Switzerland. (g) The Swiss-Model 7TM Interface for the modelling of the helices of 7TM receptors maintained by M.C.Peitsch at the Geneva Biomedical Research Institute, Geneva, Switzerland. (h) The Viseur program for the visualisation, management and integration of GPCR related information: maintained by F.Champagne at the Laboratoire de Chimie Théorique, Nancy, France.

information can help in answering several different kinds of questions, e.g., questions related to ligand design. Making phylogenetic trees is a whole science in itself. The WHAT IF program uses a neighbor joining algorithm. Although the resultant phylogenetic trees represent, as good as possible, the pairwise identities between the sequences rather than their evolutionary relationships, there is a striking resemblance with phylogenetic trees based on an accepted-mutation parsimony method (20). The phylogenetic trees presently available in the GPCRDB were made of manually selected representative subsets of sequences. We envisage producing these trees automatically, in parallel with the multiple sequence alignments.

Correlated mutation data. Correlated mutation analysis is a computational method to identify pairs of sequence positions that remained conserved or mutated in tandem during evolution. The idea behind the search for such pairs of residues is that when a mutation occurs at a functionally important site, the protein either becomes non-functional or may acquire its original or a different function due to a compensatory mutation at another position. Residues detected by the CMA method are often involved in intermolecular interactions (between ligands and receptors or G-proteins and receptors) (9,10). Although a detailed explanation for this phenomenon is beyond the scope of this article, it must be clear that the automatic detection of 'important' residues is a relevant aspect of the GPCRDB effort.

Snake like diagrams. Experimentalists in the GPCR field prefer to represent their data using snakes (Fig. 2). The Viseur (15) program can automatically generate snakes and hyperlink them to other types of information. Snakes are used in the GPCRDB to represent two kinds of data. One set of snakes is hyperlinked to the TinyGRAP mutant database. The second set is used to indicate the location of residues detected in the CMA analyses. This second set is hyperlinked to the corresponding HSSP alignment files.

Three dimensional models. The GPCRDB server holds atomic coordinates of 3D models of GPCRs. Different modelers used different alignments and different modeling techniques to build these models, and consequently a wide variety of models have been proposed. As it is at present not possible to decide which models are right and which are wrong, we have decided to store every suggested model in the GPCRDB. The models are grouped per depositor. For each model one can either download the coordinates or view them using a WWW helper application like Rasmol (21).

Other data

The GPCRDB additionally contains pointers to other GPCR related databases (Fig. 3). General information about GPCRs, articles about the GPCRDB or elements of its contents, lists of GPCR specialists addresses, pointers to external pages of different levels of relevance for GPCR research (information resources, articles, group pages, GPCR related diseases, etc.) are available. The SWISS-PROT files allow for navigation to other databases like Medline (22), OMIM (23), EMBL (24), PIR (25), Prosite (26) and several organism specific databases such as FlyBase (27) for *Drosophila* sequences. The mutation data in GRAP is hyperlinked to Medline and OMIM.

DISSEMINATION FACILITIES

GPCRDB has been conceived to provide fast and easy access to all information related to GPCRs. It should be an information tool that makes it easier for the user to think about GPCRs, and it should make suggestions for future research. For this purposes we have implemented, (and are still implementing) the four basic information system tools: browsing, retrieval, query and inferencing.

Browsing

The GPCRDB organization is based on the pharmacological classification of receptors and access to the data is obtained via a hierarchical list of known families in agreement with this classification. For one specific family, one can access the individual sequences, the multiple alignments, the profile used to perform the latter, the snakes and a phylogenetic tree. Each type of data is displayed in a WWW page with hyperlinks to other data

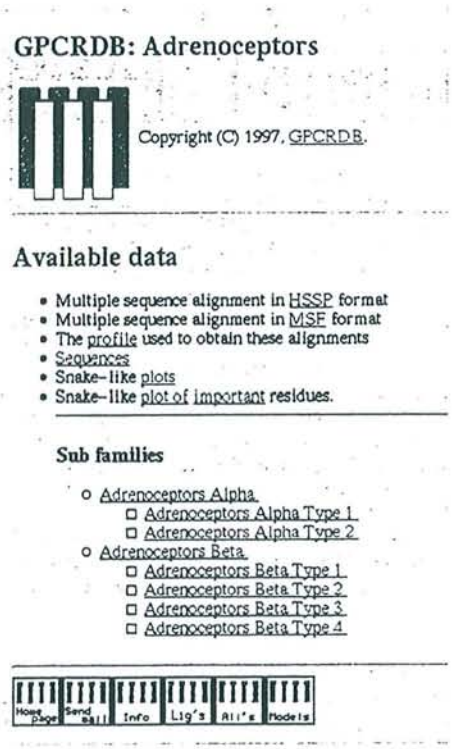


Figure 4. Example of one of the 151 subfamilies specific pages. All underlined words and icons are hyperlinked for navigation through the GPCRDB.

where appropriate. Figure 4 shows the WWW page for the adrenergic receptor class as an example of the data organization.

Retrieval

Often a user wants to work on certain data at home, independent of the GPCRDB environment. Therefore most data can be retrieved in its native form using the 'save as' option of the WWW browsers or via anonymous FTP from www.gpcr.org, data being stored in the /7tm directory.

Query

A BLAST server allows the user to scan one GPCR sequence, or the fragment of it against all GPCRDB sequences. This can give for new sequences an impression about the family they belong too.

A query system is under development. In due time it will answer complicated questions such as, for example: 'are there any mutant or CMA data for position 340 of the α2 adrenergic receptors?' The current version supports simple sequence queries by specifying query conditions on specific fields using pattern matching by means of regular expressions that can be freely combined using the logical connectors 'and, 'or' and 'not'. It will be soon possible to combine these typical database queries with less exact ones such as the sequence similarity defined by FASTA. This query facility is accessed via a WWW interface that performs conversion of the users input to the query interface of the underlying database system and a formatted presentation of query results.

Inferences

Lacking high resolution structure data, correlated mutation analyses are the most powerful tool available to date for the computational discovery of novel facts about GPCRs. The CMA clearly provide a powerful inference engine. At the cost of little CPU time, the important residue positions are selected from among the tens of thousands of residues in each alignment. So far we only calculate these residues and make them available for browsing purposes by displaying them as snakes with the appropriate hyperlinks. We will try to combine this information with experimental data (i.e. mutation data and ligand binding data) to strengthen the conclusions that can be drawn and to make it easier for the GPCRDB users to investigate their relevance.

DISCUSSION

The GPCRDB is now officially one year old. In this first year we have created a computer infrastructure that flexibly allows for extension with new data types, experimental as well as computational. Many improvements can still be made. These improvements will be directed along four lines: (i) addition of new experimental data types (ligand information, preferred G protein, cellular localization, secondary messenger, disease pattern, chimera data); (ii) addition of new theoretical data types (docked ligands, codon usage); (iii) addition of hyperlinks to other databases (mouse knock-out database, genetic diseases, etc.); (iv) user-friendliness. It is envisaged that the GPCRDB will function in the same way as an encyclopedia. One opens it with the aim of answering a question, but multiple good ideas later one has forgotten what the original question was. In other words, the GPCRDB should be more than intuitive—suggestive.

The most complicated aspect of the GPCRDB effort is the actual incorporation of data. The vast majority of all useful data is stored in articles rather than in computer readable form. We are providing tools to enter data into the database, but convincing experimentalists that it is in their own interest to participate with their data for usage by the community is another important aspect.

USAGE AND AVAILABILITY

The GPCRDB is accessible from http://www.gpcr.org/7tm . The underlying data files (alignments, models, etc.) can be downloaded by anonymous FTP from www.gpcr.org/7tm. Access to the GPCRDB is free for academic and industrial scientists. Industries can download the entire GPCRDB for in-house usage from the same FTP site. In the first year of its existence the GPCRDB has been 'visited' on average 250 times per week.

ACKNOWLEDGEMENTS

The GPCRDB is an EU sponsored project (PL 950224). We thank K.Aberer, R.Bywater, G.Casari, U.Gsbel, R.W.W.Hooft, F.Kolakowski, K.Kristiansen, W.Kuipers, L.Oliveira, C.Sander, F.Rippmann, A.Valencia, E.M.van der Wenden and A.P.IJzerman for stimulating discussions. F.C. acknowledges Centre Charles Hermite and Sanofi Recherche for support. G.V. thanks the BMFT for support to the RELIWE project. We also thank Novo Nordisk Copenhagen Denmark, Solvay Weesp the Netherlands, E. Merck Darmstadt Germany and IRBM Rome Italy for testing aspects of the GPCRDB.

REFERENCES

1 Baldwin,J.M. (1993) *EMBO J.*, **12**, 1693–1703.
2 Unger,V.M. and Schertler,G.F.X. (1995) *Biophys. J.*, **68**, 1776–1786.
3 Schertler,G.F.X., Villa,C. and Henderson,R. (1993) *Nature*, **362**, 770–772.
4 Yeagle,P.L., Alderfer,J.L., Salloum,A.C., Ali,L. and Albert,A.D. (1997) *Biochemistry*, **36**, 3864–3869.
5 Yeagle,P.L., Alderfer,J.L. and Albert,A.D. (1996) *Mol. Vis.*, **2**, 12.
6 Yeagle,P.L., Alderfer,J.L. and Albert,A.D. (1995) *Biochemistry*, **34**, 14621–14625.
7 Drews,J. (1996) *Nature Biotechnol.*, **14**, 1516–1518.
8 Singer,M.S., Oliveira,L., Vriend,G. and Shepherd,G.M. (1995) *Receptors Channels*, **3**, 89–95.
9 Kuipers,W., Oliveira,L., Paiva,A.C.M., Rippman,F., Sander,C. and IJzerman,A.P. (1996) In Findlay,J. (ed.), *Membrane Protein Models*. Bios Scientific Publishers Ltd, Oxford, pp. 27–45.
10 Oliveira,L., Paiva,A.C.M. and Vriend,G. (1995) In Kazonaya,P.T.P. and Hodges,R.S. (eds), *Peptides: Chemistry, Structure and Biology*. Mayflower Scientific Ltd, Kingswinford, UK, pp. 408–409.
11 Oliveira,L., Paiva,A.C.M. and Vriend,G. (1993) *J. Comp.-Aid. Mol. Des.*, **7**, 649–658.
12 Bairoch,A. and Apweiler,R. (1997) *Nucleic Acids Res.*, **25**, 31–36. [see also this issue (1998) *Nucleic Acids Res.* **26**, 38–42].
13 Kristiansen,K., Dahl,S.G. and Edvardsen,Ø. (1996) *Proteins: Struct. Funct. Genet.*, **26**, 81–94.
14 See http://www-grap.fagmed.uit.no/GRAP/homepage.html
15 See http://www.lctn.u-nancy.fr/viseur/viseur.html
16 Seeman,P. (1993) *Receptor Tables, vol.2: Drug Dissociation Constants for Neuroreceptors and Transporters*. SZ Research, Toronto.
17 Vriend,G. (1990) *J. Mol. Graph.*, **8**, 52–56.
18 Sander,C. and Schneider,R. (1991) *Proteins*, **9**, 56–68.
19 Devereux,J. (1989) *The GCG Sequence Analysis Software Package, Version 6.0*. Genetics Computer Group, University of Wisconsin Biotechnology Center, 1710 University Avenue, Madison, Wisconsin, USA, 53705.
20 Kolakowski,L.F. Jr (1994) *Receptors Channels*, **2**, 1–7.
21 Sayle,R. and Milner White,E.J. (1995) *Trends Biochem. Sci.*, **20**, 374–376.
22 See http://www4.ncbi.nlm.nih.gov/PubMed
23 On-line Mendelian Inheritance in Man (OMIM), a catalog of human genes and genetic disorders. McKusick,V.A. *et al.* Johns Hopkins University. See http://www3.ncbi.nlm.nih.gov/omim
24 Stoesser,G., Sterk,P., Tuli,M.A., Stoehr,P.J. and Cameron,G.N. (1997) *Nucleic Acids Res.*, **25**, 7–13 [see also this issue (1998) *Nucleic Acids Res.* **26**, 8–15].
25 George,D.G., Dodson,R.J., Garavelli,J.S., Haft,D.H., Hunt,L.T., Marzec,C.R., Orcutt,B.C., Sidman,K.E., Srinivasarao,G.Y., Yeh,L.S.L., *et al.* (1997) *Nucleic Acids Res.*, **25**, 24–27 [see also this issue (1998) *Nucleic Acids Res.* **26**, 27–32].
26 Bairoch,A., Bucher,P. and Hofmann,K. (1997) *Nucleic Acids Res.*, **25**, 217–221.
27 The FlyBase consortium (1997) *Nucleic Acids Res.*, **25**, 63–66 [see also this issue (1998) *Nucleic Acids Res.* **26**, 85–88].

Deuxième partie

A la recherche de la réutilisabilité

peek et poke sont perplexes...

Les logiciels de Chimie informatique sont complexes, donc difficiles à modifier ou à faire évoluer. Ils sont souvent construits, sans méthode particulière,[15] pour répondre à un besoin à court terme (tester et valider une nouvelle méthode du domaine d'application). Cette situation étaient observée il y a une vingtaine d'années dans tous les domaines où l'informatique était utile, avant la mise au point de méthodes de conception de logiciels. L'adoption de méthodes de programmation modulaire [YC79] puis de la programmation par objet [Mey97] pour ne citer que celles-ci a permis à l'industrie informatique d'améliorer sensiblement la qualité des logiciels produits, d'augmenter la complexité des outils, donc des services proposés aux consommateurs.

Un nombre de personnes croissant réalise que l'adoption des méthodes de développement qui ont changé l'informatique généraliste pourrait produire les mêmes résultats appliquées au développement de logiciels de Chimie informatique. Essentiellement,

- diminution du nombre de défaut du logiciel par réutilisation de parties déjà testées et meilleure compréhension du logiciel par les développeurs;
- accélération du cycle de développement : diminution du délai entre le développement d'un outil et son utilisation;
- plus grande flexibilité pour essayer de nouvelles approches.

La création de la liste d'échange «Standardization of computational chemistry software», à la suite du 35^{eme} symposium de Sanibel (1995) [Deu98] confirme que le problème a été remarqué depuis plusieurs années, mais que des solutions restent à être trouvées (la standardisation des logiciels n'étant pas obligatoirement l'approche la plus adaptée).

On observe quelques initiatives où l'approche de ce problème consiste à réécrire complètement un logiciel («sur de nouvelles bases»). Par exemple, [SWI+98] décrit un logiciel de chimie quantique réécrit pour être plus facile à modifier et permettre des calculs sur les actinides. Dans le domaine de la dynamique moléculaire, citons les outils DL_POLY [SF98] et DDGMQ [BCOY93], [BMM97], réécrits pour tirer parti des architectures parallèles.

Mais ces initiatives restent limitées par la quantité de travail requise par la réécriture d'un logiciel et par le manque d'expérience des auteurs des techniques de réutilisation.

Le chapitre suivant décrit les méthodes actuelles de recherche conformationnelle. Les outils logiciels qui implantent ces méthodes nous semblent de bons exemples pour montrer comment certaines techniques de réutilisation permettent de tirer parti des logiciels du patrimoine[16] de la Chimie informatique.

15. Il est question ici de méthode de développement seulement.

16. Le terme anglais, souvent utilisé dans la littérature qui discute de réutilisation est *legacy software*.

3

Introduction à la recherche conformationnelle

La recherche conformationnelle est l'activité qui consiste à déterminer par la simulation les conformations stables d'une molécule (à l'équilibre thermodynamique). Les pages qui suivent ont pour but de présenter les méthodes utilisées par la recherche conformationnelle. Cette présentation permettra de mieux comprendre les constats qui parlent pour la standardisation des logiciels de la Chimie informatique. Nous présenterons, dans le prochain chapitre, les composantes d'un outil de recherche conformationnelle de manière plus détaillée.

3.1 Objectifs

Bien qu'il existe des méthodes expérimentales de détermination de structures moléculaires, leur utilisation n'est pas possible dans les projets qui tentent de concevoir une molécule adaptée à une fonction particulière (fonction biologique, physico-chimique, etc.) : la molécule n'existe pas aux premières étapes de sa conception et donc, les méthodes expérimentales ne sont d'aucune aide pour tenter d'évaluer sa fonction. Comment évaluer la fonction d'une petite molécule à partir des types de ses atomes et de leurs connectivités ? La forme d'une molécule détermine en grande partie sa fonction, quelque soient les propriétés biologiques ou physiques considérées. C'est la raison pour laquelle beaucoup d'efforts sont dirigés vers la mise au point de méthodes de recherche conformationnelles fiables et rapides.

3.2 De nombreux types de méthodes

Les méthodes de recherche conformationnelle peuvent être regroupées en deux catégories. Les méthodes de la première catégorie tentent de simuler l'évolution d'un système physique par dynamique moléculaire, alors que les méthodes de la seconde catégorie tentent d'atteindre les configurations d'énergie minimum sans postuler de règles d'évolution du système.

3.2.1 Les méthodes à base de dynamique moléculaire

Les méthodes de la dynamique moléculaire consistent à considérer une molécule comme un ensemble de points matériels aux centres des atomes et soumis aux lois de la mécanique classique. Les atomes de la molécule sont considérés comme soumis à l'action d'un champ de forces, déterminé par dérivation de l'énergie de la molécule par rapport aux coordonnées des atomes. Par abus de langage, on appelle champ de forces, d'une part les expressions analytiques qui permettent de calculer l'énergie d'un fragment de molécule à partir des positions atomiques, d'autre part les paramètres de ces expressions en fonction des atomes considérés. L'énergie totale de la molécule permet d'accéder aux forces

sur chacun des atomes et l'intégration numérique des équations du mouvement, par rapport au temps, permet d'obtenir les positions atomiques au moment $t + \delta t$. La donnée de tous les conformères de la molécule considérée constitue ce que l'on appelle la trajectoire de dynamique, par analogie avec la trajectoire d'un mobile.

Les méthodes de recherche conformationnelles basées sur la dynamique moléculaire calculent un grand nombre de conformations (trajectoire de dynamique suffisamment longue), parmi lesquelles certaines sont conservées pour une minimisation finale.[17] Cette méthode présente quelques inconvénients :

- elle explore préferentiellement l'espace conformationnel local,
- elle montre une sensibilité importante à la conformation de départ, qui doit être physiquement correcte (une structure non physique contient des interactions internes si fortes que la structure se dissocie sous leur effet),
- elle est sensible à la durée de la simulation et au nombre de trajectoires calculées,
- les algorithmes de dynamique moléculaire sont intrinsèquement séquentiels (très mauvais potentiel de parallélisation),

Ces inconvénients limitent son application automatique à partir de structures de faible qualité (comme celles construites directement à partir de la topologie moléculaire).

3.2.2 Les méthodes de recherche conformationnelle directes

La seconde catégorie regroupe toutes les méthodes de recherche conformationnelle directe. Ces méthodes présentent en général l'avantage de fonctionner avec des structures initiales de très mauvaise qualité (système non physique) et de produire un échantillonnage de l'espace conformationnel probablement plus distribué pour un petit nombre de conformations calculées que ne le permet la dynamique moléculaire (voire la figure 3.1 pour une illustration).

Recherche exhaustive

Une approche très simple de l'analyse conformationnelle, par exemple par arbre de recherche en coordonnées internes [LS88] consiste à évaluer l'énergie de chaque conformère d'un ensemble obtenu en faisant varier les angles dihèdres par pas de n degrés (par exemple $n = 60\,\mathrm{deg}$). Cet algorithme est inutilisable sur des molécules relativement simples du fait de sa complexité exponentielle par rapport au nombre de dihèdres. La recherche exhaustive est aujourd'hui abandonnée pour les systèmes moléculaire au profit des méthodes stochastiques (juste une partie de l'espace conformationnel est visité, sur la base de l'algorithme et du hasard).

Recherche stochastique

Ces méthodes ne garantissent pas que le minimum de l'énergie sera atteint, mais permettent, en pratique, d'obtenir des conformères de faible énergie dans un temps raisonnable.

La plupart des algorithmes stochastiques ont été appliqués au problème de la recherche conformationnelle. Il faut citer l'utilisation des algorithmes de *Monte Carlo* de type Metropolis [MpRR+53] (coordonnées cartésiennes [Sau87], coordonnées internes [CGS89]), de *recuit simulé* [KGV83], *recuit simulé adaptatif* [Ing89] [WP96].

Les méthodes de type *distance geometry* [CH88] se différencient des méthodes précédentes. Elles représentent la molécule comme un ensemble de valeurs limites aux distances entre atomes ou groupes d'atomes (limites supérieures et inférieures, dépendant du système chimique: rayons de van der Walls, contraintes de chiralité, etc.). Un choix est réalisé au hasard à l'intérieur des limites pour chacune des distances puis ces contraintes n-dimensionelles sont plongées dans un espace à trois dimensions, où

17. La minimisation de l'énergie d'une molécule est l'opération qui consiste à minimiser l'énergie de la molécule considérée comme une fonction des paramètres de la conformation. La minimisation aboutit généralement à un minimum local de la fonction énergie. La sélection des conformation retenues pour minimisation est souvent réalisée au hasard, en retenant un conformère tous les n calculés.

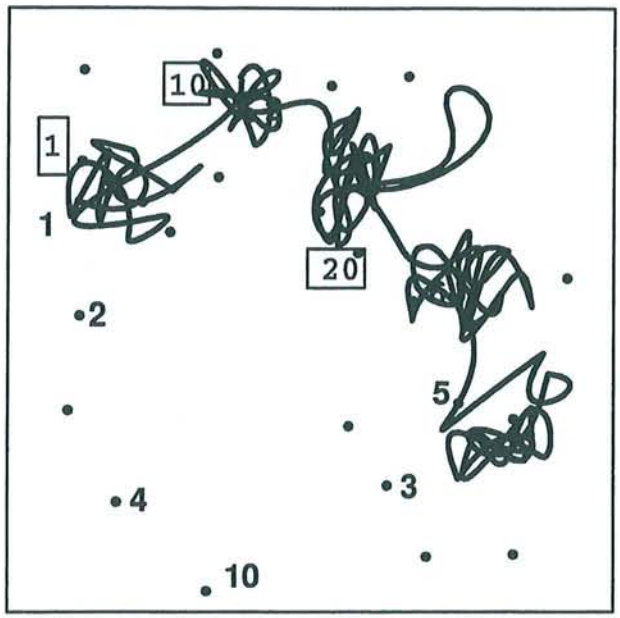


FIG. 3.1 – *Représentation schématique de la couverture de l'espace conformationnel par les méthodes de recherche conformationnelle*
L'espace conformationnel est représenté par la surface enfermée dans le carré ci-dessus (un élément de surface représente une conformation). Trajectoire continue : ordre d'obtention de la conformation encadré; recherche conformationnelle directe : numérotation simple. Illustration de la distribution de l'échantillonnage conformationnel.

l'énergie de la conformation est minimisée. Se reporter à [SWBB97] et références incluses pour une description des méthodes de *distance geometry* appliquées à la recherche conformationnelle.

Autres algorithmes

D'autres algorithmes d'optimisation ont été appliqués au problème de la recherche conformationnelle. Il faut citer les *algorithmes génétiques* [Hol75] [DJ92] [JJMP93] qui cherchent à mimer l'évolution des gènes. Dans ce type d'approche l'information à optimiser est encodée sous la forme d'un "chromosome" qui subit des mutations ponctuelles et des recombinaisons. L'information décodée (exprimée) est évaluée par rapport au critère de l'optimisation et le chromosome d'origine conservé ou rejeté par comparaison avec les performances d'autres chromosomes.

Les approches à base de systèmes experts [DLK87] s'écartent de l'approche physique puisque l'énergie du système n'est plus considérée. L'élaboration d'un de ces systèmes consiste à déterminer des règles qui permettraient de choisir les conformations les plus stables dans l'espace conformationnel. Les difficultés rencontrées pendant l'élaboration de ces systèmes sont *(i)* déterminer les règles à appliquer, *(ii)* d'exprimer ces règles dans le cadre du formalisme retenu.

Méthodes et outils

Chacune des publications citées ici décrit une méthode, mais surtout un programme d'analyse conformationnelle — un outil —, puisque les performances des méthodes sont mesurées par comparaison des performances des outils qui les implantent. Il est donc difficile d'évoquer une méthode en passant sous silence le développement logiciel inhérent à sa mise au point. Pourtant, les publications que nous avons pu parcourir donnent peu de détail sur les implantations.

Le parcours de la littérature dédiée aux méthodes de recherche conformationnelle suggère que

- les méthodes de recherche conformationnelle n'ont pas encore atteint leur maturité. Cheuk-San Wang [Wan97] décrit un algorithme de recherche conformationnelle dédié aux molécules macrocycliques environ dix fois plus rapide que les algorithmes précédents (sa méthode introduit des opérateurs de passage d'un conformère à l'autre bien adaptés aux systèmes macrocycliques),
- les outils de recherche conformationnelle directe sont développés à partir de codes différents mais dont les fonctions sont très semblables (entrée-sortie, calcul de l'énergie, moteur d'optimisation, conversion molécule vers paramètres).

Ce dernier point indique que ces outils peuvent-être développés à partir d'un petit ensemble de composants logiciels réutilisables, de façon à réduire le nombre de modifications à apporter au logiciel pour tester l'intérêt d'une nouvelle méthode. Nous avons donc décidé d'explorer les moyens de faciliter la réutilisation dans le cadre du développement d'outils de recherche conformationnelle. Le prochain chapitre va plus loin dans la présentation des parties d'un programme d'analyse conformationnel réutilisables.

4

Analyse d'un outil de recherche conformationnelle directe

L'outil de recherche conformationnelle que nous présentons ici utilise l'algorithme de recuit simulé adaptatif (précédemment évalué pour les peptides [WP96]). Ce chapitre décrit le rôle de chacun de ses constituants.

4.1 Entrées/sorties

Dans les systèmes d'exploitations actuels, la fonction de persistance[18] des données est habituellement délivrée par les systèmes de gestion de fichiers. L'utilisateur a appris à « sauver un fichier » correspondant à la partie de l'information qu'il veut conserver avant d'arrêter le programme qui la manipule.

Une partie du programme de recherche conformationnelle doit donc réaliser la lecture ou l'écriture des conformations dans des fichiers. Le nombre de formats de fichiers qui contiennent des informations moléculaires (topologie et/ou conformations) est supérieur à soixante dix. Ce décompte est obtenu en considérant les formats de conversion supportés par l'outil Babel [SWSD94], le bien nommé. Ces 70 formats de fichiers différents ont été mis au point pour répondre aux besoins particuliers de presque autant de logiciels de calculs, de visualisation, ou de présentation d'informations moléculaires.

Babel a été développé suite au constat que la réalisation d'une tâche implique très souvent l'utilisation de plusieurs outils dont les formats de stockage sont incompatibles.

Définition	**Inter-opérabilité**
L'inter-opérabilité est la capacité de plusieurs outils de fonctionner ensemble. Voir [SK93] pour un des premiers usages du terme "inter-opérable" et [Bou96] pour une description d'architectures visant à favoriser l'inter-opérabilité d'outils.	

On peut dire que Babel présente une solution au problème d'inter-opérabilité entre les logiciels de Chimie informatique. Cette solution consiste à utiliser le programme Babel pour convertir l'information produite par un outil (stockée sur disque dans le format dédié à cet outil) en information de forme utilisable par l'outil suivant dans la chaîne de traitement. C'est une solution peu optimale puisqu'elle demande l'intervention de l'utilisateur qui doit demander explicitement la conversion d'un format d'information à un autre. Nous préférons croire que l'intervention de l'utilisateur doit être limitée aux étapes du travail qui ne peuvent pas être automatisées de manière satisfaisante.

18. Persistance des données pendant les périodes de non traitement. Par exemple pendant les moments où le programme qui les utilise n'est pas actif.

Beaucoup d'outils d'analyse conformationnelle ne peuvent lire la topologie de la molécule dont ils vont chercher l'espace conformationnel que dans un seul format de fichier. Ce format doit contenir :

- la topologie moléculaire,
- la conformation utilisée pour commencer la recherche (parfois optionnel),
- les types des atomes pour le calcul de l'énergie d'une conformation,
- les charges partielles atomiques.

Nous avons choisi une approche différente : le programme que nous avons développé à l'aide de méthodes de réutilisation repose sur le logiciel Babel pour lire ou écrire topologie et structure des molécules. Nous n'avons réécrit aucune routine d'entrée sortie : nous avons réutilisé les quelques 70 fonctions précédemment développées puis testées, par les auteurs et les utilisateurs de Babel. Le développeur bénéficie de la réutilisation du logiciel : l'effort investi dans l'apprentissage du fonctionnement de la bibliothèque Babel est à peu près équivalent à l'effort d'écriture — et de tests — de routines d'entrées sorties dédiées. La différence est que la connaissance du fonctionnement de la bibliothèque alors acquise peut-être réutilisée dans le cadre d'autres projets.

Notre programme d'analyse conformationnelle peut donc lire ou écrire des conformations dans tous les formats supportés par Babel. L'utilisateur bénéficie aussi de la réutilisation du logiciel.

4.2 Évaluation de l'énergie d'une conformation

La recherche conformationnelle par l'algorithme de recuit simulé adaptatif impose que l'on puisse calculer l'énergie d'une conformation. C'est à dire, que l'on sache évaluer la fonction $E = f(T, X, C, P)$ où T est la topologie du système (connaissance des liaisons entre atomes), X les positions cartésiennes des atomes, C leur charges, et P l'ensemble des paramètres du champ de force.

Pour une molécule donnée, T et P sont des termes constants, et C peut-être considéré quasi-constant (en première approximation, valide dans la limite des modèles de modélisation moléculaire).

On doit donc évaluer $E = g(X)$ où g est une fonction de T, C, et P.

Comme la construction de la fonction g est une opération coûteuse, la plupart des logiciels de calcul d'énergie la réalisent une fois pour toute à l'initialisation de la conformation. Construire g consiste à associer les paramètres du champ de force à la conformation de façon à ce que le calcul $g(X)$ soit le plus rapide possible.

Nous avons réutilisé des procédures de calcul d'énergie conformationnelle développées précédemment au laboratoire pour un logiciel de minimisation, programmé en langage Fortran 90.

4.3 Optimisation de paramètres

Une procédure d'optimisation détermine les paramètres Z tels que un critère c, calculé à partir de Z, soit minimum (ou maximum). C'est à dire que l'optimisation doit déterminer l'ensemble des paramètres $(z_1, z_2, \ldots, z_n)$ qui minimisent (ou maximisent) $c(z_1, z_2, \ldots, z_n)$, quelque soit la fonction c. C'est un problème plus complexe que la minimisation puisque l'optimisation cherche à atteindre le minimum global du critère, sans connaissance, a priori, de la fonction c.

L'algorithme de recuit simulé adaptatif que nous avons utilisé est une procedure d'optimisation qui a été développées à partir de considérations physiques. Nous ne décrirons pas cette méthode particulière puisque nous nous intéressons aux méthodes d'optimisation dans leur généralité.

Dans le cas de la recherche conformationnelle que nous décrivons, les paramètres Z sont les coordonnées cartésiennes des atomes ou les coordonnées internes de la molécule. Le critère est l'énergie de la conformation. Cette mise en correspondance des paramètres d'optimisation et des paramètres qui déterminent une conformation joue un grand rôle dans l'efficacité de la méthode de recherche.

Essayons d'illustrer pourquoi. Nous avons vu que la recherche exhaustive n'est pas possible, même pour des molécules de taille modeste. Il s'en suit que seule une partie de l'espace conformationnel peut être explorée pendant la simulation. Il faut donc trouver une paramétrisation de la conformation qui favorise la description des conformations physiques (au sens du chimiste) au détriment des conformations

théoriques (un ensemble de paramètres qui ne corresponde pas à un conformère, par exemple lorsque les coordonnées de deux atomes imposent à leurs sphères de van der Walls de trop se chevaucher).

Une autre façon d'expliquer ce principe est de dire qu'il faut favoriser les jeux de paramètres tels que le rapport $r = \frac{card(P^{physique})}{card(P)}$ soit le plus près possible de 1. Avec $P^{physique}$ l'ensemble des n-uplets de paramètres qui codent pour une conformation physique et P l'ensemble total des n-uplets possibles, en considérant seulement les valeurs numériques accessibles aux n paramètres. On augmente ainsi la probabilité de choisir (c'est l'algorithme d'optimisation "qui choisit", parmi P) des valeurs de paramètres correspondant à des conformations ayant un sens physique.

Il est préférable aussi, puisque le nombre total d'évaluations d'énergie est limité, que le changement continu d'un paramètre transforme la molécule en des conformations toujours physiques, au fur et à mesure des changements. C'est le cas dans les méthodes à bases de dynamique moléculaire. Les méthodes directes devraient essayer de conserver cette propriété tout en améliorant la couverture de la recherche. Examinons ceci plus en détail.

4.4 Opérateurs de passage entre conformères et paramétrisation

La dynamique moléculaire détermine le passage d'un conformère à l'autre par l'intégration des équations du mouvement. Le nouveau conformère est celui obtenu au temps $t + 1$, dans le cadre du modèle de la dynamique moléculaire.

Considérer un opérateur de passage d'un conformère à l'autre, pendant la recherche conformationnelle, est simplement un moyen de généraliser une caractéristique des différents algorithmes de recherche que nous avons décrits. Nous verrons que cette généralisation, de plusieurs algorithmes différents à travers un cadre conceptuel unique, est une caractéristique importante des approches de réutilisation. Nous pensons que c'est aussi une façon efficace d'expliquer les points communs et les particularités de ces différents algorithmes.

Pour une optimisation type Monte-Carlo en coordonnées cartésienne, l'opérateur de passage qui agit sur une conformation, ou de façon équivalente sur un n-uplet de paramètres (ici $3n$ paramètres lorsque la molécule possède n atomes) peut s'écrire $p(c_i)$, ou encore $p((z_{i,1}, z_{i,2}, \ldots, z_{i,3n}))$. $z_{i,1}$ est la coordonnée x du premier atome, $z_{i,2}$ sa coordonnée y, $z_{i,3}$ sa coordonnée z; $z_{i,4}$ la coordonnée x du deuxième atome, etc.

L'opérateur p combine l'information contenue dans les paramètres avec des connaissances supplémentaires (intensité de la perturbation à appliquer à chaque paramètre, dureté des paramètres ou sélection de paramètres sur laquelle appliquer les perturbations) qui sont dépendantes de l'algorithme et du problème d'optimisation. L'opérateur p peut donc contenir la mémoire de l'apprentissage d'un algorithme d'optimisation qui en disposerait.

Il faut noter néanmoins qu'une re-paramétrisation du problème permet souvent d'intervenir de façon satisfaisante au niveau de l'opérateur de passage, sans bouleverser la structure des logiciels existants. Nous verrons comment dans le chapitre 6.

5

Techniques de réutilisation

Les techniques de réutilisation sont étudiées dans le cadre du génie logiciel mais elles peuvent aller au delà de considérations purement informatiques. Nous allons décrire ici quelques-unes des techniques de réutilisation disponibles aux adeptes de la programmation par objets. Nous ne parlerons presque pas de conception orientée-objet, en tout cas pas plus que nécessaire pour accompagner à la compréhension des travaux que nous présentons ici, aussi le lecteur intéressé est-il invité à se reporter à quelques ouvrages didactiques sur le sujet [MO92], [Boo93], [Mey97].

Le rapport sur les meilleures pratiques de réutilisation logicielle [AE95] commandé par le Département de la Défense des États-Unis d'Amérique décrit les motivations qui poussent les organisations dans la voie de la réutilisation et les *meilleures pratiques observées* tant dans l'industrie que dans les institutions gouvernementales.

En résumé, il ressort de ce rapport que les organisations qui mettent en place des cadres de travail qui favorisent la réutilisation des logiciels qu'elles développent, bénéficient après un temps relativement faible d'un retour sur investissement conséquent, et surtout de cycles de développement réduits. Elles sont plus à même de s'adapter aux demandes changeantes du marché et produisent des logiciels de meilleure qualité.

Il est tentant de se demander si ces avantages ne pourraient pas être obtenus dans le cadre des développements de logiciels scientifiques, en dehors de toute organisation autre que celle d'un laboratoire ou d'un ensemble de laboratoires réunis par un projet commun. Pour répondre à cette question, il est d'abord nécessaire d'examiner ce qui entre en jeu dans ce que l'on peut appeler *les techniques de réutilisation*.

5.1 Un jeu de construction

Le jeu de construction pourrait être l'idéal de la construction logicielle.[19]

D'une part, un jeu de construction bien conçu (nous reviendrons sur ce point) permet d'assembler des composants pour construire virtuellement n'importe quel ouvrage que l'enfant peut imaginer. Le jeu de construction est souple.

D'autre part, le jeu de construction simplifie considérablement l'activité de construction. Il n'est pas question de souder, de coller, de découper des pièces pour les ajuster l'une à l'autre. Le jeu de construction a été conçu pour que toutes les combinaisons de composants qui sont intéressantes soient faciles à réaliser. L'assemblage des éléments du jeu de construction est simple. Il dirige la construction dans des directions pré-établies, mais c'est de peu d'importance puisque c'est la direction où l'enfant veut aller.

Personnellement, ce qui m'amusait le plus étant enfant, c'était de construire un élément d'un ensemble plus grand que j'allais assembler avec ceux que construisaient mes amis pour fabriquer ce que

19. Il nous semble utile, lorsque la définition précise d'un concept nous amènerait trop loin de notre sujet, d'introduire ce concept par l'intermédiaire de métaphores, qui si elles ne sont pas des modèles exacts de la réalité, permettent d'en appréhender plusieurs aspects clés.

nous voulions faire. Les discussions sur la façon de construire les parties qui devaient "s'emboîter" étaient souvent difficiles, mais nous savions de nos succès précédents que le jeu en valait la peine.

Nous ne savons pas si le jeu de construction est toujours parmi les premiers jouets vendus pour Noël, mais quoi qu'il en soit, ceci nous a offert une métaphore que nous considérons très appropriée à l'explication des mécanismes qui permettent la réutilisation.

Concevoir un système réutilisable, c'est avant tout le découper en composants qui s'assemblent facilement, de façon à ne pas perdre de temps pour essayer une nouvelle approche, ou adapter un système existant à des besoins nouveaux. Le développement logiciel est un peu plus compliqué que le jeu de construction, aussi acceptera-t-on encore pour un temps — le temps de trouver les outils méthodologiques ou techniques qui permettront de se passer de cette contrainte, si c'est possible — d'écrire de nouvelles parties du logiciel, à la condition qu'elles n'aient pas déjà été écrites.

De ce point de vue, le développement logiciel s'apparente un peu à la publication de méthodes ou techniques. Il devient inconcevable de réécrire une partie de programme qui existe déjà, comme de re-développer une méthode déjà publiée, à moins d'avoir de sérieuses raisons de le faire.

Certaines raisons sont légitimes. Elles ont souvent trait au logiciel existant, qui n'est :

- pas disponible pour réutilisation (logiciel commercial notamment),
- pas assez portable,
- peu performant,
- pas conçu en accord avec les besoins actuels.

S'il est difficile de discuter les premières causes de réécriture, il n'en est pas de même de la dernière. S'il s'agit d'un problème de conception, alors peut-être faut-il déterminer dès que possible comment organiser les parties de logiciel qui vont bientôt être développées pour s'assurer qu'elles pourront, du moins dans la grande majorité des cas, être adaptées aux besoins futurs.

Se donner les moyens de s'adapter à quelque-chose que l'on ne connaît pas encore n'est pas simple. Il est donc plus prudent de partir de l'expérience passée. Il est possible de considérer cette expérience en regardant où les modifications des systèmes ont été les plus fréquentes et d'inférer que les modifications futures porteront plus fréquemment à ces endroits qu'à d'autres. On détermine ainsi "des endroits" du logiciel où il est souhaitable de disposer d'une *organisation* qui favorise le changement. Nous verrons que le concept d'*interface* permet d'atteindre cet objectif.

Une autre approche, complémentaire, consiste à déterminer ce qui ne changera pas. Ou encore, comment découper le problème en morceaux qui bien qu'ils ne soient pas identiques, se ressemblent assez pour que l'on puisse considérer qu'ils jouent des rôles identiques au sein du problème. Une utilisation différente du concept d'interface permet d'atteindre cet objectif.

5.2 L'interface au cœur de la réutilisation

Il est donc temps d'introduire le concept d'interface, qui dérive de l'expérience acquise pendant les développements orientés-objets de grands projets informatiques.

Une façon de comprendre ce qu'est une interface est de penser aux prises de courant que l'on trouve sur tous les appareils électriques. La prise qui permet à l'utilisateur de connecter facilement un robot électroménager à son alimentation électrique (le sous-réseau EDF de sa résidence) est une interface. C'est un morceau de plastique qui encapsule des éléments métalliques de telle façon qu'il se placent exactement dans l'alignement des conducteurs de la prise murale. Un aspect intéressant de cette métaphore est qu'elle permet de montrer très simplement qu'une interface peut-être plus qu'une commodité. Une interface peut-être conçue pour être utilisée simplement dans un but correct, mais très difficilement pour une utilisation que l'on cherche à éviter. Essayez de mettre votre sous-réseau EDF en court-circuit, avec une rallonge...

Vous n'achèteriez pas un appareil électrique dont le vendeur vous dirais que les cosses de la prise sont *légèrement plus rapprochées*, pour gagner de la place par exemple; n'est-ce pas? Vous venez de comprendre que l'interface, une fois acceptée doit-être permanente pour être efficace.

Maintenant, comment peut-on gérer l'adaptabilité avec une interface dont la caractéristique essentielle est la permanence ? Ne branchez-vous pas radio-réveil, chaîne HIFI, robot de cuisine, etc. avec la même interface ? L'interface garantit un service. Tous les composants qui peuvent se contenter du service tel que proposé par une interface peuvent fonctionner avec celle-ci.

5.2.1 Architecture collaborative

Si l'interface est au cœur de la réutilisabilité, c'est parce qu'elle est presque immatérielle. En quelque sorte, elle est du domaine des idées : il y a plusieurs façons de penser la manière dont on va découper un logiciel en composants et en interfaces. Ce qu'il faut trouver, pour atteindre les objectifs de réutilisabilité du logiciel que nous nous sommes fixés, c'est une bonne façon de couper. C'est à dire une façon qui permette de changer le logiciel en remplaçant une de ses composantes[20] par une autre qui présente les mêmes interfaces. On peut toujours changer une rallonge électrique blanche par une bleue si la couleur vient à mieux convenir à la couleur de la tapisserie...

Ce qu'il convient donc de mettre en place, c'est une architecture du changement : une certaine façon d'organiser un logiciel tel qu'il soit facile de le construire par morceaux, de changer certains composants par d'autres, de la façon la plus simple possible.

Parce que les logiciels sont plus complexes que le réseau électrique de votre résidence les interactions de leurs composants peuvent-être plus nombreuses que des relations du type `a-besoin-d-une-source-de-courant, fournit-une-source-de-courant`. Les architectes logiciels développent donc souvent des réseaux collaboratifs d'objets, qui fonctionneront ensemble pour atteindre un objectif, et pourront être modifiés par partie seulement, pour être adaptés à une tâche.

5.2.2 Les modèles de conception orientés-objet réutilisables

La connaissance accumulée pendant la conception de nombreuses architectures logicielles réutilisables (donc d'un plus grand nombre de logiciels) peut être collectée, analysée et décrite sous la forme d'un modèle de conception logiciel.[21] Une très bonne source d'inspiration pour la conception d'architectures réutilisables est donnée par le catalogue de modèles de conception réutilisables [GHJV96]. Les modèles de conception ont été remarqués comme des formes récurrentes qui apparaissent dans les logiciels que leurs développeurs ont modelé au cours du temps pour augmenter leur potentiel de réutilisabilité. De ce point de vue, il faut remarquer qu'il s'agit souvent de formes vers lesquelles des solutions logicielles ont convergé sous la pression des contraintes de réutilisabilité, dans des environnements parfois très différents. Par exemple, un type de collaboration entre objets, étiqueté comme le modèle Visiteur dans [GHJV96] est utilisé dans le compilateur Smalltalk-80 [Par90] (pendant la phase d'analyse de code source de ce langage de programmation orienté objet) et dans IRIS Inventor [Str93] (la bibliothèque graphique 3D précurseur de OpenInventor, utilisé dans Viseur pour le programmation de la vue moléculaire). Les deux environnements réalisent des activités très différentes : analyse de code et représentation tri-dimensionelle d'objets dans une scène graphique, pourtant leur architecture est semblable pour partie.

Une caractéristique remarquable des modèles de conception réutilisables est qu'ils fonctionnent très souvent par collaboration entre plusieurs objets. Ces modèles découpent l'action d'une tâche en morceaux d'une façon qui n'est souvent compréhensible que par des considérations de réutilisation. Le résultat est une architecture qui permet de réaliser une tâche par la collaboration des objets qui la composent.

Nous serons amenés à décrire certains modèles de conception réutilisables plus en détail dans la suite de ce document, mais pour le moment, examinons la question suivante.

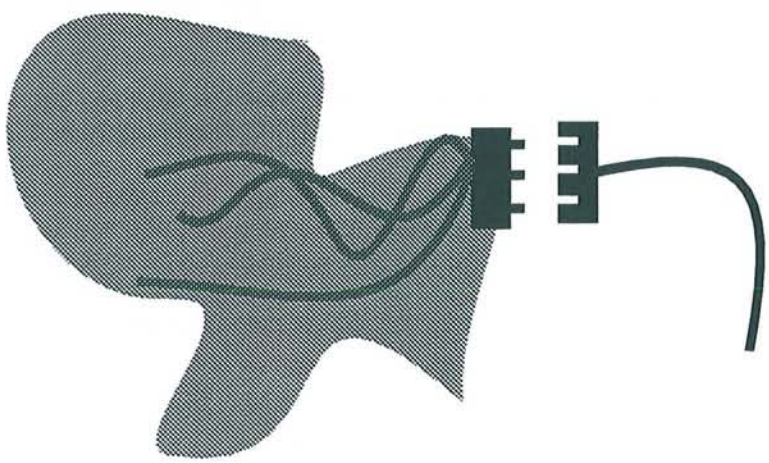
Comment les modèles de conception réutilisables sont-il reliés aux interfaces ? Très simplement : le modèle réutilisable utilise le concept d'interface pour découper la tâche à réaliser en objets interchangeables.

20. L'industrie du composant préfère le masculin, *un composant*, par l'ellipse de logiciel (le composant logiciel).
21. Traduction du terme anglais *design pattern*.

5.2.3 Illustrations du concept d'interfaces

Il est peut-être temps, maintenant, de proposer quelques illustrations des concepts que nous avons introduits de façon rapide, simplifiée, peu formelle, mais sans doute suffisante à une première compréhension.

La figure 5.2.3 nous amène à considérer que si l'interface est avant tout une idée, la spécification précise d'un service qu'un objet garantit à l'utilisateur, il faut à un moment donné programmer cette idée pour lui donner un peu de substance.



(a) Logiciel, interface et logiciel d'interface

(b) Logiciel (composant)
et interface

(c) Composition de composants logiciels

FIG. 5.1 – *Représentation schématique du concept d'interface*
(a) une partie de logiciel est représentée en gris clair, l'interface — une certaine idée d'un service — en noir est reliée au logiciel par un logiciel d'interface, en gris foncé. Le logiciel est parfois nommé implantation cible de l'interface. (b) La représentation graphique utilisée en (b) ne représente pas explicitement le logiciel d'interface. C'est celle que nous utiliserons dans la suite de ce document, lorsque nous nous intéresserons à la combinaison de plusieurs composants et que nous n'aurons pas besoin de parler du logiciel d'interface. Le logiciel (rectangle noté "composant") propose une interface (notée par un rectangle contenant un symbole qui désigne le type de l'interface). (c) Présente la composition de deux composants logiciels (`A` et `B`) à l'aide de leurs interfaces (de type `x`). Deux composants ne peuvent être composés que par des interfaces de même type. Le sens du lien entre les composants `A` et `B`), par l'interface `x` dépend de la fonction de l'interface (transfert de données ou de références de `A` à `B`, commande de `A` par `B`, etc.).

Interface et logiciel d'interface,... par l'exemple

Bien que les spécifications d'interfaces soient indépendantes de l'implantation, nous trouvons pratique de les présenter dans le cadre d'un langage de programmation. La syntaxe du langage, bien documentée, fournit un cadre formel à l'expression des interfaces. Dans la deuxième partie de ce document, nous utiliserons la syntaxe du langage C++ [Str91], pour définir les interfaces que nous présenterons. Des éléments de syntaxe du langage C++ nécessaires à la compréhension des extraits de sources présentés sont donnés en annexe D.

Le logiciel d'interface a pour rôle de traduire la représentation des données et l'organisation des traitements (en procédures, fonctions ou méthodes), retenus par une implantation, pour les adapter à la conception de l'interface, ou réciproquement.

Par exemple, le programme Babel est une implantation du traitement qui consiste à lire ou écrire une molécule dans un fichier. L'interface très simplifiée d'un tel traitement (`Babel_IO`) peut se concevoir, puis s'écrire ainsi :

```
class Babel_IO : public Molecule_IO {};

class Molecule_IO {
public:

  virtual molecule lire(string nom_fichier, string format_fichier);
  virtual void ecrire(string nom_fichier, string format_fichier, molecule mol);
}
```

La première ligne identifie l'interface `Babel_IO` avec celle de `Molecule_IO`. Ceci permet, par exemple, à d'autres développeurs d'écrire des parties du logiciel qui fonctionneront avec `Babel_IO` alors que l'implantation du logiciel d'interface est en cours : ces développeurs utiliseront l'interface `Molecule_IO` pour réaliser leurs développements, et puisque `Babel_IO` est un `Molecule_IO` ces développements seront compatibles avec l'implantation Babel dès la disponibilité de `Babel_IO`. Pour dire les choses autrement, `Babel_IO` accepte le contrat dont les termes sont décrits par l'interface de `Molecule_IO`.

Les dernières lignes décrivent l'interface `Molecule_IO`. Cette interface dispose de deux méthodes (l'objet sait réaliser deux opérations) : `lire` et `ecrire`. La première méthode permet d'obtenir un objet qui répond à l'interface `molecule` à partir d'un nom de fichier et d'un format. La définition de l'interface `molecule` n'est pas précisée ici, mais l'interface de `molecule` doit permettre d'accéder aux données qui modélisent les atomes ou les liaisons d'une molécule.

Décrivons maintenant les prérequis à l'activité de développement du logiciel d'interface. Cette activité nécessite

- la connaissance de l'implantation cible (le logiciel qui fait vraiment le travail, cf. figure 5.2.3).
- la connaissance de l'interface (pour obtenir les données ou les transmettre).

Nous décrivons dans le paragraphe suivant — à l'aide d'un exemple fictif choisi parmi les cas les plus difficiles —, à quoi correspond la connaissance d'une implantation cible et les moyens de l'obtenir.

Pour cela, supposons qu'un développeur ait besoin d'écrire une molécule dans un format non supporté par Babel, par exemple dans le format connu des seules routines Fortran `rm` et `wm`, dont la documentation a malheureusement été égarée. Le développeur en charge de la partie du programme qui doit écrire les objets qui supportent l'interface `molecule` dans le format de fichier connu de la procédure `wm` comprend que `wm` doit être appelée (`CALL wm`) après que les tableaux `ATM` et `BND` aient été initialisés correctement. Sans oublier quelque variables dont les noms évocateurs n'ont pas besoin d'être reproduits ici. Cet exemple peut faire rire, mais il est souvent assez typique des *logiciels du patrimoine*, aussi continuons. Le développeur vient de passer une heure à analyser les pré-requis à l'appel de `wm`, à la lecture (difficile), de son code. Il a maintenant bon espoir de comprendre comment `wm` écrit ces données dans un fichier, mais il pense préférable de ranger les données en bon ordre en mémoire et de laisser faire `wm`... Le développeur se souvient d'une situation similaire où la réécriture d'une procédure Fortran lui a fait découvrir beaucoup d'aspects de son fonctionnement interne (intime?).

Le paragraphe précédent décrivait un moyen d'obtenir la connaissance d'une implantation qui est nécessaire au développement du logiciel d'interface. Montrons maintenant comment utiliser cette connaissance pour développer le logiciel d'interface.

`class wmrm_IO : public Molecule_IO ;` De façon analogue à ce qui a été fait pour `Babel_IO`, le développeur déclare que `wmrm_IO` possède la même interface que `Molecule_IO`.

Ensuite, il définit le logiciel d'interface de `wmrm_IO`. Par exemple, la méthode `ecrire` doit transférer la molécule à écrire dans les tableaux `ATM, BND`, fixer quelques variables, puis faire `CALL wm`, ce qui écrit la molécule dans un fichier. Les informations à écrire dans la représentation mémoire connue de `wm` sont obtenues à travers l'interface `molecule`.

Dans cet exemple, le logiciel d'interface a

- traduit la représentation des données (de l'interface `molecule` vers `ATM, BND`,etc.) et
- organisé des traitements retenus par une implantation (les procédures `wm, rm`) pour les adapter à la conception de l'interface (`Molecule_IO::lire, Molecule_IO::ecrire`).

Une fois le logiciel d'interface développé, les autres développeurs peuvent utiliser les procédures Fortran `wm` et `rm` sans connaissance de leurs détails. Ils utilisent uniquement leur connaissance de l'interface de `Molecule_IO`.

```
wmrm_IO a_wmrm_io; // crée un objet qui possède l'interface wmrm_IO
molecule mol=a_wmrm_io.lire("fichier.wmrm","wmrm");
Babel_IO a_babel_io.ecrire("fichier.xyz","xyz",mol);
```

Ces quelques lignes transféreraient une molécule contenue dans le fichier `"fichier.wmrm"` (format lu et écrit par les routines `rm` et `wm`) `wmrm` dans le nouveau fichier `"fichier.xyz"`, au format xyz, lu par exemple par l'outil de visualisation moléculaire RasMol [Say].

5.3 Vers des développements organisés

Après cette introduction aux techniques de base de la réutilisation nous sommes mieux armés pour répondre à la question posée au début de ce chapitre. Dans les conditions fixées par le cadre d'un laboratoire ou d'un projet collaboratif de laboratoires, comment peut-on accroître le potentiel de réutilisation des logiciels de Chimie informatique pour bénéficier des avantages classiques de mise en œuvre de ces techniques ?

Commençons par examiner la mise en œuvre de techniques de réutilisation en dehors des contraintes du cadre de développement, pour un logiciel de Chimie informatique.

5.3.1 Détermination des composantes naturelles des logiciels du domaine

Il faut tout d'abord examiner les besoins de développement puis décider du découpage en composants des logiciels qui devrons être produits. Rappelons que les composants devront-être interchangeables (leur échange permettant de modifier le comportement des logiciels produits) et pourront fonctionner en collaboration pour effectuer une tâche. Nous avons brièvement rappelé les développements méthodologiques dans le domaine de l'analyse conformationnelle. Nous avons aussi décrit les composantes d'un outil d'analyse conformationnelle particulier. Cette analyse avait pour but de présenter une partition adaptée à l'ensemble des outils d'analyse conformationnelle par optimisation directe de l'énergie.

Résumons les composantes de ces outils :

- composant d'entrées/sorties (encapsulation des formats de fichiers),
- composant d'évaluation d'énergie (encapsulation de la méthode, du champ de force, etc.),
- composant d'optimisation (encapsulation de l'algorithme d'optimisation),
- composant de paramétrisation de la recherche conformationnelle (encapsulation des opérateurs de passage et de la paramétrisation).

5.3.2 Discussion des interfaces du domaine d'application

Pour garantir l'interchangeabilité de ces composantes, il faut spécifier (c'est à dire définir formellement) les interfaces nécessaires à leurs collaborations. Il faut préciser leur fonction générale puis les détails de leur modélisation dans un formalisme donné.

Dans le cadre de ce travail nous préférons le formalisme orienté-objet, mais la spécification d'interfaces dans le cadre d'un formalisme procédural est parfaitement possible. A l'appui de notre choix, il faut noter que le passage du formalisme orienté-objet vers le formalisme procédural est automatisable. A ce propos, voire la transcription des interfaces orienté-objet exprimée dans le *Interface Definition Language* (IDL) des spécifications CORBA[22] [Obj96] vers les langages procéduraux (le langage C par exemple).

Nous avons expliqué qu'une interface détermine un contrat entre les composants qui l'utilisent pour collaborer. Ce contrat fixe clairement ce qui peut être obtenu des objets qui implantent l'interface. Pendant la conception d'une interface, il faut donc réfléchir avec précaution à ce que l'on attend de son utilisation présente et future. D'autres considérations de conception dépendent des implantations des composants que l'on veut relier. Une interface bien conçue doit pouvoir être implantée (écriture du logiciel d'interface) sur n'importe quel composant qui manipule les données ou réalise les traitements que l'interface modélise.

La réflexion sur les interfaces d'un domaine d'application doit aboutir à la spécification d'interfaces qui tiennent compte des contraintes que nous avons décrites. Nous présenterons, dans le chapitre 6 la réflexion qui nous permet de proposer des spécifications d'interfaces adaptées au domaine de la Chimie informatique.

5.3.3 Implantation du logiciel d'interface

A l'aide de la spécification d'interface il est possible d'implanter le logiciel d'interface sur un composant logiciel pré-existant. Nous préférons présenter tout de suite les langages et techniques que nous avons employés pour réaliser une implantation de taille réelle (condition *sine qua non* d'un test d'applicabilité).

Nous avons choisi le langage C++ [Str91] pour plusieurs raisons. Le prototype d'outil d'analyse conformationnel que nous avons développé devant combiner des modules implantés dans les langages Fortran (évaluation d'énergie) et C (bibliothèque Babel, moteur d'optimisation ASA), le langage C++ est le langage orienté-objet vers lequel les passerelles étaient les plus simples à mettre en œuvre au moment du développement. De plus, nous verrons que ce langage offre la possibilité d'implanter efficacement les algorithmes génériques (cf. section 6.2.2) à l'aide de ses structures et classes paramétrables (*template*).

Programmation Mixte

Le lecteur curieux de connaître les méthodes de programmation mixtes[23] qui mélangent Fortran 77, Fortran 90, C, et C++ dans un même programme est invité à consulter [Cam97] qui renvoie vers des solutions *portables*. Il faut noter en effet qu'à la date de rédaction de ce document aucune convention ne fixe les règles de nommage des symboles des procédures Fortran ou l'ordre de passage des paramètres des procédures (tous deux nécessaires à leur utilisation dans le langage C). Il est donc possible de réaliser des passerelles Fortran/C ou Fortran/C++, mais sauf à utiliser des techniques particulières, ces passerelles ne sont pas correctes sur un autre système d'exploitation, ou même avec d'autres compilateurs.

5.4 Réutilisation et contexte de recherche universitaire

Les sous-sections 5.3.1, 5.3.2 et 5.3.3 décrivent les tâches à accomplir pour mettre en œuvre des techniques de réutilisation pendant le développement de logiciels de Chimie informatique. L'industrie

22. Voire la partie III pour une introduction à CORBA et IDL.
23. Une recherche de l'expression "Mixed language programming" permet de trouver quelques éléments de réponse.

informatique fournit de nombreux exemples d'utilisation de ces techniques mais le contexte du développement de nombreux logiciels de Chimie informatique n'est pas comparable. Ces développements ont souvent pour cadre un laboratoire universitaire, ou le contexte d'une collaboration de laboratoires, sur un projet commun.

Le rapport [AE95] insiste sur l'importance du développement d'une véritable politique d'entreprise autour de la promotion des techniques de réutilisation qui comprend souvent des techniques de contrôle des résultats intermédiaires ("benchmarking" de la réutilisation). Ces méthodes de travail semblent à tout le moins incompatibles avec les processus universitaires. Nous ne voulons pas sous-estimer leur impact sur la réussite d'un projet de réutilisation, aussi voudrions-nous considérer des solutions de motivation équivalente, mais plus adaptées au contexte où sont conduits les développements.

La première activité de détermination des composantes des logiciels (cf. sous-section 5.3.1) ressemble beaucoup à l'activité classique de *reviewing*, qui pourrait être complétée, ou modifiée, pour faire apparaître la structure des logiciels associés aux méthodes. Cette partie ne pose pas de difficulté majeure (il y a en général consensus sur le découpage logique, en composantes, d'un logiciel).

La troisième activité (sous-section 5.3.3) peut se concevoir dans le cadre d'un laboratoire puisque la spécification de l'interface rend l'implantation du logiciel d'interface relativement simple. La partie 5.2.3 présentait un exemple d'une telle activité (logiciel d'interface pour `wm`).

La deuxième activité : la conception d'interfaces, est plus délicate à mener à bien. Essentiellement parce qu'une interface devrait être testée sur plusieurs implantations avant d'accéder au statut d'interface réutilisable. Trop de développeurs considèrent qu'une interface est réutilisable parce qu'elle est orientée-objet. Étant donné l'expérience actuelle de la réutilisation, [GHJV96] remarque qu'une interface réutilisable est surtout une interface qui a été adaptée suite à des tentatives infructueuses de réutilisation. En outre, pour que le processus soit efficace les implantations testées doivent être suffisament différentes dans leur conception. Puisqu'il peut être difficile de trouver l'expérience de plusieurs logiciels suffisament différents au sein d'un laboratoire il est probable que la conception d'interfaces devrait impliquer un travail collaboratif entre des chercheurs de laboratoires différents. La réalisation de spécifications intermédiaires utilisée pour des tests d'implantation peut aussi aider à la spécification d'interfaces. Nous détaillons une discussion de spécification d'interfaces pour la chimie informatique dans le chapitre suivant.

6

Vers la spécification d'interfaces pour la Chimie informatique

6.1 Exécution d'un outil d'analyse conformationnelle

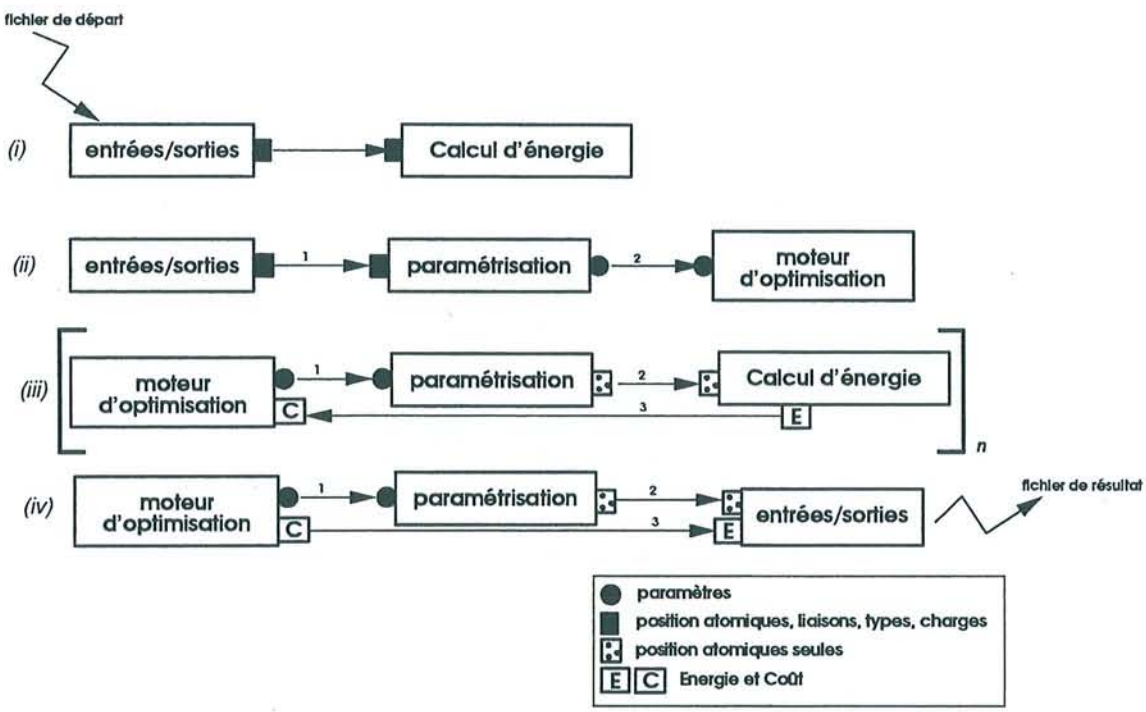


FIG. 6.1 – *Flux d'informations dans un outil d'analyse conformationnelle pendant les quatre phases de fonctionnement*
Les interfaces des différents composants sont présentées comme des symboles dont la sémantique est indiquée en légende. Ces symboles sont repris dans la suite du texte (et dans l'index) au moment de la spécification de l'interface correspondante. Les flux d'informations transitent d'un composant à l'autre par des interfaces compatibles. L'ordre des transferts d'information est indiqué, à l'intérieur de chaque étape *(i)-(iv)* par un chiffre au dessus de la flèche qui symbolise le transfert. Les composants présents sur cette figure ont été présentés dans le chapitre 4.

La figure 6.1 schématise les flux d'informations nécessaires à l'analyse conformationnelle par optimisation directe de l'énergie. La phase *(i)* transfère la totalité de l'information moléculaire du composant d'entrées/sorties vers le composant d'évaluation d'énergie. Ceci assure que toutes les informations nécessaires au calcul de l'énergie sont disponibles à ce composant, qui va être piloté par le moteur d'optimisation, en phase *(iii)*, après mise à jour de certaines informations. La phase *(ii)*, optionnelle, permet d'initialiser l'optimisation avec les paramètres correspondants à la structure lue en entrée. La phase *(iii)* requiert la conversion des paramètres utilisés par le composant d'optimisation dans la représentation moléculaire acceptée par le composant de calcul d'énergie. Ce transfert d'information est répété avant chaque calcul d'énergie demandé par le moteur d'optimisation. La phase *(iv)*, enfin, transfère le résultat de l'optimisation (meilleurs paramètres et meilleure valeur du critère d'optimisation) dans la représentation moléculaire du composant d'entrées/sorties qui produit le fichier résultat.

Cette figure illustre qu'il nous faut réfléchir d'une part à des interfaces qui permettent le transfert de données moléculaires, (plus ou moins complètes) et d'autre part à des interfaces pour gérer les collaborations des composants. Les prochaines sections expliquent les contraintes auxquelles doivent répondre ces différentes interfaces puis proposent des spécifications répondant à ces contraintes.

6.2 Transfert de paramètres

Un composant de paramétrisation a pour rôle de transformer tout ou partie de l'information moléculaire (positions atomiques, liaisons, types, charges) en un jeu de paramètres qui représenteront les degrés de liberté du système moléculaire.

Par exemple, ces paramètres pourront représenter les coordonnées des atomes de la molécule et donc être codés par $3n$ nombres flottants. Ou alors les paramètres seront les coordonnées internes de la molécule et on aura $3n - 6$ nombres flottants. On voit que le nombre de paramètres et la manière de les calculer à partir de certaines informations moléculaires sont variables, mais que leur type ("des nombres flottants") ne change pas.

La raison principale est que les algorithmes d'optimisation imposent des contraintes aux paramètres qu'ils peuvent accepter. Ces contraintes sont dépendantes de la nature des opérations que l'algorithme doit effectuer sur les paramètres. Par exemple, un algorithme qui utilise la dérivée numérique de l'énergie par rapport aux paramètres impose que $\frac{\partial E}{\partial z_i}$ ait un sens et soit calculable. Ces conditions peuvent se ramener à des conditions sur les paramètres et la fonction de coût, fonction des paramètres. On note que des paramètres flottants ou entiers remplissent les conditions de cette classe d'algorithmes d'optimisation. Ce n'est pas obligatoirement vrai pour des paramètres chaînes de caractères bien que l'on puisse imaginer des cas où ces conditions sont réunies (par exemple $(i), (ii), (iii), (iv), (v), \ldots (mcv)$ rempliraient ces conditions une fois définies l'addition et la division pour ce type particulier de paramètres !).

Pour revenir aux paramètres de l'optimisation, il doit être clair maintenant que l'on peut les modéliser comme un ensemble de nombres (flottants ou entiers). De nombreuses interfaces ont déjà été proposées pour ce type de besoin. Nous allons décrire celle que nous avons retenue, et les raisons qui nous ont conduit à ce choix.

6.2.1 L'exemple d'une solution sous-optimale

La spécification d'une interface qui donnerait accès aux paramètres d'un ensemble est une tâche apparemment simple qui peut conduire à beaucoup de mauvaises solutions. Nous décrivons une de ces solutions sous-optimales ici, ce qui nous sert de prétexte à montrer les problèmes soulevés par toute tentative de spécification.

```
class set_of_float {

public:
virtual float get_value_number(int index);
virtual void  set_value_number(int index, float value);
```

```
virtual int   size(void);
virtual void  reserve(int new_length);

}
```

Cette interface permet de fixer le nombre d'éléments de l'ensemble (`reserve`), de le déterminer (`size`) lorsque l'on ne le connaît pas, de lire une valeur de l'ensemble (`get_value_number`), d'inscrire une valeur dans l'ensemble (`set_value_number`).

Le code suivant illustre comment il est possible de copier les éléments d'un ensemble dans un autre à l'aide de cette interface.

```
void copy(set_of_float set_source,set_of_float set_dest) {

  int len=set_source.size();
  set_dest.reserve(len);
  for (int i=0; i<len; i++) set_dest.set_value_number(i,set_source.get_value_number(i));
}
```

Cette interface encapsule certaines opérations et données du concept d'ensemble mais introduit l'artefact de l'identifiant entier. C'est à dire que chaque paramètre est associé à un nombre entier, `index` dans la spécification, qui permet de le retrouver dans l'ensemble.

La programmation du logiciel d'interface au dessus d'une représentation de type tableau est simple (l'index dans le tableau *est* l'index de la spécification d'interface). Certains composants qui manipulent des paramètres pourraient-être implantés, pour des raisons de performances, avec une représentation de type liste ou arbre.[24] L'implantation du logiciel d'interface au dessus de tels composants pose les problèmes de performances liés au calcul d'un identifiant entier. Par exemple, dans le cas d'une liste, retrouver le n^{ieme} paramètre revient à parcourir les $n-1$ éléments précédents. Ceci montre que la complexité d'un algorithme (cf. l'algorithme de copie) peut-être changée par une spécification d'interface inadéquate.

Cette situation difficilement acceptable trouve ici une solution dans le modèle de l'itérateur. On peut en effet remarquer que l'opération copie peut s'exprimer comme suit.

```
void copy(set_of_float set_source,set_of_float set_dest) {

  iterator i;
  iterator end=set_source.end();
  for (i  =set_source.begin();
       i != end;
       i++) set_value_number(i,get_value_number(i));
}
```

L'itérateur `i` possède une interface qui supporte les opérations d'incrémentation (`i++`), d'affectation (`end=set_source.end();`) et de comparaison (`i!= end`). Les entiers supportent cette interface et sont un cas particulier (une spécialisation) du concept d'itérateur. L'itérateur va plus loin puisqu'il peut encapsuler un entier, un pointeur, un identificateur dans une base de donnée, et bien d'autre choses (c'est un objet...).

Voici une interface ensemble de nombres flottants qui utilise le modèle d'itérateur.

```
class set_of_float {
public:
virtual iterator begin(void);
virtual iterator end(void);
virtual float get_value_number(iterator position);
```

24. C'est assez peu probable pour des paramètres, mais nous préférons introduire ces notions, applicables aux informations moléculaires, sur cet exemple simple.

```
virtual void  set_value_number(iterator position, float value);
virtual void reserve(int new_length);
}
```

6.2.2 Programmation générique et Standard Template Library

D. R. Mussler et A. A. Stepanov ont développés le concept de *programmation générique* [MS89][MS94] puis l'ont appliqué à la définition des interfaces d'une bibliothèque d'algorithmes et de stockages de données. Ces interfaces, connues sous le sigle STL pour *Standard Template Library* ont été adoptées récemment par le comité de normalisation du langage C++ [Org9X] [Pla96]. Les implantations basées sur ces interfaces commencent à être distribuées avec les compilateurs récents.

STL est une bibliothèque qui propose des structures de données et des algorithmes pour les manipuler, de manière complètement uniforme. L'approche de programmation générique suivie par les auteurs de la spécification de STL garantit qu'une opération est effectuée sans perte de performance du fait de l'interface (cf. discussion de la sous-section précédente).

La bibliothèque STL propose

des conteneurs qui sont des structures de données répondant à une même interface. Par exemple des files, piles, listes, ensembles, tables de hashage, etc.

des itérateurs qui encapsulent le moyen de parcourir une structure de données.

des algorithmes qui agissent sur les structures de données, parfois en leur appliquant une opération. Quelques exemples : copy, transform, for_each, sort, etc.

des objets fonctionnels qui permettent de construire l'équivalent de fonctions simples en C++ (création de fonction par combinaison d'objets fonctionnels).

Par exemple, l'algorithme `copy` est défini comme suit. C'est la définition que nous retiendrons dans la suite de ce document.

L'algorithme `copy(first, last, result)` copie les éléments de l'intervalle $[\texttt{first}, \texttt{last})$ vers l'intervalle $[\texttt{result}, \texttt{result} + (\texttt{last} - \texttt{first}))$. C'est à dire qu'il réalise les affectations

$$*\texttt{result} = *\texttt{first}, *(\texttt{result} + 1) = *(\texttt{first} + 1),$$

et ainsi de suite.
Généralement, pour chaque entier n, de 0 à `last - first`, `copy` réalise l'affectation

$$*(\texttt{result} + n) = *(\texttt{first} + n)$$

Les affectations sont réalisées dans l'ordre des valeurs croissantes de n.

6.2.3 Spécification des conteneurs de paramètres

Il apparaît que les interfaces de STL fournissent une solution très soignée au problème de l'interface vers les paramètres de l'optimisation. Nous avons donc décidé de réutiliser ces spécifications pour implanter les logiciels d'interfaces des paramètres au dessus des composants d'optimisation et de paramétrisation.

Ceci nous permet de réutiliser les implantations des algorithmes et objets fonctionnels déjà développées et testées pour la bibliothèque STL.

6.3 Échange de données moléculaires

6.3.1 Accès sélectif aux données moléculaires

Le premier type d'interaction entre les composants d'un outil d'analyse conformationnel consiste à échanger des données moléculaires. Par exemple, il faut transférer la molécule lue par le module d'entrées/sorties dans le module d'évaluation d'énergie. Un retour à la figure 6.1 nous rappelle que l'interface que nous devons spécifier doit permettre de transférer les données moléculaires aussi bien dans leur intégralité que par parties (positions atomiques seules par exemple).

6.3.2 Justifications

En poussant la réflexion plus loin dans le sens de la réutilisation, on peut se demander si accéder spécifiquement à d'autres parties de l'information moléculaire ne serait pas intéressant, pour certaines tâches. On pourrait distinguer atomes, liaisons, charges, valences par exemple.

Interfaces indépendantes et besoins indépendants

A l'appui de cette remarque il faut noter que certains outils de Chimie informatique ne disposent pas de représentation pour la totalité de ces concepts. Par exemple, un logiciel de dynamique moléculaire conçu pour simuler des systèmes atomiques n'a pas de représentation pour le concept de liaison. Les méthodes qu'il implante n'en font pas naître le besoin.

Interfaces indépendantes et distinction lecture/écriture

D'autres outils de Chimie informatique vont calculer des charges partielles à partir de la donnée des atomes et des liaisons. Ils devraient donc disposer, en entrée, des interfaces atomes et liaisons, mais pas de l'interface charges; en sortie des interfaces charges, atomes, et liaisons. Certaines implantations de calcul d'énergie par modélisation moléculaire réorganisent les données de liaisons dans une étape préliminaire au calcul de l'énergie. Il est souhaitable dans ces conditions que l'accès aux liaisons soit restreint à l'écriture dans la représentation interne de l'implantation. Ceci pour rendre impossible la lecture de données incorrectes.

Interfaces indépendantes et programmation générique

Nous avons vu que la conception d'une interface doit veiller autant que possible à préserver le potentiel de performance qui s'exprime lorsqu'une représentation adaptée rencontre un algorithme efficace. Nous devons trouver un moyen pour que le logiciel d'interface puisse être implanté sans sur-coût algorithmique sur l'ensemble des implantations qui utilisent, transforment, ou fournissent des données moléculaires.

Commençons par examiner quelques représentations de données moléculaires. En toute logique, la nature de ces représentations devrait avoir été dictée par le type des opérations à effectuer sur les données (puisque le coût d'une opération dépend de la représentation choisie). On s'attend donc à observer plusieurs représentations de données différentes.

Nous devons remarquer que nous allons ici à l'encontre de ce que la plupart des chimistes entendent par *standardisation*. S'il est vrai que la standardisation passe par l'homogénéisation des façons d'accéder à des services distincts mais semblables — homogénéisation des interfaces donc — il est trop précipité de conclure que l'homogénéisation implique le choix d'une représentation unique. C'est bien sûr une solution, mais elle implique des contraintes (performances sous-optimales, bouleversement des habitudes de développement, etc.) qui l'empêchent de rencontrer l'adhésion nécessaire à son adoption. Ces deux façons de comprendre l'activité de standardisation son reprises sur la figure 6.2.

Les représentations moléculaires les plus courantes sont les suivantes.

Représentation par tableaux séparés. Les logiciels développés en Fortran utilisent préférentiellement ces représentations, qui offrent un temps d'accès aux informations constant. On peut illustrer ces représentations de cette façon.

Représentation par structures ou objets. Les logiciels qui doivent gérer des molécules dont le nombre d'atomes et/ou de liaisons varie au cours du traitement sont souvent programmés dans des langages tels que C ou C++. Les types de haut niveau disponibles dans ces langages — structures ou classes — favorisent les représentations où tous les attributs d'un atome sont regroupés. Ces langages proposent aussi des types tableaux qui peuvent être choisis par les développeurs lorsque ces représentations sont plus adaptées.

Notons enfin que des représentations moléculaires intermédiaires entre ces deux extrêmes existent aussi.

Une interface orientée-objet non générique

Le logiciel VMD [HDHL98] ("Visual Molecular Dynamics") permet de visualiser et d'analyser les systèmes macro-moléculaires (protéines, acides nucléiques, assemblages de bi-couches lipidiques, etc.) du point de vue statique mais aussi dynamique.

En accord avec ces besoins, les interfaces conçues pour développer ce logiciel séparent très clairement les informations qui n'évolueront pas au cours d'une simulation de dynamique moléculaire (types des atomes, masses, charges, liaisons, etc.) des informations qui changeront (position des particules, vitesses, etc.). Par contre, l'interface d'accès aux informations statiques regroupe (information extraite du guide du programmeur VMD [HDHL96]) :

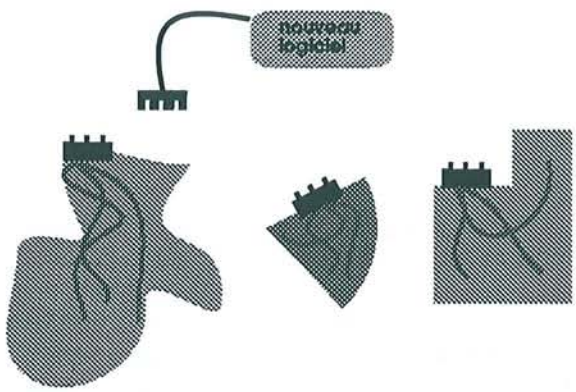
```
Mass
Charge
Atom name
Atom type
Residue name
Residue ID number
Chain identifier
Segment name
List of bonds
```

Cette interface est adaptée à la représentation des atomes contenus dans des macro-molécules, mais sa réutilisation pour accéder à l'information moléculaire contenue dans des logiciels adaptées aux petites molécules peut poser problème. Par exemple, quel sens donner aux attributs `Residue name`, `Residue ID number`, `Chain identifier`, `Segment name` dans ce contexte ?

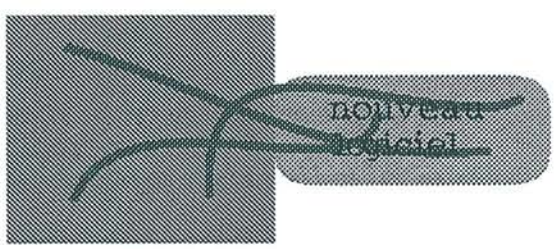
Certains proposeront d'utiliser l'artifice de « valeurs par défaut ». C'est à dire fixer `Chain identifier=MAIN_CHAIN`, par exemple, lorsque le concept d'identificateur de chaîne n'est pas applicable. Ce type de pratique pose problème lorsque les valeurs par défaut sont utilisées dans le cadre de calculs ultérieurs. Quel crédit peut-on apporter aux résultats de calculs dont certains paramètres n'ont pas de sens dans le contexte ? Ajoutons à cela qu'il est très difficile de décider si tel calcul ou tel autre peut-être utilisé sans risque sur telles ou telles données. On aimerait pouvoir se fier au mécanismes de typage des compilateurs pour automatiser le rejet des combinaisons données/calcul inconsistants.

C'est pourquoi nous pensons qu'il est préférable de scinder l'information moléculaire (■) en parties (⚄,▷,⊡). Le choix d'implanter telle combinaison d'interface ou telle autre sur une partie de logiciel dépendant des données que traite ou stocke cette composante.

Lorsque `calcul_P` utilise les positions atomiques d'une molécule pour calculer la propriété P, définir `calcul_P`(⚄) est préférable à `calcul_P`(■). Dans le premier cas, le compilateur acceptera la combinaison de `calcul_P` avec tous les composants logiciels qui implantent l'interface ⚄. Dans le second cas,



(a) Standarisation des interfaces



(b) Standarisation des logiciels

FIG. 6.2 – *Deux manières de comprendre la "standardisation"*
(a) Les parties de logiciel sont connectées à l'aide d'interfaces (cf. chapitre 5). (b) Le code d'interface existe aussi mais il est "intégré" aux logiciels. Par exemple, le "nouveau logiciel" est construit en utilisant la représentation des atomes des logiciels existants (choix de la représentation "historique"). Le code de l'interface, qui après le développement est inséparable du "nouveau logiciel" n'est pas réutilisable pour le connecter à d'autres parties du logiciel puisque ce code est spécifique des logiciels maintenant "intégrés".

le compilateur empêchera l'utilisation de `calcul_P(⚃)` puisque `calcul_P` est une fonction de ■ et que ■ "n'est pas" ⚃ (■ contient ⚃, plus peut-être d'autres parties).

Une interface orientée-objet générique

Puisque la réutilisation implique que ■ doit s'écrire ⚀,⚁,⚂, il est tentant de réutiliser le modèle des conteneurs STL et de spécifier l'interface d'un composé moléculaire, compound (■), ainsi :

```
class compound {

  atom_container    atoms();
  bond_container    bonds();
  valence_container valences();
  charge_container  charges();

}
```

Cette interface propose le moyen d'accéder à des interfaces plus spécialisées qui remplissent chacune leur rôle. Il est ansi possible de composer une interface `compound` en aggrégeant les interfaces spécialisées `atom_container`, `bond_container`, `valence_container`, ou tout autre sous-ensemble de ces interfaces. Nous disposons ainsi du moyen de composer des interfaces élémentaires (adaptées à une fonction précise) pour construire les interfaces d'implantations particulières.

L'interface `atom_container` est un conteneur au sens STL, on peut donc transférer les atomes d'une représentation (`source`) dans une autre (`dest`) en écrivant (cf. définition de l'algorithme `copy`, section 6.2.2).

```
copy(source.atoms().begin(), source.atoms().end(),
     dest.atoms().begin());
```

`source.atoms().begin()` est l'itérateur qui représente le premier atome. Utiliser un itérateur permet de garantir l'indépendance de l'interface par rapport à la représentation de l'atome dans l'implantation cible. L'itérateur peut encapsuler l'index de l'enregistrement de l'atome dans un tableau, ou le pointeur sur la structure ou l'objet "atome".

Le transfert de liaisons doit différer de celui des atomes puisqu'une liaison relie deux atomes. Le transfert doit traduire les itérateurs vers les atomes de la représentation source en itérateurs vers les atomes équivalents dans la représentation cible. Pour cette raison et pour d'autres qui deviennent apparentes à l'écriture de l'algorithme de transfert de liaison, que nous reprennons ci-dessous pour illustration, l'algorithme de transfert de liaison accepte les composés source et destination comme arguments.

```
transfer_bonds(source, dest);

template <class src_compound_t,class dest_compound_t>
INLINE void transfer_bonds(const src_compound_t &comp_source,
   dest_compound_t &comp_dest) {

      // déclarations de types (la difficulté provient de ce que
      // l'on utilise le type du composé (src_compound_t) pour accéder au type de
      // l'itérateur dont l'algorithme a besoin). bond_container est
      // déclaré par typedef dans l'interface de src_compound_t
      // vers le type réel.
    src_compound_t::bond_container::iterator first=comp_source.bonds().begin();
    src_compound_t::bond_container::const_iterator last=comp_source.bonds().end();
    dest_compound_t::bond_container::iterator dest=comp_dest.bonds().begin();
```

```
    while (first != last) {        // pour chaque liaison a transférer.

      // déterminer les identifiants uniques relatifs des atomes
      // aux extrémités de la liaison.
int idff =comp_source.atomIdentifier((*first).from(comp_source));
int idft =comp_source.atomIdentifier((*first).to(comp_source));

      // fixer la liaison (dest) dans le composé destination  (comp_dest)
      // à l'aide des itérateurs construits à partir des identifiants uniques relatifs

        (*dest).set(comp_dest,comp_dest.atomIdentifier(idff),
                    comp_dest.atomIdentifier(idft));

      // fixer l'ordre de la liaison dest à partir de l'ordre de la liaison first

(*dest).order((*first).order());

      // incrémentation des itérateurs de liaison
first++; dest++;
    }

};
```

La donnée des conteneurs d'atomes sources et destination est indispensable puisque le seul moyen de garantir la correspondance des itérateurs est de

1. traduire un itérateur en identifiant unique relatif (action réalisée par `int atomIdentifier(atom_iterator)`),
2. traduire l'identifiant relatif unique en itérateur dans le conteneur cible (`atom_iterator atomIdentifier(int)`).

Choisir l'identifiant unique relatif au conteneur comme la position de l'itérateur dans son conteneur est valide à condition que l'ordre de parcours des itérateurs par l'incrémentation soit constant.[25] *Si* `it` *est un itérateur du conteneur* `c`, on peut définir la position de `i` dans `c` par :

```
int position(container::iterator i,container c) {

 container::iterator cursor=c.begin();
 int pos=0;

 while (cursor++ != i) pos++;
 return pos;
}
```

L'interface que nous proposons est générique parce que nous n'avons fait aucune hypothèse sur l'implantation cible autre que celles nécessaires à l'écriture des algorithmes de transfert d'atomes et de liaisons.

6.3.3 Potentiel de Réutilisation

Nous avons discuté précédemment de la difficulté de construire des spécifications d'interfaces réutilisables. Notamment, et ceci s'applique aux interfaces que nous proposons ici, il est nécessaire d'essayer les interfaces sur plusieurs implantations différentes avant de leur accorder le statut d'interfaces réutilisables. De ce point de vue, les interfaces que nous avons développées et présentées ici sont de qualités

25. C'est le cas la plupart du temps, mais pas toujours. Dans le cas d'un programme parallèle, une autre tâche pourrait insérer des atomes dans le conteneur d'atomes et changer l'ordre de parcours à moins de contrôler l'accès à cette ressource.

différentes. Les interfaces d'échanges de paramètres, que nous avons seulement réutilisées sont certainement les plus stables. Puisque nous avons implanté l'interface d'échange de données moléculaires au dessus de deux logiciels très différents (Babel en langage C, un minimiseur en Fortran 90), nous pouvons supposer que cette interface possède de bonnes qualités de réutilisabilité. [26]

Les interfaces que nous décrivons dans la suite de ce chapitre :

- composant d'entrées/sorties (`coBabelInOut`),
- évaluateur d'énergie (`lou_eval`),
- moteur d'optimisation (`optimizer`),

devraient donc être considérées comme des hypothèses de travail succeptibles d'être discutées et optimisées pour accroître leur potentiel de réutilisation.

6.4 Composant d'entrées/sorties

Nous avons choisi d'interfacer la bibliothèque Babel à notre outil d'analyse conformationnel mais nous pourrions, dans le futur, vouloir remplacer Babel par une autre bibliothèque. Pour s'assurer de cette possibilité, nous devons définir une interface qui permet de commander la bibliothèque, pour lire ou écrire une molécule, sans connaissance de son fonctionnement interne. L'exemple de `wm`, et `rm` décrivait une interface de ce type, et c'est en fait approximativement celle, reprise ci-dessous, que nous utilisons.

```
class coBabelInOut {
public:
    coBabelInOut(); // constructeur et destructeur gèrent l'interaction
   ~coBabelInOut(); // avec la bibliothèque ou les routines..
    babel_compound *read(string filename, string type);
    bool write(babel_compound *c, string filename, string type);
}
```

Le gros défaut de cette interface est qu'elle impose de passer un objet de type `babel_compound` lors de l'écriture, alors que l'on voudrait pouvoir écrire n'importe quel composé, sans se soucier de sa représentation. Bien sûr, cette limitation est due pour partie à la bibliothèque Babel, qui ne peut écrire sur disque que les représentations moléculaires qui se conforment à sa conception (essentiellement des représentations créées par la bibliothèque elle-même). L'interface de transfert de données moléculaires que nous avons présentée dans la section précédente nous permet de lever cette limitation : il suffit de définir la méthode `write` telle qu'elle transfère les données de la molécule `c` dans une représentation `babel_compound` avant d'invoquer la fonction d'écriture de la bibliothèque Babel.

```
class coBabelInOut {
public:
    coBabelInOut(); // constructeur et destructeurs gèrent l'interaction
   ~coBabelInOut(); // avec la bibliothèque ou les routines..
    compound *read(string filename, string type);
    bool write(compound *c, string filename, string type);
}

bool coBabelInOut::write(compound *c, string filename, string type){

 babel_compound tmp=new babel_compound();
 copy_compound(c,tmp);
 internal_write(tmp,filename, type);
}
```

26. Comme nous le notions dans la section 5.4 il n'existe pas de mesure exacte du potentiel de réutilisabilité d'un logiciel (ou d'une interface). Le nombre de réutilisations réussies peut néanmoins servir d'indicateur.

6.5 Évaluation de l'énergie d'une conformation

Le composant de calcul de l'énergie dont nous disposions peut évaluer l'énergie d'un molécule décrite dans sa représentation, mais d'une seule.[27] Ce type de limitation était assez courant pour les logiciels implantés en Fortran77 — du fait de la difficulté de gérer la mémoire dynamique — et on peut souhaiter la voire disparaître des nouvelles implantations réalisées en Fortran90. En attendant, il est utile de trouver un moyen de limiter le nombre de molécule stockées dans ce type de représentation.

L'interface que nous proposons doit donc permettre de créer la représentation d'une molécule mais pas d'une deuxième. La solution à ce problème de conception est décrite dans le modèle du singleton [GHJV96].

La composante de calcul d'énergie peut alors se décomposer en deux parties.

6.5.1 Représentation moléculaire singleton

```
class lou_compound {
private:
   lou_compound(); // cet objet ne peut pas être créé par la méthode habituelle

public:
   // la méthode suivante
   // 1/ renvoie un objet lou_compound s'il n'en existe encore aucun
   // 2/ déclenche une exception (interruption du programme)
   //    si elle est appelée alors qu'une représentation est déjà utilisée.
   static lou_compound the_compound() throw "lou_compound: too_much_object_requested";

   lou_atom_container atoms();
   etc... // suite de l'interface d'accès aux données moléculaires.
}
```

La classe `lou_compound` permet de représenter une molécule stockée dans l'implantation Fortran90 que nous avons utilisé. Nous présentons ici la partie de cette interface (implantation du modèle singleton) qui permet de restreindre le nombre de molécule stockée par l'implantation à un.

6.5.2 Évaluateur d'énergie

Nous voudrions écrire `double eval.energy(■);` pour calculer l'énergie d'une molécule stockée dans une représentation moléculaire quelconque, à l'aide des paramètres et champs de forces encapsulés par eval. Une première étape consiste à proposer l'interface suivante.

```
class lou_eval {
public:

   lou_eval(lou_compound &c); // Il n'est pas nécessaire d'utiliser le modèle
   // du singleton ici, mais comme il n'existe qu'une seule instance de
   // lou_compound, la référence (&) est indispensable.

   double energy(); // calcule l'énergie de la conformation stockée dans
   // le composé dont la référence a été donnée en argument du constructeur.
}

// exemple d'utilisation:
```

27. Il est possible de calculer l'énergie d'un système composé de plusieurs molécules en interaction, mais nous ne sommes pas intéressé par cette fonctionnalité.

```
  co_lou_compound c=co_lou_compound.the_compound();

...  initialisation de représentation moléculaire ...

  lou_eval eval=new lou_eval(c);

  double energy=eval.energy();

  copy_compound(other_mol, c);
  double other_energy=eval.energy();
```

Pourquoi séparer évaluation d'énergie et représentation moléculaire alors que le logiciel Fortran qui implante le traitement déclaré par l'interface est homogène? Pour garder la possibilité d'implanter l'interface au dessus de logiciels qui ne sont pas homogènes. On pourrait en effet concevoir des codes d'évaluation d'énergie implantés pour utiliser l'interface ( ,  , )[28] implantée par une autre — ou d'autres — parties de logiciel.

6.6 Moteur d'optimisation

L'interface d'un moteur d'optimisation (l'implantation d'un algorithme d'optimisation) doit permettre :

1. de configurer la fonction de coût avant l'optimisation,
2. de fixer les paramètres de départ avant l'optimisation,
3. de fixer les limites de variation des paramètres avant et pendant l'optimisation.
4. de déclencher l'optimisation,
5. d'accéder aux meilleurs paramètres (ce ne sont pas obligatoirement les derniers rencontrés par la fonction d'évaluation),
6. d'accéder, à tout moment, au meilleur coût déterminé pendant l'optimisation.

Les points 2–6 rapprochés des discussions de la section 6.2 peuvent nous amener à construire l'interface suivante.

```
class optimizer {
public:

   optimizer(); // constructeur pour initialisation de l'implantation
  ~optimizer(); // destructeur (ménage de l'implantation)

  parameter_container         parameters(void);          // point 2
  min_parameter_container min_parameters(void);          // point 3
  max_parameter_container max_parameters(void);             //

  void   optimize(void);                                 // point 4
  const  best_parameter_container best_parameters(void); // point 5
  double best_cost(void);                                // point 6

}
```

28. L'intérêt de telles implantations est discutable lorsque l'on compare le temps de calcul de l'énergie et le temps de transfert d'une représentation à une autre.

Le point 2 nous amène à doter le moteur d'optimisation d'un conteneur de paramètres, dont l'interface a été précédemment discutée. Le point 3 donne le moyen de fixer les limites de variations des paramètres par la donnée de valeurs minimums et maximums. Ceci signifie simplement que le moteur d'optimisation est autorisé à choisir la valeur du paramètre z_i telle que (`opt` étant un objet possédant l'interface `optimizer`) :

$$\texttt{opt.min\_parameters}()[i] \leq z_i \leq \texttt{opt.max\_parameters}()[i]$$

soit vrai, à toute étape de l'optimisation. z_i correspond bien sûr à `opt.parameters()[i]`.

Un dernier détail : le type du conteneur renvoyé par `best_parameters()` est indiqué `const`. Ceci précise que les paramètres de ce conteneur sont constants, ou pour dire les choses autrement, que le compilateur refusera toute tentative d'écriture dans ces paramètres. Un accès en écriture "accidentel" pendant une optimisation modifierait certainement le fonctionnement de l'algorithme d'optimisation et doit donc être évité.

6.6.1 Problèmes éventuels

L'interface que nous proposons pour le moteur d'optimisation peut poser les problèmes suivants.

1. L'approche qui restreint les limites de l'optimisation par des bornes minimum et maximum sur les paramètres permet-elle d'encapsuler efficacement tous les types d'algorithmes d'optimisation ?
2. Comment encapsuler efficacement les différentes fonctions de coût que peut utiliser une implantation particulière de l'algorithme ?

Il est difficile d'apporter des réponses à ces points sans considérer plus d'implantations que nous ne l'avons fait ici. Aussi n'aborderons-nous pas davantage ces problèmes dans le cadre de ce document. Nous noterons simplement que l'interface que nous proposons est suffisante pour réaliser l'outil d'analyse conformationnelle présenté en annexe E.

6.7 Paramétrisation

Nous allons discuter ici comment les interfaces que nous avons décrites dans les sections précédentes permettent de changer la paramétrisation de la recherche conformationnelle. La conception du composant de paramétrisation est un exemple d'adaptateur [GHJV96].

En gardant à l'esprit la figure 6.1, rappelons que le composant de paramétrisation transforme ■ en • et réciproquement. Définissons plus précisément sa sémantique.

Nous aimerions pouvoir écrire

```
Babel_IO IO;
babel_compound *c;
c=IO.read("start_conformation.xyz","xyz");

coord_adaptator_t coord_adaptator= new coord_adaptator_t(c);


copy(coord_adaptator.begin(),
     coord_adaptator.end(),   opt.parameters().begin());
```

pour transférer les paramètres correspondants à la molécule c dans la représentation interne de l'optimiseur. `coord_adaptator_t` est le type du composant de paramétrisation, dont une nouvelle instance est crée `new coord_adaptator_t(c);` à partir d'une instance `compound`.

L'utilisation de l'instance de `coord_adaptator_t` (par l'algorithme générique `copy` ne présuppose rien sur la façon dont les paramètres ont été choisis. On peut dire que `coord_adaptator_t` encapsule la paramétrisation des paramètres.

Le transfert des paramètres contenus dans la représentation interne de l'optimiseur vers la représentation moléculaire associée à `coord_adaptator` peut s'écrire, de façon analogue :

```
copy(opt.parameters().begin(),
```

```
        opt.parameters().end(),        coord_adaptator.begin());
```

On dit que `coord_adaptator` est un adaptateur parce qu'il permet à deux composant de fonctionner ensemble en adaptant l'interface de l'un pour la rendre compatible avec celle de l'autre. On pourrait facilement changer le nombre et la nature des paramètres à optimiser en remplaçant l'instance de `coord_adaptator` par une instance d'une classe dont l'interface est compatible.

Nous donnons en appendice F deux exemples d'adaptateurs. Le premier transforme les coordonnées cartésiennes d'une représentation moléculaire en paramètres. Le second transforme les coordonnées internes d'une représentation moléculaire en paramètres. Ces exemples illustrent qu'un adaptateur peut réaliser tout aussi bien un travail simple (mise en correspondance d'interfaces) qu'un travail conséquent (transformation des coordonnées internes vers les coordonnées cartésiennes).

Le passage d'une paramétrisation à une autre doit en général s'accompagner d'une re-définition des limites de l'optimisation. Par exemple, si l'optimisation en coordonnées cartésiennes peut fonctionner en fixant les bornes d'optimisation à $\pm$ 0.1 Å autour de la conformation courante (pour chaque étape de l'optimisation); il en va autrement de l'optimisation en coordonnées internes.

Mais commençons par examiner comment fixer les bornes de l'optimisation à l'aide des interfaces que nous proposons, dans le cas de l'optimisation en coordonnées cartésiennes.

```
transform(coord_adaptator.begin(),
          coord_adaptator.end(),
          opt->min_parameters().begin(),
          bind2nd(plus<double>(), -0.1));

transform(coord_adaptator.begin(),
          coord_adaptator.end(),
          opt->max_parameters().begin(),
          bind2nd(plus<double>(), +0.1));
```

Quelques explications s'imposent. `transform` est l'*algorithme générique* STL qui permet de transformer les éléments d'un conteneur `source` alors qu'il sont écrits dans un conteneur `dest`. La définition de `transform` est la suivante.

```
template <class InputIterator, class OutputIterator, class UnaryFunction>
OutputIterator transform(InputIterator first, InputIterator last,
                         OutputIterator result, UnaryFunction op);
```

> L'algorithme `transform(first, last, result, op)` réalise l'opération `op(*i)` pour chaque itérateur `i` dans l'intervalle `[first, last)` et assigne le résultat de cette opération à `*o`, où `o` est l'itérateur de sortie (`result`). C'est à dire que pour chaque n tel que $0 \leq n < last - first$, `transform` réalise l'assignation `*(result + n) = op(*(first + n))`. La valeur de retour est $result + (last - first)$.

L'objet fonctionnel `bind2nd(plus<double>(), +0.1)`, construit dynamiquement pour l'occasion, ajoute (`plus`) `0.1` à la valeur (de type `double`) qui lui est passée en argument. L'algorithme invoque l'objet fonctionnel avec l'élément courant puis place le résultat dans l'élément destination. Ceci est répété autant de fois que l'opération doit-être réalisée.

Examinons maintenant comment fixer les limites d'optimisation de coordonnées internes.

```
transform(coord_adaptator.begin(),
          coord_adaptator.end(),
          opt->min_parameters().begin(),
          set_min_value);

transform(coord_adaptator.begin(),
          coord_adaptator.end(),
          opt->max_parameters().begin(),
          set_max_value);
```

Le code est très semblable. Mais examinons les définitions des fonctions `set_min_value` et `set_max_value`

```
  double set_min_value(coordinate_adaptor_t::reference &it) {
    switch (it.isAngle()) {
    case false: return double(it); // min bond length equals length;
    case true:
        switch (it.isBending()) {
        case true:
            return (double(it)); // min bending angle equals angle
        case false: //torsion
            return -180; // min angle;
        //  return double(it)-50; // use this version to explore
                                  //  locally around the current dihedral
}
    }
}

double set_max_value(coordinate_adaptor_t::reference &it) {
    switch (it.isAngle()) {
    case false: return double(it); // max bond length equals length
    case true:
        switch (it.isBending()) {
        case true:
            return (double(it)); // max bending angle equals angle
        case false: //torsion
            return +180; // max angle;
//          return double(it)+50;  //  use this version to explore
                                   //  locally around the current dihedral
}    }
}
```

Ces fonctions s'appuient sur l'interface de l'itérateur interne du composant de paramétrisation : `coordinate_adaptor_t::reference` est le type d'une référence à cet itérateur. Ce type particulier d'itérateur propose une interface enrichie qui permet de déterminer quel est le type de coordonnées qu'il référence. Ainsi, `it.isAngle()` vaut `true` lorsque le paramètre référencé par l'itérateur est un angle. Si le paramètre n'est pas un angle, alors c'est une longueur de liaison. La méthode `it.isBending()` discrimine les angles de valence[29] des angles de torsion.

Dans le cadre des spécifications d'interface que nous proposons, il est donc possible de changer la paramétrisation de la recherche conformationnelle au moyen d'un composant de paramétrisation qui proposerait l'interface suivante.

```
class parametrization {
public:

    parametrization(compound &c);
   ~parametrization(compound &c);

    ...

    para_iterator begin();          // interface du conteneur de paramètre
    para_iterator end();
```

29. Bending angles.

```
    ...

    parameters_to_compound();          // déclencheurs de conversions
    parameters_from_compound();        // (utiles quand la conversion
                                       //  est coûteuse)
    ...

    double min_value(para_iterator); // détermination des limites
    double max_value(para_iterator); // de l'optimisation.
}
```

Cette interface n'est pas précisément celle que nous avons implantée mais dérive de l'expérience acquise pendant l'implantation de la partie paramétrisation du logiciel. Ceci illustre que la spécification d'interface est une activité de va et vient entre l'implantation, la réflexion sur les interfaces, l'ajustement de l'implantation, une nouvelle réflexion sur la base des difficultés rencontrés pendant la phase d'implantation, etc.

De ce point de vue, le travail que nous avons présenté dans ce chapitre devrait être considéré comme un point de départ pour l'établissement de spécifications d'interfaces pour la Chimie informatique.

7

Bilan

7.1 Réutiliser l'existant

La première chose à retenir de ce travail est que nous montrons qu'il est possible de construire de nouveaux logiciels de Chimie informatique, *à l'aide des codes disponibles*, en appliquant les méthodes qui favorisent la réutilisabilité des logiciels.

La réutilisation des logiciels du patrimoine de la Chimie informatique peut se conduire en trois étapes.

1. détermination des composantes logicielles fréquemment rencontrées dans le domaine d'application,
2. spécifications des interfaces partagées par les composantes relevées en 1.,
3. implantation des logiciels d'interfaces au dessus des codes qui proposent des services compatibles.

L'étape numéro 2., assez délicate, devrait être conduite par une procédure de proposition de spécification/réflexion/implantation/test de réutilisation/retour sur les spécifications.[30]

L'étape 1. ne présente pas de difficulté particulière et pourrait-être réalisée lors de l'activité de *reviewing*.

L'étape 3., peut être réalisée facilement par collaboration de deux personnes : la première connaissant bien le code à interfacer, la seconde, les spécifications de l'interface.

Les *rares* approches de développement de logiciels de Chimie informatique orienté-objet insistent sur la réécriture du nouveau logiciel, mieux conçu que ses prédécesseurs. Nous préférons passer plus de temps à réfléchir à l'architecture globale du nouveau logiciel et utiliser moins de temps pour implanter et tester les logiciels d'interface au dessus de codes pré-existants — donc déjà testés et validés...

Cette approche de la réutilisation ne nécessite pas de bouleversement des habitudes de développement puisque des parties du logiciel peuvent continuer à être développées en Fortran (par exemple) puis combinées au reste du logiciel en développement.

Enfin, réfléchir à la spécification des interfaces d'un composant permet souvent de mettre en relief les points faibles d'une implantation particulière (qui limitent la réutilisation du code). Certains codes, très mal conçus, doivent parfois être réorganisés pour pouvoir être réutilisés; d'autres doivent-être légèrement adaptés, d'autres sont immédiatement réutilisables.

7.2 Perspectives

30. La procédure de développement des spécifications CORBA est exemplaire de ce processus.

7.2.1 Performance des codes et programmation générique

Les techniques de programmation générique, et plus particulièrement l'instanciation d'algorithmes adaptés à une représentation[31] (utilisation des constructions *template* du langage C++ par exemple) donnent les moyens de produire des logiciels adaptables et optimisés.

7.2.2 Utilisation distante de logiciels de Chimie informatique

Les logiciels de Chimie informatique — visualisation mise à part — sont gros consommateurs de puissance de calcul. Pour cette raison, le logiciel de calcul fonctionne souvent sur une machine d'un centre de calcul ou sur une machine puissante distincte de la station avec laquelle l'utilisateur prépare les simulations et analyse les résultats. Si nous voulions faciliter le suivi d'un de ces calculs, depuis la station d'analyse et peut-être la modification interactive des paramètres décidée dans le cadre du suivi, nous chercherions des moyens de transférer des données et des directives entre la machine de calcul et la machine de commande. Les spécifications d'interfaces nous seraient alors très utiles, couplées au modèle Procuration [GHJV96]. La figure 7.1 illustre cette architecture.

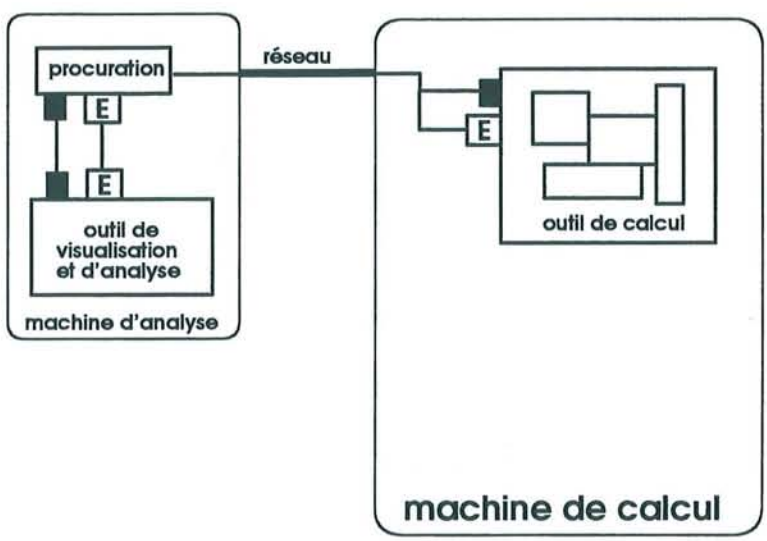


FIG. 7.1 – *Le modèle procuration*
Le composant noté `procuration` est responsable de transmettre les requêtes et les résultats de, et vers, un outil de visualisation et d'analyse. Ce composant se substitue à l'outil de calcul, situé sur une autre machine, pour transmettre les communications à travers le réseau. Tout se passe comme si l'outil de visualisation était directement connecté à l'outil de calcul.

7.3 Conclusion de la deuxième partie

7.3.1 A propos de la réutilisabilité

Notre recherche de la réutilisabilité approche de sa fin (temporaire...). Nous nous sommes placés dans le contexte de la recherche conformationnelle pour introduire plusieurs concepts indispensables à la compréhension des méthodes de la réutilisation. Bien sûr, le contexte de la recherche conformationnelle n'était pas qu'un prétexte, comme les exemples trop vites choisis pour illustrer un manuel. L'introduction aux algorithmes de recherche conformationnelle qui débutait cette partie montre clairement que

31. Nous n'avons pas présenté ces particularités un peu techniques. L'appendice F présente des exemples de mise en œuvre.

ce champ de recherche est actif. Pour cette raison, la réalisation d'un logiciel facilement adaptable à de nouveaux besoins, suffisamment modulaire pour permettre l'échange d'un algorithme d'optimisation par un autre, le changement de la paramétrisation de la recherche conformationnelle, du champ de force, etc. est motivée.

A côté de cet aspect "applicatif", le logiciel, ou plus exactement l'ensemble des composants logiciels que nous avons développés est un outil idéal pour expérimenter et raffiner les spécifications d'interfaces adaptés au domaine de la Chimie informatique et les techniques d'implantations. Ces dernières joueront un rôle capital au niveau des performances des codes développés à l'aide de techniques de réutilisation de composants. Or les logiciels de Chimie informatique sont des logiciels forts consommateurs de puissance de calcul. Il est donc prudent de vérifier, avant de démarrer tout projet d'envergure à l'aide de ces méthodes et techniques, qu'elles ne détériorent pas les performances des logiciels produits. L'outil que nous avons développé peut permettre ce type de vérification.

En dernier lieu, nous voudrions revenir rapidement sur l'information contenue dans la figure 6.1. Ce diagramme synoptique présente une vue unificatrice des algorithmes de recherche conformationnelle. Il permet d'expliquer rapidement — et précisément — ce que fait un programme de recherche conformationnelle. Le programme que nous avons développé se conforme précisément à ce schéma. En particulier, les composants du programme sont clairement séparés par un petit nombre d'interfaces. Les spécifications d'interfaces et les composants logiciels sont donc aussi des outils d'enseignement qui permettent de présenter aux étudiants les informations les plus pertinentes.[32]

Nous pouvons maintenant répondre aux questions laissées en suspens en fin de première partie.
Qu'est-ce que la réutilisation? une mesure d'économie

- des efforts qu'il faut faire pour comprendre le fonctionnement d'un logiciel (notez comme nous avons réutilisé les mêmes interfaces et composants),
- des efforts de mémorisation (il faut se souvenir de moins de choses que l'on réutilise plus souvent),
- des efforts de développement et de test (moins de logiciel nouveau à écrire et à tester),

Comment favoriser la réutilisation? en développant le potentiel de réutilisation des logiciels — leur réutilisabilité. Nous avons décrit une démarche potentielle dans les chapitres précédents.

7.3.2 Retour à la bioinformatique

Nous avons vu comment favoriser la réutilisation des composants d'un outil de recherche conformationnelle. Comment pourrions-nous adapter cette expérience au contexte particulier de la bioinformatique, pour favoriser le développement d'outils d'aide à l'analyse de données biologiques bien adaptés à une tâche?

C'est le thème que nous allons discuter dans la troisième partie de ce document.

32. A condition de se limiter à l'utilisation des composants. Voire l'annexe E pour un exemple de code utilisateur.

D

Éléments de syntaxe du langage C++

Les éléments de syntaxe présentés ici permettent d'interpréter les fragments de sources présentés au cours du texte mais ne permettent pas d'aborder les extraits de sources présents des prochaines annexes. Consulter [Str91] pour les compléments.

Certains articles sont extraits du manuel de référence de C++ [Str91], les citations sont entourées de guillemets et suivies du numéro de section dans cet ouvrage (entre crochets). Des renvois vers ce manuel sont notés de la même façon pour des compléments.

Identificateurs

«Un identificateur est une suite de lettre et de chiffres de longueur arbitraire. Le premier caractère doite être une lettre ; le souligné '_' compte pour une lettre. Les majuscules et les minuscules sont distinctes.» [2.2]

Exemples d'identificateurs valides: `un_identificateur`, `UnAutre`, `_encore_un_autre123456789`.

Types fondamentaux

Les types de base du langage C++ que nous utilisons dans ce manuscrit sont les suivants.

int représente un entier,

float représente un nombre flottant simple précision.

double représente un nombre flottant double précision.

[4.5]

Variables

Les variables du langages sont déclarées en faisant suivre l'identificateur d'un type valide (type de base ou nom de classe) par un identificateur. [8.3]

L'affectation de deux variables consiste à transferer le contenu d'une variable dans l'autre. Elle est réalisée à l'aide de l'opérateur binaire '=' :

```
int a=3;  // affectation de la valeur 3 à la variable a
int b=4;  // ...
int c=a;  // affectation de a dans c

// c vaut maintenant 3.
```

Profils des fonctions et méthodes

Le profil d'une méthode (ou d'une fonction) correspond au nom de la méthode, à son type de retour, et à la liste des types de ses arguments.

Par exemple, le profil de la fonction `Fonction`, de nom "Fonction" est donné par:

```
int Fonction(int premier_argument, float second_argument);
```

Cette information indique que `Fonction` calcule un entier (int à la gauche du nom de la fonction) à partir d'un entier (`int premier_argument`) et d'un nombre réel (`int second_argument`).

Le mot-clé `void` note l'absence d'un argument ou d'une valeur de retour. Ainsi, `void setA(int nouvelleValeurDeA)` est une fonction qui ne retourne aucun résultat.

Les profils sont utiles lors de la déclaration des classes du langages pour préciser les méthodes applicables aux objets.

Classes

La structure syntaxique suivante déclare une classe de nom `nom-de-la-classe`.

```
class nom_de_la_classe {
}
```

Constructeurs/destructeurs

Des méthodes spéciales, appellées constructeur et destructeur, sont appelées juste après l'instanciation d'un objet de la classe, respectivement juste avant sa destruction.

```
class nom_de_la_classe {
        nom_de_la_classe(); // constructeur
       ~nom_de_la_classe(); // destructeur
}
```

Types pointeurs

Les types pointeurs sont des types qui supportent l'opérateur d'indirection. Cette opérateur, noté `*ptr` lorsque `ptr` est le nom d'une variable de type pointeur, est évaluée à la valeur du type pointé par `ptr`. Le sens de "pointer" varie en fonction du type pointeur considéré puisque l'opération d'indirection peut être définie pour un type objet (cf. `opérator*` [7.2]). Appliqué à un itérateur, l'opérateur d'indirection permet d'obtenir la valeur référencée par l'itérateur (l'itérateur est la généralisation du pointeur).

Types objets des bibliothèques standards

La classe `string` permet de stocker et de manipuler des chaînes de caractères.

Attributs de classes et méthodes

La syntaxe du langage C++ ne différencie pas l'interface d'une classe et l'implantation d'une classe mais permet d'exprimer ces deux concepts.

Interface d'une classe

Les interfaces peuvent être exprimées dans le langage C++ en ne définissant que des profils de méthodes à l'intérieur de la classe.

```
classe molecule {
  public:
         int nombreAtomes(void);
}
```

La classe molecule dispose d'une méthode publique (utilisable) qui retourne un entier.

Implantation d'une classe

```
class implantation_molecule : public molecule {
public:
          int NbreAts;
          void nombreAtomes(int val}; // interface
          void nombreAtomes(int val} { NbreAts=val; } // implantation
}
```

La classe `implantation_molecule` hérite de l'interface `molecule` et implante un attribut de classe : `int NbreAts;`

Instantiation des classes

Les classes du langage ou définies par l'utilisateur peuvent être instanciées. C'est à dire que la classe est utilisée pour créer un nouvel objet dont le comportement est décrit par la classe. Une métaphore utile est celle du moule. Une classe est un moule (comme un moule à gâteau, ou un moule à château de sables) qui sert à créer des gâteaux ou des châteaux de sables presque identiques au moule. Dans le langage C++, l'objet créé ne permet pas de créer de nouveaux objets, c'est une prérogative de la classe.

```
// définition de la classe Gateau (le moule à gâteau) :
class Gateau {
... // définition de la classe (atributs, méthodes, droits d'accès)
};

// instantiation d'un objet Gateau (création d'un nouveau gâteau à l'aide du moule.
Gateau baba= new Gateau();
```

L'exemple précédent illustre l'instantiation d'un nouvel objet à partir de la classe `Gateau`.

Utilisation des objets

Le programmeur utilise les objets contenus dans des variables après instanciation en manipulant leurs attributs et méthodes.

```
// utilisation de implantation_molecule :

implantation_molecule mol1=new implantation_molecule();

mol1->NbreAts=3;
// la variable NbreAts de l'objet mol1 est fixée à 3.
mol1->nombreAtomes(4);
// la méthode nombreAtomes de l'objet mol1 est invoquée.
// son implantation fixe la variable NbreAts de l'objet mol1 à la valeur 4.

implantation_molecule mol2;
 mol2.NbreAts=3;
```

L'opérateur -> permet d'accèder aux membres d'un objet créé avec le mot clé `new`.

L'opérateur point . permet d'accèder aux méthodes ou aux attribut d'un objet lorsque l'on dispose d'une référence directe (sans pointeur). C'est le case pour l'objet `mol2`.

Ojets fonctionnels

Certains objets sont munis d'un opérateur `operator()` qui permet de les utiliser comme des fonctions.

```
class objet_fonctionnel {
int operator()(int a) {return a+1;}
}

  objet_fonctionnel func;
  int i=0;
  int j=func(i); // appel de la méthode operator()\index{Opérateur!() (fonction-
nel)} de l'objet func.

  // j vaut 1.
```

Les algorithmes `copy` et `transform` de STL utilisent cette fonctionnalité (parmi d'autres).

E

Point de vue de l'utilisation

```
/* Recherche conformationnelle en coordonnées internes
   Certaines interfaces décrites dans le manuscrit ne sont
   pas utilisée ici. Par contre le lecteur trouvera le code
   qui est à l'origine de leur spécifications.
*/

#define INLINE

#include <math.h>
#include <vector.h>
#include <algo.h>
#include <iostream.h>
#include <iomanip.h>


#include <CoTypes.H>
#include <CoLou.H>
#include <CoBabelInOut.H>
#include <CoGeometryASA.H>
#include <CoLouAssignTypes.H>
#include <CoTranslateTypesCFF93.H>

typedef lou_compound compound;
typedef asa_cost_function cost_evaluation;


typedef ums_compound compound_for_coord_adaptor;
typedef  internal_parameters<compound_for_coord_adaptor> coordinate_adaptor_t;

typedef geometry_optimization_ASA< coordinate_adaptor_t::iterator,
              cost_evaluation>  optimizer_t;

template<class optimizer, class container>
void calculateParameterMinMax(optimizer *opt,container cp);

void print_para(double value) {

cout << value << newl;
}

// we want to avoid creation, initialization and destruction
// of lou_compound objects. We use a static global reference
// as ASA code doesn't provide a way to pass user defined
// data to the cost function.

static compound energy_eval_compound=CoLouCompound() ;

static compound_for_coord_adaptor *coord_adaptor_compound=NULL;
static coordinate_adaptor_t *our_global_cp=NULL;
static lou_energy_evaluator<compound> cff93_energy(energy_eval_compound);
static double *a_min_max;

void print_atom(const lou_atom &at )  {

    cerr << "at : " << at.type() << " x: " << at.x()
         << " y: " << at.y()
         << " z: " << at.z()
         << newl;
}

double
internal_cff93_cost_function (double *x,
               double *,
               double *,
               double *,
               double *,
               long int   * parameter_dimension,
               int *,
               int *cost_flag,
               int *,
```

```
                USER_DEFINES * OPTIONS)
{
//    cout << "cff93_cost_function"<<newl;

    // put the parameters back into the compound.
    // cp takes care of putting the parameters at
    // the right place.

//     cout <<" new parameters: ====================" << newl;
//    for_each(x, x+*parameter_dimension, print_para);

     copy(x,x+*parameter_dimension,(*our_global_cp).begin());

     coord_adaptor_compound->internals().toCartesian( coord_adaptor_compound->ato
ms(),
                                                      coord_adaptor_compound->bon
ds());

     copy_atoms(coord_adaptor_compound->atoms(),energy_eval_compound.atoms());

     optimizer_t *opt= (optimizer_t*)OPTIONS->Asa_Data;

     ccs_energy energy=cff93_energy.energy();

     if ( ! energy.isPhysical()) {
         // reject this configuration
         *cost_flag=FALSE;
         return 10000;
     }
     if (OPTIONS->Best_Cost && energy.value() < *(OPTIONS->Best_Cost)){
         cout << "energy: " << energy << newl;
         cout.flush();
     }
     if (OPTIONS->Locate_Cost == 12){
         calculateParameterMinMax(opt,*our_global_cp);
     }
     *cost_flag=TRUE;
     return energy.value();
};

int print_internal_coord(ums_internal_coord ic ,ums_compound c){

     const ums_atom_iterator nil=c.atoms().end();
     const ums_atom_iterator start=c.atoms().begin();
     const ums_atom_iterator first=ic.first(c);
     const ums_atom_iterator second=ic.second(c);
     const ums_atom_iterator third=ic.third(c);
     const ums_atom_iterator fourth=ic.fourth(c);

     cout  <<first-start << " " <<  (*first).type() << " " ;
     int coord=0;
     if (second!= nil)     { cout <<second-start << " " <<  (*second).type() << "
"; coord++;}
     if (third!= nil)      { cout <<third-start << " " <<  (*third).type() << " ";
 coord++;}
     if (fourth!= nil)     { cout <<fourth-start << " " << (*fourth).type() << " "
; coord++;}
     if (coord >=1) cout << "    r() " << ic.r();
     if (coord >=2) cout << " bend() " << ic.bend();
     if (coord >=3) cout << " torsion() " << ic.torsion();

         cout << newl;
    return 0;
}

#define newl "\n"

main(int ac, char **av) {
```

```
      if (ac <3) {

        cout << "usage : tBabel -i<input format> file " << newl;
        exit(0);
    }
    if (strncmp(av[1],"-i",2)!=0) {
        cout << "usage : tBabel -i missing " << newl;
        exit(0);
    }
    char *input_filename;
    char input_format[20];

    strcpy(input_format,&av[1][2]);
    input_filename=av[2];
    coBabelInOut bab;
    coUMSCompound *comp;


    comp=bab.read(input_filename,input_format);
    if (!comp) {
        cerr << "error : reading " << input_filename
            << " (format : "<<input_format<<")" << newl;
        exit(20);
    }
      {
          { //assign cff93 types:
              coActionPatternAssignTypes<ums_compound> patmatch(*comp,
                                                                ExpandC3,
                                                                sizeof(ExpandC3)
);

              patmatch.assignTypes();

// The following ugly declaration creates an object called transl
// specialized to tranlate from babel to CFF93,
// source and destination compounds are ums_compound
              TranlationTypeEquivTableCFF93(ums_compound,ums_compound,transl);

              transl.translate(*comp,*comp);
          }


          energy_eval_compound.atoms().reserve(comp->atoms().size());
          energy_eval_compound.bonds().reserve(comp->bonds().size());

          coord_adaptor_compound =new compound_for_coord_adaptor();
          coord_adaptor_compound->atoms().reserve(comp->atoms().size());
          coord_adaptor_compound->bonds().reserve(comp->bonds().size());
          transfer(*comp,*coord_adaptor_compound);
          fflush(stdout);

          transfer(*comp,energy_eval_compound);

          our_global_cp=new coordinate_adaptor_t(*coord_adaptor_compound);
          coordinate_adaptor_t &cp=*our_global_cp;
          cff93_energy.compound(energy_eval_compound);

          if (cff93_energy.assignForceFieldParameters()
              == 0) {
              cout << "Error: something got wrong while assigning ForceField typ
es to the compound " << newl;
              exit(10);
          }

          CoLouCompound().consolidateRepresentation();

          cout << "Energy from initial configuration: "
               << cff93_energy.energy()
```

```
                << newl;

        if (ac<4) {
            cerr << " usage: " << av[0] << " arg 3: <asa_option_filename>" << "\
n";
            exit(10);
        }
        string asa_opt_fn(av[3]);

          optimizer_t opt(&internal_cff93_cost_function);
          opt.readOptions(asa_opt_fn);

          int  paras_number=cp.size();
          a_min_max=  new double[paras_number]; //reserve space to construct min
_max
          opt.reserve(paras_number); //reserve storage for   parameters.
          // parameter bounds before import into opt.
//        for_each(cp.begin(), cp.end() , print_para);

          opt.parameters.reserve(cp.size());
          opt.para_lower_bounds.reserve(cp.size());
          opt.para_upper_bounds.reserve(cp.size());

          copy(cp.begin(), cp.end(),opt.parameters.begin());

          calculateParameterMinMax(&opt,cp);
          opt.startFromInitialConfiguration(true);
          opt.optimize(); //optimize according to options.
          opt.printExitReason(cout);
          cout << "final energy: " << opt.bestCost() <<newl;
          cout.flush();

          int again=1;
          double last_cost=opt.bestCost();
          do{

              copy(opt.bestParameters(),
                   opt.bestParameters()+paras_number,
                   opt.parameters.begin());
              // best configuration found in the previous ASA run as
              // the starting configuration.
              calculateParameterMinMax(&opt,cp);
              opt.optimize(); //optimize according to options.


              opt.printExitReason(cout);
              cout << "final energy: " << opt.bestCost() <<newl;
              if (opt.bestCost() <(last_cost -abs(last_cost/10)))
                  { opt.startFromInitialConfiguration(true);
                  cout <<" keeping last configuration" << newl;
                  last_cost=opt.bestCost();
                  }
              else {
                  opt.startFromInitialConfiguration(false);
                  cout <<" not keeping last configuration, starting from random"
 << newl;
                  last_cost=opt.bestCost()+10e9; // be sure we will continue
                  //from the new random->refined conformation.
              }

              cout.flush();
          } while (--again);

//      opt.writeTestFile(); // writes a file that conforms itself to
        // ASA original user_out file. with the result of the optimization.
        // useful for checking out if the OO wrapper works like the original
        // ASA routines..

          delete []a_min_max;
          double *best_parameters=opt.bestParameters();
```

```
            copy(best_parameters,best_parameters+paras_number,cp.begin());
            coord_adaptor_compound->internals().
                toCartesian( coord_adaptor_compound->atoms(),
                             coord_adaptor_compound->bonds());

            copy_atoms(coord_adaptor_compound->atoms(),comp->atoms());
            bab.write(comp,"out.car","mdl");
          exit(0);
    }

}

double set_min_value(coordinate_adaptor_t::reference &it) {
    switch (it.isAngle()) {
    case false: return double(it); // min bond length;
    case true:
        switch (it.isBending()) {
        case true:
            return (double(it)); // - 5 deg. less that the original bending angl
e.
        case false: //torsion
            return -180; // min angle;
//          return double(it)-50; // min angle;
        }
    }
}

double set_max_value(coordinate_adaptor_t::reference &it) {
    switch (it.isAngle()) {
    case false: return double(it); // max bond length;
    case true:
        switch (it.isBending()) {
        case true:
            return (double(it)); // + 5 deg. less that the original bending angl
e.
        case false: //torsion
//          return double(it)+50;
            return +180; // min angle;
        }    }
}

template<class optimizer, class container>
void calculateParameterMinMax(optimizer *opt,container cp){

    int paras_number=cp.size();
//    cout << "paras_number" << paras_number <<newl;

    transform(cp.begin(),cp.end(),
              opt->para_lower_bounds.begin(),
              set_min_value);



    transform(cp.begin(),cp.end(),
              opt->para_upper_bounds.begin(),
              set_max_value);


}
#endif
```

F

Composant de paramétrisation

Le lecteur intéressé trouvera ici l'implantation de la partie conteneur du composant de paramétrisation.

```
#ifndef CO_GEOMETRY_ASA_H
#define CO_GEOMETRY_ASA_H

#include <CoASA.H>

#include <iterator.h>


template <class compound>
class internal_compound_iterator
    : public random_access_iterator<double,ptrdiff_t> {
public:
    typedef double &reference;
    typedef compound::internal_container::iterator internal_it;
    typedef internal_compound_iterator self;


/*    friend ostream& operator<<(ostream& out, cartesian_compound_iterator &it){
        return out << it.at << "coord: " <<it.coord << newl;
    }
    */
    internal_compound_iterator(const internal_it &a,unsigned int coordinate) : a
t(a),
        coord(coordinate) {}
    const self &operator=(self &rhs) {
        coord=rhs.coord;
        at=rhs.at;
        return *this;
    }
    const self operator++(int) {

        self tmp=*this;
        if (((signed int)coord) <
            (((signed int)(*at).parameterNumber())-1)) { coord++; }
        else { at++; coord=MIN_COORD; }
        return tmp;
    }
    const self operator++() {

        if (((signed int)coord) <
            (((signed int)(*at).parameterNumber())-1)) { coord++; }
        else { at++; coord=MIN_COORD; }
        return *this;
    }
    const self  operator+(int inc) {
        self tmp(*this);


        tmp.at += ((coord+inc)/3);
        tmp.coord = ((coord+inc) % 3) ;
        return tmp;
    }
    int operator-( self rhs) {
        return (at - rhs.at) *3 + (coord - rhs.coord);
    }
    bool operator==( const self rhs) const {
        if (at != rhs.at) return 0;
        if (coord != rhs.coord) return 0;
        return 1;
    }
    internal_it internal() { return at; } // used to find the internal to which
    // the parameter belongs to.

    bool isAngle() {return coord != R_COORD;}
    bool isBondLength() {return coord == R_COORD;}

    class ici_proxie {
    private:
        internal_it at;
        unsigned int coord;
```

```
    public:

        ici_proxie(internal_compound_iterator &it)
            :at(it.at),coord(it.coord) {}


        ici_proxie &operator=(double value) { //lvalue use
            switch (coord) {
            case R_COORD:
                (*at).r(value);
            case BEND_COORD:
                (*at).bend(value);
            case TORSION_COORD:
                (*at).torsion(value);
            default: ;
            }
            return *this;
        }
        operator double() const { //rvalue use

             switch (coord) {
             case R_COORD:
                 return (*at).r();
             case BEND_COORD:
                 return (*at).bend();
             case TORSION_COORD:
                 return (*at).torsion();
             default:
                 return 0.0;
             }
         }
        operator double() { //rvalue use

             switch (coord) {
             case R_COORD:
                 return (*at).r();
             case BEND_COORD:
                 return (*at).bend();
             case TORSION_COORD:
                 return (*at).torsion();
             default:
                 return 0.0;
             }
         }
        bool isAngle() {return coord != R_COORD;}
        bool isBondLength() {return coord == R_COORD;}
        bool isBending() {return coord == BEND_COORD;}
        bool isTorsion() {return coord == TORSION_COORD;}
    };
    ici_proxie operator*()  {
        return ici_proxie(*this);
    }

    friend class cci_proxie;


private:
        static unsigned const int R_COORD=0,BEND_COORD=1,TORSION_COORD=2;
        static unsigned const int MAX_COORD=TORSION_COORD;
        static unsigned const int MIN_COORD=R_COORD;

        internal_it at;
        unsigned int coord;

};
template <class compound>
class internal_parameters {

public:
    typedef internal_compound_iterator<compound> iterator;
```

```
    typedef const internal_compound_iterator<compound> const_iterator;
    typedef internal_compound_iterator<compound>::ici_proxie reference;
    typedef internal_parameters self;
    internal_parameters(const compound &c) : comp(c) {};
    static unsigned const int MIN_COORD=0;
    static unsigned const int MAX_COORD=3;
    iterator begin() const { return ++iterator(comp.internals().begin(),MIN_COOR
D); }
    iterator end()  const { return const_iterator(comp.internals().end(),MIN_COO
RD);};
    const_iterator begin()  { return ++iterator(comp.internals().begin(),MIN_COO
RD); }
    const_iterator end()   { return iterator(comp.internals().end(),MIN_COORD);}
;
    unsigned int size() const {
        return comp.internals().size()*3 -6; // 3n -6 valid parameters
    }
    unsigned int size()  {
//        cerr <<"count for size: begin()"  <<  "calculated size: "
//<<comp.internals().size()*3 -6<< newl;
        return comp.internals().size()*3 -6; // 3n -6 valid parameters
    }
    self operator=(self &rhs) { comp=rhs.comp; }

private:
    const compound &comp;
};

template <class compound>
class cartesian_compound_iterator
    : public random_access_iterator<double,ptrdiff_t> {
public:
    typedef double &reference;
    typedef compound::atom_container::iterator atom_it;
    typedef cartesian_compound_iterator self;


/*    friend ostream& operator<<(ostream& out, cartesian_compound_iterator &it){
        return out << it.at << "coord: " <<it.coord << newl;
    }
    */
    cartesian_compound_iterator(const atom_it &a,unsigned int coordinate)
        : at(a),
          coord(coordinate) {}

     self &operator=(self &rhs) {
        coord=rhs.coord;
        at=rhs.at;
        return *this;
    }
     self operator++(int) {

        self tmp=*this;
        if (coord <MAX_COORD) { coord++; }
        else { at++; coord=MIN_COORD; }
        return tmp;
    }
    self operator++() {
        if (coord <MAX_COORD) { coord++; }
        else { at++; coord=MIN_COORD; }
        return *this;
    }
     self  operator+(int inc) {


        self tmp(*this);

        tmp.at += ((coord+inc)/3);
        tmp.coord = ((coord+inc) % 3) ;
        return tmp;
```

```
    }
    int operator-( self rhs) {
        return (at - rhs.at) *3 + (coord - rhs.coord);
    }
    bool operator==( const self rhs) const {
        if (at != rhs.at) return 0;
        if (coord != rhs.coord) return 0;
        return 1;
    }
    atom_it atom() { return at; } // used to find the atom to which
    // the parameter corresponds.

    class cci_proxie {
    private:
        atom_it at;
        unsigned int coord;
    public:

        cci_proxie(cartesian_compound_iterator it)
            :at(it.at),coord(it.coord) {}


        cci_proxie &operator=(double value) { //lvalue use
            switch (coord) {
            case X_COORD:
                (*at).x(value);
            case Y_COORD:
                (*at).y(value);
            case Z_COORD:
                (*at).z(value);
            default: ;
            }
            return *this;
        }
        operator double() const { //rvalue use

             switch (coord) {
             case X_COORD:
                 return (*at).x();
             case Y_COORD:
                 return (*at).y();
             case Z_COORD:
                 return (*at).z();
             default:
                 return 0.0;
             }
         }
        operator double() { //rvalue use
             switch (coord) {
             case X_COORD:
                 return (*at).x();
             case Y_COORD:
                 return (*at).y();
             case Z_COORD:
                 return (*at).z();
             default:
                 return 0.0;
             }
         }

    };
    cci_proxie operator*()  {
        return cci_proxie(*this);
    }

    friend class cci_proxie;


private:
         unsigned const int X_COORD=0,Y_COORD=1,Z_COORD=2;
```

```
         unsigned const int MAX_COORD=Z_COORD;
         unsigned const int MIN_COORD=X_COORD;
        atom_it at;
        unsigned int coord;

    };

template <class compound>
class cartesian_parameters {

public:
    typedef cartesian_compound_iterator<compound> iterator;
    cartesian_parameters(const compound c) : comp(c) {};
    unsigned const int X_COORD=0;
    unsigned const int Y_COORD=1;
    unsigned const int Z_COORD=2;
    iterator begin() const { return iterator(comp.atoms().begin(),X_COORD); }
    iterator end()  const { return iterator(comp.atoms().end(),X_COORD);};
    unsigned int size() const { return comp.atoms().size() *3; /* 3n cartesian c
oordinates */ }

private:
    compound &comp;
};

template <class parameter_iterator,
          class cost_function_ptr,
          class minmax_iterator=double*>
class geometry_optimization_ASA
    : public asa_optimization_algorithm <parameter_iterator,
                                         cost_function_ptr,
                                         minmax_iterator> {

public:


        geometry_optimization_ASA(cost_function_ptr cfp)
            :asa_optimization_algorithm<parameter_iterator,
            cost_function_ptr,
            minmax_iterator>(cfp)  {

        }
        void optimize(void) {
            mandatoryOptions();
            supertype::optimize();
        }
private:
        typedef asa_optimization_algorithm <parameter_iterator,
            cost_function_ptr,
            minmax_iterator> supertype;

        void mandatoryOptions(void) {
            /* place here options that have to override

            1/ default parameters
            2/ option_file parameters

            because they are crucial for the behavior of this
            class.
            */


            USER_DEFINES *OPTIONS=&_USER_OPTIONS;

            OPTIONS->Curvature_0=1;
//          OPTIONS->User_Initial_Parameters=FALSE;
//          OPTIONS->User_Initial_Parameters=TRUE;
//          OPTIONS->Maximum_Cost_Repeat=0;
            initial_cost_temperature=100;
            OPTIONS->Temperature_Anneal_Scale=parameterNumber() *1.5;
```

```
            initial_cost_temperature=2000;
        }
};

#endif //CO_GEOMETRY_ASA_H
```

Troisième partie

A propos d'un système d'aide à l'analyse de données biologiques

8

La tâche d'analyse de données biologiques

Ce chapitre présente une description de l'activité et des outils d'analyse de données (biologiques). Nous construirons cette description en présentant plusieurs approches que nous pensons complémentaires. Dans le prochain chapitre, munis de la description de la tâche d'ADB que nous aurons construite, nous serons en mesure d'expliciter le cahier des charges d'un système d'aide à l'ADB.

8.1 L'analyse de données biologiques en question

Pour trouver les moyens d'accélérer une tâche, il est primordial de déterminer les étapes coûteuses — en temps — puis, ensuite, de chercher les moyens de les optimiser. Dans le cadre de l'aide à l'analyse de données biologiques, il peut être utile de s'asseoir à côté d'un utilisateur aux prises avec un problème particulier et de noter la nature des étapes qui lui demandent plus de temps que d'autres. Que l'on réalise cette expérience en prêtant attention aux étapes les moins intéressantes, du point de vue de l'utilisateur : "quand ça n'avance pas" et l'on pourra s'apercevoir que ces étapes n'ont pas grand chose à voir avec ce qui est habituellement dénommé "analyse de données". Mais peut-être que l'analyse de données biologiques est une activité qui regroupe d'autres aspects que ceux qu'englobe l'analyse de données.

Examinons brièvement quelques points de terminologie. Dans le cadre de ce document, on parlera d'analyse de données (AD) pour désigner des méthodes type "détermination des composantes principales", ou autres (cf. [Ben73] pour une description de méthodes de ce type), qui s'appliquent à des données de forme simple (tableaux de nombres, par exemple).

De la même manière, l'approche statistique propose des méthodes théoriques pour analyser des ensembles de données de forme attendue (valeurs de variables numériques, continues ou discrètes). Les calculs de moyennes, écarts types, régressions linéaires, etc. sont des exemples de méthodes définies par cette discipline. Les approches statistiques enseignées dans la plupart des disciplines scientifiques ne disent rien de la façon de réunir les données ou d'implanter les outils théoriques pour réaliser l'analyse en pratique. D'une certaine façon, que l'analyse statistique soit conduite à l'aide d'un papier et d'un crayon, d'un boulier, ou d'un ordinateur, ne changera rien aux méthodes statistiques (et bien sûr au résultat de l'analyse).[33]

Ceci comporte plusieurs avantages, notamment celui de la généralité, mais aussi, et nous nous intéresserons maintenant plutôt à cet aspect des choses, des inconvénients partagés par les méthodes d'analyse de données. Par exemple, l'approche statistique contiendrait les solutions aux problèmes

33. Notons néanmoins que les chercheurs de ce domaine peuvent avoir des préocupations plus larges. Nous nous intéressons ici aux méthodes statistiques telles qu'elles peuvent-être utilisées, en routine, pendant l'activité d'analyse de données biologiques.

d'analyse de données[34] si l'analyste était un être idéal et intemporel. En effet, un problème de l'activité d'analyse de données est de réaliser les analyses pertinentes (peut-être en nombre important, ou de mise en oeuvre difficile) dans le temps imparti à l'activité d'analyse, qui est borné. Comment donc faciliter la pratique de l'analyse de données ? Ou pour expliciter le terme "la pratique", comment choisir rationnellement les moyens de faciliter le processus d'analyse de données en tenant compte notamment,

- de l'accès aux données et...
- de l'analyste.

Les travaux de recherche sur l'extraction de connaissances dans les bases de données (ECBD)[35], peuvent nous aider à mieux cerner les problèmes soulevés par l'activité d'analyse de grandes quantité de données. C'est en considérant les différences entre l'activité d'ECBD et l'activité d'analyse de données biologiques (ADB), telle que le programme Viseur permet de la réaliser, que nous décrirons comment la tâche d'ADB se différencie de la tâche d'ECBD. Cette approche nous amènera à proposer un modèle de l'activité d'ADB, utile pour concevoir les outils logiciels qui faciliteront la réalisation de cette activité.

8.2 Extraction de Connaissances dans les Bases de Données (ECBD)

Une première façon de définir l'ECBD est de préciser ses buts. Cette discipline a l'ambition d'aider à la découverte de connaissances à partir d'une quantité, potentiellement très importante, de données.

8.2.1 Mais qu'est-ce que la connaissance ?

Il n'existe pas de réponse établie à cette question. La connaissance peut-elle se réduire à sa forme, ou contient-elle le sens ? La réponse à cette question conditionne la possibilité qu'une machine formelle (un ordinateur par exemple) puisse simuler les processus de la pensée. Jacques Arsac pose ce problème dans un article [Ars98] du *Dictionnaire de l'ignorance* [Sci98] et conclut, à défaut d'une réponse scientifique, par l'énoncé de sa croyance personnelle.

Qu'aucune réponse scientifique ne soit disponible pour nous éclairer sur la nature de la connaissance semble ne pas simplifier le problème de l'ECBD qui cherche à l'extraire des données. En réalité, les chercheurs en ECBD considèrent que la connaissance est formelle. Par exemple, Yves Kodratoff définit la connaissance, dans le cadre de l'ECBD, à partir du concept de *bloc de connaissance* :

« Un bloc de connaissance est une phrase dans un langage pré-défini qui présente les propriétés suivantes :

1. Cette phrase contient deux sortes d'informations. Une partie est la connaissance elle-même (par exemple une implication et un coefficient de croyance dans cette implication) et l'autre partie est la méta connaissance expliquant comment on peut utiliser la connaissance, par exemple en fixant les contextes dans lesquels une implication est intéressante à employer et ceux dans lesquels elle est inutile, bien que l'implication soit valable partout.
2. Il existe au moins un agent, un être vivant, ou une machine, qui se comportera différemment dans le monde selon qu'il possède ou non ce bloc de connaissance. Le degré de pertinence d'un bloc de connaissance pour une action particulière est souvent le sujet de nombreuses discussions. Cet agent a des buts qui déterminent l'emploi et la pertinence de la connaissance.

».

Cette définition sous-entend l'utilisation de représentations formelles des blocs de connaissances. La figure 8.1 illustre la différence entre la connaissance au sens de l'ECBD (sens IA) et la connaissance au sens commun.

34. Nous verrons plus loin que les approches statistiques n'apporteraient une solution que partielle.
35. Le sigle anglais est KDD pour *Knowledge Discovery in Databases*.

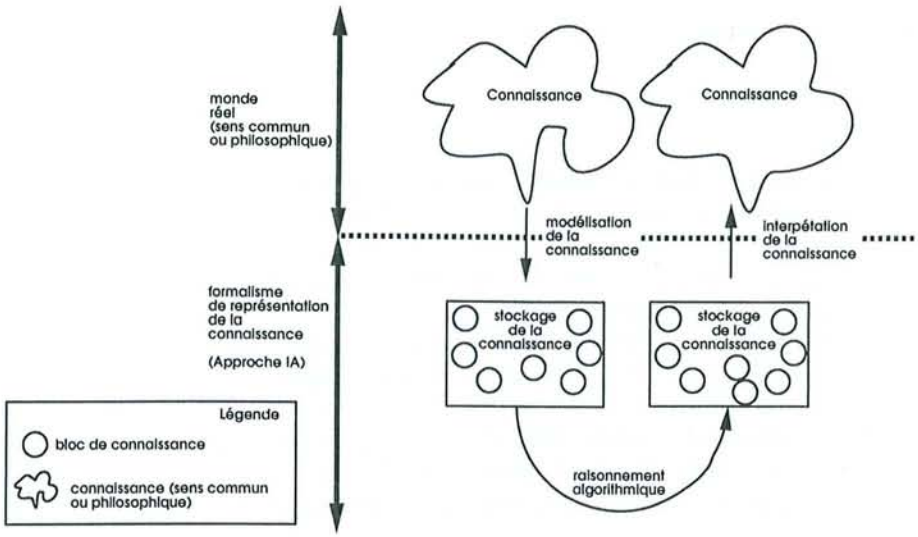


FIG. 8.1 – *Modélisation de la connaissance en IA*

Notons que les recherches d'ECBD s'appuient sur différents formalismes de représentations de la connaissance parce qu'aucun formalisme n'est suffisament général pour permettre de traiter la totalité des problèmes. Ces formalismes sont, par exemple, l'utilisation de règles ou clauses inspirés de la logique du premier ordre, les réseaux sémantiques, etc. (voir [MNC+89] et références incluses pour une description des formalismes de représentation des connaissances à l'origine de plusieurs langages à objets).

L'ECBD aide à la découverte de connaissances en sélectionnant pour l'analyste les connaissances *pertinentes et intelligibles* extraites des données. Il est important de comprendre en effet qu'il est possible de dériver beaucoup d'informations secondaires d'un petit ensemble de données. Il est facile de programmer un ordinateur pour obtenir ce résultat. Il est par contre beaucoup plus difficile de déterminer les blocs de connaissances *pertinents* (suffisamment fiables, utilisables, en accord avec les buts de l'analyste) puis de les présenter sous une forme *intelligible* (présentation adaptée à celui qui doit les comprendre).

Après avoir présenté les buts de l'ECBD, passons rapidement en revue ses méthodes.

8.2.2 Les méthodes de l'ECBD

D'après Kodratoff, L'ECBD utilise et adapte les méthodes de domaines de recherche précédemment séparés : les Bases de Données, les Statistiques, l'Apprentissage Symbolique Automatique (ASA). Il faut sans doute ajouter à cette liste la Visualisation qui développe les méthodes de présentation d'une information intelligible. Voici une description rapide des buts très différents vers lesquels se sont développés ces domaines de recherche en dehors de considérations d'Extraction des Connaissances dans les Bases de Données.

Les Bases de Données : stockage et accès performant aux données, gestion des conflits de mise à jour et d'accès, développement d'un langage de requête déclaratif pour retrouver les données répondant à certains critères;

Les Statistiques : calcul de descripteurs rigoureux pour comparer ou décrire des ensembles de grandeurs équivalents mais différents, découverte de règles d'évolution des données ou de relations entre elles, etc.

L'Apprentissage Symbolique Automatique : les buts de l'Apprentissage Automatique Symbolique (qui traite de symboles, un sur ensemble qui contient les chiffres) est d'apprendre à partir des données et de présenter le résultat de l'apprentissage à l'utilisateur de manière intelligible;

La Visualisation : l'intelligibilité est un but essentiel de la Visualisation qui cherche les méthodes de représenter des données pour faciliter leur interprétation. On observe aussi, avec les systèmes de visualisation hypertexte, le nouveau besoin de trouver les meilleures solutions pour visualiser le parcours dans une information potentiellement trop vaste pour être appréhendée en une fois.

De tous ces domaines, l'Apprentissage Automatique Symbolique (ASA) est sans doute le moins familier des lecteurs intéressés par la bioinformatique, bien que ses méthodes soient applicables aux problèmes biologiques. L'ASA regroupe l'ensemble des méthodes de représentation de connaissances (cf. figure 8.1) et regroupe donc des méthodes de modélisation de la connaissance et des méthodes de raisonnement à partir des connaissances modélisées.

Les recherches en ASA tentent de proposer des méthodes formelles permettant d'extrapoler de nouvelles données à partir de données de départ. L'ASA propose des méthodes adaptées à certains contextes ou à certaines données. Par exemple, certaines données sont adaptées aux méthodes de classification (détermination d'un ensemble de classes plus petit que le nombre de données, cf. figure 8.2), d'autres à des méthodes de détermination de règles (si ... alors ...), d'autres encore à la détermination d'arbres de décision (aide au choix d'une alternative parmi beaucoup), etc.

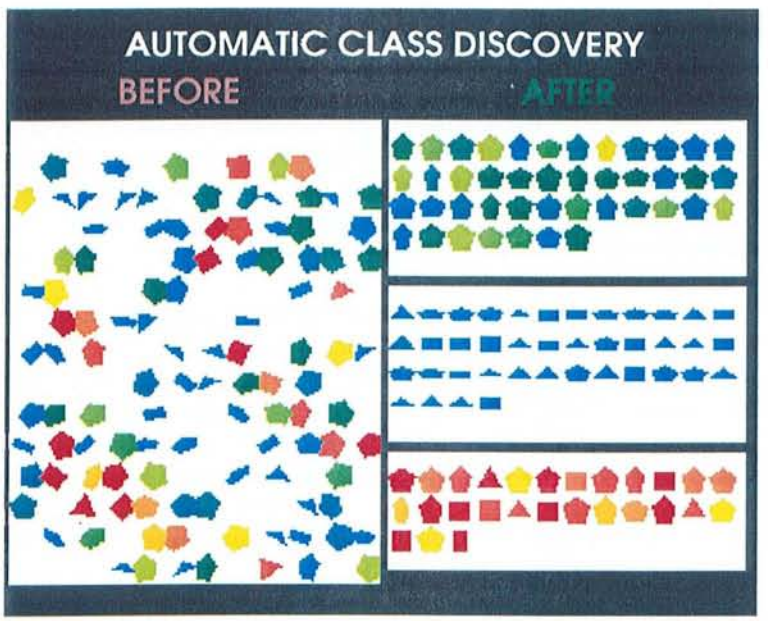


FIG. 8.2 – *Illustration visuelle de la classification automatique*
Les algorithmes de classifications automatiques déterminent des classes de données et l'appartenance des données aux classes. Le côté gauche de la figure présente un ensemble d'objets possédant des propriétés diverses (taille, couleur, aire, etc.). Le côté droit de la figure présente une partition automatique de ces objets en trois classes. Illustration extraite de [Aut98].

L'examen des buts des domaines de recherches que nous avons repris ici montre que ces quatre disciplines ont évolué pour répondre à des besoins différents, et que l'ECBD, pour réutiliser leurs résultats, doit les adapter à ses buts [Kod97].

Il n'est pas suffisant de juxtaposer les méthodes de ces quatre domaines de recherche. Pour le comprendre, considérons la réaction d'un utilisateur qui se voit proposer les résultats statistiques (très significatifs) de l'analyse d'une grande quantité de données (extraites d'un SGBD[36]) dont il n'avait jamais pris connaissance auparavant. Dans un premier temps, si l'utilisateur n'est pas familier des outils statistiques utilisés, il peut ne pas comprendre que les résultats qui lui sont présentés sont très significatifs. Dans le cas favorable où l'utilisateur a déjà rencontré les résultats d'analyses statistiques équivalentes, il peut ne pas les reconnaître immédiatement : présentation des résultats inhabituelle pour lui, ou

36. Système de Gestion de Bases de Données, *DataBase Management System (DBMS)*.

inadaptée aux données (tableau de plus de 200 nombres plutôt qu'une représentation graphique "équivalente"). Enfin, la connaissance contenue dans le résultat de l'analyse statistique peut-être brouillée par des informations *statistiquement exactes*, mais que l'utilisateur considérera accessoire dans cette phase de son analyse.

Les Statistiques ne répondent pas au problème de la sélection des informations pertinentes, la Visualisation propose des techniques de représentation mais ne précise pas quand et comment les utiliser pour représenter les données. Les recherches d'ECBD essaient de résoudre ces problèmes.

8.2.3 Quelques résultats des études d'ECBD

Définition	**ECBD**
Les recherches dans le domaine de l'Extraction de Connaissances à partir des Bases de Données (ECBD) essaient d'aider à extraire des connaissances (formelles) intéressantes et intelligibles à partir des données stockées dans une base de donnée, de préférence de façon automatique ou semi-automatique.	

Brachman et Anand [BA96] insistent sur le rôle central que joue l'utilisateur dans l'analyse de données et sur la distinction nécessaire entre les applications d'ECBD et les environnements d'aide à l'ECBD. Les premières permettent à un utilisateur non spécialiste de l'ECBD, mais spécialiste de son domaine, d'obtenir de nouvelles connaissances à partir de nouveaux jeux de données. Les seconds permettent de construire les applications d'ECBD pour des buts d'extractions plus spécifiques, mais requièrent de l'utilisateur une compréhension des méthodes d'ECBD.

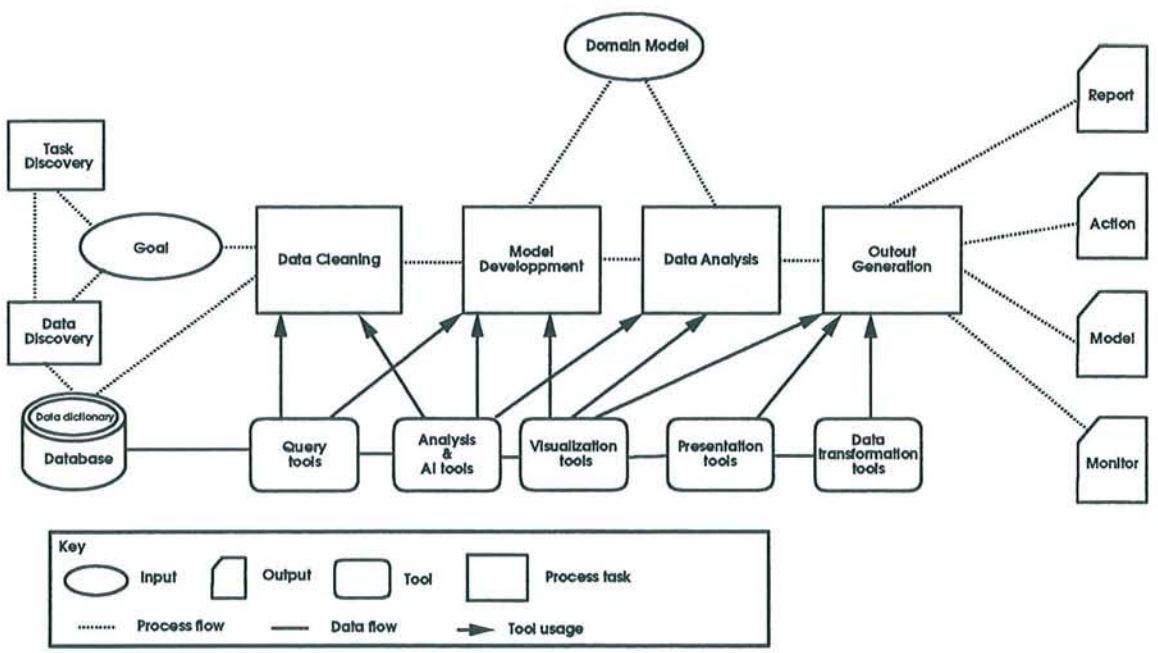


FIG. 8.3 – *Le processus d'extraction de connaissances dans les bases de données*
Reproduit à partir de "The Process of Knowledge Discovery in Databases: A Human-Centered Approach" Ronald J. Brachman and Tej Anand in Advances in Knowledge Discovery and Data Mining, Usame M. Fayad et al., editors, AAAI Press / The MIT Press, 1996.

La figure 8.3, d'après [BA96], décrit le processus d'Extraction de Connaissances *à partir d'une base*

de données. Dans la suite de cette sous-section, nous traduisons et explicitons les termes correspondants aux tâches ou aux outils du processus d'ECBD qui apparaissent sur cette figure.

Domain Model. Le modèle du domaine, c'est à dire le formalisme de représentation des connaissances retenu pour analyser les données du domaine.

Database. Le système de gestion de base de donnée qui contient les données à analyser.

Data Cleaning. Nettoyage des données. C'est la phase de l'analyse au cours de laquelle l'analyste découvre si les données peuvent permettre de réaliser le type d'analyse qu'il cherche à mener à bien, et si non, les corrige ou restreint les données qu'il prendra en compte. Les problèmes qui peuvent rendre nécessaire un nettoyage des données incluent la présence de bruit dans les données, dû à un prétraitement inadéquat, à des données incomplètes, etc.

Statistics and AI tools. Les outils statistiques, d'analyse de données et d'IA (ASA essentiellement) à disposition de l'analyste.

Model Development. La phase de développement du modèle. Précisons ici que le sens du mot modèle, dans le cadre de l'ECBD, désigne un modèle empirique, exprimé dans le formalisme retenu (*Domain Model*). Il peut donc s'agir de déterminer des règles, des classes, mais aussi des modèles mathématiques empiriques (régressions linéaires par exemple), etc.

Data Analysis. La tâche d'analyse de données ressemble à l'activité d'analyse de donnée confirmative : l'analyste cherche à confirmer ou infirmer une hypothèse à propos des données et utilise les méthodes et outils à sa disposition, dans ce but. Cette phase comprend le choix du modèle (ou sa spécification), la spécialisation éventuelle du modèle et son évaluation par rapport aux données.

Visualization tools. Les outils de visualisation qui permettent de visualiser les données dans le cadre des modèles.

Output Generation. Création de sorties. Les résultats de l'analyse sont mis en forme et préparés pour faciliter leur compréhension par les personnes à qui ils sont destinés.

Presentation tools. Les outils de présentations permettent de réaliser des présentations de données de qualité, pour diffusion des résultats de l'analyse.

Data transformation tools. Des outils de transformation de données pour supprimer certaines données des présentations ou modifier leur apparence, par exemple.

8.2.4 L'activité d'ADB vue à travers le processus d'ECBD

Les lecteurs familiers de l'analyse de données biologiques auront remarqué qu'il existe de fortes ressemblances entre l'ADB et le processus d'ECBD tel que décrit précédemment. Explicitons cette ressemblance en reprennant les étapes ou outils du processus d'ECBD et en décrivant leur équivalent ADB. Pour les besoins de la comparaison nous prendrons l'exemple du programme Viseur, décrit en première partie de ce document, ou d'autres outils d'aides à l'ADB.

Le programme Viseur n'est pas suffisamment général pour prétendre représenter l'Analyse de Données Biologiques dans son ensemble mais il présente de nombreux éléments décrits par la figure 8.3.

Domain Model. Les "formalismes" de représentation de connaissances proposés par le programme Viseur pour analyser les données du domaines sont essentiellement ... non formels. En effet, rappelons que la *représentation de la connaissance* se compose d'une partie stockage et d'une partie raisonnement (cf. figure 8.1). Or si le programme Viseur stocke les connaissances de façon formelle (c'est un logiciel) il n'implante aucun algorithme de raisonnement à partir de ces connaissances. Ce programme *présente* des connaissances biologiques à l'analyste qui les utilise pour conduire son *raisonnement*. La figure 8.4 illustre ce processus.

Database. Le programme Viseur maintient une banque de données locale (dédiée à un projet).

Data Cleaning. Viseur réalise quelques tests de conformité sur les données garants du bon fonctionnement de ses algorithmes. Le nettoyage des données, au sens de l'ECBD, est le plus souvent réalisé par l'utilisateur, à l'aide des outils de visualisation du programme (lien qui n'apparaît pas sur le schéma 8.3). Notons que le nettoyage de données est aussi une préocupation de l'ADB (outils de tests de la qualité de structures moléculaires, [HVSA96] [HSV96] par exemple).

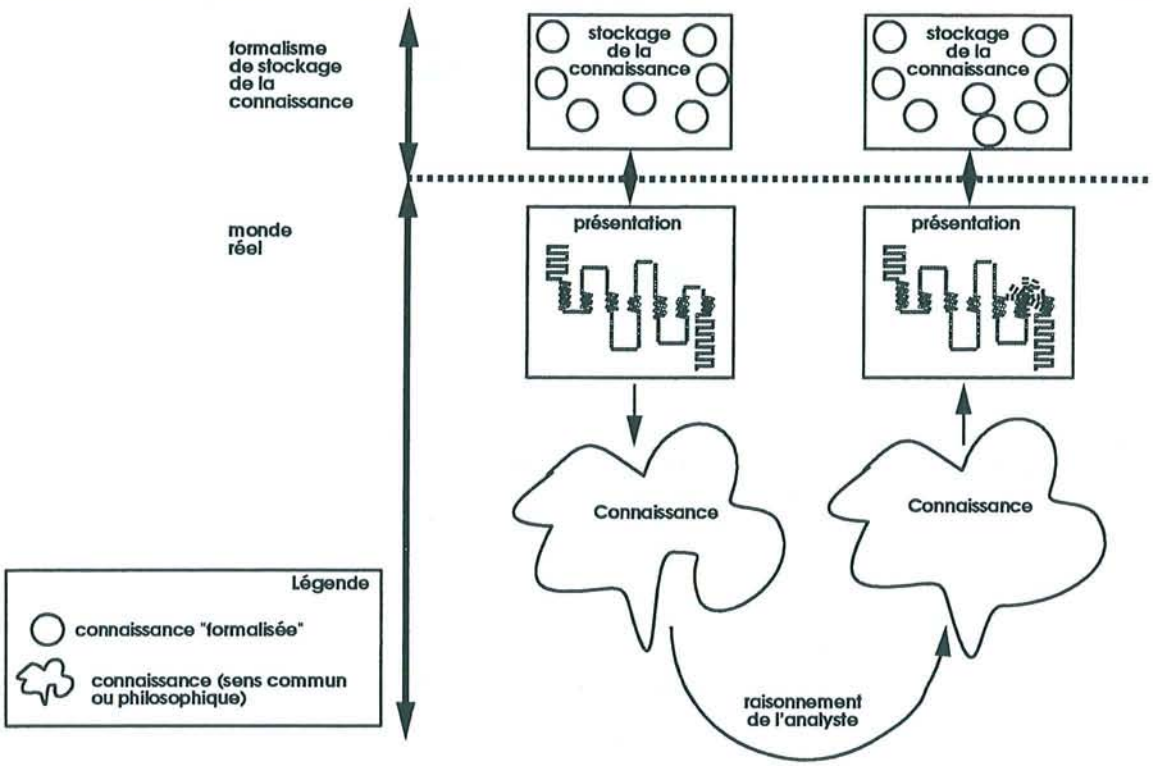


FIG. 8.4 – *Stockage de Connaissances, présentation et raisonnement (outil Viseur)*

Statistics and AI tools. Ces méthodes ne sont pas utilisées par le programme Viseur : les types de modèles auxquels se conforment les données sont connus puisque le programme est bien adapté au système étudié. Par contre, dans les cas où il faut spécifier ou choisir le modèle pendant l'analyse, les méthodes statistiques et d'AD sont aussi utilisées en ADB. Par exemple : utilisation de tests statistiques — calcul des *scores* [ABGW94] — implantés dans l'outil FASTA pour déterminer si l'homologie de deux séquences est significative (choix de modèles d'alignements parmi tous les alignements potentiels)

Model Development. La phase de développement de modèle consiste, dans le cadre de Viseur, à diminuer peu à peu les degrés de liberté du modèle (le champ des possibles) jusqu'à obtenir un modèle spécialisé pour la tâche de modélisation. Par exemple, le modèle en serpent peut potentiellement représenter n'importe quelle protéine à hélices transmembranaires. Fixer la séquence de la protéine spécialise une première fois le modèle. L'utilisateur détermine ensuite la position des domaines transmembranaires (fixe de nouveaux paramètres) et spécialise ainsi encore le modèle. Même chose pour le modèle moléculaire (paramètres : structure de départ, alignement, séquence, rotation/translation des hélices).

Dans le cadre d'un outil plus général, non spécialisé, une première étape consisterait à déterminer un modèle consistant avec les données (spécification ou choix du modèle).

Visualization tools. Viseur propose plusieurs outils de visualisations (diagrammes en serpent, modèle moléculaire,...) très utiles à l'analyse des données. Nous avons déjà remarqué, en première partie, que représentation et modèle sont intimement reliés.

Presentation tools. Les représentations PostScript, HTML ou VRML sont des exemples de fonctionnalités de Viseur qui rentrent dans la catégorie des outils de présentation, c'est à dire les outils qui permettent de préparer la diffusion de résultats intermédiaires ou finaux.

Data Analysis. L'analyse des données est reliée à la spécialisation des modèles dans le cas étudié : l'analyse a lieu pendant que le modèle est raffiné. Elle est obligatoirement relative au modèle considéré (c'est sans doute pour cette raison que le schéma 8.3 place le développement du mo-

dèle avant l'analyse de données, alors que ce sont deux tâches concomitantes).

Output Generation. Lorsque l'analyse est satisfaisante, ou pour discuter des résultats intermédiaires avec des collègues, l'utilisateur réalise des représentations des données (modèle en serpent, d'alignement, moléculaire)

8.2.5 Différences entre l'ECBD et l'ADB

Examinons plus particulièrement les différences entre les étapes des processus d'ADB et celles du processus d'ECBD décrit par [BA96].

Complexité des modèles de données

La première chose à noter est que le niveau de complexité des modèles diffère. Brachman et Anand décrivent un processus d'ECBD qui s'applique à des données dont l'analyste ne connaît, a priori, que peu de choses. Par exemple, ces données peuvent être les chiffres de ventes d'une chaîne de supermarchés et les modèles pourraient être des relations empiriques (numériques ou autres) qui relient les volumes des ventes à la date de la vente, ou à la position des articles dans le magasin. Le programme Viseur offre des modèles théoriques qui se sont constitués lentement au cours de l'évolution des sciences (modèles issus de la théorie atomique, modèle d'évolution des séquences biologiques, etc.). Leur complexité est beaucoup plus grande et il n'est pas envisageable de les redécouvrir à partir des données en quelques heures ou quelques jours, seulement de les spécialiser, comme nous l'avons précédemment expliqué.

Les données des bases de données biologiques (BDB) sont souvent aussi complexes que les modèles qui permettent de les analyser. Or les méthodes d'apprentissage symbolique automatique ont été développées pour des modèles de données relativement simples (la simplicité du modèle de stockage de la connaissance facilite le développement de méthodes de raisonnement formel).

Reprenons l'exemple de la classification non supervisée des données biologiques pour expliquer pourquoi ce type de méthode est peu employé pour l'ADB. Le modèle de données du classifieur AutoClass [CKS+88] permet de représenter toutes données qui peuvent se mettre sous la forme de n-uplets d'attributs. Voici l'exemple reproduit de [CS95] :

```
medical case #8, described as
   (age=23, blood-type=A, weight=73.4kg, ...)
```

Les attributs peuvent être de type énumération (une valeur parmi les éléments pré-définis d'une liste), entiers ou réels. Les attributs structurés[37] ne sont pas autorisés alors qu'ils sont indispensables à la représentation des données biologiques. La représentation la plus simple d'une séquence biologique, par exemple, fait intervenir au moins la séquence de résidus. Cet attribut ne peut pas être modélisé par une énumération : `(sequence=AVALAG...VALEL)` il faudrait que l'énumération possède autant d'éléments qu'il existe de séquences. En d'autres termes, chaque attribut serait unique. Pas de classification possible avec un seul attribut. Une autre représentation : `(AA1=A, AA2=V, AA3=A ...)` ne permet de classer que des séquences de même longueur. C'est cette représentation que les auteurs ont du utiliser pour classer les sites donneurs et accepteurs d'épissage de l'ADN en trois catégories. Ces trois catégories sont C riches, TA riches et G riches respectivement. Ceci signifie que chaque base du site donneur ou accepteur de la première catégorie a une probabilité significativement supérieure à la moyenne d'être une cytosine. Cette propriété est à rapprocher de la notion d'usage du code, dont une présentation plus détaillée est disponible dans [Mos96]. Cette classification automatique de données biologiques est décrite dans [CS95].

Cet exemple montre que certaines techniques d'apprentissage automatique symbolique pourraient être utilisées avec profit pour l'analyse de données biologiques mais que certaines contraintes sur les modèles de données qu'ils supportent freinent leur utilisation — quand ces méthodes ne sont pas simplement ignorées des analystes. Nous montrerons comment contourner ce problème dans la suite de ce document (sous-section 10.1.1).

37. Un attribut stucturé est composé de plusieurs types simples.

Bien que les niveaux de complexité des modèles diffèrent il est appréciable de constater que le processus d'ECBD décrit correspond assez bien au schéma du programme Viseur. Examinons les deux dernières différences : le programme Viseur ne crée que deux types de sorties parmi les quatre indiqués dans le processus d'ECBD.[38] La conception du programme Viseur laisse-t-elle de côté un aspect important de l'analyse de données? Oui et non. Oui, parce que les actions et moniteurs décrits par Brachman et Anand sont des *outils* puissants, très utiles à l'analyste. Non, parce que ces outils sont essentiellement utiles dans les cas où les données de la base sont mises à jour automatiquement. L'action serait déterminée puis exécutée automatiquement à la fin de l'analyse, en fonction de ses conclusions (pour le moment l'action est du domaine de la science fiction !). Le moniteur est beaucoup plus pragmatique puisque son rôle est de surveiller l'évolution des données et de prévenir l'analyste de changements dans les conclusions des analyses déjà réalisées, par exemple.

Accès aux données biologiques

Une partie de la tâche d'ADB consiste à collecter les données biologiques nécessaires à la tâche. Cette activité, que nous dénommons *accès aux données biologiques*, est absente du processus d'ECBD.

Classiquement, le processus d'ECBD commence lorsque les données à analyser sont disponibles dans un SGBD, un outil optimisé pour offrir les meilleurs temps d'accès pour des tailles de bases importantes (plusieurs dizaines de Go à l'heure actuelle). La base de donnée est peuplée de données très souvent obtenues d'autres bases de données — les bases de données du client de l'analyste — qui ont été peuplées après saisie de réponses à des questionnaires ou par des méthodes d'acquisition automatique. Rappelons l'exemple du supermarché : prix, nature des articles et heure d'achat peuvent être obtenus de manière fiable par l'enregistrement automatique des codes barres.

Au contraire, l'accès aux données biologiques est une part importante du processus d'ADB (en temps notamment : trouver les données, les récupérer, les convertir, etc.). Par exemple, le programme Viseur maintient une banque de données locale que l'utilisateur constitue puis complète au fur et à mesure de ses besoins. Il était difficilement envisageable d'utiliser un SGBD pour stocker les données du fait du coût de ces systèmes.[39]

Mais comment l'utilisateur constitue-t-il la banque de données qu'il va analyser? A partir du contenu des Bases de Données Biologiques (BDB) ou des données expérimentales auxquelles il a accès directement. Étant donné l'importance de l'accès aux données biologique dans l'activité d'ADB, nous détaillons cette tâche dans la prochaine sous-section.

38. Viseur ne réalise pas de rapport d'analyse... mais facilite la rédaction d'un rapport scientifique à l'aide de ses outils de présentation

39. Les fonctionnalités des SGBD auraient grandement facilité l'implantation de certaines parties de Viseur, mais chaque site utilisateur du programme devrait supporter le coût du SGBD si nous avions retenu cette solution.

8.2.6 Problématique de l'accès aux données biologiques

Définition **Accès aux données biologiques**

L'accès aux données biologiques est l'activité qui consiste à réunir les données nécessaires à la tâche d'ADB à l'aide des outils bioinformatiques disponibles (interfaces WWW des BDB, ftp, email, etc.). Cette activité peut consister à

- trouver les BDB pertinentes (trouver le site WWW de la BDB),
- trouver et choisir les données pertinentes dans la BDB sélectionnée — utilisation des outils (requête, visualisation,etc.) disponibles sur le site WWW de la BDB,
- télécharger les données vers un espace de stockage local (réseau ou machine de l'analyste),
- traduire ces données dans le formalisme des outils d'analyse (transformation de formats de fichiers, cf. deuxième partie) et classer ces données pour pouvoir y accéder facilement au cours de l'analyse.

Dans le cadre du projet Viseur, l'assistance aux utilisateurs semble indiquer que l'accès aux données et leur conversion pour utilisation avec le programme (aspect que nous assimilons à l'accès aux données) pose des problèmes comparables, en terme de consommation de temps, aux autres aspects de l'analyse. En outre, la collecte de données est une activité récurrente du processus d'analyse de données : le résultat d'une analyse peut conduire l'analyste à collecter de nouvelles données pour vérifier une nouvelle hypothèse.

Pour construire un schéma du processus d'ADB équivalent à celui proposé pour l'ECBD par Brachman et Anand, nous sommes donc amené à remplacer le bloc `Database` de l'ECBD par la figure 8.5, qui reflète mieux la tâche d'accès au données biologiques.

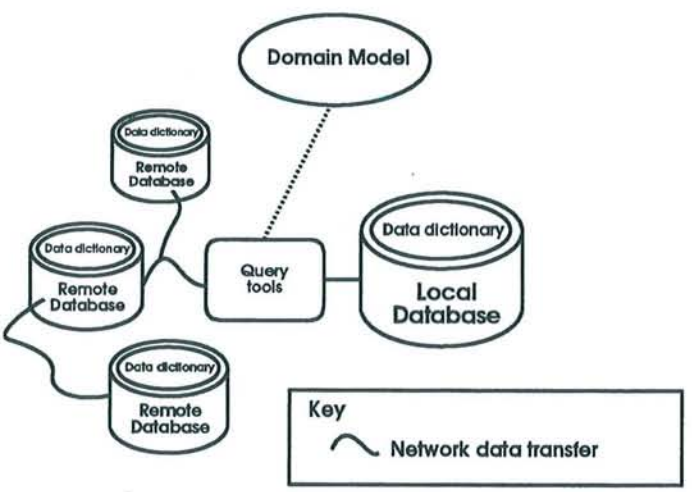


FIG. 8.5 – *Accès aux Données Biologiques*

Réseaux et Bases de Données Biologiques

Tout d'abord, la légende indique un accès aux bases de données distantes par un réseau. C'est maintenant chose courante pour un analyste de données biologiques que de collecter les données de l'analyse sur le réseau. Le numéro spécial *Databases* de la revue Nucleic Acid Research 1998, qui recense l'essentiel des bases de données d'intérêt biologique, compte la description de 102 bases de

données. Ce nombre est en augmentation de plus de 25% par rapport à 1997. Toutes ces bases de données sont consultables à travers le réseau (WWW et/ou ftp).

Ensuite, les outils de requête (`Query tools`) jouent un rôle dans la collecte d'informations : ils sont utilisés pour sélectionner les données distantes qui vont être copiées localement pour analyse. Il y a beaucoup trop de données dans les BDB pour envisager des copies locales de toutes ces données. La copie locale intégrale est réservée aux bases très souvent consultées parce que la réalisation d'une copie locale, puis son maintien à jour, sont consommateurs de ressources (matérielles : stockage et humaines).

Automatisation des copies locales partielles

Comme nous l'avons vu en première partie, le programme Viseur permet d'interroger TinyGRAP, une BDB réseau distante, pour constituer et mettre à jour sa banque de données locale de mutations. Cette fonctionnalité est très appréciée des utilisateurs qui sont déchargés de la collecte de ces données et peuvent se concentrer sur leur analyse.

Le programme Viseur montre aussi des aspects plus classiques, tel que la possibilité de charger des fichiers de séquences, qui offrent la souplesse attendue de ce type d'outil mais pourrait souvent être remplacé par une fonctionnalité du type "obtenir les séquences S de la base B". Par exemple, l'utilisateur demanderait toutes les séquences de la base SwissProt dont l'identificateur contient la chaîne 5HT (récepteurs sérotoninergiques) et pourrait continuer à travailler avec ces séquences dans le programme après que celui-ci se soit connecté à la base de donnée et ait obtenu — si besoin traduit — les séquences demandées...

Ce type de fonctionnalité permettrait aux utilisateurs de gagner beaucoup de temps et sa réalisation ne pose pas de problème technique particulier.[40] Les pages "problems and solutions" disponibles sur le site de Viseur contiennent le témoignage des problèmes et pertes de temps que la mise à disposition de cette fonctionnalité aurait permis d'éviter.

Lorsque la copie locale automatisée n'est pas implantée, comment les utilisateurs font-ils pour obtenir les données biologiques nécessaires à l'analyse ? Ils utilisent le système d'exploitation de leur ordinateur, et les outils réseaux à leur disposition (WWW, *gopher*, *ftp*, *email*).

Cette section nous a permis de présenter l'extraction de connaissances à partir des bases de données puis les différences entre le processus d'ECBD et l'activité d'analyse de données biologiques. Dans les deux prochaines sections nous affinerons notre description de l'ADB à travers la description de quelques applications conçues pour l'AD ou l'ADB.

8.3 Applications pour l'aide à l'analyse de données

Dans cette section, nous décrivons quelques applications conçues pour faciliter les tâches de visualisation et le traitement statistique des données. Ces applications sont présentées pour leur caractère illustratif, en insistant sur certains points de leur conception que nous réutiliserons dans la suite de ce document.

8.3.1 Visualisation

Les applications d'aide à la visualisation de données sont un premier exemple d'outils développés pour faciliter une des tâches de l'analyse de données. Parmi ces nombreux outils, nous voudrions décrire deux applications d'aide à la représentation de données dans un modèle tri-dimensionnel.

Pour reprendre la terminologie utilisée en première partie, ces applications facilitent l'injection de données quelconques dans un modèle 3D (les composantes de ce modèle sont des formes de cer-

40. Les utilisateurs indiscrets qui ont lu ces lignes voudront bien attendre un peu avant de demander l'implantation immédiate de cette fonction !

taines couleurs à différentes positions de l'espace 3D). Dans ce domaine d'applications, citons Explorer [Ltd95], et AVS/Express [Sys]. Les caractéristiques communes de ces deux systèmes sont

1. système interactif,
2. manipulation graphique des modules de transformation de données (pour paramétrisation et connection des modules entre eux),
3. transfert de données entre les modules gérés par modèle d'événement,
4. types de données fixes et spécialisés pour le domaine (représentation de données 3D et quelques types généralistes : paramètres entiers, réels, chaînes de caractères),
5. nombre de types de données spécialisés limité.

Plusieurs caractéristiques rappellent les considérations discutées en deuxième partie. Dans le cas d'Iris Explorer, les données sont représentées par des types, alors que pour AVS/Express les données sont des objets.[41]

La figure 8.6 présente une vue graphique de l'application Explorer annotée pour expliciter visuellement les concepts de modules et de transfert de données. Dans cet outil, les modules sont connectés par "des cables" qui symbolisent le *medium* de communication de données. On peut remarquer les analogies module↔composant et type↔interface. Chaque module possède un ou plusieurs *slots*fonctionnels. La légende de la figure 8.6 donne un exemple de connection. Elle indique aussi que les types de données acceptés ou fournis par les *slots*appartiennent à un ensemble réduit: `Geometry`, `Lattice`, `Pyramid`, `Parameter`. Pour prendre l'analogie avec les composants décrits en deuxième partie, le nombre de sortes d'interfaces supportés par les composants est restreint ce qui conserve une forte inter-opérabilité entre composants.

Les points 4. et 5. concourent à faciliter la réutilisabilité des modules en diminuant le nombre des types de données disponibles dans le système tout en garantissant que ces types sont adaptés aux données du domaine.

8.3.2 Statistiques

L'application GenStat

Ce programme permet de réaliser des analyses statistiques à partir d'ensembles de données lus dans des fichiers ou saisis interactivement par l'utilisateur. Plusieurs types de fichiers d'entrées sont supportés.

Les type de données proposés par le programme sont très simples mais bien adaptés aux analyses statistiques : des tableaux de nombres à une ou plusieurs dimensions semblables à ceux que l'on peut trouver dans un tableur. L'utilisateur dispose d'un langage qui permet de formuler puis de déclencher le type d'analyse qu'il souhaite réaliser. Une interface graphique permet de construire des phrases de ce langage pour doter le système d'une interface plus conviviale. Les phrases du langage Genstat ainsi construites peuvent être sauvegardées, modifiées et/ou réutilisées par l'utilisateur. Nous retrouvons ici la notion de script interne à une application pour l'automatisation des traitements. Le programme Genstat offre la possibilité de représenter graphiquement le résultat de certaines analyses (types de graphiques variés).

8.4 Applications pour l'analyse de données biologiques

Quelques systèmes d'aide à l'analyse de données ont été développés, sans toutefois proposer de modèle explicite de la tâche d'ADB. Le système décrit par [MMAD95], récemment dénommé ImaGene est conçu pour

– gérer les données biologiques et le résultat de leurs analyses (cf. point 1. du cahier des charges à mettre en place),

41. Rappelons que l'objet se compose d'un type et de méthodes qui permettent de manipuler la valeur du type.

- aider l'analyste à choisir les méthodes les plus apropriées à une tâche,
- permettre à l'analyste "de chaîner" plusieurs méthodes pour conduire l'analyse.

Les auteurs de ce système utilisent un modèles de représentation de connaissances par objets [RU91] qui permet de représenter la complexité des données biologiques. Le système Scarp [Wil94] [MWCR93], pour l'aide à la résolution de problèmes leur permet d'assister l'analyste dans le choix des outils à utiliser pendant l'ADB, qui semble donc être considérée comme une activité de résolution de problèmes (rappelons que la tâche d'ADB n'est pas explicitement décrite par les auteurs).

Le système Minos est un prototype d'outil d'aide à l'analyse de données biologiques. Nous avions envisagé de réutiliser une partie du système Minos dans une première phase de conception de l'environnement que nous décrivons. L'impossibilité de fait d'obtenir le logiciel nous a amené à revenir sur cette décision et à accorder une plus grande importance à la disponibilité pratique (par opposition à l'accord de disponibilité) des composantes d'un système d'aide à l'analyse de données. Il n'est pas suffisant qu'un système existe pour être utile : il doit être disponible pour être utilisé. C'est le point principal que nous retiendrons de ce prototype, qui par ailleurs essayait de montrer l'intérêt des représentations de connaissances biologiques par les méthodes de treillis de concept [Wil92], ou l'utilisation d'un SGBD[42] objet pour la modélisation des données biologiques.

8.5 Contexte de l'activité d'ADB

Comme le fait remarquer [CWA97], le développement logiciel devrait être considéré comme un développement de système et à ce titre, prendre en compte le contexte organisationnel dans lequel va s'intégrer le nouvel outil logiciel. Nous nous intéressons donc maintenant à la façon dont le contexte de réalisation de l'activité d'ADB peut augmenter les contraintes de conception du nouveau système.

8.5.1 Prise en compte de l'évolution des connaissances du domaine

Les modèles des systèmes biologiques sont sujets à évolution. Les méthodes se perfectionnent et les outils spécialisés qui implantent ces méthodes ou permettent de simuler ces modèles doivent évoluer aussi. Au niveau informatique, cela signifie qu'une nouvelle version d'un programme, ou un nouveau programme, devra être utilisé pour bénéficier des avancées méthodologiques du domaine.

Ce fait a des implications fortes pour l'analyse de donnée biologiques puisque les progrès méthodologiques du domaine peuvent influencer :

- les outils de requête réseau (mise en place d'une nouvelle base de données ou proposition de nouveaux services distants,...)
- les outils d'analyse ou d'ASA (nouvelle méthode de calcul de propriétés dérivées des données biologiques, nouvel algorithme de classification,...),
- les outils de visualisation (nouvelle méthode de visualisation d'alignement multiples...).

Les évolutions logicielles ne sont pas un gros problème en soi mais posent des problèmes à l'analyste qui est spécialiste de son domaine mais pas de l'installation et de la mise à jour de logiciels. Le paradoxe est que l'analyste de données biologiques a besoin des nouvelles versions des logiciels mais n'a pas envie de les installer (cette tâche est souvent considérée comme ingrate et il n'est pas rare d'attendre «quelques numéros de version de plus» avant de mettre à jour un programme).

Ce type de problème se rencontre dans tous les domaines d'utilisation des outils informatiques mais est critique pour l'analyste qui ne sait pas installer un logiciel. Les logiciels pour l'analyse de données biologiques sont souvent distribués sous forme de sources, et ont besoin d'être compilés puis configurés avant qu'aucune utilisation ne soit possible.

L'industrie informatique commence à prendre en compte ce problème et développe des mécanismes de mise à jour des logiciels simplifiés lorsque la machine de l'utilisateur est connectée au WWW. Un programme offre alors à l'utilisateur la possibilité de le mettre à jour par l'intermédiaire d'un de ses

42. Système de Gestion de Bases de Données (DBMS en anglais).

menus (technologie *pull*). Ou bien la mise à jour est automatique à chaque *release* du logiciel par le distributeur (technologie *push*, [Cas97]).

Complémentaire, la technologie Java [Sun97a] permet la portabilité des logiciels sur un grand nombre de plateformes et propose des mécanismes (archives *jar* et ressources notamment) qui facilitent l'installation des logiciels par un utilisateur novice.

8.5.2 Autres considérations

Adaptabilité à l'expérience de l'analyste

Nous avons expliqué en première partie que des représentations comme les diagrammes en serpent semblent plus adaptées aux centres d'intérêts des biologistes que ne le sont les diagrammes moléculaires. Pour expliquer cette différence nous avons avancé des hypothèses qui pourraient être classées dans le cadre de la psychologie cognitive. Cette discipline, tout comme l'ergonomie, est intéressante pour la conception d'interfaces hommes/machines [Cou91]. Nous pensons qu'il est important de prendre en compte des résultats de ces disciplines s'ils peuvent aider à comprendre le contexte de l'ADB.

Dans le cadre de ces disciplines, plusieurs études ont cherchés à mettre en évidence l'avantage des représentations graphiques de données numériques par rapport aux présentations textuelles. Ces travaux montrent qu'il est préférable de présenter des ensembles de plus de vingt nombres sous forme graphique. Pour les ensembles plus réduits certaines de ces études aboutissent à des résultats contradictoires [CR97]. A la lecture des résultats de ces études, il nous semble encore plus clair que la représentation d'un ensemble de données doit-être adaptée à l'individu qui va l'utiliser. Plutôt que de chercher les raisons qui vont rendre une représentation plus efficace qu'une autre, ce que nous ferions pour concevoir une nouvelle représentation, nous préférons donner à l'analyste la possibilité de choisir parmi les représentations disponibles celle qui convient le mieux à son expérience.

Cette considération est reliée à celle de la section 8.5.1 : la représentation adaptée à l'expérience de l'analyste pouvant avoir été développée par un tiers.

Adaptabilité au but de la recherche en cours

Si le système doit permettre d'adapter la représentation pour permettre à différents analystes de l'utiliser selon leur préférence on doit considérer aussi la possibilité de pouvoir effectuer des transformations des données qui mettrons en valeur les relations que l'analyste cherche à confirmer. Par exemple, pour comparer les valeurs de deux ensembles, injecter la différence, ou le rapport de leurs éléments dans un représentation graphique ou toutes autres grandeurs statistiques estimées plus faciles à appréhender que les valeurs brutes peut être mieux adapté au but de la recherche que d'injecter les variables elles-mêmes.

Les applications Genstat, Explorer ou AVS et [MMAD95] (MMVD) (en partie) ont été mises au point pour faciliter ce type d'adaptabilité aux buts de l'analyste. Les outils de connection de composants répondent à ce problème lorsqu'on les associe à la spécification d'interfaces adaptées au domaine d'application.

Adaptabilité aux contraintes technologiques

Nous avons suggéré en première partie que le développement du WWW n'a pas résolu les problèmes de portabilité des représentations graphiques par rapport au matériel ou au logiciel. Les variantes de HTML (versions 2.0, 3.2, 4.0) avec ou sans *frames*, feuilles de styles, JavaScript, Java, etc... supportées différemment selon les navigateurs rendent très difficile de proposer la représentation la plus adaptée (considérations précédentes) compatible avec le navigateur de l'utilisateur.

Une solution consiste à proposer des variantes technologiques et à laisser à l'utilisateur, ou à un mécanisme automatique lorsqu'ils seront disponibles, le choix de la représentation adéquate.

Nous verrons dans la section 10.3 comment gérer l'adaptabilité des présentations à ces trois facteurs.

8.6 Modélisation de l'activité d'ADB

Nous pouvons conclure ce chapitre en résumant notre conception — ou modèle — de l'activité d'analyse de données biologiques. L'activité d'ADB peut être considérée comme constituée de sous-activités. L'analyste qui réalise une ADB partage son temps entre les différentes tâches présentées sur la figure 8.7. Les outils à disposition de l'analyste et les transferts de données entre ces outils sont représentés de la même façon que sur le schéma du processus d'ECBD.

Les modèles construits pendant l'ADB peuvent être classés en deux catégories. L'analyste peut développer

- des modèles empiriques construits à l'aide des outils statistiques, d'AD, ou d'ASA à partir des données et d'un formalisme de représentation de la connaissance (formalisme mathématique ou représentation au sens IA),
- des spécialisations de modèles théoriques.

Les modèles du deuxième type sont développés par interaction entre l'analyste et des *présentations* (cf. figure 8.4). Ce type particulier de visualisation présente les données à l'analyste sous une forme familière (associée à un modèle théorique qu'il connait) qui l'aide à conduire son raisonnement.

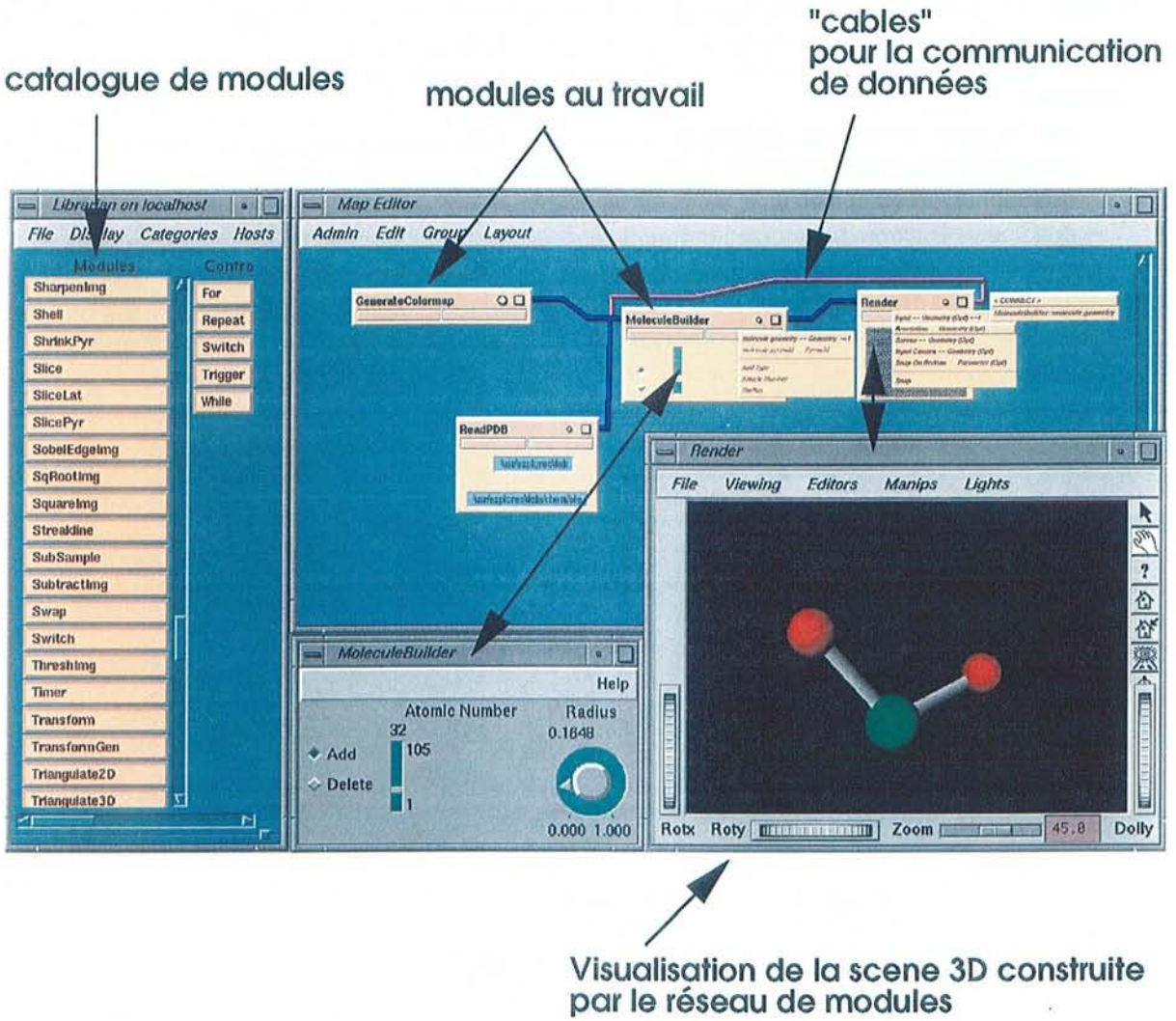


FIG. 8.6 – *L'Application d'aide à la visualisation de données Explorer*
La fenêtre `Render` en bas à droite propose la visualisation des données dans un modèle tri-dimensionnel. La visualisation est construite à l'aide de modules (choisis parmi ceux du catalogue de modules, à gauche). La fenêtre `Map Editor` permet de connecter les entrées et les sorties des modules. Les cinq *slots* d'entrées du module `Render` sont affichés : `Input`, `Annotation`, `Input Camera`, `Snap On Redraw`. Les types des *slots* sont indiqués dont les valeur possibles sont `Geometry`, `Parameter`, `Pyramid`. Le slot de sortie `molecule geometry` du module `MoleculeBuilder` est relié au slot d'entrée `Input` du module `Render`. Les doubles flèches relient les représentations simplifiées des modules (en haut) à leur représentation effective (en bas).

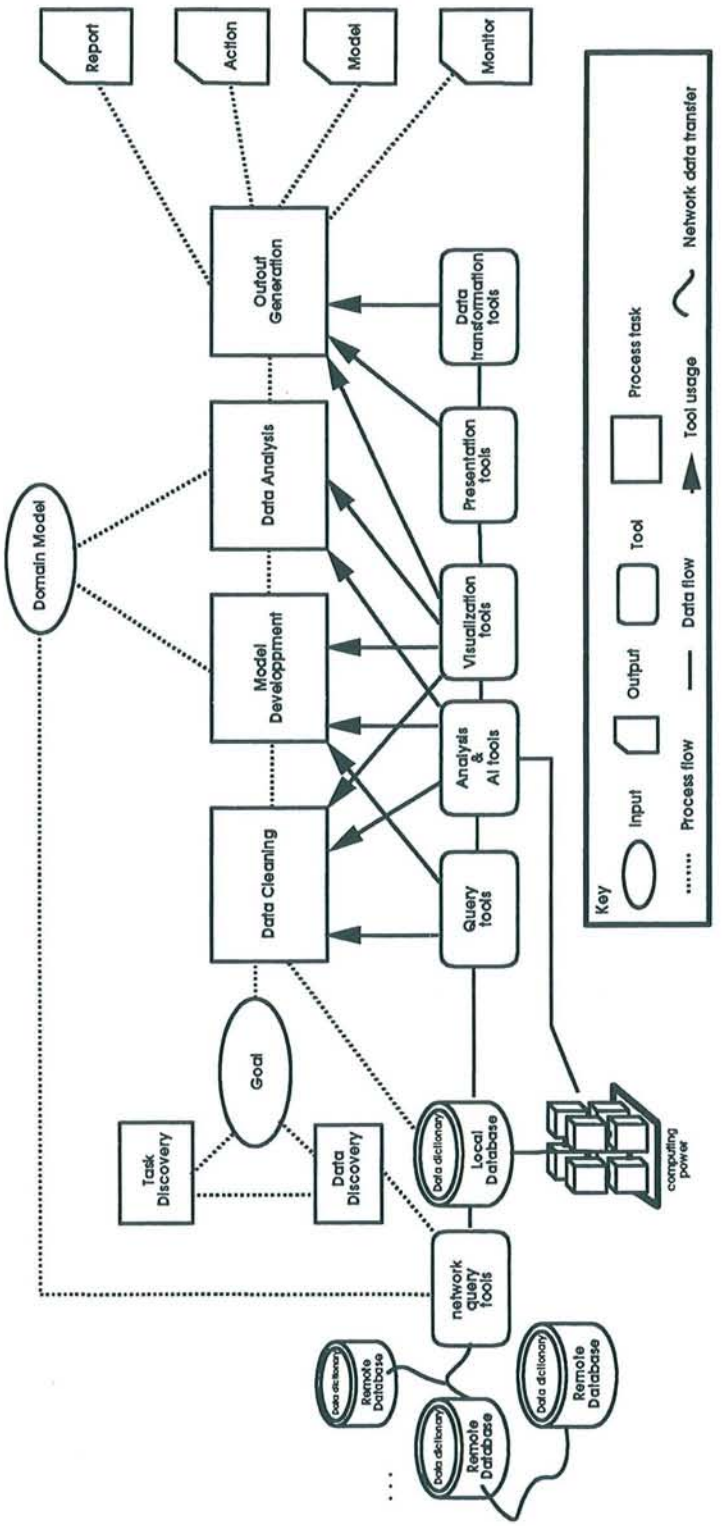


FIG. 8.7 – *Le processus d'Analyse de Données Biologiques*
Cette figure complète le processus d'ECBD (figure 8.3), modifié pour faire apparaître le rôle des outils de visualisation dans le nettoyage des données, l'accès aux données biologiques (figure 8.5) et l'utilisation de machines dédiées à une tâche de calcul.

9

Le Cahier des charges

Nous pouvons déduire du modèle de l'ADB que nous avons présenté dans le chapitre précédent (cf. figure 8.7) les points essentiels que l'environnement que nous cherchons à concevoir doit supporter pour faciliter l'activité d'ADB. En accord avec ce modèle, un environnement d'aide à l'analyse de données devrait donc proposer :

1. un accès adapté aux données biologiques (outils de requêtes adaptés aux réseaux, aux collections de données et aux données de représentation complexe);
2. des outils d'Analyse de Données (certains, domaine dépendants) et d'Apprentissage Symbolique Automatique;
3. des outils de Visualisation adaptés aux données biologiques et aux connaissances de l'analyste, expert du domaine;
4. des outils de présentation pour communiquer les résultats intermédiaires, préparer les rapports ou les présentations des résultats;
5. un mécanisme qui maximise l'inter-opérabilité entre les outils (par exemple réduction des problèmes de transfert de données);
6. des facilités de mise à jour des parties du système qui permettent à l'analyste d'utiliser les outils les plus récents sans difficulté d'installation.
7. la possibilité d'automatiser certains traitements pour les appliquer à des ensembles de données, de les sauvegarder pour réutilisation ultérieure, etc...
8. un mécanisme qui facilite l'adaptabilité à des modes de présentations différents (bibliothèques de visualisation 2D, 3D, WWW : HTML 2.0, 3.2, 4.0, Cascading Style Sheets, plugins, Java, JavaScript,...).

Certains de ces points sont couverts par des outils existants. Par exemple il existe beaucoup d'outils d'analyse (statistiques, analyse de données), d'apprentissage symbolique automatique, de présentation graphique, etc. qu'un environnement d'aide à l'ADB devrait pouvoir réutiliser.

Par contre, il n'existe pas d'outils (programme ou application) pour favoriser les points 5. et 6. parce que leur prise en compte nécessiterait la conception de systèmes d'exécution d'outils. Les systèmes de visualisation graphique que nous avons décrits (section 8.3.1) sont des exemples de tels systèmes. Les systèmes d'exploitation servent souvent de pis-aller, à défaut de mécanismes performants pour couvrir les points 5. et 6., lorsque les outils sont des programmes ou des applications.

Décrivons maintenant la manière dont l'architecture de l'environnement d'aide à l'ADB peut influencer le déroulement de l'ADB.

9.0.1 Le processus d'analyse de données idéalisé

Afin de mettre en évidence les problèmes qui compliquent la tâche d'un analyste, intéressons-nous au processus, idéalisé, d'une analyse de données biologiques.

La figure 9.1 illustre ce processus. L'analyste dispose d'un ensemble d'outils qu'il va pouvoir assembler pour construire des traitements. Les traitements s'appliquent aux données et produisent des visualisations ou des présentations. Les outils disponibles à l'analyste sont choisis parmi les catégories suivantes — reprises de la figure 8.7 :

- accès aux données,
- analyse statistique ou symbolique (IA, classification ou arbre de décision),
- visualisation ou présentation (sur machine locale ou pour un intranet ou internet),

Toutes ces catégories d'outils peuvent fonctionner de façon complètement locale ou partiellement à l'aide de ressources distantes. L'analyste doit pouvoir utiliser les outils dont il a besoin de la même façon, que les outils soient locaux à son système ou utilisent des ressources réseaux. Les considérations de la section 8.5.1 impliquent que l'analyste puisse facilement compléter l'ensemble des outils qu'il peut utiliser et que ceux-ci s'intègrent à son environnement sans efforts inutiles.

Pour prendre quelques exemples, l'outil réseau qui accède aux données sur la figure pourrait être le moyen de sélectionner une séquence particulière d'une base de données biologiques, par exemple à l'aide de son numéro d'identification dans la base. L'outil réseau de transformation de données, réalisé par un autre analyste (ou par un groupe de recherche spécialisé dans le domaine ou dans une technique d'analyse) pourrait être un moyen d'accéder à une ressource dédiée à la recherche de séquences homologues dans une base de donnée (accélérateur matériel). Ces deux outils reliés à un outil de présentation d'alignement multiple constituent un réseau d'outil qui permet de visualiser l'alignement d'une séquence avec les régions qui lui sont les plus homologues parmi les séquences

9.0.2 Le processus d'analyse de données dans les faits

Comparons la situation de l'analyste à la description de la sous-section précédente pour montrer les problèmes qui existent actuellement.

Accès aux données biologiques

L'analyste accède manuellement aux données biologiques au travers des interfaces WWW des bases de données biologiques. Chaque BDB présente une interface différente que l'analyste doit apprendre avant de pouvoir récupérer une donnée dans la base. Des BDB qui proposent des données similaires présentent souvent des interfaces utilisateurs différentes. Le système *Sequence Retrievial System* (SRS) [EA93] a été spécifiquement conçu pour apporter une solution à ce problème, dans le cadre des données de séquences : SRS propose une interface uniforme pour interroger de nombreuses bases de données de séquences biologiques. Pour les autres types de données, l'analyste ne dispose pas d'un système pour l'aider à trouver une donnée nécessaire à son analyse quelque soit la BDB dans laquelle la donnée est stockée. L'analyste est obligé de passer d'une BDB à l'autre et d'utiliser l'interface WWW de chaque BDB avant d'accéder à la donnée,... ou à la BDB suivante.

Évolution de la boîte à outils

Tous les problèmes dont nous avons discuté dans la section 8.5.1 se posent ici. Les outils sont des applications ou des programmes qu'il faut installer puis configurer avant de les utiliser. Les programmes qui utilisent des ressources logicielles distantes sont encore rares et leur installation pose encore problème (Entrez [SEOJ96] est l'un d'eux, par exemple).

Automatisation des traitements

Les solutions classiques à ce type de besoins sont les langages de scripts. D'une part les langages de scripts du système d'exploitation, d'autre part les langages internes des applications. Les premiers permettent d'automatiser les traitements qui utilisent plusieurs outils, les seconds de configurer précisément un outil pour réaliser un traitement. Les scripts permettent de réutiliser un traitement, parfois complexe, sur plusieurs données et sont donc très utiles dans le cadre de l'analyse de données, mais

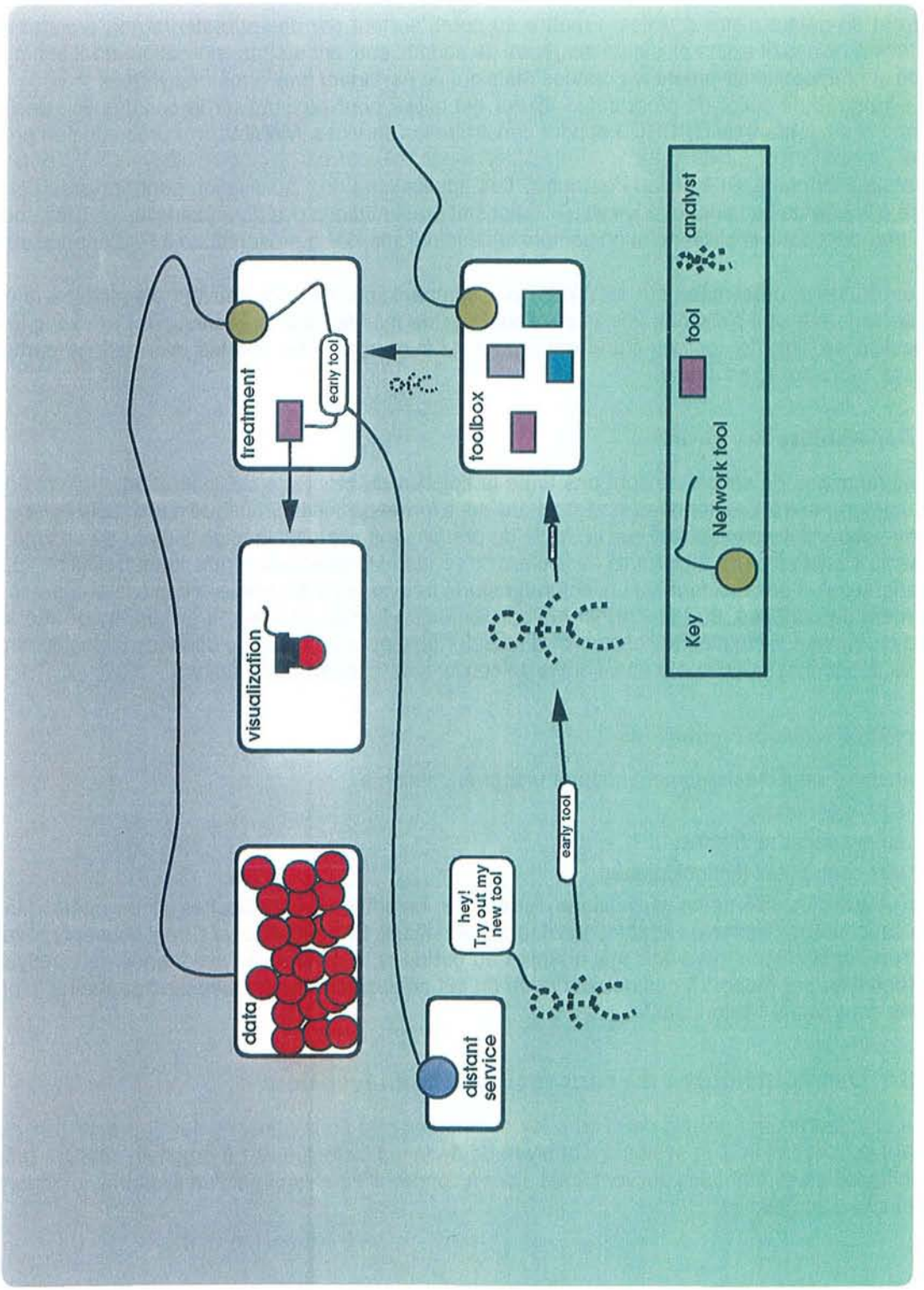


FIG. 9.1 – *Une autre vision de l'Analyse de Données Biologiques*
Par rapport à la figure 8.7 l'emphase est mise ici sur l'assemblage d'un traitement (par l'analyste) dont les constituants sont évolutifs. Ces constituants sont des outils choisis parmi ceux d'une boîte à outils. La constitution et la mise à jour du contenu de la boîte à outils est faite par l'analyste en fonction de ses besoins. Des outils récents, développés par des tiers peuvent ainsi être ajoutés au contenu de la boîte à outils et permettre l'accès à des services distants développés par le fournisseur de l'outil, ou réaliser une transformation de données localement (telle qu'un calcul ou la construction d'une visualisation). Le traitement construit peut être stocké pour utilisation ultérieure, réutilisé, adapté à une tâche semblable, complété d'outils récents. Le traitement produit une (des) représentation(s) à partir de données potentiellement distantes.

souffrent du défaut d'être difficiles à mettre au point, surtout par des utilisateurs non programmeurs. Ajoutons à cela qu'il existe plusieurs langages de scripts, souvent un par application, dont les fonctionnalités sont approximativement les mêmes mais qui ne partagent pas la même syntaxe.

Le langage de script du programme Viseur est utilisé pour automatiser la construction des représentations en serpent de GPCRDB et pour construire les services WWW Viseur présentés en première partie.

La situation varie en fonction des outils. Les applications de visualisation sont conçues pour permettre à l'analyste de construire les visualisations et présentations des données telles qu'il les souhaite. Elles prennent donc implicitement en compte le besoin d'adapter la présentation à l'expérience de l'analyste.

Les données présentées sur le WWW, par comparaison, ne sont souvent disponibles que sous une seule forme et il n'est pas possible à l'analyste de modifier leur présentation. Par exemple, il est impossible de filtrer le contenu d'une page pour ne faire apparaître que les informations pertinentes (dans le cadre de la recherche).

Inter-opérabilité des outils

Les langages de scripts ne sont pas toute la solution du problème de l'utilisation conjointe de plusieurs programmes. Les données sont stockées sous forme de fichier. Pour que deux outils puissent être utilisés séquentiellement, il faut que la sortie du premier soit compatible avec l'entrée du second. Étant donné qu'il existe plus de 20 formats de fichiers de séquences différents et que chaque outil ne supporte en général qu'un ou deux formats en lecture/écriture, trouver les outils nécessaires à la préparation d'un traitement de données, ou transformer les données d'un format de fichier à un autre, est une activité assez courante. La situation est encore plus inconfortable pour les types de données biologiques autres que les séquences, s'il n'existe pas d'outils de conversion (pseudo-)automatique.

Adaptabilité des présentations

L'analyste peut théoriquement adapter une présentation à

- son expérience,
- au but de sa recherche,
- aux contraintes technologiques,

en utilisant les différentes applications disponibles. En pratique, les difficultés d'inter-opérabilité entre outils concourent à limiter les adaptations des présentations. Les données ne seront souvent présentées que d'une seule façon, peut-être mal adaptée au but de la recherche, à l'expérience de l'analyste, ou sous-optimale par rapport à ces deux critères du fait de solutions techniques plus ou moins imposées par des contraintes "historiques".

9.0.3 Des contraintes de conception à la conception

Ce chapitre nous a permis de décrire les contraintes que nous pensons devoir prendre en compte pendant la conception d'un système d'analyse de données biologiques. Le prochain chapitre présente les technologies et méthodes qui vont nous aider à concevoir l'architecture d'un système qui répond aux contraintes résumées ici.

10

Confrontation cahier des charges/technologie : vers un prototype

10.1 Traitements et technologie composants

Les contraintes liées aux traitements suggèrent l'utilisation des technologies composants [Szy98]. Nous avons présenté en deuxième partie un exemple de conception de système à base de composants. Pour résumer très rapidement, les composants sont des objets dont les interfaces sont conçues pour s'assembler simplement. Les systèmes de composants évolués proposent des méthodes d'assemblage visuel pour assembler des composants pré-existants sans écriture de code. Les outils d'assemblage de composants les plus simples d'emploi permettent à des utilisateurs de composer un outil sans programmation. Étant donné la facilité de composer un nouveau programme à l'aide de ces outils et d'une bibliothèque de composants pré-existants, les besoins d'échanges de composants se sont développés et les outils d'assemblage proposent maintenant les moyens de construire des distributions de composants, pour diffusion en direction d'autres développeurs.

Dans le système que nous proposons, le traitement de données nécessaire à une étape de l'analyse est construit à partir de composants logiciels. Ces composants sont choisis parmi des composants spécifiques au domaine d'application, à l'analyse de données, ou construits par l'analyste expérimenté pour les besoins d'une analyse particulière.

Le résultat est un nouveau composant qui peut être utilisé pour transformer plusieurs jeux de données, réutilisé plus tard, distribué à d'autres utilisateurs, etc...

Nous privilégions le choix des environnements d'assemblage de composants visuels qui permettent de mettre au point un traitement de façon interactive. La mise au point consiste à

- ajouter ou enlever des composants,
- créer ou modifier les connections entre les composants,
- paramétrer le fonctionnement interne de chaque composant,
- modifier les données qui servent à tester le traitement.

Choisir les composants et les outils d'assemblage de composants nous permet de tenir compte des points suivants du cahier des charges.

1. et 5. Les composants dérivent du paradigme objet et supportent des interfaces objets qui sont capables d'encapsuler des données complexes. Certaines spécifications de composant (JavaBeans [Jav97]) permettent aux composants de réagir à la modification d'une de leur valeur d'entrées.

6. Les spécifications et les outils composants sont conçus pour simplifier l'échange de composants entre développeurs.

7. Un assemblage de composants peut être transformé en composant. Ceci facilite la réutilisation des traitements. Certains outils d'assemblage de composants facilitent la gestion d'une boîte à outils.
8. Les modèles de conception [GHJV96] sont issus de la même philosophie de réutilisation logicielle que les composants et proposent des solutions maintenant classiques à ce problème. Citons les architectures MVC [Gol84] (Modèle Vue Contrôleur) et PAC [Cou87].

10.1.1 Introspection et outils d'analyse de données

Dans des environnements où les données sont obtenues de sources distantes, peuvent présenter une structure complexe (richesse du modèle objet) et ne sont pas obligatoirement connues a priori, il est parfois très utile de pouvoir déterminer la structure d'un objet au moment de l'exécution.

Par structure, il faut comprendre la liste des méthodes des interfaces associées à un objet, et les paramètres de ces méthodes.

Considérons une interface (exprimée dans le langage IDL) qui permet d'encapsuler certains éléments d'une séquence biologique.

```
interface BiologicalSequence {

string getShortName();

void setShortName(
  in string arg0
);

string getResidueCodes();

void setResidueCodes(
  in string arg0
);

};
```

Cette interface possède quatre méthodes (nommées `getResidueCodes`, `setResidueCodes`, `getShortName`, `setShortName`). La première méthode renvoie une chaîne de caractères (`string`) qui représente un nom que les outils de présentation peuvent utiliser pour désigner la séquence (ce nom doit être connu de l'analyste et être suffisament court — moins de dix caractères — pour être compatible avec n'importe quelle représentation graphique de séquence. La seconde méthode permet de fixer le nom d'une séquence particulière et ne renvoie aucune donnée (`void`). Les méthodes suivantes ont des fonctions différentes (elles permettent de lire ou de fixer la séquence des résidus).

La simplicité de l'interface est dictée par les considérations exposées en deuxième partie, pour la conception d'interfaces pour la chimie informatique.

Cette interface doit être réutilisable, donc donner accès au maximum d'implantations différentes de l'objet séquence (BLAST [AGM+90], FASTA [PL78] [Pea78], Swiss-Prot, Viseur, ...). Elle présente les données les plus utiles dans le cadre de la présentation de données biologiques (un nom pour la séquence et la séquence des résidus). Nous ne chercherons pas à justifier davantage cette interface qui est encore préliminaire. En effet, elle devrait certainement encore inclure un accès aux annotations associées et la mention de son origine (BDB, et détails sur le dépôt, outil qui la manipule, etc.) mais les interfaces pour ces mentions restent à discuter.

Définition	**Introspection**
L'introspection est la propriété d'un objet de pouvoir observer sa propre structure [MN87]. C'est à dire, l'ensemble de ses méthodes, y-compris les méthodes d'accès à ses attributs. Les mécanismes d'introspection peuvent être utilisés pour découvrir la structure d'autres objets.	

L'introspection permet de déterminer, à partir de la référence à un objet qui implante l'interface `BiologicalSequence`, que cet objet propose quatre méthodes de noms :

- getResidueCodes,
- setResidueCodes,
- getShortName,
- setShortName.

Puis il est possible de déterminer le nombre et la nature de chacun des arguments des méthodes.

En général, l'introspection fonctionne en associant à chaque objet un objet *class* qui peut être utilisé pour obtenir une description de la classe de l'objet initial. Ces classes particulières, qui contiennent une information à propos d'une autre classe, sont dénommées méta-classes. La première utilisation des méta-classes date de Smalltalk-76 [Ing78]. [Hat96] propose une introduction à l'application des métaclasses pour l'instrospection (*reflection* en anglais).

Montrons sur un exemple comment un programme peut utiliser les mécanismes d'introspection. Les instructions suivantes détermineraient que l'objet `seq`, de type `BiologicalSequence` propose la méthode `getResidueCodes`.[43]

```
//objet seq est construit par un service distant, il est disponible ici.

Class c=seq.getClass();              // Obtient de l'objet seq un
                                     // objet c qui documente la
                                     // classe de seq.
Method m=c.getMethod(first_method);  // utilise c pour obtenir un
                                     // objet, m, qui décrit une méthode.

System.out.prinln("l'interface de l'objet seq propose la  méthode "
                  + m.name() + " qui accepte " + m.numArgs()
                  + " arguments");

/* Les trois lignes précédentes afficheraient, à l'exécution du programme :
"l'interface de l'objet seq propose la  méthode getResidueCodes qui ac-
cepte 0 arguments"
*/
```

L'introspection est donc utile pour déterminer la structure d'un objet de manière dynamique mais le véritable intérêt de cette technique est de permettre de construire des appels aux méthodes de l'objet inspecté.

```
//objet seq est construit par un service distant, il est disponible ici.

Class c=seq.getClass();
Method m=c.getMethod(fourth_method);
m.setArgument(first_arg, "new sequence name");
m.invoke();    // exécute seq.setShortName("new sequence name");
```

L'introspection permet de construire des représentations pour des objets dont la définition est postérieure à la date du développement du logiciel de présentation. Les systèmes à objet, qui proposent l'introspection, utilisent souvent cette possibilité pour présenter les objets aux développeur dans les outils d'aide à la correction de défauts.

De notre point de vue ce type de mécanisme présente un intérêt certain pour l'aide à l'ADB puisqu'il permet de décomposer tout objet en valeurs élémentaires. Ces valeurs dont le type est simple

43. Les instructions présentées s'inspirent du langage Java mais simplifient le mécanisme d'introspection pour montrer les principes de son utilisation.

(chaîne de caractère, entiers, réels) peuvent alors être traduites dans le formalisme de représentation de connaissances des outils d'apprentissage symbolique (cf. l'exemple des problèmes de traduction de données séquences biologiques pour utilisation avec le classifieur AutoClass, section 8.2.5.0).

Cette tâche de traduction n'est certainement pas simple et nous n'avons pas entrepris de la mettre en pratique dans le cadre de ce travail. Néanmoins, nous remarquons que les mécanismes d'introspection, qui sont une des bases des technologies composants — donc disponibles dans le système que nous présentons — peuvent permettre de réaliser le pont entre la complexité des données biologiques et les modèles simplifiés des outils d'analyse IA et statistiques.

10.2 Outils réseaux et CORBA

Le problème de l'accès aux ressources distantes (données ou traitements) impose d'utiliser des technologies adaptées aux réseaux. Dans le cadre de Viseur nous avons utilisé le protocole HTTP [Num97] pour effectuer la communication des données entre TinyGRAP et Viseur. Le processus de transfert d'information n'était pas suffisant et il a fallu mettre en place un protocole (l'équivalent d'un contrat) entre le client et le serveur.

Nous avons présenté le modèle procuration en fin de deuxième partie (cf. figure 7.1) qui propose une solution orientée-objet à la communication entre objets distribués. En fait, ce modèle est à la base de la solution retenue par les spécifications d'un standard pour une architecture d'objets distribués. CORBA (Common Object Request Broker Architecture) est une architecture logicielle qui fixe un cadre général à la coopération d'objets inter-connectés par un réseau.

Pour une introduction générale à CORBA, le lecteur est invité à consulter [OHE97]. Des détails de la mise en oeuvre de cette technologie à l'aide de différentes implantations sont décrits par [OH97]. Une description, même sommaire, de CORBA sortirait du cadre de ce document, aussi ne présenterons-nous ici que quelques éléments de vocabulaire utiles au propos. L'OMG ("Object Management Group") qui est un consortium de plus de 700 entreprises ou organisations intéressées par les technologies objets est en charge du processus de développement des spécifications de CORBA. La figure 10.1 présente l'infrastructure générale des spécifications CORBA :

Client Le concept de client est commun à toutes les architectures client/serveur dont CORBA est un exemple. Le client est la machine, le processus, ou l'objet qui invoque une opération distante. Cette opération peut être l'accès à une donnée ou l'exécution d'un traitement. (Ex. Viseur, navigateur WWW).

Serveur Le serveur reçoit les requêtes, les exécute et renvoie le résultat au client. (Ex. TinyGRAP, serveur HTTP).

IDL (*Interface Definition Language*). IDL est le langage déclaratif[44] qui permet de décrire les interfaces du système d'objets distribués. C'est un moyen formel de décrire le contrat entre le client et le serveur.

ORB (*Object Request Broker*). L'ORB est le service CORBA responsable de transmettre une requête exprimée dans le formalisme objet, à travers le réseau, jusqu'au serveur responsable de son exécution. Le résultat de la requête, s'il y en a un, est transmis par l'ORB dans le sens inverse vers le client.

Amorce (stub) Un objet présent sur le client qui a pour rôle de transmettre les requêtes au sujet.

Sujet (skelton) Un objet présent sur le serveur qui réceptionne les requêtes envoyées par l'amorce et les redirige vers l'objet destinataire.

Contrairement aux architectures client/serveur plus traditionnelles, CORBA permet de considérer une même machine comme client ou serveur, en fonction du moment de l'interaction.

44. IDL n'est pas un langage de programmation : il permet de décrire les interfaces, pas de les implanter.

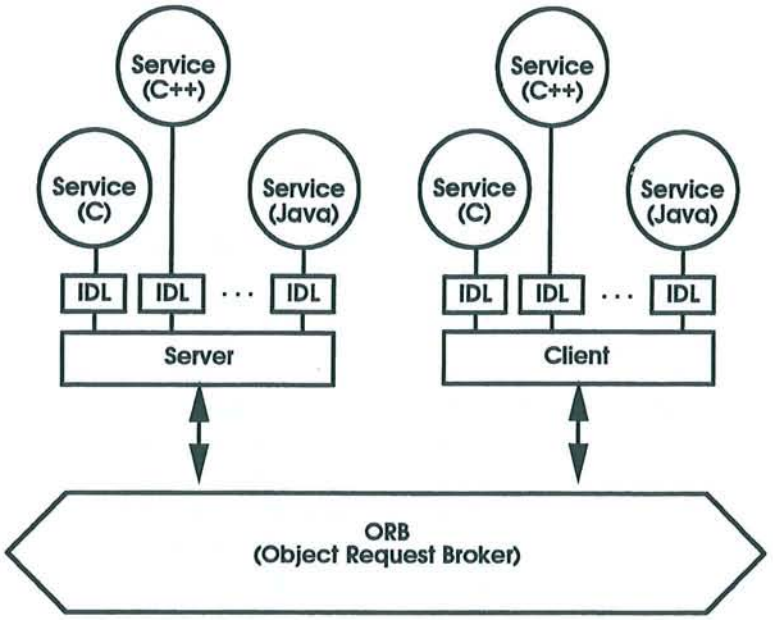


FIG. 10.1 – *Architecture générale de l'ORB CORBA.*

CORBA offre les moyens de faire coopérer des objets écrits dans des langages différents, certains pouvant jouer le rôle de serveur, d'autres le rôle de client, d'autres encore les deux à la fois. Les moyens sont exprimés dans le langage IDL (*Interface Definition Language*) et sont des interfaces vers des services implantés dans un langage ou un autre. L'ORB fournit le moyen de faire communiquer les objets distribués qui implantent les services à travers un réseau, de façon transparente.

10.2.1 Quelques avantages de l'approche CORBA

Utiliser les implantations de CORBA pour réaliser les interactions réseaux présente plusieurs avantages :

- CORBA est construit sur le paradigme objet. Pour CORBA tout objet est un composant;
- CORBA masque complètement la diversité des protocoles réseaux — et leur complexité;
- beaucoup de langages sont supportés directement : C, C++, Java, Cobol, Smalltalk, Ada: les applications existantes peuvent être interfacées au système;
- bénéfices des spécifications et implantations des services généraux (services de nom, d'événements, etc...);
- du point de vue du client, un serveur CORBA est toujours actif (des dispositifs de reprise de panne peuvent être mis en place de manière transparente au logiciel client et à l'utilisateur), ceci simplifie la conception et la réalisation des applications clientes;
- ...

Mises à part ces considérations, qui sont suffisantes à elles seules pour s'intéresser à l'architecture CORBA, il faut noter que l'OMG a mis en place en 1994 un groupe de travail chargé de développer les spécifications de services pour les domaines de la biologie et de la chimie : CORBAmed [Gro96].

10.2.2 Inconvénients de l'approche CORBA

Les inconvénients liés au choix de CORBA sont essentiellement autres que techniques :

- le prix des environnements de développement et des licences de déploiement (compter environ 25kf pour un environnement commercial complet),[45]

45. Notons que Orbacus [Obj98] est une implantation disponible gratuitement pour des utilisations non commerciales et que certaines compagnies proposent leurs produits gratuitement à la comunauté scientifique académique).

– la difficulté de trouver les ressources humaines formées à la technologie CORBA.

10.2.3 Autres alternatives

Examinons rapidement les alternatives au choix de CORBA.

DCOM. DCOM est le concurrent de CORBA, développé par Microsoft pour ses propres besoins. DCOM offre la majorité des avantages de CORBA mais n'est pas portable en dehors de l'environnement Microsoft et est un système essentiellement fermé (les logiciels sont souvent disponibles bien avant les documentations, pour ne pas parler des spécifications, il n'est pas possible d'influencer l'évolution du produit[46])

RMI. RMI est le protocole tout-Java de Sun pour l'appel de méthode entre objets Java distants. L'utilisation de RMI est comparable à celle de CORBA mais l'architecture est beaucoup moins générale.

Sockets. Cette couche de programmation établie au-dessus de TCP/IP a été introduite sur systèmes UNIX depuis 1981. Elle est maintenant disponible sur tous les systèmes d'exploitations. Le langage Java implante les sockets en standard. Ce type de protocole n'est pas orienté-objet ce qui complique la transmission de données complexes à travers le réseau.

HTTP/CGI. Il est possible d'utiliser les mécanismes CGI [CGI98] ou équivalents qui reposent sur HTTP pour connecter un client et un serveur. Tout se passe comme si un utilisateur réalisait l'interaction (voir la figure 2.5) mais c'est ici un programme qui utilise le protocole pour communiquer avec un autre programme. HTTP/CGI n'est pas une solution orientée-objet et pose des problèmes de modélisation des interactions et de performances non négligeables.

[OH97] propose un comparatif beaucoup plus complet (170 pages) et conclut que CORBA est sans doute une très bonne solution par rapport à ses concurrents mais les auteurs n'envisagent que des arguments techniques, sans prendre en compte les contraintes organisationnelles du développement logiciel (voir à ce propos [CWA97]). Dans le cadre de l'aide à l'analyse de données biologiques et des contraintes propres aux acteurs du domaine, il nous semble que cette conclusion doit être modulée. Les inconvénients que nous avons indiqués prennent tout leur sens dans le contexte de la recherche universitaire où les laboratoires qui gèrent une base de données n'ont pas toujours les ressources suffisantes pour mettre en place et administrer un serveur CORBA. Dans ce cas, l'approche HTTP/CGI peut être compétitive, étant donné que le laboratoire investit déjà des ressources dans la mise en place de l'accès WWW.

Il apparaît donc que si de grands centres de recherche disposent de l'infrastructure nécessaire au développement de serveurs CORBA pour les bases de ressources biologiques (données et traitements), les centres plus modestes se tourneront vers d'autres technologies (HTTP ou RMI). Il faut donc tenir compte de la diversité des solutions disponibles pour la communication réseau dans l'architecture du système d'aide à l'analyse de données. Le modèle des composants permet ce type de souplesse ; montrons comment.

10.2.4 Conséquences pour l'analyse de données

Des composants spécialisés pour l'accès aux bases de ressources (les outils réseaux sur la figure 9.1) permettent d'encapsuler les détails de la communication avec le centre de ressources. Ces composants, du point de vue de l'utilisateur, ne se distinguent des outils qui accèdent à des données locales que par le fait que l'analyste les a obtenus auprès de la base de ressources.

Dans le scénario que nous évoquons, les centres de ressources proposent, en plus d'un serveur de ressources, les moyens d'accéder à la ressource. Ce moyen est un composant dédié. Les centres qui choisissent de développer des serveurs CORBA distribuent des composants qui accèdent aux ressources en utilisant les amorces CORBA des objets serveurs, tandis que les centres qui proposent d'autres types d'accès (HTTP, Sockets, RMI, ...) peuvent fournir des composants qui communiquent par le protocole choisi et qui adhèrent aux conventions de l'interaction définies par eux. Du point de vue de l'analyste, si le composant est bien conçu (paramètres du composant indépendants du protocole de

46. Les spécifications de CORBA, au contraire, sont largement diffusées et discutées avant leur adoption.

communication) il ne doit y avoir aucune différence mis à part les différences de performances inhérentes au protocole de communication. Le composant livré propose une ou des interfaces adaptées au domaine d'application.

10.3 Adaptabilité des présentations

Cette section décrit très rapidement le modèle de conception MVC (Modèle Vue Contrôleur) [Gol84] et comment il apporte à l'architecture la flexibilité nécessaire pour intervertir facilement plusieurs présentations d'une même donnée. Ce modèle n'est pas une solution générale à ce type de problèmes, mais dans d'une architecture à base de composants où l'utilisateur a la liberté de construire un traitement en combinant des composants, il semble approprié. Dans la suite de cette section nous prendrons l'exemple de la construction d'une représentation de séquence biologique dans le cadre du système que nous présentons.

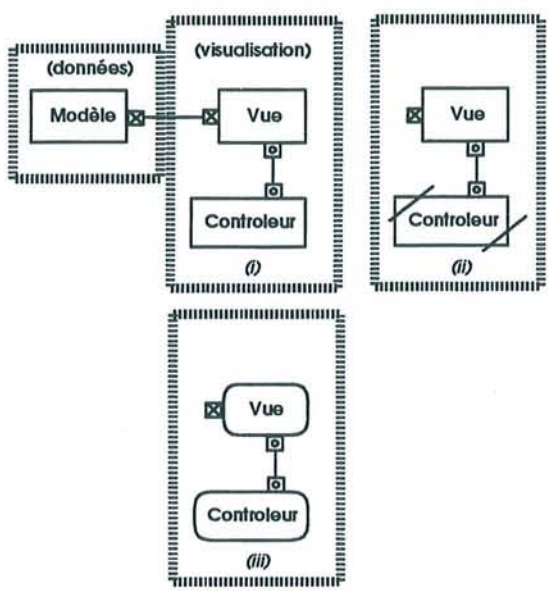


FIG. 10.2 – *Architecture et adaptabilité du modèle MVC*
Les interactions des composants Modèle, Vue et Contrôleur sont explicitées. L'adaptabilité du modèle est illustrée par *(i)* le couple V/C correspondant à un type de visualisation (HTML par exemple), *(ii)* le couple V/C′ qui propose un contrôleur différent pour le même type de visualisation (interaction utilisateur différente avec les paramètres de la vue), *(iii)* un couple v/c très différent qui peut changer notablement la construction de l'interface graphique par rapport à *(i)* (passage de HTML à un environnement de fenêtrage par exemple).

10.3.1 Le modèle (MVC)

Le modèle contient les données à présenter. Par exemple, une séquence biologique (d'amino-acides ou d'acides nucléiques). Dans le cadre d'un système à base de composant, le modèle est accessible à travers une interface. Comme nous l'avons vu en deuxième partie utiliser le concept d'interface permet de modifier des parties du logiciel de façon simple.

10.3.2 La Vue (MVC)

La Vue est le composant qui construit ce que l'analyste voit. Dans un premier temps nous avons réalisé un composant qui présente une séquence biologique sous la forme d'une phrase du langage HTML. Nous avons présenté en première partie [CM98] deux méthodes de représentation d'alignement multiples en HTML. Ces méthodes sont applicables à la représentation de séquences seules. Le problème est ici de gérer le choix entre ces deux type de représentations. Typiquement ce choix est un paramètre du composant Vue.

10.3.3 Le Contrôleur (MVC)

Le Contrôleur est le composant qui permet de présenter à l'analyste les paramètres de la Vue (choix de représentations). Le Contrôleur peut proposer une traduction et une explication des choix possibles de façon à ce que le l'utilisateur comprenne bien ce qu'il peut changer. Le Contrôleur joue un rôle un peu similaire aux "Preferences" disponibles dans le programme Viseur. Comme dans ce cas, lorsque les paramètres de la Vue changent, la représentation construite par la vue devrait être mise à jour pour refléter le nouvel état. Dans ce modèle de conception, la Vue est indépendante du Contrôleur : on peut changer la façon d'obtenir les choix de l'analyste (le Contrôleur) sans rien changer au composant de présentation des données. Par contre le Contrôleur est dépendant de la vue puisqu'il utilise une de ses interfaces pour fixer les paramètres qu'il contrôle.

10.3.4 Changement de type de représentation

Pour changer complètement de type de représentation, il faut changer le couple Vue/Contrôleur. Tant que l'interface d'accès aux données du Modèle permet d'obtenir les données pour construire la représentation il est inutile de modifier le Modèle.

Une conséquence qui devrait s'imposer après la lecture des exemples contenus dans la deuxième partie est qu'il faut spécifier avec soin les interfaces qui permettent d'accéder aux données du domaine de façon à ne jamais les changer. Ainsi, des développement tiers qui utilisent l'interface du Modèle peuvent présenter des représentations originales (composées du couple Vue/Contrôleur) qui peuvent s'intégrer immédiatement à l'environnement d'aide à l'analyse. La figure 10.2 résume l'architecture du modèle MVC.

10.4 Construction d'un prototype dans le cadre d'un sous-problème

Construire un outil d'aide à l'analyse est une tâche ambitieuse qui peut nécessiter beaucoup de temps et de travail. Il peut donc être intéressant de tester les idées que nous proposons dans un cadre plus restreint, à condition que ce cadre soit analogue au problème de l'analyse de données biologiques. Examinons comment le sous-problème que nous avons choisi remplit cette condition.

10.4.1 Présentation du sous-problème, justifications

Nous remarquons que la construction de présentation de données biologiques sur le WWW ou sur un intranet est très analogue au processus d'ADB. Examinons rapidement cette activité à travers le schéma d'ADB (figure 8.7).

Local Database. Les informations présentées sur le WWW sont souvent extraites d'une base de donnée locale ou d'un système de fichier,

Query tools. Les outils de requête de la base de données ou les outils du système d'exploitation (`find`, `grep`,...) remplissent ce rôle pour les systèmes de fichiers.

Analysis and AI tools. Certains outils d'analyse peuvent être invoqués pour construire les pages HTML. Exemple le plus courant : requête BLAST ou FASTA. La puissance de machines de calcul distinctes du serveur HTTP peut être utilisées dans cette étape (certaines machines peuvent être dédiées à une tâche et leur configuration optimisée pour la réaliser). Tous les outils d'analyse du domaine peuvent être utiles.

Visualisation tools. Des outils dédiés à la visualisation comme les outils qui construisent des images, ou des pages HTML, ont leur place ici.

Presentation tools. Ces outils sont voisins des outils de visualisation. On placera plutôt dans cette catégorie les outils qui construisent des sorties dont la qualité est compatible avec le cadre d'une présentation de résultats.

Data transformation tools. Outils pour transformer les résultats des outils de visualisation et de présentation : réduction du nombre de couleur d'une image GIF, extraction des pages d'un fichier PostScript pour les traduire en fichiers EPS...

Network Query tools. Certaines présentations nécessitent d'accéder à des données distantes. Par exemple, construire le diagramme en serpent d'un GPCR, sur requête, en affichant l'information de mutagenèse la plus à jour, demanderait deux requêtes réseaux : une pour obtenir la séquence de Swiss-Prot, la seconde pour obtenir les mutations de TinyGRAP.

Une distinction importante entre le processus d'ADB et la présentation d'informations biologiques sur le WWW est que l'ADB pose les questions « Quelles données ? Comment les analyser ? Quelles connaissances ? Comment les présenter ? » alors que l'individu qui veut construire une présentation WWW de données ou de connaissances biologiques devrait connaître la plupart des réponses à ces questions avant de commencer sa tâche. Nous appellerons cet individu concepteur dans la suite de ce texte, non parce qu'il développe des présentations mais surtout parce qu'il doit concevoir les réponses aux questions pré-posées avant de commencer ses développements. Le concepteur utilise approximativement les mêmes outils que l'analyste mais connaît son but dès le départ alors que l'analyste le modifie pendant sa confrontation avec les données. En conséquence, un environnement qui propose les huit points du cahier des charges (chapitre 9) devrait aider aussi bien l'analyste pendant le processus d'ADB que le concepteur de présentations de données biologiques sur le WWW. Les différences entre les deux tâches imposeront sans doute qu'un environnement plus performant serait préféré pour réaliser l'ADB (plus grande quantité de données, temps de réponse à réduire pour conserver l'interactivité, ...).

11

Description du prototype Crover

Ce chapitre décrit la réalisation du prototype d'un système pour simplifier la présentation de données biologiques sur le WWW — prototype que nous dénommons Crover. Nous avons expliqué (section 10.4) que l'activité de présentation de données biologiques sur le WWW se compose d'un sous-ensemble des tâches d'ADB.

11.1 Assemblage des traitements

Crover est un prototype qui permet de construire des traitements dédiés à la présentation de données biologiques sur le WWW. Un traitement, inspiré de la figure 9.1 est constitué de composants logiciels assemblés de telle façon qu'ils

- utilisent des données et
- construisent une présentation.

L'assemblage des composants est à la charge du concepteur qui construit le traitement adapté à la tâche de présentation. La version actuelle du prototype fournit un ensemble de composants "Java beans" (voir figure 11.1) que le concepteur peut assembler dans l'outil "beanbox".

Cet outil de composition de "Java beans" est distribué gratuitement par *Sun microsystems* afin d'aider au développement des composants Java. D'autres systèmes de développement à base de composants "Java beans" existent, commerciaux, qui offrent un plus grand confort de développement (visualisation des connections après établissement, par exemple) mais leur usage implique d'adapter les composants développés à leurs spécifications. Étant donné qu'il existe plus de cinq outils de ce type et qu'ils requièrent tous une adaptation particulière des composants, que la tendance générale du développement de ces outils va vers des systèmes d'adaptation automatique à partir des composants "Java beans" originaux, nous pensons qu'il est judicieux de patienter. Nous notons néanmoins que parce que ces systèmes sont plus ergonomiques et plus puissants que l'outil "beanbox", leur support permettra à des utilisateurs non spécialistes d'assembler des composants pour produire un traitement. De ce point de vue, l'outil *Java Studio* [Sun97b] est caractéristique : son ergonomie nous semble supérieure à celle d'AVS/Express lorsqu'il s'agit de connecter les composants du système.

Pour assembler deux composants à l'aide de l'outil "beanbox", le concepteur sélectionne le composant à l'origine de la connection (origine de la flèche sur les schémas de réseau de composants) puis la propriété qu'il veut transferer. Une propriété est un nom qui permet de distinguer entre plusieurs interfaces de même type, mais de fonctions différentes, portées par le même composant. Par exemple, un composant qui stocke une séquence peut proposer deux interfaces de type `String` : l'une correspondant au nom de la séquence, l'autre aux résidus de la séquence. Ce composant proposera alors les deux propriétés `shortName` et `residueCodes`.

Lorsque composant source et propriété sont choisis, le concepteur est invité à sélectionner le composant destination, puis la propriété destination de la connection. Le type des interfaces des propriétés est vérifié par l'outil "beanbox" qui s'assure que la connection est valide avant de l'établir. Lorsque les

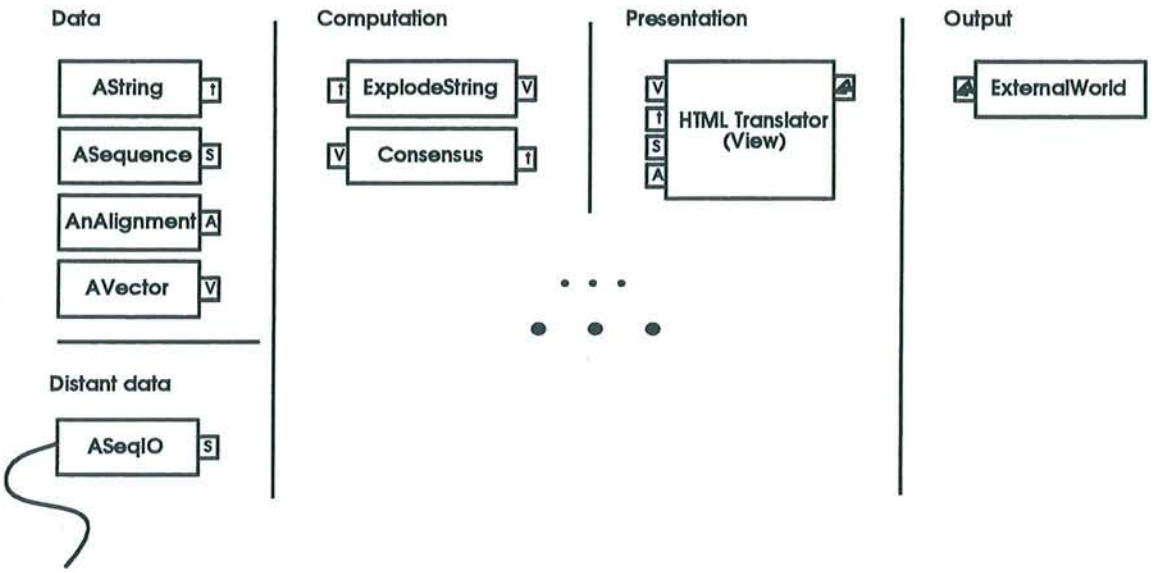


FIG. 11.1 – *Les premiers composants Crover (au moment de la rédaction)*
Les composants `data` et `Distant data` correspondent aux composants `data` de la figure 11.4. Les types des composants et ceux de leurs interfaces les plus importantes sont indiqués. La présentation introduite en deuxième partie est utilisée pour indiquer les interfaces. Les interfaces [↑], [S], [A], [V], [🖵], correspondent, respectivement, à des données de types (objets) : chaîne de caractères, séquence (biologique), alignement (de séquences biologiques), vecteur (une collection de données), présentation. Les interfaces sur le côté gauche des composants sont des interfaces d'entrées, alors que celles présentes sur le côté droit sont des interfaces de sorties. Les données proposent en général leur interface en entrée/sortie mais seule la sortie est indiquée. `Data` et `Distant data` sont les Modèles de MVC et `HTML Translator` est la Vue.

composants sont connectés et que la propriété connectée est mise à jour dans le composant source, elle est transmise au composant destination qui peut réagir à ce changement (par exemple en effectuant un calcul dont le résultat dépend de la propriété mise à jour). Dix secondes suffisent à connecter deux composants de cette manière. Le concepteur a manipulé des objets graphiques et n'a saisi aucune ligne de programme. C'est en assemblant d'autres composants de cette façon que le concepteur réalise des présentations HTML.

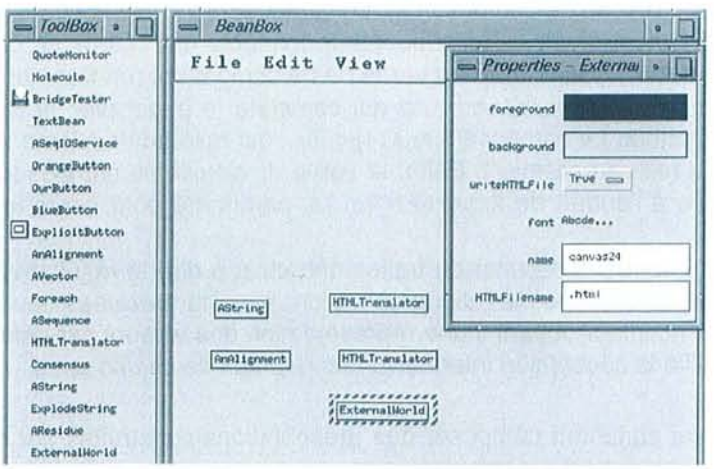
Les composants Crover sont des composants Java (des "Java beans"), écrits en langage Java, donc portables d'une machine à une autre sans recompilation. De cette façon, ils peuvent être assemblés sur une machine (poste du concepteur), transférés par le réseau vers une machine proche des données (site WWW de la base de donnée), puis exécutés sur cette machine de traitement. Cette approche respecte un grand principe de la recherche de performance en informatique distribuée : les traitements doivent être proches des données qu'ils manipulent.[47]

Nous reviendrons bientôt sur les mécanismes qui permettent l'échange de traitements entre le poste du concepteur et le serveur WWW. Avant cela, présentons un exemple de réseau de composants Crover pour mieux comprendre comment assembler les composants du prototype Crover pour construire une représentation HTML de données biologiques.

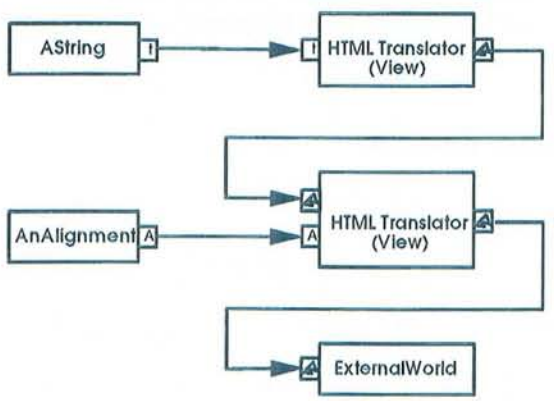
11.2 Présentation d'un alignement multiple

La figure 11.2 présente un premier exemple du développement de page WWW à l'aide du prototype Crover. Crover est utilisé pour composer une page HTML comportant un texte suivi d'un fragment d'ali-

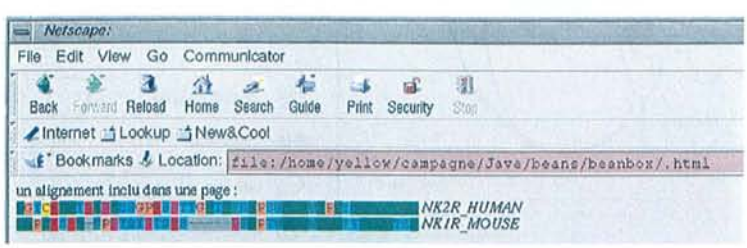
47. Dans les calculateurs parallèles les logiciels sont optimisés pour faire migrer les données plus près des processeurs qui les transforment, ici on transfère le traitement vers les données qu'il manipule.



(a) Composants assemblés dans l'outil "beanbox".



(b) Réseau de composants.



(c) Visualisation HTML/CSS du résultat du traitement.

FIG. 11.2 – *Un exemple de développement avec les composants Crover*

gnement multiple de deux séquences. La méthode de présentation de l'alignement est celle présentée en première partie, variante feuilles de style (CSS). Sur la figure, (a) montre les composants Crover à la fin de l'assemblage dans l'outil "beanbox".

Le schéma, noté (b) sur la figure 11.2, explicite les connections entre les composants utilisés pour construire le traitement (réseau de composants). Le composant `ASring` contient la chaîne de caractères à afficher avant l'alignement multiple. Le concepteur connecte une référence du composant `ASring` à l'entrée du composant `HTMLTranslator`. La sortie de ce composant (un fragment d'HTML) est connectée à l'entrée d'un autre `HTMLTranslator`, ce qui complète la page avec la représentation constuite par ce deuxième traducteur. Le composant `AnAlignment` est relié au deuxième traducteur (de la même façon que `ASring` est relié au premier). Enfin, la sortie du deuxième composant de traduction (`HTMLTranslator`) est reliée à l'entrée de `External World`, paramétré pour écrire la présentation dans un fichier nommé `.html`.

La sous-figure (c) montre le résultat du traitement, c'est à dire le résultat HTML interprèté par un navigateur WWW. Les séquences de l'alignement n'ont pas été précisées pendant la construction du traitement, aussi le concepteur obtient-il une représentation des valeurs par défaut du composant `AnAlignment`. Ceci simplifie la conception interactive des réseaux de composants.

Cet exemple illustre comment composer des présentations construites par des outils différents — encapsulés dans des composants manipulables sans programmation — lorsque ces outils construisent une représentation HTML. Ce principe de construction de traitement permet aussi de réaliser des calculs non visuels. La construction de traitement par assemblage de composants et l'utilisation du modèle MVC répondent aux points 4. à 8. du cahier des charges du chapitre 9.

Dans la prochaine section, nous décrivons un moyen de rendre les traitements indépendants de leur environnement d'exécution, et donc de les rendre plus facilement transférables d'une machine à une autre.

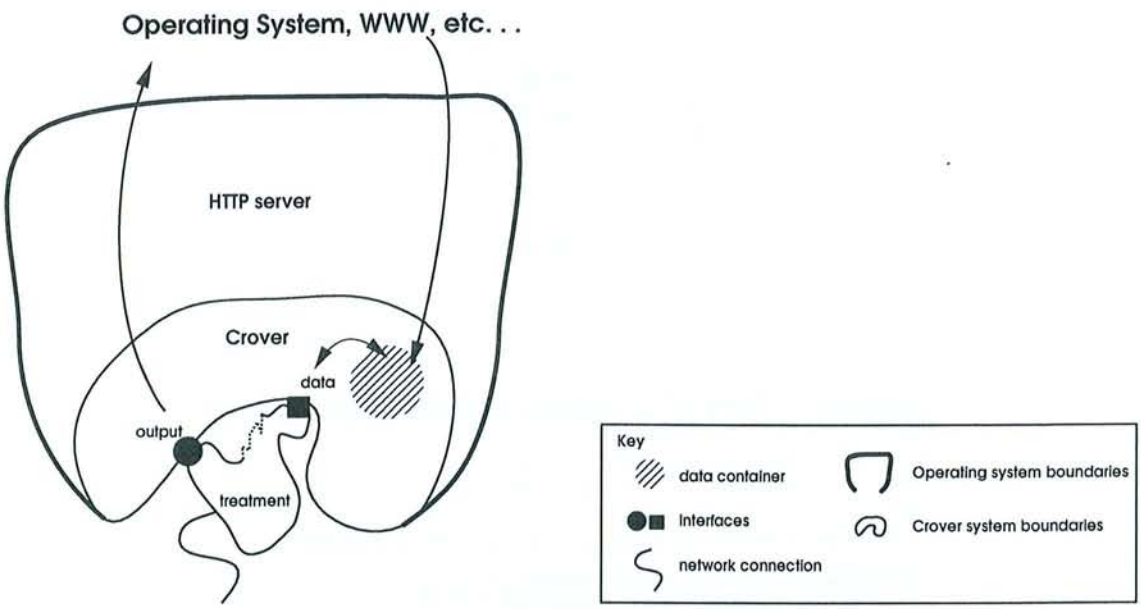


FIG. 11.3 – *Vue d'ensemble du système Crover*

Le serveur HTPP fonctionne dans un environnement constitué du système d'exploitation et du réseau. L'environnement d'exécution Crover (EEC) définit les interfaces `output` et `data` qui permettent d'accueillir les traitements construits à l'aide des composants Crover (cf. figure 11.4). Le traitement baigne dans le système d'exploitation et peut utiliser le réseau si des mécanismes de sécurité ne restreignent pas ces accès. Les flux de données sont indiqués par des flèches. L'EEC maintient un ensemble de données locales conservées entre les différentes exécutions du traitement.

11.3 Environnement d'Exécution Crover (EEC)

Le prototype que nous décrivons ici se compose des composants Crover (figure 11.1), qui permettent de construire les traitements, et d'un environnement d'exécution pour ces traitements. Cet environnement isole le traitement de la diversité des environnements d'exécutions (systèmes d'exploitation et/ou serveurs WWW différents). Cet environnement d'exécution, que nous désignerons par Environnement d'Exécution Crover, ou EEC, s'interface avec un serveur HTTP de façon à transmettre le résultat du traitement (un document HTML) vers le navigateur WWW de l'utilisateur.

Dans cet environnement, les traitements disposent de deux types d'interfaces : des interfaces pour obtenir ou stocker des données et une interface pour transmettre la visualisation/présentation construite à la fin du traitement (optionnel). La figure 11.3 présente les interactions de l'environnement d'exécution Crover avec le serveur HTTP et le traitement.

La figure 11.4 illustre comment nous traduisons le concept de traitement dans les environnements à base de composants. Les interfaces *données* et *sortie* de l'environnement d'exécution sont utilisées par des composants spécialisés (`Data`, `External World`, sur la figure). Ces composants, directement manipulables dans les environnements de développement graphique à base de composants, peuvent être combinés, par le concepteur, avec d'autres composants pour construire le traitement `treatment`. La partie `specific treatment` peut être constituée d'un mélange de composants du système avec des composants développés par des tiers.

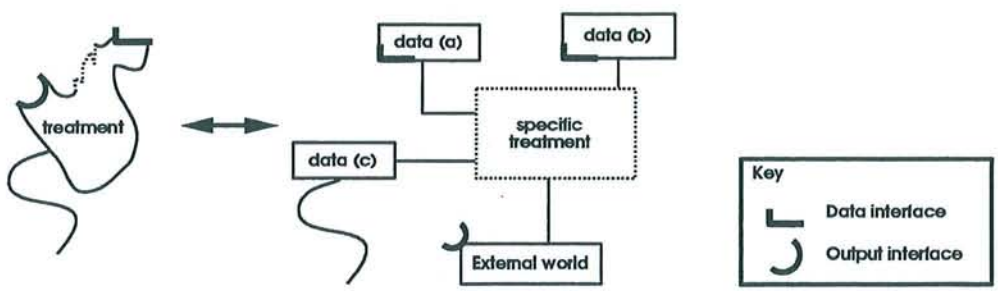


FIG. 11.4 – *Traduction des interfaces Crover dans le formalisme des composants*
La représentation unitaire du traitement de la figure 11.3 est éclatée en ses composants constitutifs. Les composants `data` utilisent l'interface d'accès aux données de l'EEC et proposent la donnée correspondante sur leur interface de sortie. Le composant `External World` propose une interface d'entrée de type `output`. Pour une présentation WWW, le type `output` permet de stocker du HTML. Le composant `data (c)` est un composant d'accès à des données distantes qui fournit la donnée obtenue via le réseau sur son interface de sortie. Du point de vue de l'utilisateur, `data (a)` et `data (c)` peuvent s'utiliser de la même façon.

11.4 Implantation

Cette section est destinée aux spécialistes qui apprécieront quelques détails sur l'implantation de l'EEC et des composants Crover.

11.4.1 L'Environnement d'Exécution Crover (EEC)

L'environnement d'exécution Crover est implanté dans le serveur Netscape Enterprise Server, à l'aide du service WAI [Cor97] de Netscape, basé sur une implantation CORBA de Visigenic.[48] L'implantation est réalisée avec le langage Java pour permettre d'exécuter des traitement écrits dans ce langage.

L'EEC peut être implanté à l'intérieur de tout serveur HTTP qui propose un mode d'exécution de requêtes avec conservation d'état entre les invocations. CGI/BIN est un protocole sans état mais peut

48. Un service WAI est une application qui implante un objet serveur CORBA pour communiquer avec le serveur HTTP.

permettre d'émuler un système à états (voir [OH97] pages 226–227). Nous lui préférerons toutefois les solutions Servlet ou WAI lorsqu'elle sont disponibles, pour des raisons de performances. La conservation d'état entre les invocations permet d'implanter l'interface `data` de l'EEC qui gère la communication de données entre deux invocations d'un traitement.

Le composant `ASeqIO`

Le composant `ASeqIO` lit une séquence ou un alignement de séquences sur le disque dur d'une machine du réseau, parmi les vingt-deux formats de séquences supportés par le logiciel SeqIO [Kni96].

Son implantation se compose de deux parties : un serveur CORBA sur la machine du réseau dont le disque dur fournit les séquences, un client CORBA encapsulé dans le composant `ASeqIO`. L'implantation du composant `ASeqIO` est présentée en annexe G. Elle montre que programmer des requêtes réseau, bien que simplifiée par l'utilisation de CORBA, reste une activité de programmation, et gagne donc à être encapsulée dans un composant.

11.5 Exemples et Résultats

Cette section complète la description du système Crover en montrant comment il répond aux huit points du cahier des charges que nous avons développés dans le chapitre 9.

11.5.1 Présentation d'un alignement obtenu par accès distant

La figure 11.5 illustre comment des composants réseaux peuvent être intégrés au traitement. Le composant ASeqIO encapsule la logique d'un client CORBA et permet d'obtenir un alignement stocké sur le disque dur d'une machine du réseau. Le serveur CORBA est responsable d'obtenir l'alignement à partir d'un nom de fichier. Il est possible de concevoir des composants analogues qui permettraient d'obtenir les séquences ou alignements, ou autres données stockées dans les bases de données biologiques. Le composant `ASeqIO` illustre comment répondre au point 1. du cahier des charges.

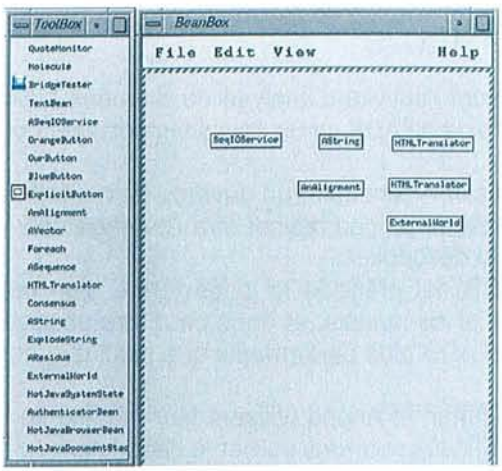
La figure 11.5 (c) montre l'interface que l'EEC propose au concepteur pour installer un nouveau traitement, sans effort, sur le serveur WWW. Après assemblage le traitement est sauvegardé sous la forme d'un *package* Java, qui est un fichier d'un format particulier qui contient la description de classes Java et d'autres données. Le *package* Java doit être déposé sur la machine qui héberge le serveur HTTP, dans un répertoire à partir duquel les classes qu'il contient peuvent être instanciées par l'EEC. Pour cela, le concepteur télécharge le fichier *package* à l'aide d'une requête POST vers le serveur HTTP (bouton "Download new treatment and run it"). Lorsque le fichier *package* est téléchargé l'EEC active automatiquement le traitement qu'il contient et présente le résultat produit ou le détail des exceptions ou erreurs qui l'ont interrompu.

Ce mécanisme offre des perspectives très intéressantes pour la construction de ressources WWW plus interactives, mais avant cela il nous faudra mettre au point les mécanismes de sécurité indispensables à une utilisation publique du transfert de traitement.

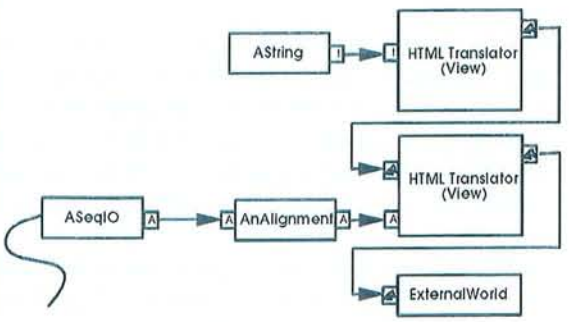
11.5.2 EEC intégré à l'environnement de développement

Cet exemple présente comment l'utilisation combinée des composants Crover et d'un composant qui implante un navigateur HTML permet de disposer d'un environnement de développement et de mise au point interactif. La figure 11.6 (a) montre le réseau de composants dans l'environnement de développement. Lorsque le concepteur modifie le contenu des champs de saisie `TextBean` qui contiennent le nom de la séquence et la séquence des résidus, la représentation HTML interprétée par le composant `HotJavaBrowserBean` est mise à jour.

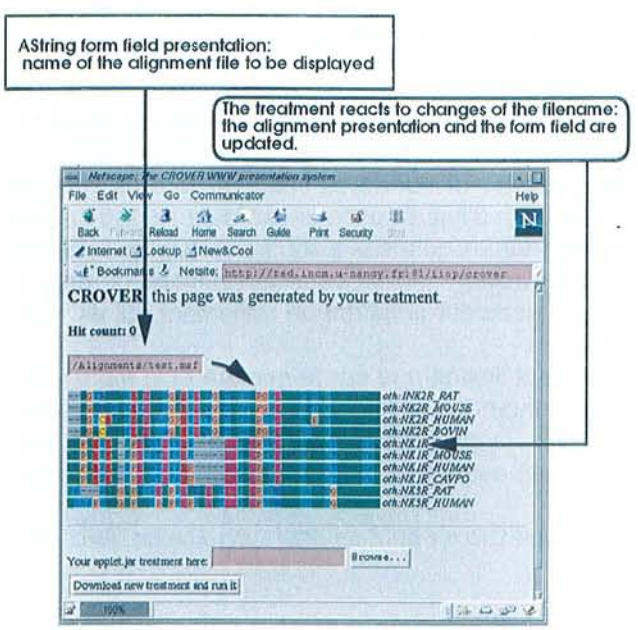
Le traitement, montré en (b) construit deux représentations d'une séquence biologique. La première, adaptée à ce type de donnée, la seconde plus adaptée pour les collections de données de petite taille. Le vecteur construit dans le cadre de ce traitement, à partir de la séquence des résidus, pourrait être



(a) Composants assemblés dans l'outil "beanbox".



(b) Réseau de composants.



(c) Visualisation HTML/CSS du résultat du traitement.

FIG. 11.5 – *Composant ASeqIO dans le système CROVER*

réutilisé par des traitements qui fonctionnent avec des collections de données (calcul du consensus, classifieur, etc.).

11.6 Bilan et perspectives

11.6.1 Intérêts d'un modèle de l'activité d'ADB

Le travail présenté dans cette quatrième partie décrit l'activité d'analyse de données biologiques (ADB). Cette description, souvent omise des travaux d'aide à l'ADB, nous semble importante pour deux raisons.

Premièrement il est nettement plus probable de réussir à construire un ouvrage (texte, logiciel, système,etc.) lorsque l'on dispose d'une description précise de ce que devrait être l'ouvrage une fois terminé plutôt que de commencer sa réalisation sans cette description.

Deuxièmement, puisque notre conception de l'ADB est précisée ici sous forme d'un modèle[49] d'autres recherches pourront venir compléter ou modifier ce modèle. et donc peut être poser de nouvelles bases pour la conception de systèmes d'aide à l'ADB plus performants que celui que nous proposons.

En tout état de cause, de la même façon que Brachman et Anand utilisent leur modèle de l'ECBD pour évaluer les environnements d'aide à l'ECBD [BA96] nous pourrons utiliser la description du processus d'ADB et le cahier des charges qui s'en déduit pour comparer les environnements d'ADB de façon plus méthodique — l'environnement (a) facilite-t-il les accès aux données biologiques par le réseau depuis leurs sources, comment? même question pour l'environnement (b), etc.

11.6.2 Apports du prototype Crover

Crover, le prototype de système d'aide à la présentation de données biologiques sur le WWW que nous avons développé, permet d'illustrer comment les technologies composants et/ou distribuées permettent de réaliser un outil qui remplit le cahier des charges de l'aide à l'ADB. Dans ce contexte, il montre que les technologies réseaux, et leur développement vers les systèmes d'objets distribués — tel CORBA —, qui suscitent beaucoup d'intérêt dans le cadre de l'activité d'intégration des BDB, peuvent être utiles pour l'ADB, pour amener les données jusqu'au bureau de l'analyste.

Au titre de ses points forts, Crover simplifie considérablement la mise à jour d'information à partir de données qui évoluent : le concepteur construit un traitement qui est déclenché dès qu'une de ses données d'entrée est modifiée.

Les usages de Crover ne sont limités que par le nombre et la nature des composants qu'il permet de manipuler. Or nous avons conçu Crover de façon à permettre son utilisation avec des composants développés par des tiers. En fait, le système permet déjà d'utiliser tous les composants "Java beans" disponibles et bénéficie de ce fait des efforts de développements consentis par de grands acteurs du secteur informatique.[50]

Le développement du système Crover peut se poursuivre selon deux axes.

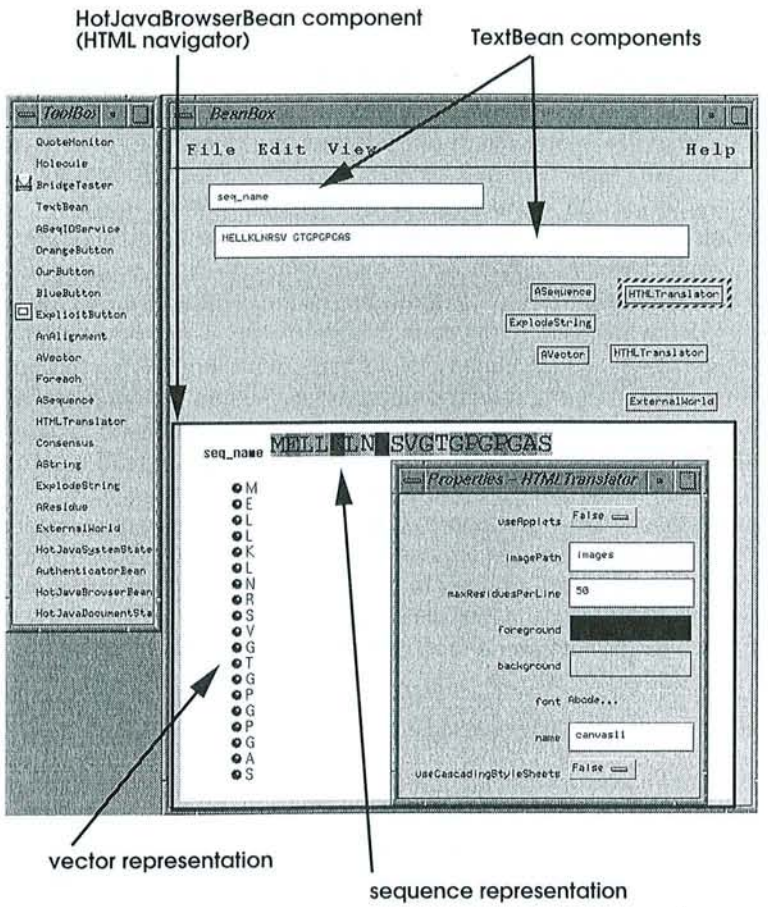
Environnement d'Exécution Crover

Premièrement l'EEC peut être étendu pour faciliter son utilisation. Plusieurs directions sont possibles.

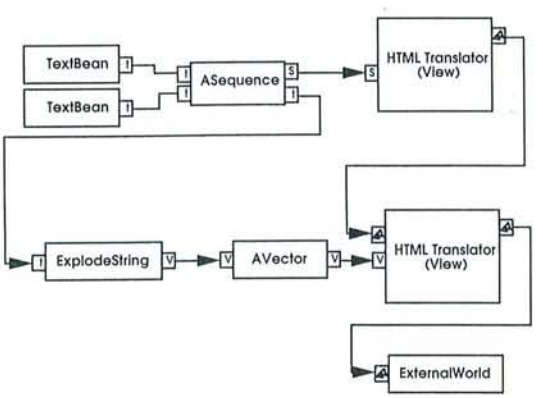
- adaptation à plusieurs serveurs WWW,
- développement d'un EEC indépendant d'un serveur WWW,
- gestion de plusieurs traitements simultanés,
- optimisation du temps de mise à jour des traitements,

49. Ce modèle comprend : tâches et outils de l'activité d'ADB, transferts de données et passages d'une tâche à l'autre au cours de l'activité.

50. C'est une des applications de cet adage : « Évitez de faire ce que d'autres peuvent faire à votre place », que pourrait compléter, « surtout s'il le font mieux ».



(a) Composants dans l'outil "beanbox".



(b) Réseau de composants.

FIG. 11.6 – *Environnement de développement Crover avec un composant de navigation HTML*

– raffinement du modèle d'événement pour améliorer les temps de réponse des requêtes.

Composants Crover ou composants tiers

Deuxièmement, il est possible d'étendre le système très simplement en développant de nouveaux composants. Ces composants spécialisés pour la biologie, la modélisation moléculaire, l'analyse de données, la classification automatique, etc. bénéficieront dès leur écriture de l'existence des composants déjà développés à condition d'utiliser les mêmes interfaces que les composants Crover ou de fournir des adaptateurs. Ceci répond au points 2. et 3. du cahier des charges.

Notons que le développement d'un composant est plus simple que celui d'une applet[51] et son intégration à l'outil de développement ou d'ADB est très simple (copier un fichier dans un répertoire).

Enfin, un apport très important de ce système est de fixer un cadre de travail, d'expérimentation et de développement qui favorise les extensions méthodiques du système d'ADB. Ainsi, Crover permet d'envisager de développer des systèmes bioinformatiques de manière distribuée. Par exemple, un groupe de recherche peut construire certains composants du système alors que d'autres se concentrent sur les composants qui implantent les traitements dont ils sont spécialistes. Les versions préliminaires des composants peuvent être échangées très facilement à travers le réseau, au cours du processus de développement, ce qui facilite les tests du système en cours de construction.

Pour conclure cette description des apports du prototype Crover, nous voudrions remarquer que de notre point de vue de développeur de services WWW, le système Crover permet de mieux structurer les développements, de réutiliser l'existant facilement, de simplifier la présentation de pages complexes,... mais qu'il est clair que la validation de ce système ne peut venir que de l'accueil et de l'utilisation que la communauté bioinformatique en fera, et pas de notre point de vue, certainement subjectif.

11.6.3 Ouverture vers d'autres domaines d'analyse

Le système Crover a été conçu à partir du cahier des charges dérivé du modèle de la tâche d'analyse de données que nous avons présenté. Il est donc probable que son architecture peut permettre de faciliter d'autres activités que l'ADB, si ces activités sont analogues, ou incluses dans notre description de l'activité d'ADB. Par exemple, plusieurs discussions nous amènent à penser que l'architecture du système Crover faciliterait l'analyse des informations disponibles sur internet — informations en langage naturel notamment — si tant est que des composants adaptés à ce domaine de recherche soient conçus puis développés.

51. Pour le comprendre il suffit de considérer que le composant ne doit implanter qu'une partie du travail d'une applet équivalente. Si le composant transforme des données il ne fait que cela. Quand une applet transforme des données, elle doit saisir les données, faire la transformation et présenter le(s) résultat(s). Le composant sera connecté à d'autres composants pour obtenir le même effet.

Conclusion

Le projet Viseur

Le travail présenté dans cette thèse se compose de trois volets. Le premier volet, présenté dans la première partie de ce manuscrit, décrit le travail de conception et de développement réalisé sur le thème de l'aide à la modélisation des récepteurs couplés aux protéines G. C'est le projet Viseur. Dans le cadre du développement de ce projet, nous avons été amené à concevoir des méthodes pour améliorer l'intégration de données hétérogènes, pour faciliter l'échange d'informations biologiques à travers les réseaux informatiques (internet, intranets), et pour simplifier les confrontations entre les modèles moléculaires de ces systèmes avec les données expérimentales.

Le nombre de sites utilisateurs du programme Viseur et les retours des utilisateurs, qui consultent les représentations en serpent construites pour GPCRDB, témoignent clairement du succès de ce projet.

Du point de vue de sa conception, le programme Viseur est un logiciel très classique, construit par étapes à mesure que les directions de recherches s'esquissaient, sur la base de son utilisation et sous l'impulsion de ses utilisateurs. C'est le cas du développement de beaucoup de logiciels scientifiques et ne facilite pas l'évolution de ces outils, intimement couplée à l'évolution des connaissances — connaissances du domaine pour lequel l'outil a été conçu.

Vers des méthodes pour favoriser la réutilisation logicielle

Dans le deuxième volet de ce travail, nous avons expliqué les méthodes actuelles d'aide à la réutilisation des logiciels à travers l'exemple du développement d'un outil d'analyse conformationnelle. L'outil que nous avons réalisé, qui pourrait être appliqué à l'analyse conformationnelle des ligands des GPCRs, illustre comment réutiliser des logiciels existants pour construire des logiciels plus évolutifs. Dans le cadre de cette activité nous avons proposé les spécifications de quelques interfaces adaptées au domaine de la Chimie informatique : interfaces pour échanger des données moléculaires, interfaces pour l'évaluation d'énergie d'une conformation, pour l'optimisation et la paramétrisation de l'optimisation.

Aide à l'analyse de données biologiques

Dans le troisième volet de ce travail, nous présentons une réflexion à propos de systèmes d'aide à l'analyse de données biologiques. Ces systèmes, dont le programme Viseur est un exemple, sont les outils dont nous avons besoin pour tirer parti de la quantité de données produite par les projets génomes et les évolutions des biotechnologies. Nous suggérons que toute amélioration méthodologique dans le domaine de la production de données incite à améliorer les méthodes destinées à faciliter leur analyse et leur interprétation. Mais comment développer ces nouvelles méthodes ?

Pour essayer d'améliorer l'activité d'analyse de données biologiques, nous avons choisi de la modéliser, puis de proposer des méthodes susceptibles de simplifier la tâche décrite dans le cadre du modèle. Notre approche a donc au moins deux faiblesses : *(i)* notre modèle de l'activité d'ADB peut être incorrect ou incomplet, *(ii)* les conclusions que nous tirons de l'étude du modèle peuvent-être erronées. Ces faiblesses sont le fait de toutes les entreprises de prédiction basées sur des modèles et nous savons qu'il est possible de les contourner, en raffinant le modèle ou en améliorant son exploitation.

Nous terminons le troisième volet de ce travail par la description d'un prototype logiciel, le système Crover, qui illustre les points que nous pensons importants pour faciliter l'activité d'analyse de données biologiques. Le système Crover est destiné dans un premier temps à faciliter la présentation de données biologiques sur le WWW, mais nous avons avancé (section 10.4) que cette activité est une sous-activité de la tâche d'ADB. Le système Crover illustre comment les technologies à base de composants et les technologies client/serveur (CORBA, DCOM, RMI, HTTP, etc.) se complètent. Il montre comment construire un cadre de développement ou d'analyse qui intègre les ressources distantes dans le traitement des données. Il suggère aussi que ce cadre peut être adéquat à l'analyse de données de structure complexe, telles que peuvent être les données des sciences biologiques et chimiques. Nous voudrions noter, pour terminer, que la prise en compte de tous les facteurs importants pour concevoir des systèmes d'aide à l'analyse et à la compréhension de données impose, à notre avis, d'adopter une

approche pluridisciplinaire qui peut et devrait se traduire par des collaborations entre les spécialistes de trois domaines. Cette approche pluridisciplinaire devrait comprendre :

- la connaissance du domaine d'application — pour comprendre le fonctionnement des modèles,
- les résultats des sciences du traitement de l'information formelle : l'informatique — pour construire les représentations logicielles des modèles,
- et peut-être, les résultats d'approches psychologiques qui permettraient d'améliorer le transfert bidirectionnel entre l'information formelle et la connaissance détenue par les experts du domaine — pour apprendre comment construire les représentations les plus adaptées à une tâche, et donc ne plus aider seulement à l'analyse mais aussi à la compréhension des données.

G

Le composant SeqIO

Les pages suivantes contiennent l'implantation des serveur CORBA et composant client `ASeqIO`.

```
package seqio;
import org.omg.CORBA.*;

import croverCORBA.*;
import seqio.*;
import crover.Sequence;
import netscape.WAI.Naming;

public class SeqIOServer {

   static public void main(String args[]) {

       try {
           if (args.length <1) {
            System.err.println("seqio: usage: java seqio.SeqIOServer $CROVERSERV
ER/bin");
            System.exit(20);
           }

       ORB orb =ORB.init();
       BOA boa=orb.BOA_init();
       SeqIOImpl seqio=new SeqIOImpl("seqIO");

       seqio.bin_path=args[args.length-1]; // last argument
                                           // is the path to fmtseq


       boa.obj_is_ready(seqio);
 // Register the object with a name service
//      netscape.WAI.NameUtil.registerObject(InetAddress.getLocalHost().getHostN
ame(), "/NameService/NetscapeBank", manager);

      netscape.WAI.Naming.register("http://red.incm.u-nancy.fr:81/SeqIO", seqio)
;
      System.out.println("seqio: Implementation is ready");

/*       BiologicalSequenceImpl impl=new BiologicalSequenceImpl();
     System.out.println("impl: "+impl);
       BiologicalSequence bsi= new _tie_BiologicalSequence(impl);
     System.out.println("bsi: "+bsi);

       bsi.setResidueCodes("AAASDFG");
        System.out.println("bsi:  is ready: bsi.getResidueCodes() "+bsi.getResid
ueCodes());
       String[] files={"/home/yellow/campagne/5HT/SwissProt/A1AC_RAT.sw"};

                croverCORBA.BiologicalSequence seqs[]=seqio.sequences(files);
                Sequence seq=new Sequence();
                seq.setResidueCodes(seqs[0].getResidueCodes());
                seq.setShortName(seqs[0].getShortName());
   System.out.println("seq:  seq.getResidueCodes() "+seq.getResidueCodes()
+" X-X " +seqs[0].getResidueCodes()); */
       boa.impl_is_ready();

       } catch( Exception e) {

         System.err.println(e);
       }
   }
}
```

```
package seqio;

import java.io.*;
import java.lang.*;
import crover.Sequence;
import java.util.Vector;
import croverCORBA.*;
import org.omg.CORBA.*;
/**
<p>
<ul>
<li> <b>Java Class</b> seqio._example_SeqIO
<li> <b>Source File</b> seqio/_example_SeqIO.java
<li> <b>IDL Absolute Name</b> ::seqio::SeqIO
<li> <b>Repository Identifier</b> IDL:seqio/SeqIO:1.0
</ul>
<b>IDL definition:</b>
<pre>
    interface SeqIO {
      typedef sequence&lt::croverCORBA::BiologicalSequence&gt sequence_of_crover
CORBA_BiologicalSequence;
      typedef sequence&ltstring&gt sequence_of_string;
      ::seqio::SeqIO::sequence_of_croverCORBA_BiologicalSequence sequences(
        in ::seqio::SeqIO::sequence_of_string arg0
      );
    };
</pre>
</p>
*/
public class SeqIOImpl extends seqio._sk_SeqIO {
  /** Construct a persistently named object. */
  public SeqIOImpl(java.lang.String name) {
    super(name); reset();
  }
  /** Construct a transient object. */
  public SeqIOImpl() {
    super(); reset();
  }
  /**
  <p>
  Operation: <b>::seqio::SeqIO::sequences</b>.
  <pre>
    ::seqio::SeqIO::sequence_of_croverCORBA_BiologicalSequence sequences(
      in ::seqio::SeqIO::sequence_of_string arg0
    );
  </pre>
  </p>
  */
  public croverCORBA.BiologicalSequence[] sequences(
    java.lang.String[] filenames
      ){

     croverCORBA.BiologicalSequence[] result;
   System.err.println("::croverCORBA::sequences: filenames.length " +filenames.l
ength);
     Vector set=new Vector();  // set of sequences to be read from the files
     for (int s=0; s<filenames.length; s++) {

        String filename=filenames[s];
        System.err.println("::croverCORBA::sequences("+filename+")");

        int format_number=0;
        String format="0";
        if (format_number>0 && format_number<=22) format=String.valueOf(format_n
umber);
        else {
         if (filename.endsWith(".sw"))format="sprot" ;
         if (filename.endsWith(".SW"))format="sprot" ;
         if (filename.endsWith(".msf"))format="gcg-msf" ;
         if (filename.endsWith(".MSF"))format="gcg-msf" ;
```

```
         if (filename.endsWith(".bout"))format="bout" ;
         if (filename.endsWith(".blast"))format="bout" ;
        }
        String seq_AC ;
        String ShortName;
        StringBuffer residues=null;
        StringBuffer description=null;
        croverCORBA.BiologicalSequence seq=new_seq();
        try {
           Process process=Runtime.getRuntime().exec(
                   bin_path+"/fmtseq -inf="
                                   +format
                                   +" -ffasta " + filename );
           process.waitFor();
           if (process.exitValue() !=0) System.err.println("seqio: fmtseq: error
 returned: "+ process.exitValue()
+ " the file \"" + filename + "\" was probably not found");
           else {
              DataInputStream is=new DataInputStream(process.getInputStream());

              String line;
              while ((line=is.readLine()) !=null) {

                   if (line.startsWith(">")) { // new sequence,
                                               // ID line with origin..

                       if (residues != null && seq != null)  {

                         seq.setResidueCodes(residues.toString());
                         set.addElement(seq);
                       }

// SendMsg.HelloImpl hi = new SendMsg.HelloImpl("TieExample");
                       seq = new_seq();

//                       seq=new croverCORBA.BiologicalSequence();
                       int end_of_shortname=line.indexOf('|');
                       if (end_of_shortname<0) end_of_shortname=line.indexOf(' ');
                       if (end_of_shortname<0) end_of_shortname=
                                            Math.min(line.length(),
                                                     10);
                       seq.setShortName(line.substring(1, end_of_shortname));

                       System.err.println("seqio: fmtseq " +s+": shortName"
                                          +seq.getShortName());

                       residues=new StringBuffer();
                       description=new StringBuffer();

                   } else  if (line.startsWith(";")) { // description lines

                        description.append(line.substring(1));
                   } else { //residue lines
                        residues.append(line);
                   }
              }

           }
        }
        catch (java.io.IOException e ) {
            System.err.println("SeqIOImpl: SeqIO IO problem reading "
             + filename
             + "exception details: " +e);

        }
        catch (java.lang.InterruptedException e){
            System.err.println("SeqIOImpl: SeqIO: fmtseq exec was interupted whi
le parsing "
             + filename
             + "exception details: " +e);
```

```
        }
        seq.setResidueCodes(residues.toString());
        System.err.println("::croverCORBA::sequences ready to return seq:"
                    +seq.getShortName() + " // "  +seq.getResidueCodes());
        set.addElement(seq);



      }
      result=new croverCORBA.BiologicalSequence[set.size()];
      set.copyInto(result);
      return result;

  }
  public croverCORBA.BiologicalSequence new_seq() {
                    //transient object
     return new_seq(null);
  }
  private transient ORB orb;
  private transient BOA boa;

  private void readObject(java.io.ObjectInputStream s)
                        throws java.lang.ClassNotFoundException,
                               java.io.IOException {
      s.defaultReadObject();
      reset();
  }
  void reset() {

    orb=ORB.init();
    boa=orb.BOA_init();
  }

  public croverCORBA.BiologicalSequence new_seq(String name) {
                    //transient object
//      if (boa == null) reset();
     croverCORBA.BiologicalSequence seq;
     if (name != null) seq=new _tie_BiologicalSequence(new BiologicalSequenceImp
l(),name); else
     seq=new _tie_BiologicalSequence(new BiologicalSequenceImpl());

     boa.obj_is_ready(seq);
     return seq;
  }
/**
  <p>
  Operation: <b>::seqio::SeqIO::dispose</b>.
  <pre>
    void dispose(
      in ::seqio::SeqIO::sequence_of_croverCORBA_BiologicalSequence arg0
    );
  </pre>
  </p>
  */
  public void dispose(
    croverCORBA.BiologicalSequence[] to_release
  ) {

    return; /*
     // tell the BOA we don't use these objects any longuer
      for (int i=0; i<to_release.length; i++) {
System.err.println("::croverCORBA::dispose seq, len: "+
to_release.length + " i: "+ i + " to_release[i]: " + to_release[i]);
          boa.deactivate_obj(to_release[i]);
     } */
  }

  public String bin_path="/usr/local/bin";
}
```

```
 package croverCORBA;

import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.Serializable;
import java.util.*;

import crover.NotAVisualBean;
import crover.Sequence;

import seqio.*;
import org.omg.CORBA.*;

class ASeqIOServiceException extends Exception{


};


// this non graphical bean allows you to manipulate the CROVER
// seqIO service graphically and interactively.

public class ASeqIOService extends crover.NotAVisualBean
                        implements Serializable,
                                   java.beans.PropertyChangeListener  {

    public ASeqIOService() {
        super("SeqIOService");
        reset();
    }
    private void readObject(java.io.ObjectInputStream s)
                         throws java.lang.ClassNotFoundException,
                                java.io.IOException {
      s.defaultReadObject();
      reset();
    }
    private transient ORB orb;
    private transient BOA boa;

    private String server_host;
    private Integer server_port;

    public String getServerHost() { return server_host;}


    private void reset() {
    // find the SeqIO CORBA service
    try {
       int port_number=14002;
       String args[]={"ASeqIOService -ORBagentAddr 193.50.239.27 -ORBagentPort "
 +port_number };
       java.util.Properties props = new java.util.Properties();
        props.put("ORBagentAddr", "193.50.239.27");
          props.put("ORBagentPort", String.valueOf(port_number));

       orb=ORB.init(args,props);//  ARGS
       BOA boa = orb.BOA_init();
      // bind to SeqIO service
      seqio=SeqIOHelper.bind(orb,"http://red.incm.u-nancy.fr:81/NameService/SeqI
O");
//       seqio=SeqIOHelper.bind(orb,"seqIO");
      sequences=null;
    } catch (Exception e) {

        Error er= new Error("crover: unable to bind to SeqIO service, reason: ex
ception occured: " +e);
        throw er;
    }
```

```
    if (seqio==null) {
        Error er= new Error("crover: unable to bind to SeqIO service, seqio==nbu
ll");
        throw er;
      }
    }
    String filenames[];
/*
    public void setSequenceAsString(String seq_in_string) {

     // sequence is given as a text String. For example a SwissProt file
     // is passed in the String. The String is sent to the SeqIO service,
     // written to a file, converted by the SeqIO tool, the result is
     // retrieved through the BiologicalSequence interface

     sequence=seqio.sequenceFromFileInString(seq_in_string);
     changes.firePropertyChange("sequenceVector", null,getSequence() );
    }
    private transient croverCORBA.BiologicalSequence sequence;
*/
    public void setFilenameVector(Vector names){

      if (names.isEmpty()) return;

      filenames= new String[names.size()];
      names.copyInto(filenames);
      sequences=seqio.sequences(filenames);
System.err.println("ASeqIOService: setFilenameVector:sequences has "
+sequences.length + " elements");
      changes.firePropertyChange("sequenceVector", null,getSequenceVector() );
    }
    private transient croverCORBA.BiologicalSequence sequences[];

    public Vector getSequenceVector() {
      Vector result=new Vector();
      if (sequences != null) {

        Sequence seq;
        for (int i=0;i<sequences.length;i++) {
           seq=new Sequence();
           seq.set(sequences[i]);

           result.addElement(seq);   // return CORBA stubs as Sequences
        }
        seqio.dispose(sequences); // tell to seqIO service we
                                  // don't need these objects anymore
      }
      return result;
    }
/*    public Sequence getSequence() throws ASeqIOServiceException {

      if (sequence != null) {

        Sequence seq;

           seq=new Sequence();
           seq.set(sequence);

        seqio.dispose(sequence); // tell to seqIO service we
                                 // don't need these objects anymore
        return seq;
      }
      else  throw new ASeqIOServiceException("no sequence to get");

    }*/
    private transient seqio.SeqIO seqio=null;

    public void propertyChange(java.beans.PropertyChangeEvent e) {

    }
```

```
/**
     * The specified PropertyChangeListeners <b>propertyChange</b> method will
     * be called each time the value of any bound property is changed.
     * The PropertyListener object is addded to a list of PropertyChangeListener
s
     * managed by this button, it can be removed with removePropertyChangeListen
er.
     * Note: the JavaBeans specification does not require PropertyChangeListener
s
     * to run in any particular order.
     *
     * @see #removePropertyChangeListener
     * @param l the PropertyChangeListener
     */
    public void addPropertyChangeListener(PropertyChangeListener l) {
        changes.addPropertyChangeListener(l);
    }

    /**
     * Remove this PropertyChangeListener from the buttons internal list.
     * If the PropertyChangeListener isn't on the list, silently do nothing.
     *
     * @see #addPropertyChangeListener
     * @param l the PropertyChangeListener
     */
    public void removePropertyChangeListener(PropertyChangeListener l) {
        changes.removePropertyChangeListener(l);
    }

    private PropertyChangeSupport changes = new PropertyChangeSupport(this);
    private Vector pushListeners = new Vector();


    public void setAlignmentFileName(String filename) {
       Vector vec=new Vector();
       vec.addElement(filename);
       setFilenameVector(vec);
       changes.firePropertyChange("alignment",null,getAlignment());
    }
    public crover.Alignment getAlignment() {

      crover.Alignment align=new crover.Alignment();

      if (sequences == null) return align;
        Sequence[] seqs=new Sequence[sequences.length];
        Sequence seq;
        for (int i=0;i<sequences.length; i++) {
          seq=new Sequence();
          seq.set(sequences[i]);
          seqs[i]=seq;
        }
        seqio.dispose(sequences);
        align.setSequenceSet(seqs);
        return align;
    }

}
```

Index

Bibliographie

[ABGW94] S.F. Altschul, M.S. Boguski, W. Gish, and J.C. Wootton. Issues in searching molecular sequence databases. *Nat Genet*, 6(2):119–129, February 1994. 8.2.4

[AE95] Applied Expertise (AE) and EDS. Report of the Software Reuse Best Practices Study. Technical report, Department of Defense (DOD) Software Reuse Initiative (SRI), 1995. The report was commissioned under contract DCA100-93-D-0066, URL: `http://sw-eng.falls-church.va.us/ReuseIC/lessons/benchmark/html/bench.htm\#1.1.` 5, 5.4

[AGM+90] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990. 10.1.1

[Ars98] Jacques Arsac. L'informatique et le mur du sens. dans [Sci98] (pp 213–234), 1998. 8.2.1

[Aut98] The autoclass project. URL: `http://ic-www.arc.nasa.gov/ic/projects/bayes-group/group/auto` –1998. 8.2, G

[BA96] R.J. Brachman and T. Anand. The Process of Knowledge Discovery in Databases. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 37–57, Menlo Park, California, 1996. AAAI Press / MIT Press. 8.2.3, 8.2.3, 8.2.5, 49

[BCOY93] D. Brown, J. H. R. Clarcke, M. Okuda, and T. Yamazaki. A Domain Decomposition Parallelization Strategy For Molecular-Dynamics Simulations on Distributed Memory Machines. *Computer Physics Communications*, 74:67–80, 1993. 15

[Ben73] J.-P. et collaborateurs Benzécri. *L'analyse de données.* Dunod, 1973. 8.1

[BMM97] D. Brown, H. Minoux, and B. Maigret. A domain decomposition parallel processing algorithm for molecular-dynamics simulations of systems of arbitrary connectivity. *Computer Physics Communications*, 103:170–186, 1997. 15

[Boo93] Grady Booch. *Object-oriented Analysis and Design with Applications.* Benjamin/Cummings Pub. Co. - Redwood City, Calif., Benjamin Cummings Redwood City, 2nd edition, 1993. ISBN 0-8053-5340-2. 5

[Bou96] N. Boudjlida. Environnement logiciels intégrés et bases de données. In A. Ferchichi and J.-P. Giraudin, editors, *FIIA 96*, pages 131–164. ESIG, 1996. 18

[Cam97] F. Campagne. Informal Collaboration for Computational Chemistry Software Design and Reuse: Mixed language programming technics. URL: `http://www.lctn.u-nancy.fr/design/mixedlanguages.html`, 1997. 23

[Cas97] Castanet white paper. Marimba, Inc.; 440 Clyde Avenue, Mountain View, CA 94043-2232 www.marimba.com, URL: `http://www.marimba.com/products/`, December 1997. see also URL: `http://www.marimba.com/products/`. 8.5.1

[CBM97] F. Campagne, J.-M. Bernassau, and B. Maigret. The Viseur program. Laboratoire de Chimie théorique de Nancy, WWW: `http://www.lctn.u-nancy.fr/viseur/viseur.html`, 1995–1997. Viseur project home page. 2.3.1

[CGI98] Introduction to the CGI-BIN protocol. *URL: `http://hoohoo.ncsa.uiuc.edu/cgi`*, 1998. 2.3.2, 46

[CGS89] G. Chang, W. E. Guida, and W. C. Still. An Internal Coordinate Monte Carlo Method for Searching Conformational Space. *J. Am. Chem. Soc.*, 111(12):4379–4386, 1989. 3.2.2.0

[CH88] M. Crippen, G. and F. Havel, T. *Distance Geometry and Molecular Conformation.* Research Studies Press, Ltd., Taunton, Somerset, England, 1988. ISBN: 0 86380 073 4. 3.2.2.0

[Chr98] Tom Christiansen. The Perl Language Home Page. *URL: http://language.perl.com/index.html*, 1998. 2.3.2

[CJR+98] F. Campagne, R. Jestin, J.L Reversat, J.M. Bernassau, and B. Maigret. Visualisation and Integration of G Protein-Coupled Receptor Related Information Help the Modelling: Description and Applications of the Viseur program. *J. Of Computer Aided Molecular Design*, 1998. in press. 2.3.1

[CKS+88] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. AutoClass: A Bayesian Classification System. In Ann Arbor, editor, *Proceedings of the Fifth International Conference on Machine Learning*, pages 54–64, San Francisco, June 1988. Morgan Kaufmann Publisher. 8.2.5.0

[CM97] F Campagne and B. Maigret. Protein Sequence in HTML: colored, possibly hyperlinked, compact representations. In *2nd Molecular and Modelling Conference (MGM EC-2)*, October 1997. `http://www.vei.co.uk/mgmec2`. 2.3.2

[CM98] F Campagne and B. Maigret. Multiple Sequence Alignment in HTML: colored, possibly hyperlinked, compact representations. *J. Of Molec. Graph. & Mod.*, 1998. in press. 2.3.1, 10.3.2

[Cor97] Netscape Communications Corporation. Writing Web Applications with WAI, Netscape Enterprise Server/FastTrack Server. Technical report, Netscape Communications Corporation, April 1997. indicative URL: `http://developer.netscape.com:80/docs/manuals/enterprise/wai/contents.htm` 11.4.1

[Cou87] J. Coutaz. PAC : an implementation Model for Dialog Design. In *Interact'87*, pages 431–436, September 1987. 10.1

[Cou91] J. Coutaz. Interfaces Homme-Machine: un regard critique. *Techniques et Sciences Informatiques*, 10(1):53–64, 1991. 8.5.2.0

[CR97] M. Carswell, C. and C. Ramzy. Graphing small data sets: should we bother? *Behaviour and Information Technology*, 16(2):61–71, 1997. 8.5.2.0

[CS95] Peter Cheeseman and John Stutz. Bayesian classification (AUTOCLASS): Theory and results, 1995. URL: `http://ic-www.arc.nasa.gov/ic/projects/bayes-group/group/images/kdd-95.ps` referenced by [Aut98]. 8.2.5.0, 37

[CWA97] W. Clegg, Chris, E. Waterson, Patrick, and M. Axtell, Carolyn. Software development: some critical views. *Behaviour and Information Technology*, 16(6):359–362, 1997. 8.5, 10.2.3

[DBB+87] H.G. Dohlman, M. Bouvier, J.L. Benovic, M.G. Caron, and R.J. Lefkowitz. The multiple membrane spanning topography of the beta 2-adrenergic receptor. localization of the sites of binding, glycosylation, and regulatory phosphorylation by limited proteolysis. *J. Biol. Chem.*, 262(29):14282–14288, 1987. 1.2

[Deu98] Erik Deumens. Standardization of computational chemistry software. *URL: http://www.qtp.ufl.edu/softstand.html*, 1995-1998. This discussion list is an extension of a discussion session started at the 35th Sanibel Symposium held in St. Augustine, Florida in 1995. 15

[Dis96] Discussion panel. 24th Aaron Katzir - Katchalsky Conference 25th Anniversary of the Protein Data Bank, 10th Anniversary of the Swiss-Prot Database. In *Special Session :The Future of Bioinformatics*, JERUSALEM (Israel), November 1996. 0.2

[DJ92] K.A. De Jong. Genetic Algorithms are NOT function optimizers. In D. Whitley, editor, *Foundations of Genetic Algorithms: Proceedings 24–29 July 1992*, Vail, CO, 1992. 3.2.2.0

[DLK87] D. P. Dolata, A. R. Leach, and Prout K. WIZARD: AI in conformational analysis. *J. Of Comp.-Aid. Molec. Des.*, 1:73–85, 1987. 3.2.2.0

[DT93] P. Darlu and P. Tassy. *La reconstruction phylogénétique, Concepts et méthodes.* Masson, 1993. épuisé. 1.2

[EA93] T. Etzold and P. Argos. SRS an indexing and retrieval tool for flat file data libraries. *Comput. Appl. Biosci.*, 9:49–57, 1993. 9.0.2.0

[GHJV96] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns — Catalogue de modèles de conception réutilisables.* International Thompson Publishing France, 1996. ISBN: 2-84180-054-7. 21, 21, 5.4, 27, 6.7, 7.2.2, 10.1

[Gol84] A. Goldberg. *Smalltalk-80 : The interactive Programming Environment.* Addison Wesley, 1984. see p 158 for MVC. 10.1, 10.3

[Gro96] Object Management Group. CORBAmed RFI. OMG Document # corbamed/96-01-01, January 1996. URL: ftp://ftp.omg.org/pub/docs/corbamed/96-01-01.htm. 10.2.1

[Hat96] R. J. Hathaway. Editor's corner, part 2: Meta systems and reflection. *Object Currents Hypertext Journal*, 1(4), April 1996. 655 West Irving Park Road, Suite 5417; Chicago, Illinois 60613-313775 USA, URL: `http://www.sigs.com/publications/docs/oc/9604/oc9604.d.edit.html`. 10.1.1

[HDHL96] W. Humphrey, A. Dalke, K. Hamer, and J.and J. Phillips Leech. *VMD Programmer's Guide – 1.1 Draft.* Theoretical Biophysics Group, University of Illinois and Beckman Institute, 405 N. Mathews, Urbana, IL 61801, May 1996. URL: `http://www.ks.uiuc.edu/Research/vmd/pg/pg.html`. 6.3.2.0

[HDHL98] W. Humphrey, A. Dalke, K. Hamer, and J.and J. Phillips Leech. VMD — Visual Molecular Dynamics. URL: `http://www.ks.uiuc.edu/Research/vmd/`, 1994–1998. 6.3.2.0

[Hol75] J. Holland. *Adaptation in Natural and Artificial Systems.* University of michigan Press, Ann Arbor, MI, 1975. 3.2.2.0

[HSV96] R.W.W. Hooft, C. Sander, and G Vriend. Verification of protein structures: Side-chain planarity. *J. Appl. Cryst.*, 29:714–716, 1996. 8.2.4

[HVSA96] R.W.W. Hooft, G. Vriend, C. Sander, and E.E. Abola. Errors in protein structures. *Nature*, 381:272–272, 1996. 8.2.4

[HWB+98] F. Horn, J. Weare, M.W. Beukers, S. Hörsch, A. Bairoch, W. Chen, Ø. Edvardsen, F. Campagne, and G. Vriend. GPCRDB: an information system for G protein-coupled receptors. *Nucleic Acids Res.*, 1:275–279, January 1998. see also URL:`http://www.gpcr.org/7tm` 2.4.1

[Ing78] D.H.H. Ingalls. The SMALLTALK-76 Programming System design and Implementation. In *Proceedings of the 5th POPL*, pages 9–17, Tucson, Arizona, 1978. 10.1.1

[Ing89] Lester Ingber. Very Fast Simulated Re-Annealing. *Mathl. Comput. Modelling*, 12:967–973, 1989. 3.2.2.0

[Jav97] Javabeans. Specification 1.01, Sun microsystems, 1997. 10.1

[JJMP93] J. R. Judson, E. P. Jaeger, Treasurywala A. M., and M. L. Peterson. Conformational Searching Methods for Small Molecules, II Genetic Algorithm approach. *J. Of Comput. Chem.*, 14(11):1407–1414, November 1993. 3.2.2.0

[KDE96] K. Kristiansen, S. G. Dahl, and 0. Edvardsen. A Database of Mutants and effects of Site-Directed Mutagenesis Experiments on G Protein-Coupled Receptors. Protein Structure. *Function, and Genetics*, 26:81–94, 1996. 2.2.3

[KGV83] S. Kirkpatrick, Jr. Gelatt, C.D., and J.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. 3.2.2.0

[Kni96] James Knight. SEQIO: A C Package for Reading and Writing Sequences. Dept. of Computer Science; Univ. of California, Davis; Davis, CA 95616, 1996. distributed by the author (knight@cs.ucdavis.edu), see indicative URL: `http://wwwcsif.cs.ucdavis.edu/~knight/`. 11.4.1.0

[Kob92] B. Kobilka. Adrenergic receptors as models for G protein-coupled receptors. *Annu Rev Neurosci*, 15:87–114, 1992. 1.2

[Kod97] Yves Kodratoff. L'extraction de connaissances à partir des données : un nouveau sujet pour la recherche scientifique. In *JICAA*, pages 539–566, 1997. 8.2.2

[LS88] M. Lipton and W. C Still. The Multiple Minimum Problem in Molecular Modeling. Searching Internal Coordinate Conformational Space. *J. Of Comput. Chem.*, 9(4):343–355, 1988. 3.2.2.0

[Ltd95] The Numerical Algorithms Group Ltd. IRIS Explorer User's Guide. URL: `http://www.nag.co.uk/visual/IE/iecbb/DOC/UG/CONTENTS.html`, 1995. 8.3.1

[Mey97] Bertrand Meyer. *Object-Oriented Software Construction.* Prentice Hall, 1997. ISBN: 0-1362-9155-4. 15, 5

[MMAD95] C. Medigue, I. Moszer, Viari A., and A. Danchin. Analysis of a Bacillus subtilis genome fragment using a co-operative computer system prototype. *Gene*, 165(1):GC37–GC51, November 1995. 8.4, 8.5.2.0

[MN87] P. Maes and D. Nardi, editors. *Meta-Level Architectures and Reflection.* North-Holland, Amsterdam, 1987. 10.1.1

[MNC+89] G. Masini, A. Napoli, D. Colnet, D. Léonard, and K. Tombre. *Les langages à objets, langages de classes, langages de frames, langages d'acteur.* InterEditions, 1989. ISBN 2 7296 0275 5. 8.2.1

[MO92] James Martin and James J. Odell. *Object-Oriented Analysis and Design.* Prentice Hall, 1992. ISBN 0-13-630245-9. 5

[Mos96] Ivan Moszer. *Représentation et Analyse des Génomes : Application au Projet de Séquençage du Genome de Bacillus Subtilis.* Thèse d'Université, Université Paris VI, December 1996. 37

[MpRR+53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953. 3.2.2.0

[MS89] David R. Musser and Alexander A. Stepanov. Generic Programming. In *Lecture Notes in Computer Science*, pages 13–25. Springer-Verlag, 1989. 6.2.2

[MS94] David R. Musser and Alexander A. Stepanov. Algorithm-oriented Generic Libraries. *Software–Practice & Experience*, 24(7):623–642, July 1994. 6.2.2

[MWCR93] C. Médigue, J. Willamowski, F. Chevenet, and F. Rechennman. Representation tasks for problem solving in molecular biology. In *IJCAI'93 Workshop Artificial intelligence and genome*, pages 67–76, 1993. 8.4

[Num97] Numerous authors. HTTP Specifications and Drafts. W3C, URL: `http://www.w3.org/Protocols/Specs.html`, November 1997. 10.2

[Obj96] Object Management Group. CORBA 2.0/IIOP specification. Technical Report PTC/96-03-04, Framingham Corporate Center, Framingham (MA), USA, 1996. 22

[Obj98] Object Oriented Concepts Inc. Orbacus home page. URL: `http://www.ooc.com/ob/`, 1998. 45

[OH97] R. Orfali and D. Harkey. *Client/Server Programming with JAVA and CORBA.* Wiley computer publishing. John Wiley & Sons, Inc., 1997. 10.2, 10.2.3, 48

[OHE97] R. Orfali, D. Harkey, and J. Edwards. *Instant CORBA.* Wiley computer publishing. John Wiley & Sons, Inc., 1997. 10.2

[Org9X] International Standards Organization. ISO/IEC Standard 14882:199X, 199X. Once adopted, the official C++ Standard around the world. Currently still in draft form. 6.2.2

[Par90] *ObjectWorks*
Smalltalk Release 4 Users Guide. ParcPlace Systems, Mountain View, CA, 1990. 21

[Pea78] W. R. Pearson. Rapid and Sensitive Sequence Comparison with FASTP and FASTA. *Methods in Enzimology*, 183:63–98, 1978. 10.1.1

[PL78] W. R. Pearson and D. J. Lipman. Improved Tools for Biological Sequence Aalysis. *PNAS*, 85:2444–2448, 1978. 10.1.1

[Pla96] P.J. Plauger. The Dinkum® C/C++ Library Reference. Copyright © 1992-1996 Dinkumware, Ltd. 398 Main Street Concord MA 01742, 1992–1996. URL: `http://www.dinkumware.com/refxcpp.html`. 6.2.2

[Rei95] Reinhart A.F. Reithmeier. Characterisation and modeling of membrane proteins using sequence analysis. *Current Opinion in Structural Biology*, 5:491–500, 1995. 1.2

[RU91] F. Rechenmann and T. Uvietta. Shirka: An object-centered knowledge base management system. In *Artificial Intelligence in Numerical and Symbolic Calculations*, Lyon, 1991. Aléas Publishers. 8.4

[Sau87] M. Saunders. Stochastic Exploration of Molecular Mechanics Energy Surfaces. Hunting for the Global Minimum. *J. Am. Chem. Soc.*, 109(10):3150–3152, 1987. 3.2.2.0

[Say] Roger Sayle. *RasMol Version 2.6-beta-2 Manual – Molecular Visualisation Program.* Glaxo Wellcome Research and Development Stevenage, Hertfordshire, U.K. URL: `http://info.bio.cmu.edu/Courses/BiochemMols/RasFrames/RASMAIN.HTM`. 5.2.3.0

[Sci98] Bibliothèque ALBIN MICHEL Sciences, editor. *Dictionnaire de l'ignorance.* Édition Albin Michel S.A., , 22 R. Huyghens, 750014 Paris, 1998. 8.2.1, G

[SEOJ96] G.D. Schuler, J.A. Epstein, H. Ohkawa, and Kans J.A. Entrez: molecular biology database and retrieval system. *Methods Enzymol.*, 266:141–162, 1996. 9.0.2.0

[SF98] W. Smith and T.R. Forester. The DL_POLY Molecular Dynamics Simulation Package. URL: `http://www.dl.ac.uk/TCSC/Software/DL\protect\T1\textunderscorePOLY/main.html`, July 1998. contact: w.smith@dl.ac.uk. 15

[SGI97] SGI. A vrml ressource page. URL: `http://vrml.sgi.com/intro.html`, 1997. 2.4.3

[SK93] A.P. Sheth and V. Kashyap. So Far (Schematically) Yet So Near (Semantically). In E.J. Neuhold D.K. Hsiao and R Sack-Davis, editors, *Interoperable Database Systems*, North Holland, 1993. Elsevier Science. 18

[Sne93] P. H. A. Sneath. Evidence from Aeromonas for Genetic Crossing-Over in Ribosomal Sequences. *Int. J. Of Systematic Bacteriology*, 43(3):626–629, July 1993. 10

[Str91] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley, 1991. ISBN: 0-201-53992-6. 5.2.3.0, 5.3.3, D, D

[Str93] Paul S. Strass. IRIS Inventor, a 3D graphics toolkit. In *Object-Oriented Programming Systems, Languages, and Applications Conference Proceedings*, pages 192–200, Washington, D.C., September 1993. ACM Press. 21

[Sun97a] Sun microsystems Inc. Java technology home page. URL: `http://java.sun.com/`, 1997. 8.5.1

[Sun97b] Sun Microsystems, Inc. JavaStudio. URL: `http://java.sun.com/products/index.html`, 1997. 11.1

[SWBB97] David C. Spellmeyer, Alex K. Wong, Michael J. Bower, and Jeffrey M. Blaney. Conformational analysis using distance geometry methods. *J. Of Molec. Graph. & Mod.*, 15(1):18–36, February 1997. 3.2.2.0

[SWI+98] A. Simper, A. Willets, A. Ioannon, S. Spencer, C. Skylaris, and L. Gagliardi. A Relativistic Density Functional Quantum Chemistry Code for Large Systems. URL: `http://magic.ch.cam.ac.uk/`, 1998. 15

[SWSD94] A. V. Shah, W. P. Walters, R. Shah, and D. P. Dolata. Babel - A Molecular Information Interchange Hub. In Lysakowski and C. E. Gragg, editors, *Computerized Chemical Data Standards: Databases, Data Interchange, and Information Systems*, Philadelphia, 1994. American Society for Testing and Materials. ASTM STP 1214, R. 18

[Sys] Advanced Visual Systems. AVS/Express®. URL: `http://www.avs.com/`. See Products. 8.3.1

[Szy98] Clemens Szyperski. *Component Software Beyond Object-Oriented Programming.* Addison Wesley, 1998. ISBN: 0-201-17888-5. 10.1

[Wan97] Cheuk-San Wang. Efficient Algorithm for Conformational Search of Macrocyclic molecules. *J. Of Comput. Chem.*, 18(2):277–289, 1997. 3.2.2.0

[Wel90] J.A. Wells. Additivity of mutational effects in proteins. *Biochemistry*, 29:8509–8517, 1990. 1.2

[Wil92] R. Wille. *Concept lattices and conceptual knowledge systems*, pages 493–515. Pergamon Press, 1992. Lehman, F. ed. 8.4

[Wil94] J. Willamoski. *Modélisation de tâches pour la résolution de problèmes en coopération système utilisateur.* Thèse d'Informatique, Université Joseph Fourier, Grenoble I, 1994. 8.4

[WLMB89] H. Wang, L. Lipfert, C.C. Malbon, and S. Bahouth. Site-directed anti-peptide antibodies define the topography of the beta-adrenergic receptor. *J. Biol. Chem.*, 264(24):14424–14431, 1989. 1.2

[WP96] Zhiqinag Wang and Ruth Pachter. Prediction of Peptide Conformation: The Adapatative Simulated Annealing Approach. *J. Of Comput. Chem.*, 18(3):323–329, 1996. 3.2.2.0, 4

[YC79] Edward Yourdon and Larry L. Constantine. *Structures Design, Fundamentals of a Discipline of Computer Program and Systems Design.* Computing. Yourdon Press, Prentice Hall Building, Englewood Cliff, NJ 07632, 1979. 15

Monsieur **CAMPAGNE Fabien**

DOCTORAT de l'UNIVERSITE HENRI POINCARE, NANCY-I

en CHIMIE INFORMATIQUE & THEORIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER

Nancy, le 8 octobre 1998 n° 79

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ, NANCY 1 LE PRESIDENT

J.P. FINANCE

Résumé

Ce manuscrit présente, *(i)* le travail réalisé dans le cadre du projet Viseur, un ensemble d'outils et de ressources bioinformatiques dédié à l'aide à la préparation de la modélisation des récepteurs couplés aux protéines G (GPCRs); *(ii)* un exemple de réutilisation des logiciels du patrimoine de la Chimie informatique pour la réalisation d'un programme de recherche conformationnelle par algorithme de recuit simulé adaptatif (utilisation de techniques de programmation orientée-objet, de programmation générique et de programmation multi-langage); *(iii)* une réflexion de fond, illustrée par les exemples *(i)* et *(ii)*, sur la conception et la réalisation d'outils pour l'aide à l'analyse de données biologiques adaptés aux besoins des utilisateurs biologistes, biochimistes et chimistes, et aux évolutions technologiques récentes (développement des réseaux internet et intranets, technologie des composants logiciels, CORBA, etc.). Le point *(iii)* est appuyé par la réalisation d'un prototype logiciel dont l'architecture est décrite.

Mots-clés: Récepteurs Couplés aux Protéines G (GPCR), aide à la modélisation, bioinformatique, recherche conformationnelle, optimisation par recuit simulé adaptatif, Aide à l'analyse de données biologiques distribuées et hétérogènes, internet, intranet, CORBA et les bases de données biologiques

Abstract

This manuscript describes, *(i)* the work carried on in the context of the Viseur project, a set of tool and bioinformatic ressources dedicated to helping G Protein Coupled Receptors modeling (GPCRs); *(ii)* an illustration of the reuse of computational chemistry legacy software for building a conformational search program with the adaptative simulated annealing optimization algorithm (use of object-oriented technics, generic programming, and mixed-language programming); *(iii)* An in-depth reflexion about the design and realization of tools dedicated to assist the biologist, biochemist, and modeling chemist during his research work that is built upon the real examples given in *(i)* and *(ii)*, and the acknowledgment of recent technological advances (internet and intranet networks, software component technology, CORBA, etc). Point *(iii)* is concluded by the description of the architecture of a software prototype, realized according to this reflexion.

Keywords: G Protein Coupled Receptors, modeling helping, bioinformatic, conformational search, adaptative simulated annealing optimization, help to the analysis of distributed and heterogeneous biological data, internet, intranet , CORBA and the biological databases