CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS
CENTRE RÉGIONAL ASSOCIÉ DE GRENOBLE

MÉMOIRE
présenté en vue d'obtenir
LE DIPLÔME D'INGÉNIEUR C.N.A.M.

en
Informatique

par
Alain GARREAU

Étude et réalisation d'une interface cartographique d'aide à l'analyse des séquences génomiques

Soutenu le 16 juin 1995

JURY

Président	:	Mme	Véronique DONZEAU-GOUGE
Membres	:	Mme	Maria-Virginia APONTE
		M.	Gilles BISSON
		M.	Jacques COURTIN
		M.	Christian GAUTIER
		M.	François RECHENMANN

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS
CENTRE RÉGIONAL ASSOCIÉ DE GRENOBLE

MÉMOIRE
présenté en vue d'obtenir
LE DIPLÔME D'INGÉNIEUR C.N.A.M.

en
Informatique

par
Alain GARREAU

Étude et réalisation d'une interface cartographique d'aide à l'analyse des séquences génomiques

Les travaux relatifs au présent mémoire ont été effectués au sein du projet SHERPA,
commun à l'INRIA Rhône-Alpes et au laboratoire LIFIA/IMAG
du 1 mars 1994 au 28 février 1995
sous la direction de
François Rechenmann, directeur de recherche à l'INRIA

Remerciements

Mes remerciements seront tout d'abord adressés au jury :

Madame Véronique DONZEAU-GOUGE, Professeur au Conservatoire National des Arts et Métiers de Paris, me fait l'honneur de présider le jury. Je l'en remercie.

Madame Maria-Virginia APONTE, Maître de conférences au Conservatoire National des Arts et Métiers de Paris, a accepté de participer au jury. Je l'en remercie.

Monsieur Gilles BISSON, Chargé de recherches au CNRS et membre du projet SHERPA à L'INRIA Rhône-Alpes, a bien voulu s'intéresser à mon travail tout au long de cette année, et me faire part de ses remarques et attentes. Je l'en remercie.

Monsieur Jacques COURTIN, Professeur à l'Université Pierre Mendès-France et responsable du cycle Ingénieur CNAM à Grenoble, m'a suivi au cours de mes études. Je l'en remercie.

Monsieur Christian GAUTIER, Professeur à l'Université Claude Bernard à Lyon a accepté de participer au jury. Je l'en remercie.

Monsieur François RECHENMANN, Directeur de recherches à l'INRIA Rhône-Alpes et responsable du projet SHERPA, m'a accueilli dans son équipe et donné l'occasion de constater combien le vivant est une source d'inspiration importante pour le développement des sciences. Je l'en remercie.

Aux membres du projet SHERPA, j'adresse mes remerciements et plus particulièrement :

À Olivier SCHMELTZER pour ses commentaires avisés concernant l'interface cartographique.

À Jutta WILLAMOWSKI pour toutes les discussions que nous avons pu avoir.

À Patrice UVIETTA et Jean-Yves MORNEAU pour la lecture attentive du manuscrit original de ce mémoire.

À Pierre FONTANILLE pour toutes ses indications sur l'environnement informatique.

Aux membres du laboratoire de Régulation de l'Expression Génétique (REG) de l'Institut Pasteur, j'adresse mes remerciements et plus particulièrement à Claudine MEDIGUE pour s'être intéressée à mon travail et m'avoir permis de l'illustrer de manière concrète sur des données réelles.

Aux membres du laboratoire de biométrie et de génétique des populations de l'Université Claude Bernard, j'adresse mes remerciements et plus particulièrement à Thierry VERMAT pour son engagement et sa disponibilité dans le cadre du projet GREG.

Et si ce mémoire est un point d'ancrage important dans ma vie, il le doit beaucoup à la tendresse de marie, romain et julia.

Table des matières

Introduction..... 1

Chapitre 1 - Quelle interface cartographique..... 5

1.1. Données biologiques.....5
1.2. Déroulement d'un projet de séquençage.....7
1.3. Les cartes génomiques.....9
1.3.1. Les cartes cytogénétiques.....9
1.3.2. Les cartes génétiques..... 10
1.3.3. Les cartes physiques..... 10
1.3.4. Conclusion..... 10
1.4. Les logiciels de cartographie..... 11
1.4.1. ACeDB..... 11
1.4.2. SIGMA..... 13
1.4.3. HoverMaps..... 14
1.4.4. Conclusion..... 16
1.5. Extension de la réflexion..... 18
1.5.1. Abstraction..... 18
1.5.2. Les critères d'acceptation d'une interface..... 19

Chapitre 2 - Définition de l'interface.....23

2.1. Environnement informatique..... 23
2.1.1. Le langage à objets C++..... 25
2.1.2. ILOG Views : une bibliothèque de classes graphiques en C++. 26
2.1.3. Conception par objets..... 29
2.2. Les concepts élémentaires..... 30
2.2.1. Le plan de travail..... 31
2.2.2. La carte..... 32
2.2.3. Les objets..... 34
2.2.4. Les couches et les plans..... 37
2.2.5. Liens entre cartes..... 38
2.2.5.1. Lien global..... 40
2.2.5.2. Liens entre objets..... 40
2.3. Modification d'objets..... 42

Chapitre 3 - APIC : Aide à la Présentation des Interfaces Cartographiques.....43

3.1. Une bibliothèque d'objets..... 43
3.2. Architecture..... 44
3.3. Évaluation de l'interface..... 47
3.4. Le plan de travail (MULTIMAP)..... 48
3.5. Organisation des cartes (ARRANGE)..... 52
3.6. Description d'une carte (MAP)..... 53
3.6.1. Visualisation d'une carte..... 53
3.6.2. Contrôle d'une carte..... 55
3.7. Gestion de l'affichage (DISPLAY)..... 57

3.8. Disposition des objets sur la carte (LAYOUT) 59
3.9. Paramétrisation de l'affichage (SETTING) 61
3.10. Synchronisation des cartes (HOOK)..... 64
3.11. Positionnement sur un objet (GOTO)..... 65
3.12. Recherche d'un objet (FIND)..... 66
3.13. Gestion des ressources 68

Conclusion 71

Bibliographie 75

Annexes 77

Annexe 1 : Description d'une carte dans le format TINA-1..... 77
Annexe 2 : Description des unités dans le format TINA-1..... 81
Annexe 3 : Description des ressources dans le format TINA-1 83
Annexe 4 : Procédure de Positionnement des Objets 85
Annexe 5 : Exemple de classe et d'utilisation de la bibliothèque de classes d'objets graphiques ILOG Views..... 87
Annexe 6 : La bibliothèque de classes d'objets pour la construction d'interfaces cartographiques APIC. 91

Introduction

Les biologistes s'intéressent tout particulièrement à l'ADN (Acide DésoxyriboNucléique) contenu dans les cellules des organismes vivants. La raison en est simple : cette molécule est composée, entre autres, de bases azotées dont la lecture se traduit par la fabrication des protéines qui sont les principaux constituants structuraux des cellules et les catalyseurs organiques nécessaires aux réactions chimiques de l'organisme.

La totalité du matériel génétique d'un organisme vivant peut être ainsi représentée par une très longue série de lettres A, C, G, T qui sont les initiales des quatres bases Adénine, Cytosine, Guanine et Thymine.

Le problème pour les biologistes, c'est que cette série est vraiment très longue : par exemple l'organisme humain possède un génome qui compte quelques milliards de ces bases! Comment dresser la carte précise de cette succession de molécules ? Comment identifier celles qui jouent un rôle ? Et surtout comment identifier ce rôle ? C'est à cette tâche titanesque que depuis plusieurs années se sont attelés de par le monde un nombre important de biologistes moléculaires.

De nombreux projets de séquençage sont donc en cours actuellement, sur des organismes différents : bactéries, levures ou animaux supérieurs, pour obtenir la séquence complète (c'est-à-dire la suite des lettres A, C, G et T) contenue dans une molécule d'ADN. La séquence obtenue est alors analysée pour localiser des "objets biologiques" et déterminer leur fonction biologique dans l'organisme. Le but final est d'obtenir le génome entier d'un organisme, c'est-à-dire l'ensemble des gènes (objets biologiques particuliers = tronçons d'ADN) dans lesquels les bases déterminent la structure et la fonction d'une protéine.

Afin d'aider le biologiste dans sa tâche, des efforts importants ont été récemment réalisés dans le développement de systèmes informatiques adaptés à la gestion de données dans le cadre des projets de séquençage de génomes entiers. Mais ces systèmes, souvent conçus pour répondre à un besoin spécifique, éprouvent des difficultés à évoluer.

Cette constatation a débouché en 1993 sur le lancement d'un projet dans le cadre du GREG Groupement de Recherches et d'Études sur les Génomes [Greg93]. Ce projet vise à spécifier et développer un système coopératif d'aide à l'analyse de séquences génomiques. Ce système devra être capable non seulement de gérer les données et les résultats d'analyse mais aussi d'aider l'utilisateur :

- En choisissant parmi plusieurs les méthodes adaptées pour une tâche donnée (recherche de régions déjà connues, de parties codantes, etc.) et en enchaînant plus ou moins automatiquement ces méthodes pour résoudre une tâche globale d'analyse.
- En fournissant les moyens de visualiser et de confronter plusieurs résultats.

Il devra pouvoir supporter des fragments de connaissance soumis par différents laboratoires concernés par un projet de séquençage de génome dans le but de converger vers une connaissance commune et consensuelle. Le système final sera distribué dans les laboratoires de recherche impliqués dans l'analyse de génomes et devra être assez générique pour être facilement adapté à différents projets de séquençage.

Introduction

Le projet dont il est question est planifié sur une période de deux ans (1994 et 1995) et concerne quatre laboratoires dont les compétences sont complémentaires:

L'Institut National de Recherche en Informatique et Automatique (INRIA), à Grenoble	:	le projet SHERPA, spécialisé dans la construction de modèles de connaissances à objets, et qui a déjà développé un premier prototype d'un environnement coopératif de résolution de problèmes (SCARP).
La société ILOG Software à Gentilly	:	qui construit, implémente et commercialise du logiciel dans les domaines de la représentation de connaissances et des générateurs d'interfaces graphiques; elle doit, en relation avec l'équipe SHERPA, assurer les extensions futures et la maintenance des outils sur lesquels reposera le futur système.
L'Université Claude Bernard à Lyon	:	le laboratoire de biométrie et de génétique des populations pour son expérience dans l'analyse de séquences.
L'Institut Pasteur à Paris	:	qui produit de longues séquences d'ADN dans le cadre du projet de séquençage du génome de la bactérie *Bacillus Subtilis*. Ces données seront utilisées pour l'expérimentation du futur système.

Au sein de ce projet, et du projet SHERPA dirigé par F. Rechenmann, mon travail a consisté à étudier et à réaliser l'interface qui doit permettre de visualiser et de confronter les résultats issus des méthodes d'analyse de séquences.

Les organismes étudiés sont nombreux et les résultats peuvent prendre des formes différentes. Il s'agit généralement de "cartes" qui donnent la position d'objets au sein d'un génome selon une unité qui peut refléter une réalité physique (ex. : la position dans la séquence, en nombre de bases) ou une constatation génétique (ex. : le pourcentage de recombinaison des gènes d'une génération à la suivante). Mais il peut également s'agir de courbes utilisées pour représenter par exemple le résultat d'une méthode de prédiction de parties codantes. Pour ces raisons, l'étude et la réalisation d'une interface cartographique d'aide à l'analyse des séquences génomiques a rejoint **l'étude et la réalisation d'une bibliothèque d'objets adaptés à la construction d'interfaces cartographiques et suffisamment génériques pour couvrir différents types de représentation et permettre la confrontation de résultats d'origines différentes.**

Ce document comprend trois parties.

La première partie rappelle quelques notions sur l'étude des génomes. Elle a pour but de donner au lecteur une vue générale des données qui intéressent plus particulièrement les biologistes et des méthodologies employées pour les obtenir et les présenter (projet de séquençage et cartographie). A partir de la comparaison de différents logiciels de cartographie, cette partie identifie les objectifs de l'étude et fixe les contraintes à respecter. Devant la diversité des représentations, elle conclut sur la nécessité de s'extraire de la vision biologique du problème à résoudre pour n'en garder qu'une vision purement graphique et sur le besoin de concevoir une bibliothèque de classes d'objets de haut niveau pour créer des interfaces autonomes ou connectées à des projets.

La deuxième partie traite de la définition de l'interface. Elle énonce les concepts qui ont été retenus pour répondre aux principaux besoins exprimés par les biologistes. Cette partie expose les problèmes rencontrés et les solutions retenues.

La troisième partie aborde la réalisation.

La conclusion permet de faire le point sur le travail réalisé.

Chapitre 1

Quelle interface cartographique ?

Ce chapitre a pour but de rappeler quelques notions sur l'étude des génomes. Il expose les principaux *objets biologiques* étudiés et décrit les méthodologies généralement employées dans l'étude des génomes (projet de séquençage, cartographie).

1.1. Données biologiques

Eucaryote ou procaryote

On distingue deux types d'organismes vivants : les organismes eucaryotes (levures, animaux supérieurs) et les organismes procaryotes (bactéries).

Un organisme eucaryote est un organisme où chaque cellule possède un noyau qui est composé, entre autres, de plusieurs molécules d'ADN visibles au moment de la division cellulaire sous la forme de chromosomes (ex : chez l'organisme humain, on trouve 23 paires de chromosomes).

Un organisme procaryote est un organisme unicellulaire, sans noyau, qui possède une seule molécule d'ADN (encore appelée chromosome bactérien). Il est de ce fait plus facile à étudier.

L'ADN

L'ADN (Acide DésoxyriboNucléique) est constitué de deux séquences d'unités fondamentales : les nucléotides, eux mêmes composés de trois micromolécules : un sucre (désoxyribose), un phosphate, et une *base* azotée.

Les bases de l'ADN sont de quatre types : (A) Adénine, (C) Cytosine, (G) Guanine et (T) Thymine. Elles s'apparient sur les deux séquences (souvent appelées brins) selon des règles de complémentarité : A avec T, et G avec C. On parle alors de paires de bases.

Les deux brins, que l'on nomme direct et complémentaire, sont orientés. On précise le sens de leur lecture par la notation : 5'->3' pour le brin direct et 3'->5' pour le brin complémentaire.

Par exemple :

5'----ATTACCGTT----3' (brin direct)
3'----TAATGGCAA----5' (brin complémentaire)

Figure 1: Cellule eucaryote. Les bases contenues dans les molécules d'ADN, visibles au moment de la division cellulaire sous la forme de chromosomes, sont de quatre types (A) Adénine, (C) Cytosine, (G) Guanine et (T) Thymine (d'aprés [Schmeltzer95]).

Les biologistes s'intéressent tout particulièrement aux molécules d'ADN car elles contiennent toute l'information nécessaire au fonctionnement et à la reproduction de l'organisme [Beilvert93] :

- **La lecture des bases contenues dans l'ADN se traduit par la fabrication des protéines, constituants fondamentaux des tissus, organes et mécanismes biologiques de l'organisme.**

- **L'ADN constitue la mémoire à long terme de l'organisme. En effet toute l'information contenue dans l'ADN est transmise de la cellule "mère" à la cellule fille.**

Les objets biologiques étudiés

L'ADN est constitué de deux types de régions : les régions codantes (*CDS* pour CoDing Sequence) et les régions non codantes qui n'ont pas de signification connue.

Les régions codantes sont des régions de la séquence dont la lecture se traduira par la fabrication de protéines. Chacune de ces régions constitue un *gène* dés lors que l'on a pu identifier la fonction biologique du produit de la traduction, c'est-à-dire de la protéine correspondante.

La molécule d'ADN , dont les deux séquences de nucléotides forment une double hélice, après avoir été recopiée en simple hélice puis allégée de quelques caractères va être lue par un ensemble appelé ribosome. Les bases vont alors être traitées par groupe de trois (un codon) et traduites en acides aminés pour former une protéine. Le début et la fin de la fabrication d'une protéine sont signalés par des signaux spécifiques sur l'ADN : les codons *START* et *STOP* (figure 3).

La lecture d'une région de l'ADN par groupe de trois bases est orientée (directe ou indirecte), ce qui offre six phases de lecture possibles en fonction du sens de la base de départ (figure 2).

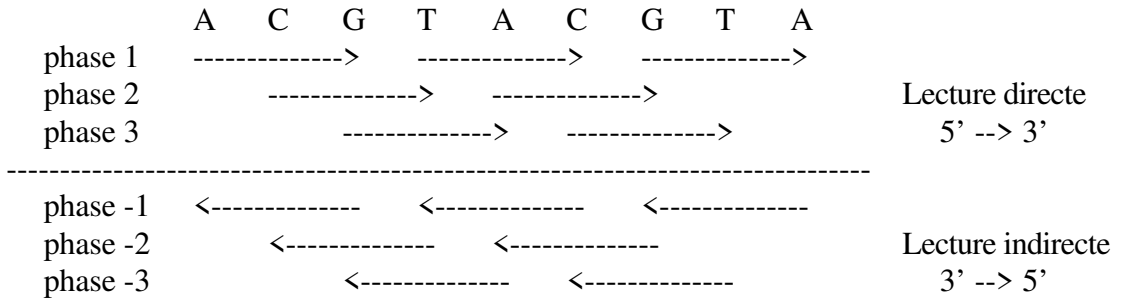


Figure 2 : Représentation des six phases de lecture possibles sur une région de l'ADN.

Le but de l'analyse d'une séquence est donc d'obtenir tous les gènes exprimables quelle que soit la phase de lecture. Pour cela on identifie sur la séquence des objets des types suivants.

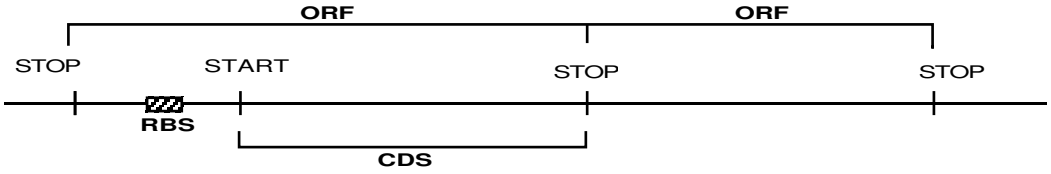


Figure 3 : Les objets biologiques étudiés.

STOP : Motifs particuliers TAA, TGA ou GAT qui signalent la fin de la synthèse d'une protéine.

START : Motifs particuliers ATG, GTG et TTG chez les bactéries, qui signalent le début de la synthèse d'une protéine.

ORF : Open Reading Frame. Il s'agit d'une phase ouverte de lecture, c'est-à-dire une portion de texte susceptible de contenir une séquence codante (CDS). Elle est délimitée par deux codons STOP.

RBS : Ribosome Binding Site : le site de fixation du ribosome. Il est situé généralement 3 à 10 paires de bases avant le codon de départ START. C'est un motif proche de la séquence "AGGAGG".

CDS : CoDing Sequence. Il s'agit du gène qui code une protéine.

1.2. Déroulement d'un projet de séquençage

Plusieurs projets de séquençage sont en cours actuellement à travers le monde afin d'obtenir la description totale du génome d'un organisme. Les objets biologiques recherchés sont ceux qui ont été décrits précédemment et un projet de séquençage se déroule généralement en quatre étapes [Beilvert93] (figure 4) :

1. *Détermination de la séquence* (c'est-à-dire de la suite de lettres A, C, G et T).
   Elle est réalisée par fragments d'ADN (quelques centaines de paires de bases). C'est un travail très lourd, 30000 à 50000 paires de bases peuvent être séquencées par an et par chercheur.

2. *Résolution des chevauchements*
   Il s'agit d'aligner les régions de recouvrement des différents fragments séquencés afin de reconstruire la séquence complète.

3. *Analyse de la séquence*
   C'est un processus itératif réalisé en fonction des connaissance du moment. Il s'agit de localiser sur la séquence des objets biologiques par application de motifs types.
   Exemple de règle : le site de fixation du ribosome (RBS) est un motif proche de la séquence AGGAGG, il précède de quelques bases (3 à 10 paires) le codon START qui est soit ATG, GTG ou TTG. Les résultats sont ensuite vérifiés par expérimentation.

4. *Recherche de la fonction biologique associée à l'objet biologique dans le fonctionnement de l'organisme*
   Elle se fera par expérimentation ou par la recherche de similitudes avec des séquences déjà connues, analysées et commentées dans des banques de données internationales (Genbank, EMBL, etc.). Des méthodes d'analyse de données et statistiques peuvent être également utilisées.
   Exemple de méthode : étude des formes fortes, cette méthode permet de mettre en évidence des ensembles d'objets homogènes et aussi de définir comment ces classes d'objets s'imbriquent les unes dans les autres.

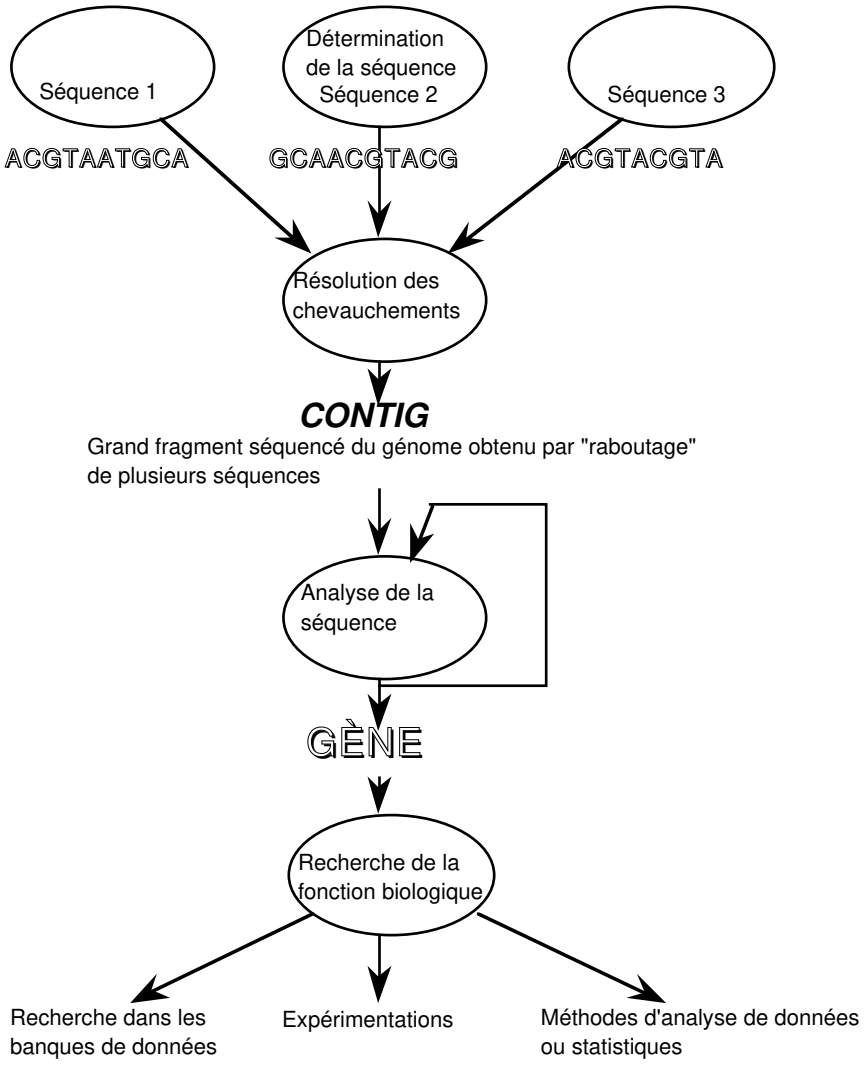


Figure 4 : Déroulement d'un projet de séquençage.

1.3. Les cartes génomiques

Nous avons présenté dans les sections précédentes les objets biologiques étudiés et le déroulement d'un projet de séquençage qui doit permettre d'identifier ces objets et leur rôle dans le fonctionnement de l'organisme étudié. Mais d'autres expériences peuvent être réalisées en parallèle pour faciliter la recherche ou confirmer les résultats.

Les cartes génomiques sont utilisées pour illustrer tous ces points de vue. Elles permettent de représenter et d'ordonner les objets biologiques issus d'expériences différentes.

On distingue généralement trois types de carte: cytogénétique, génétique et physique.

La figure 5 en montre des exemples.

Figure 5 : Les trois types de carte génomique (d'après [Schmeltzer95]).

1.3.1. Les cartes cytogénétiques

Elles sont obtenues par coloration d'un chromosome, ce qui fait ressortir des bandes claires et sombres.

Dans ce type de carte vont être représentées les bandes qui peuvent se décomposer en sous-bandes selon la résolution et des entités telles que les gènes dont on aura déterminé l'appartenance à une bande ou à un groupe de bandes [Pitaval94] [Schmeltzer95].

1.3.2. Les cartes génétiques

Une carte génétique est fondée sur le repérage des degrés de proximité entre les portions de l'ADN qui sont identifiées.

Chez les êtres sexués, il s'agit généralement d'étudier la fréquence de recombinaison des sites repérés d'une génération à la suivante. Les distances seront alors exprimées en centiMorgans (ou cM). Plus les gènes sont proches et moins ils ont de chance d'être séparés.

Une distance de 1 cM correspond à une fréquence de recombinaison de 1%, c'est-à-dire que quatre-vingt dix-neuf fois sur cent, les deux caractères seront hérités ensemble.

Chez les bactéries, il existe un processus particulier dit de *conjugaison* grâce auquel deux bactéries peuvent s'échanger du matériel génétique. Il s'agira dans ce cas d'interrompre ce processus et de regarder quels sont les gènes qui sont passés entre les deux bactéries (une donneuse et une réceptrice). Ici c'est la notion de temps qui intervient et on donne une position en minutes [Medigue91].

1.3.3. Les cartes physiques

Contrairement aux cartes précédentes dont la détermination expérimentale donne des résultats imprécis, les cartes physiques décrivent une réalité physique.

Elles sont obtenues en coupant l'ADN en fragments puis en reconstruisant leur ordre en tirant parti des recouvrements qui existent entre eux, ce qui permet d'obtenir des *contigs :* grands fragments séquencés du génome obtenus par "raboutage" de plusieurs séquences.

Leur unité est un multiple de la base (souvent le Kilobase ou Kb).

1.3.4. Conclusion

Des informations complémentaires

Les trois types de carte (cytogénétique, génétique et physique) donnent des informations complémentaires.

Ainsi, la carte cytogénétique permet de rétrécir le champ de recherche d'un gène à une région de quelques millions de bases. Le chercheur peut alors s'intéresser plus particulièrement au fragment d'ADN représentant la région puis la sonder jusqu'à isolement du gène.

Un ensemble d'objets, des représentations différentes

Une carte se compose d'objets qui peuvent prendre des représentations différentes selon le type de carte auquel ils appartiennent. Sur une carte physique, un gène sera souvent représenté par un segment alors que sur une carte génétique seul un point sera visible.

Un lien imprécis

Il n'existe pas de correspondance précise entre les unités de ces différentes cartes. On peut donner toutefois une relation "moyenne" comme 1 cM=1000 Kb entre une carte génétique et une carte physique [Jordan89]. Dans le cas de l'étude du bacile *Escherichia Coli*, où l'on dispose d'une carte génétique de 100 minutes alors que la longueur du chromosome circulaire est de 4720 Kilobases, on pourra prendre comme relation "moyenne" 1 Minute=47,20 Kb.

1.4. Les logiciels de cartographie

Il existe de nombreux logiciels d'aide à l'analyse de séquences. Nous en avons retenu trois qui nous ont paru suffisamment significatifs pour permettre une comparaison des différents moyens mis à la disposition des biologistes pour traiter les fonctionnalités propres au domaine de la cartographie génomique.

1.4.1. ACeDB

Le système ACeDB est une base de données orientée objet à l'origine développée pour représenter n'importe quel type d'information rencontrée dans le projet de séquençage du génome du nématode *Cœnorhabditis Elegans.* Il a été par la suite plusieurs fois adapté à d'autres organismes vivants. La force de ce logiciel repose sur l'affichage graphique spécialisé pour les cartes génétiques, physiques et les séquences.

La représentation d'une carte se compose d'un axe cartographique (souvent vertical pour les cartes génétiques et horizontal pour les cartes physiques) muni d'une échelle, sur lequel sont positionnées les entités cartographiques; de plus un marqueur indique la position de la partie de carte visible sur le chromosome entier [Durbin92]. La figure 6 en montre des exemples.

ACeDB dispose de fonctionnalités diverses, dont certaines sont classiques et fournies par la quasi-totalité des systèmes, comme la possibilité de zoomer sur la carte, de visualiser des informations textuelles sur les entités en "double-cliquant" dessus, etc.

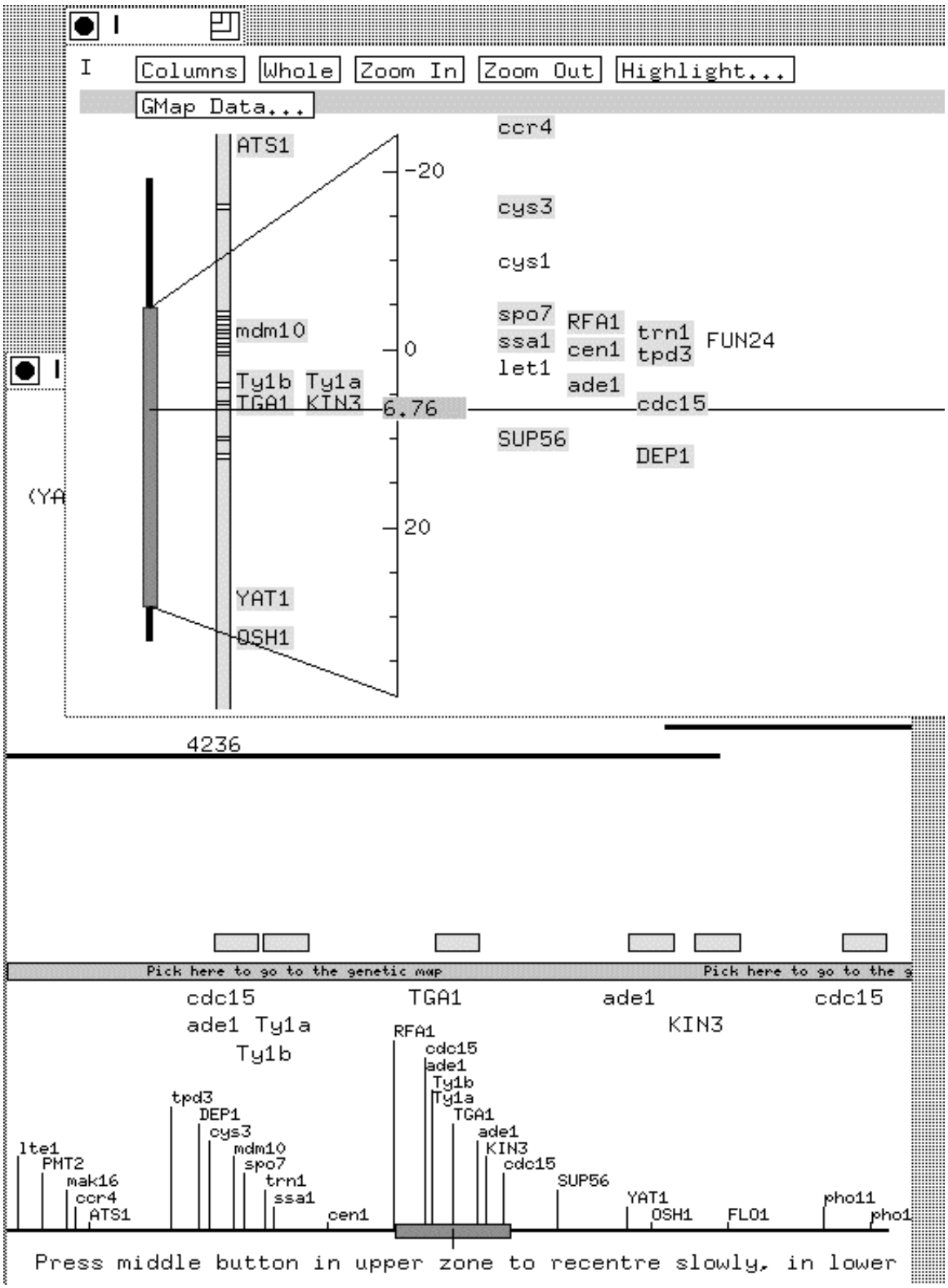


Figure 6 : Interface graphique d'ACeDB. Cette figure montre une carte génétique (verticale) et une carte physique (horizontale) issues de l'étude de la levure de boulanger (*Saccharomyces cerevisiae*). Le bouton "Columns" (en haut, à gauche) une fois activé, affiche une nouvelle fenêtre dans laquelle il est possible de choisir les types d'entités à afficher (y compris l'axe gradué). Un clic souris sur une entité (ex : DEP1) ouvre une fenêtre contenant des informations textuelles et permet d'accéder à d'autres informations sous la forme d'un hypertexte.

1.4.2. SIGMA

SIGMA, ou System for Integrated Genome Map Assembly, est un outil destiné à la création, l'édition et la maintenance de cartes génomiques de toute sorte. Il possède la particularité de représenter l'ensemble des entités (bandes cytogénétiques, gènes, etc.) dans une seule fenêtre, qui intègre toutes les cartes, quelle que soit leur résolution (figure 7).

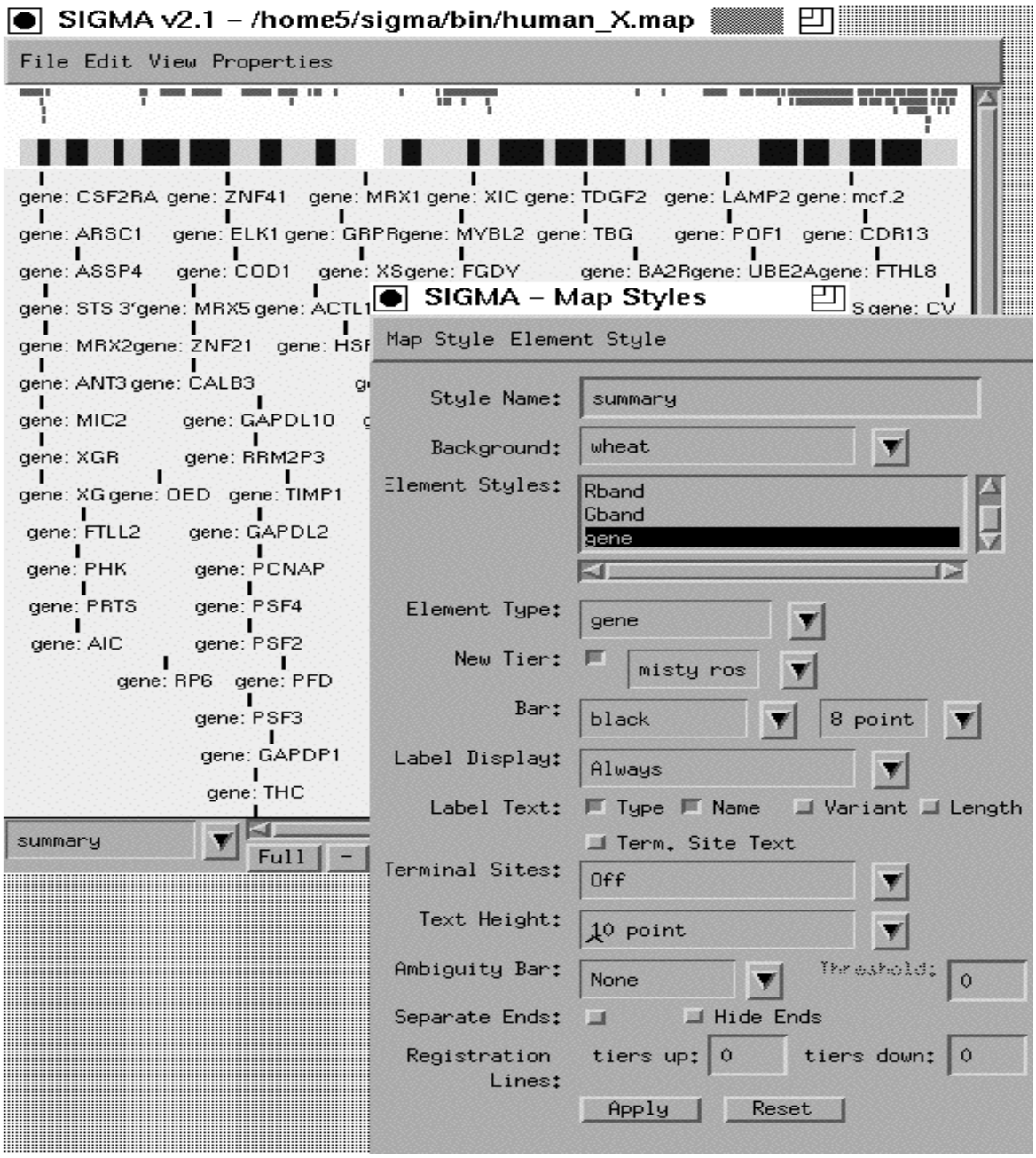


Figure 7 : Interface graphique de SIGMA. Elle se compose d'une seule fenêtre qui intègre l'ensemble des cartes. On distingue en partant du haut, des données physiques (ici des contigs) puis des bandes cytogénétiques et enfin des gènes. SIGMA offre la possibilité de définir et de sauvegarder un style de carte. Celui-ci contient, pour chaque type d'élément à afficher, des choix à prendre en compte pour sa représentation graphique (couleur, affichage ou nom du type de l'élément, etc.). Chaque type constitue une couche et l'ordre d'affichage est modifiable. Les données affichées ici sont issues de l'étude du chromosome X de l'homme.

SIGMA permet de paramètrer la représentation des différents types d'entités qui composent une carte. Ainsi il est possible de préciser la couleur, les dimensions, l'étiquetage pour un type donné et de le retenir ou de l'exclure de l'affichage. L'ensemble de ces informations qui forme le style de la carte peut être sauvé dans un fichier pour être réutilisé par la suite [Sigma92].

La documentation de SIGMA prévoit l'implémentation future d'un mécanisme qui devrait permettre de traiter le problème de conversion, au sein d'une carte, entre les différentes unités dans lesquelles sont données les positions des objets. Le peu d'explications fourni ne nous permet pas d'approfondir plus en détail cet aspect.

SIGMA fournit également des fonctionnalités plus ou moins classiques de manipulation graphique: le zoom, la sélection d'objet, etc.

1.4.3. HoverMaps

Le système HoverMaps a l'avantage de fournir une interface cartographique (figure 8) basée sur son modèle de représentation de connaissances [Dorkeld94]. C'est une différence importante avec les interfaces précédentes qui reposent sur des langages à objets mais pour lesquelles la connaissance est contenue dans de simples fichiers.

La possibilité de lier un concept de la base de connaissances à un objet graphique permet de lier plus facilement les actions sur l'interface aux objets de la base de connaissances et réciproquement. Ainsi la mise à jour d'une connaissance dans la base pourra être répercutée plus facilement dans l'interface, et inversement une modification graphique pourra entraîner automatiquement une vérification sur la validité de la modification. Enfin, le fait de n'avoir qu'à lire la description d'un objet pour l'afficher, au lieu de parcourir tout un fichier, améliore les performances de l'outil graphique.

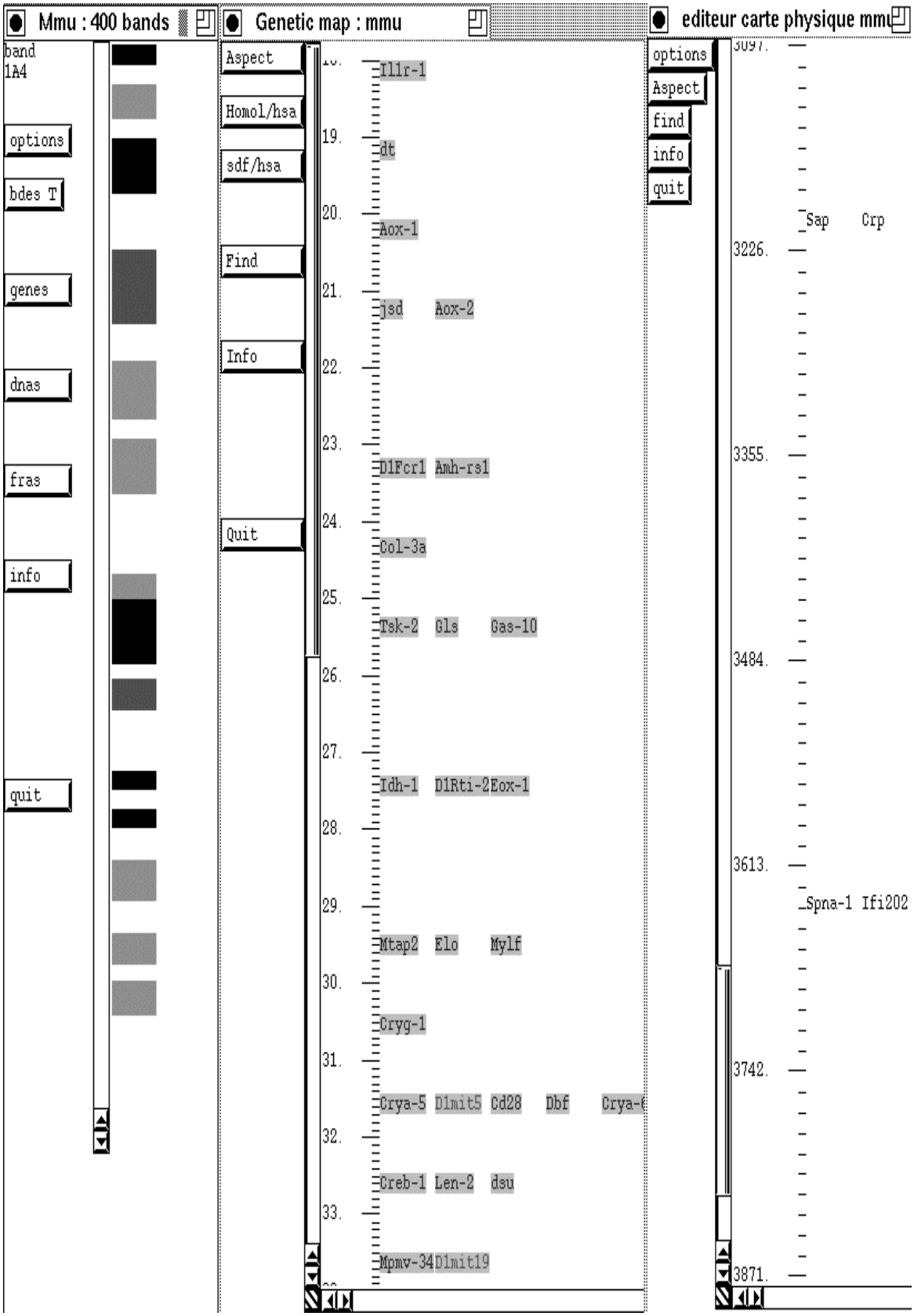


Figure 8 : Interface graphique de HoverMaps. Trois cartes (cytogénétique, génétique et physique) sont affichées. Elles sont issues de l'étude de la souris. On y retrouve les éléments cartographiques classiques (axe gradué, objets positionnés).

1.4.4. Conclusion

Il existe de nombreux autres logiciels de visualisation d'informations cartographiques mais qui procurent tous le même ensemble de fonctionnalités dans des conditions plus ou moins spécifiques :

- Affichage d'un axe gradué et d'entités.
- Zoom.
- Sélection d'entités.
- Information textuelle (plus ou moins riche).

L'objectif principal d'une interface cartographique est de permettre de provoquer par des visualisations diverses un stimulus à l'imagination :

- Au sein d'une carte : superposer les résultats de plusieurs méthodes d'analyses différentes sur un même fragment séquencé peut amener le biologiste à se poser des questions sur les méthodes employées ou le sens des résultats. Les similarités ou les différences détectées pourront alors confirmer ou infirmer les résultats obtenus.
- Entre plusieurs cartes : lier une carte physique et une carte génétique (en général moins précise) est un moyen de voir où il y a des erreurs.

Cet objectif conduit à en définir d'autres :

Un affichage multiple

L'interface doit permettre l'affichage de plusieurs cartes (ou de la même carte plusieurs fois) en limitant l'augmentation du nombre de fenêtres.

Il devra alors être possible d'arranger l'ordre d'affichage de ces cartes.

Recherche et positionnement

L'interface doit offrir des fonctions de positionnement et de changement de résolution performantes et adaptées à la contrainte des volumes de données importants.

Il doit être facile de se positionner ou de se repositionner dans une carte même si elle est volumineuse. Par exemple, offrir la possibilité de mémoriser des vues sur différentes régions d'une carte afin d'y revenir rapidement peut s'avérer très pratique sur des grandes cartes.

Enfin, offrir la possibilité de changer de résolution, et d'avoir ainsi par exemple toute la carte visible à l'écran, semble être indispensable pour identifier d'éventuels regroupements dans une carte ou entre plusieurs cartes.

Confrontation des résultats

Il s'agit là d'un point très important. Devant l'accroissement des données issues des projets de séquençage, il est indispensable d'avoir un moyen de comparer rapidement des cartes et ceci qu'elles soient ou non issues d'organismes différents et quelles que soient leurs origines, unités ou résolutions.

Ainsi, on pourra vouloir :

- Situer une sous-carte dans une carte plus globale.
- Comparer deux cartes du même type mais résultant d'expériences différentes dans un but de vérification.
- Relier des cartes d'unités différentes grâce à un facteur d'échelle (s'il est connu) ou bien sur les objets communs qu'elles se partagent, ceci pour permettre la détection d'éventuelles incohérences.

Ni ACeDB, ni SIGMA ou HoverMaps ne répondent correctement à tous ces aspects de la confrontation cartographique.

ACeDB certes fournit la possibilité de situer la partie visible d'une carte en rapport avec le chromosome entier, mais il s'agit là d'un cas particulier : il faudrait pouvoir situer n'importe quelle sous-carte dans une carte plus globale.

Ces trois logiciels permettent de visualiser simultanément plusieurs cartes, mais ils ne permettent pas de synchroniser le déplacement à l'intérieur de ces cartes, ni de faire ressortir explicitement des incohérences de positionnement.

SIGMA qui affiche à plat toutes les cartes dans la même fenêtre ne comporte de ce fait pas d'axe gradué. Il s'agit pourtant d'une information visuelle importante pour situer rapidement la position d'un objet sur une carte (surtout si celle-ci est volumineuse). Il ne fournit, pour l'instant, aucun moyen de lier des cartes d'unités différentes et/ou de résolutions différentes. Quelle que soit la solution, elle devra être facile à mettre en œuvre et à utiliser, ce qui peut paraître difficile à imaginer en conservant toutes les cartes dans une même fenêtre.

Personnalisation de la représentation

Toujours dans un but de confrontation, il semble important de pouvoir personnaliser la représentation graphique des objets contenus dans une carte (forme, couleur, etc.). D'autre part, pouvoir ordonner la représentation dans une carte en créant des couches graphiques pour une ou plusieurs familles d'objets et en choisissant les attributs à afficher par famille d'objets (sa représentation graphique, son nom, son orientation, des incertitudes, sa séquence) semble être un mécanisme graphique intéressant pour clarifier l'affichage (par exemple, pour permettre de bien distinguer les objets repérés sur chacune des six phases de lecture).

Modification d'objets

La mise à jour des entités qui constituent une carte (le changement de la position des objets, du contenu de la séquence ou la suppression d'objets) pourrait avoir lieu graphiquement dans l'interface. Mais au cas où l'interface serait connectée à une base de connaissances, cette mise à jour graphique demanderait au préalable d'être validée. Cette validation pourrait très bien entraîner alors un réajustement partiel ou total de la carte.

1.5. Extension de la réflexion

1.5.1. Abstraction

Dans le projet GREG, l'interface APIC (Aide à la Présentation des Interfaces Cartographiques) a pour objectif de permettre la visualisation et la comparaison des résultats produits par des méthodes. Plus généralement, elle doit offrir une représentation graphique des informations que l'on possède sur les séquences nucléotidiques dans les bases de connaissances (figure 9). Elle doit donc posséder un haut niveau de généricité pour pouvoir prendre en compte des nouvelle méthodes d'analyse et être ainsi l'interface "standard" unique et homogène du biologiste.

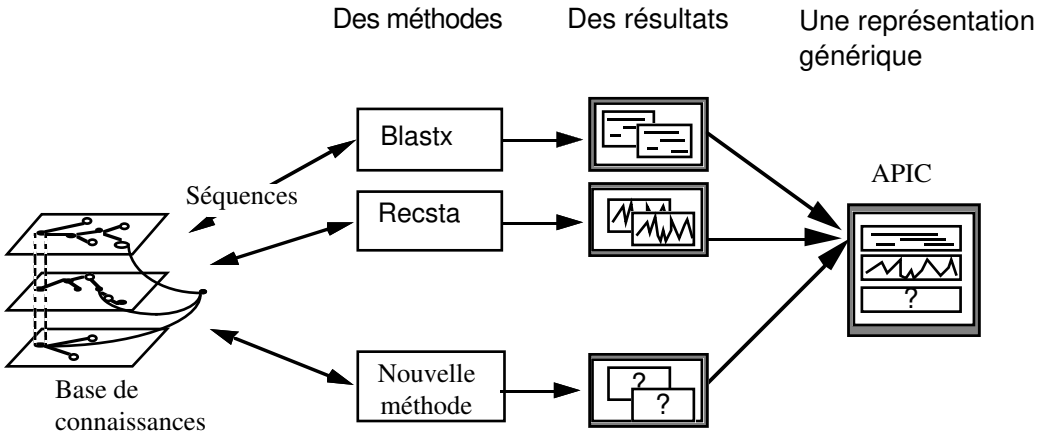


Figure 9 : L'interface APIC dans le système général.

Selon les méthodes d'analyses qui sont utilisées, les résultats obtenus peuvent se présenter sous des formes extrêmement variables : cartes génétiques, physiques, courbes, etc. De ce fait pour réaliser une interface qui soit réellement générique, il est nécessaire, dans un premier temps, de s'extraire de la vision "biologique" du problème à résoudre pour n'en garder qu'une vision purement "graphique" (figure 10). Ainsi, dans la plupart des cas, on constate qu'il s'agit de représenter sur un axe orienté et gradué (vertical ou horizontal) une succession d'objets qui peuvent être des points, des segments ou des courbes. À ces objets sont fixées des annotations diverses comme le nom de l'objet, la séquence nucléotidique ou encore l'orientation. En conclusion on voit que l'affichage met en œuvre un nombre relativement faible de concepts [Bisson94].

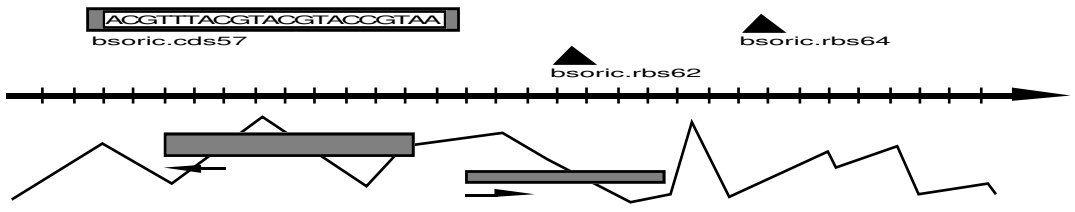


Figure 10 : Une vision purement graphique du problème.

À terme, APIC doit se présenter sous trois formes différentes. Premièrement, celle d'une bibliothèque de fonctions de visualisation de séquences, organisées sous la forme d'une API (Application Programming Interface) utilisables par les utilisateurs qui souhaitent construire une interface spécifique. Deuxièmement, celle d'une application autonome qui permettra de visualiser des informations contenues dans des fichiers externes décrits au format TINA-1 (Cf. Annexe 1). Troisièmement, dans le cadre du projet GREG, l'interface APIC constitue le module permettant de visualiser la base de connaissances et d'interagir avec elle. Bien évidemment, ces deux dernières applications ne sont en fait qu'une mise en pratique des fonctions offertes par l'API.

1.5.2. Les critères d'acceptation d'une interface

L'objectif du concepteur de l'interface est de produire une représentation perceptible la plus proche possible de la représentation mentale de l'utilisateur.

Les critères énoncés ici sont en fait des recommandations d'ergonomes aux concepteurs d'interfaces. Ils sont extraits d'un cours INRIA sur les interfaces intelligentes dans l'information scientifique et technique [Ebouenya92].

Ils forment autant de *contraintes* à prendre en compte dans la conception et la réalisation d'une interface. C'est pourquoi il nous est apparu important de les rappeler et de leur adjoindre des implications liées directement à l'étude qui nous concerne.

Ces principes sont :

- *L'aisance ou la facilité d'apprentissage*. Elle est liée au faible nombre de symboles distincts à apprendre (des objets graphiques élémentaires aux objets graphiques les plus complexes), au nombre limité de concepts particuliers liés à la modélisation (les événements utilisateurs détectables) et à une aide consistante. Participent ainsi à l'aisance : la concision et la réduction de la charge informationnelle.

⇒ L'interface cartographique, pour faciliter l'apprentissage, pourra être vue comme un *plan de travail* sur lequel plusieurs *cartes* pourront être disposées et ordonnées. Le biologiste disposera alors d'un certain nombre d'*outils* pour lui permettre de sélectionner, d'organiser et de personnaliser ce qu'il souhaite voir (codons START, gènes, etc.). Chaque outil aura un nom particulier et l'appel de cet outil provoquera l'ouverture d'une fenêtre qui contiendra tous les paramètres de réglage de l'outil.

- *La concision* qui implique que la représentation perceptible ne doit pas être alourdie de "bruits" non pertinents, qui compliqueraient sa perception et sa manipulation.

⇒ Dans l'interface cartographique, chaque carte disposera de son propre panneau de commande qui lui permettra de déclencher des actions (zoom, changement de position, etc.). Si l'on suppose que l'on puisse relier plusieurs cartes entre elles, il serait souhaitable de ne conserver qu'un seul panneau de commande qui pilote alors les actions sur l'ensemble des cartes liées. Cela permet en effet de ne pas alourdir l'affichage et de ne pas gêner la comparaison en introduisant des informations devenues inutiles.

• *La capacité de mémoire* à court terme de l'individu est réduite; l'ordinateur permet de compenser ce handicap en servant de mémoire externe que l'utilisateur peut consulter afin de "rafraîchir" sa mémoire à court terme. Pour cela, l'interface doit mettre à la disposition de l'utilisateur des outils permettant d'atteindre les informations de la mémoire externe. Les menus sont des outils classiques. Les hypertextes constituent des outils plus généraux, permettant d'atteindre, à partir d'une information, toutes les informations associées. On réduit ainsi la charge informationnelle.

⇒ Afin de pouvoir confronter des résultats, il est important que l'interface cartographique puisse être réduite aux seules informations graphiques puis amener au fur et à mesure de la demande plus d'informations textuelles. Ainsi une action de clic sur la souris pourra ouvrir une fenêtre contenant diverses informations sur l'objet sélectionné. D'autre part, offrir un mécanisme qui puisse conserver et rappeler en mémoire des régions de carte visualisées peut être une aide importante pour le biologiste qui travaille sur des cartes volumineuses.

• *La descriptivité* est la possibilité donnée à l'utilisateur non seulement de construire, mais aussi d'avoir à l'écran, à la demande, partiellement ou globalement si possible, une vision d'ensemble du système informatique. Il peut ainsi voir ce qui peut être fait, grâce à une hiérarchie de menus, des représentations iconiques, etc., compte tenu de son activité et de son profil, automatiquement. Cette descriptivité regroupe la visibilité et la mise à profit de l'intuition.

⇒ Dans l'interface cartographique, deux niveaux d'utilisation sont perceptibles :

- Le premier se situe au niveau du plan de travail. Un menu général doit permettre la sélection, la suppression ou l'ordonnancement de cartes et l'accès aux outils cités précédemment.
- Le second se situe au niveau de la carte elle-même. On préférera alors, à la place d'un menu, disposer d'un ensemble de possibilités directement accessibles (sous la forme de bouton, champ texte, etc.) qui permettront rapidement de déclencher des actions (zoom, positionnement, mémorisation, repositionnement, centrage, etc.).

• *La transparence* : les structures de données des objets graphiques, et leurs liens sont transparents pour l'utilisateur. Il n'a pas à savoir comment sont pris en compte les aspects relationnels entre objets informatiques. Il lui suffira de voir s'effectuer ses tâches.

⇒ Ainsi, par exemple, la suppression d'une carte du plan de travail devra être répercutée automatiquement sur toutes les fenêtres qui la référencent.

• *Le dialogue est à l'initiative de l'utilisateur* : le système ne doit évoluer qu'à la suite de ses actions explicites. On dit qu'il y a un besoin de contrôle de l'utilisateur.

⇒ On préférera dans le cas du paramétrage des outils demander explicitement au biologiste s'il souhaite appliquer ou non de nouvelles valeurs plutôt que de les appliquer directement.

• Lorsque, sans programmation, l'interface est utilisable par des types d'utilisateurs différents, on dit qu'elle est *adaptable*. Par exemple, l'utilisateur peut choisir la couleur de ses dossiers et d'autres paramètres. Ceux-ci représentent généralement des caractéristiques "cosmétiques" qui font appel à la perception visuelle et non à l'activité cognitive de l'utilisateur. De manière plus générale, l'adaptabilité d'une interface est l'un des critères qui participent à la convivialité et qui sont difficiles à limiter, voire cerner.

⇒ L'interface cartographique doit être suffisamment conviviale et générique pour pouvoir être acceptée et utilisée par des projets différents.

• *La représentation perceptible doit être homogène*; sa perception et les manipulations associées font référence à un cadre constant, commun aux représentations perceptibles destinées à un même utilisateur. Celui-ci peut alors évoluer dans un environnement constant sans avoir à s'adapter à chaque utilisation.

⇒ Un exemple de perception homogène dans l'interface cartographique serait de considérer, dans le cas de cartes horizontales, qu'elles aient toutes la même largeur. Un exemple de manipulation homogène pourrait être de gérer le redimensionnement des cartes affichées de manière uniforme. Ainsi, toujours dans le cas de cartes horizontales, une seule manipulation devrait permettre de redimensionner toutes les cartes en largeur.

Chapitre 2

Définition de l'interface

Nous avons énoncé dans le chapitre précédent des objectifs et des contraintes à prendre en compte dans la conception et la réalisation d'une interface cartographique. Nous allons maintenant décrire dans ce chapitre, les concepts qui ont été retenus, les problèmes qu'ils soulèvent et les solutions qui ont été choisies pour les résoudre. En introduction, nous préciserons l'environnement informatique qui a servi à développer l'interface finale. Nous rappellerons alors les grands principes de la programmation et de la conception par objets.

2.1. Environnement informatique

Le développement de l'interface a été réalisé sur une station de travail SPARC 10 de la société SUN sous le gestionnaire de fenêtres X-WINDOW. Le système d'exploitation de la station est UNIX.

Le langage à objets C++ et la bibliothèque de classes d'objets C++ ILOG Views de la société ILOG Software ont été utilisés pour l'implémentation.

Principes de la programmation par objets

La programmation par objets se différencie de la programmation classique par la connaissance d'une seule catégorie d'entités informatiques : *les objets*.

Alors que la programmation classique traite les programmes comme un ensemble de données sur lesquelles agissent des procédures, la programmation par objets considère que les programmes sont avant tout des ensembles d'objets informatiques caractérisés par les informations qu'ils connaissent (figures 11a et 11b).

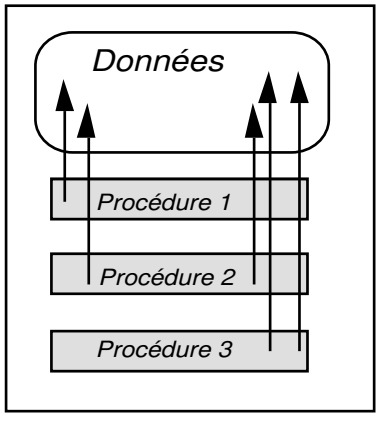


Figure 11a : Programmation classique. Découpage entre procédures et données.

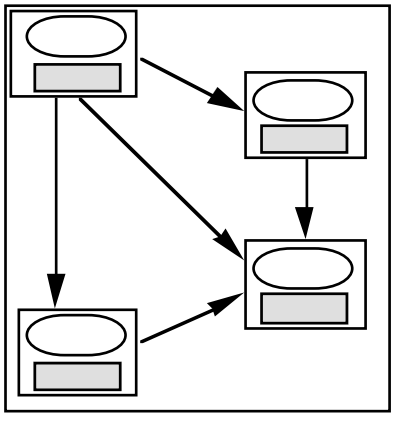


Figure 11b : Programmation par objets. Un programme est constitué d'un ensemble d'objets, chacun disposant d'une partie procédure et d'une partie données. Les objets interagissent par envoi de messages.

Avec les objets, ce sont les données qui deviennent prépondérantes. On répond d'abord à la question : "De quoi parle-t-on ?" et non pas à : "Que veut-on faire ?" comme dans la programmation classique [Ferber90].

Les notions de base de la programmation par objets

La programmation par objets repose essentiellement sur trois notions de base :

1. La classe qui va servir à modéliser ce dont on parle.
2. L'objet qui pourra être activé à l'aide de l'envoi de messages et des méthodes.
3. L'héritage qui va permettre une construction par affinage.

Classe

Plusieurs objets peuvent posséder une structure et des comportements en commun et il devient utile de pouvoir les regrouper à l'aide d'un modèle général : une classe. Une classe est donc une sorte de moule à partir duquel sont générés les objets que l'on appelle *instances* de la classe. Programmer revient alors à décrire des classes d'objets, à caractériser leur structure et leur comportement, puis à "*instancier*" ces classes pour créer des objets.

Par exemple, une carte génomique (composée d'une échelle graduée et d'objets graphiques représentant des objets biologiques) sera décrite à l'aide des classes échelle et objet graphique.

La définition d'une classe est constituée par un ensemble de *champs* (appelés aussi *variables d'instance ou attributs*) qui décrivent la structure de ses instances, et d'opérations, appelées *méthodes* qui leur sont applicables.

Un exemple plus complet sera donné par la suite dans la présentation du langage à objets C++.

Objet, méthodes et envoi de messages

La partie dynamique des objets est assurée par la notion d'envoi de message. Envoyer un message à un objet, c'est lui dire ce qu'il doit faire.

Ce comportement est décrit au niveau de leur classe par un ensemble d'opérations appelées *méthodes*.

L'héritage

L'héritage permet de factoriser toutes les informations communes à plusieurs objets à l'intérieur d'une même classe. C'est un mécanisme très puissant qui offre la possibilité de décrire des structures *génériques* ou *abstraites*. Toute classe peut alors servir de modèle à d'autres classes, appelées sous-classes de la première. Les sous-classes possèdent par défaut l'ensemble des caractéristiques (champs et méthodes) de leurs sur-classes, qui peuvent être modifiées au niveau de la sous-classe.

Ce mécanisme induit un nouveau style de programmation qui procède par affinages successifs de classes prédéfinies. Créer un programme dans ces conditions consiste à définir des sous-classes de classes plus générales, c'est-à-dire à augmenter les caractéristiques du langage en décrivant des classes plus spécifiques, plus adaptées au problème à résoudre.

2.1.1. Le langage à objets C++

Le langage à objets C++ [Mollaret89] [Khalil91] [Stroustrup92] est une extension du langage C. Il a été conçu en 1983 par Bjarne Stroustrup.

Comme nous l'avons vu précédemment, la définition d'une classe s'effectue en deux temps : d'abord la déclaration des champs et des méthodes :

```
class Objet {
    float début=0.;
    float fin=0.;
public
    void début(float);
    void fin(float);
    float début();
    float fin();
};
```

puis la définition des méthodes elles-mêmes :

```
void Objet::début(float f)
{   début=f;          }
void Objet::fin(float f)
{   fin=f;            }
float Objet::début()
{   return(début);    }
float Objet::fin()
{   return(fin);      }
```

On notera que dans la déclaration de la classe, il existe deux parties: une partie privée qui va du début de la déclaration jusqu'au mot clé *public,* puis une partie publique qui se situe après. Il est possible de définir une valeur pour chacun des champs qui sera placée lors de l'instanciation. On remarquera également qu'il est possible de surcharger une méthode en définissant plusieurs méthodes de même nom, mais avec certaines différences au niveau du nombre ou des types des arguments ou du type de message envoyé par chaque méthode.

Pour créer un nouvel objet, il suffit de déclarer une variable, "de type pointeur vers un objet" et de lui affecter une instance d'objet :

```
Objet* objet1;
objet1=new Objet;
```

L'envoi de message reprend une forme qui ne surprendra pas les adeptes de C. Il est de la forme : <objet>-><sélecteur>(<arg1>, ...,<argn>).

Ce qui donne pour changer la valeur de l'attribut début de objet1 :

```
objet1->début(300.);
```

Nous laissons au lecteur, s'il le désire, le soin d'approfondir plus en détail les possibilités qu'offre le langage C++ notamment dans la description des classes (avec ses trois sections private, protected et public) et l'emploi de l'héritage et des classes dérivées. Il est à noter que la version Sun 3.0.1 du langage C++ qui a été utilisée permet l'héritage multiple, mais demande alors que les noms des méthodes des classes héritées soient tous différents pour supprimer toute ambiguïté dans la détermination du propriétaire de la méthode appelée.

2.1.2. ILOG Views : une bibliothèque de classes graphiques en C++

ILOG Views est une bibliothèque de classes d'objets graphiques en C++ pour construire des applications graphiques [Ilog94]. La version 2.1 est portable sur différentes machines et compatible avec différents systèmes de fenêtrage et compilateurs C++.

Par exemple :

Machine	Système d'exploitation	Système de fenêtrage	Compilateur C++
SUN	Sparc (OS 4.1.2 au moins)	OpenWindows 3.1 X11 R4, R5 + MOTIF	SUN CC 2.1 3.0
PC 486	DOS OS/2	Windows 3.1 Presentation Manager	Borland 3.1 Visual C++

Son utilisation nous a permis de constater que la programmation par objets est particulièrement bien adaptée à la conception d'interfaces. Le concept d'objet est en effet facile à comprendre et il est aisé de voir une fenêtre comme un objet de dialogue qui reçoit et envoie des messages. De plus, grâce au principe d'héritage, il est facile de construire l'interface par affinages successifs et ainsi d'introduire des améliorations et des extensions sans changer les fonctionnalités de base.

D'autre part, utiliser une bibliothèque de classes d'objets graphiques est un gain de productivité : avec une bibliothèque qui implémente des caractéristiques de base dans la gestion de fenêtres, la gestion de diagrammes, de courbes, etc., il devient plus facile de s'intéresser aux concepts propres à l'application. Il est alors possible de créer sa propre collection de classes au dessus de cette bibliothèque et de l'étendre à volonté.

Nous allons maintenant présenter au lecteur comment s'organise la bibliothèque de classes d'objets graphiques ILOG Views et à travers cette présentation insister sur les principaux concepts que nous avons utilisés dans l'implémentation.

La bibliothèque ILOG Views laisse le choix au développeur entre quatre niveaux de programmation en liaison avec ce qu'il veut faire :

1. Les primitives.
2. Les objets 2D.
3. Les gestionnaires (*managers*).
4. Les applications.

Niveau 1 : les primitives

Cette couche a pour objectif d'assurer une portabilité à travers différents systèmes de fenêtres. Elle permet :

- L'utilisation de fonctions virtuelles pour dessiner des formes de base telles que des lignes, des courbes, des rectangles, des arcs, etc.
- La gestion des ressources graphiques telles que des couleurs, des polices de caractères, etc.
- La gestion des événements (clavier et souris).
  Il est possible d'identifier des types d'événements (un bouton, une touche de clavier), l'information géométrique des événements (coordonnées globales), les événements (bouton gauche, caractère ASCII).
- La manipulation de transformateurs (rotation, zoom, translation) : les fonctions de base permettent de les relier, de les inverser et finalement de les appliquer sur les objets et les vues.

Niveau 2 : les objets 2D

Ce niveau permet d'afficher et d'animer des données dans des vues graphiques et étend la bibliothèque des composants. Il est possible d'utiliser ou d'étendre plusieurs classes d'objets prédéfinis. À côté des indications géométriques (les coordonnées système d'ILOG Views utilisent des entiers) et des propriétés graphiques, les objets savent comment :

- Se dessiner eux-mêmes avec n'importe quelle transformation.
- Se dupliquer.

Les objets peuvent être sauvegardés dans un format ASCII pour être réutilisés plus tard. Il est également possible d'ajouter des propriétés sur les objets et de grouper des objets. On trouve à ce niveau la notion de *container*. Il s'agit, en quelque sorte, d'une "boîte active" qui peut englober un ensemble d'objets, et qui se caractérise par des fonctionnalités telles que :

- L'attachement de transformations à la vue complète telles que rotation, translation ou le changement d'échelle.
- L'association immédiate d'actions, aussi appelées accélérateurs (tel que associer la touche clavier "z" à l'action de zoom).
- La gestion de propriétés de base sur les objets telles que la visibilité, la sélection, le déplacement ou le nommage.

Niveau 3 : les gestionnaires (*managers*)

Un *manager* manipule les relations entre un ensemble d'objets et un ensemble de vues. Il offre comme principales fonctionnalités :

- Le stockage d'objets dans plusieurs couches (ou *layers*) pour gérer la selection et la visibilité sur des groupes d'objets : à chaque couche est assignée une priorité d'affichage.
- La gestion de vues multiples du même ensemble d'objets, et la conservation de la cohérence entre vues : cela permet d'avoir différentes manières de voir des objets sans les dupliquer.

Niveau 4 : les applications

Ce niveau contient des classes prédéfinies et des applications préconçues telles que des boîtes de dialogue ou une classe pour représenter n'importe quel graphe hiérarchique ou cyclique. C'est à ce niveau que l'on trouve un éditeur de dessin personnalisable.

Programmer avec la bibliothèque ILOG Views consiste donc à décrire des classes d'objets au dessus de la bibliothèque des classes prédéfinies. On peut ainsi construire les différentes fenêtres qui composent une interface en instanciant un certain nombre de classes existantes (bouton, label, liste de chaînes de caractères, menu, *container*, *manager*, etc.).

L'utilisation de l'éditeur de dessin *ilvedit* permet d'optimiser cette étape. Il offre la possibilité de définir interactivement l'image d'une fenêtre de l'application en choisissant ses composants, parmi l'ensemble des objets de la bibliothèque ILOG Views (figure 12).

Il est alors possible de nommer ces objets et de leur associer le nom d'une fonction (*callback*) qui sera déclenchée lorsque l'objet sera sélectionné ou un comportement par défaut (*interactor*) parmi plusieurs classes de comportements possibles (ex. : un objet pourra être redimensionné ou déplacé).

Tous ces éléments pourront alors être sauvegardés dans un fichier pour être rappelés plus tard dans le code de l'application.

Il restera alors à implémenter les fonctions et à spécialiser les classes de comportement prédéfinies pour préciser d'éventuels contrôles ou l'action finale à réaliser lors de l'interaction.

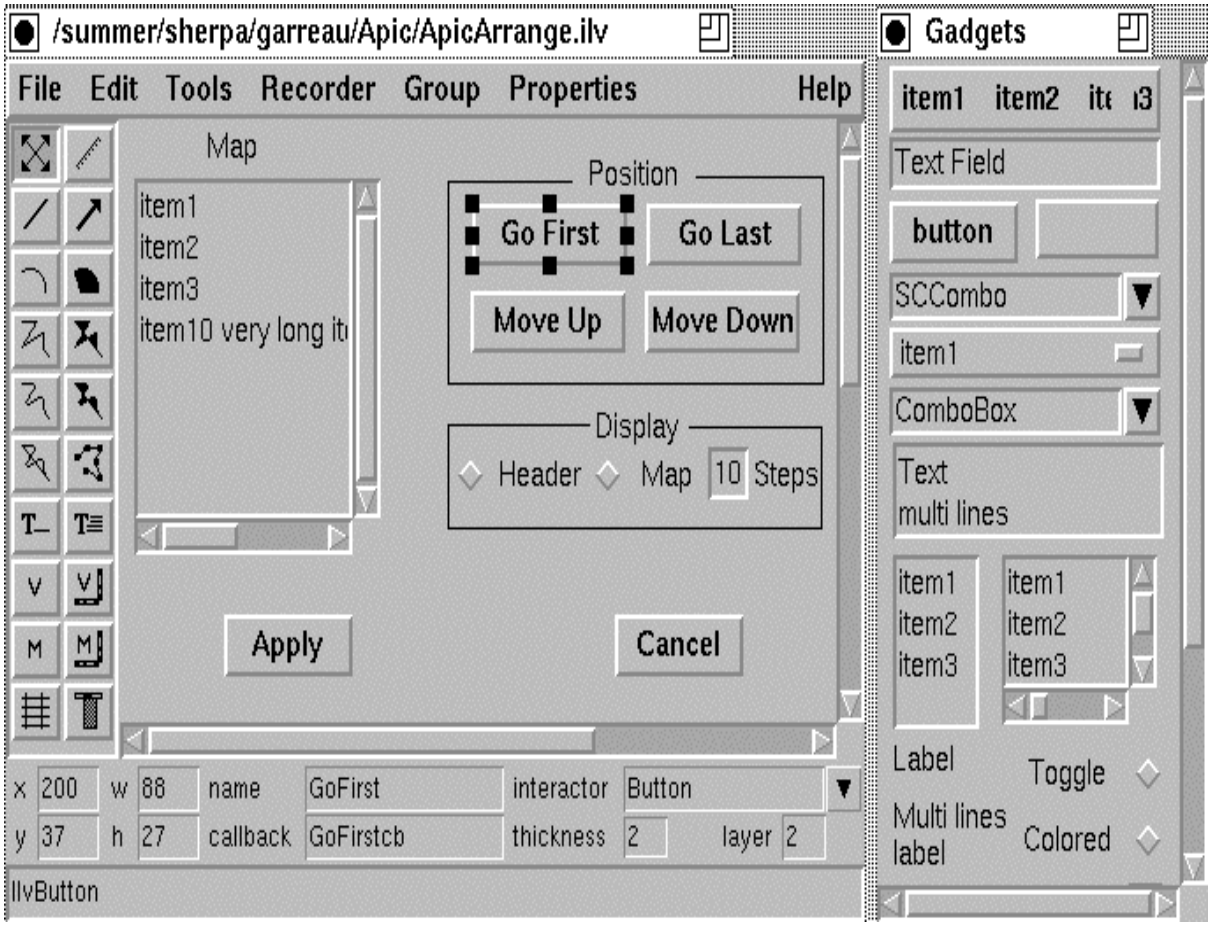


Figure 12 : l'éditeur de dessin *ilvedit*. Un objet peut être sélectionné dans la fenêtre *Gadgets* et déplacé dans la fenêtre principale. Il sera alors possible de lui associer un nom (ex : on a nommé un bouton GoFirst) et une fonction (*callback*) qui sera déclenchée lorsqu'il sera sélectionné (ici GoFirstcb).

La notion d'objets permanents utilisables par un programme comme données d'entrée n'est pas nouvelle. Dans le domaine des Boîtes À Outils (BAO), cette description est maintenue dans un fichier dit de ressources distinct du fichier contenant le code. Le fichier de ressources est une suite structurée de caractères ASCII qui pour chaque objet de représentation (appelé ressource), précise le type, l'identification et diverses informations liées au comportement de la ressource. L'accès au fichier des ressources s'effectue grâce à des primitives de la BAO [Coutaz90].

La connaissance de l'ensemble des possibilités offertes par une bibliothèque de classes d'objets prédéfinies peut prendre un temps assez long (de l'ordre de 6 mois [Coutaz90]). Il pourra être réduit par l'utilisation d'une documentation adaptée (sous la forme d'un hypertexte par exemple).

Un exemple de classe et d'utilisation de la bibliothèque de classes d'objets pour la construction d'interfaces graphiques ILOG Views est donnée en annexes (Cf. Annexe 5).

2.1.3. Conception par objets

Le but de cette section est de rappeler les grandes étapes de la conception par objets et de retenir quelques conseils.

Les grandes étapes de la conception par objets

Les méthodes classiques de conception reposent sur une approche totalement descendante : en premier lieu on effectue une analyse des besoins, puis on spécifie le logiciel permettant de répondre à ces besoins, ensuite on code le logiciel et enfin on le met au point. Même la phase d'analyse et de codage est descendante : on caractérise d'abord les grandes fonctions, puis on détaille celles-ci de manière récurrente jusqu'à parvenir à des fonctions si simples qu'elles peuvent être codées directement. Quelques allers et retours sont bien prévus entre les étapes, mais généralement on ne passe à l'étape suivante que si l'étape précédente est terminée.

Dans la conception par objets, la démarche est moins descendante et plus itérative. On réalise rapidement une première maquette, qui servira en fait à la spécification du logiciel final [Ferber90].

Un cycle de conception par objets s'effectue en cinq étapes [Ferber90]:

1. Identifier les entités du domaine.
2. Structurer le domaine.
3. Identifier les opérations.
4. Décrire plus précisément ces opérations.
5. Décrire le lancement du programme.

Identifier les entités du domaine

C'est l'étape la plus importante.

Elle s'élabore à partir de la question générale "De quoi est-il question ?". Cette étape comme les autres est effectuée de manière itérative : on détermine initialement les classes les plus importantes, puis, au fur et à mesure de l'avancée de l'analyse, les classes sont enrichies, distinguées ou fusionnées. Les classes principales correspondent généralement aux mots et aux concepts principaux du domaine. Enfin il existe une règle très simple, que l'on peut se donner au départ pour savoir quelles sont les classes du domaine : faire parler le spécialiste, et noter le vocabulaire qu'il emploie. Les noms communs qu'il utilise auront tendance à devenir des objets (et donc des classes) et les verbes à se retrouver implémentés sous forme de méthodes.

Structurer le domaine

Une fois les classes définies, il faut savoir les organiser et les relier. On distingue souvent les propriétés des relations : les premières ne mettent en œuvre que des valeurs simples (nombres, booléens, chaînes de caractères) alors que les secondes mettent en relation des objets plus complexes.

Identifier les opérations

Après avoir examiné dans les étapes précédentes les aspects statiques, il s'agit de s'intéresser aux aspects dynamiques : les opérations que l'on peut faire sur ces objets, et que l'on implémente sous forme de méthodes. Dans un premier temps on s'intéresse uniquement aux protocoles, c'est-à-dire à l'ensemble des messages auxquels sait répondre une population d'objets, sans véritablement prendre en compte la manière dont ces opérations vont être implémentées. On verra ainsi souvent des opérations "remonter" le long des classes afin de définir un protocole général d'objets caractérisés par un comportement par défaut, lequel se résume souvent à ne rien faire.

Décrire précisément ces opérations

C'est la phase de codage proprement dite. Cette étape correspond à l'implémentation de la description des méthodes dans un langage à objets.

Décrire le lancement du programme

Une fois toutes les classes définies avec leurs attributs et leurs méthodes, il suffit de mettre en route le programme. Ceci s'effectue généralement en créant quelques instances initiales (souvent une seule) et en leur envoyant quelques messages pour démarrer le processus. Ensuite de message en message, le programme se déroule. C'est ici qu'interviennent la mise au point où les interfaces élaborées, et les *debuggers* symboliques qui se révèlent des atouts précieux.

L'ordre de ces étapes n'est pas figé. Il est possible de revenir en arrière dès que les nécessités l'imposent.

Un des conseils, souvent énoncé en conception par objets et que nous retiendrons tout particulièrement, est celui de *prototyper rapidement avant de passer à une spécification complète*. La conception par objets se démarque très notablement des méthodes de conception traditionnelles en faisant une grande part au maquettage et au développement d'applications partielles. Il faut très rapidement passer à une première ébauche du logiciel. Cette ébauche servira en fait à spécifier le logiciel. Cette différence d'approche provient essentiellement de la facilité avec laquelle il est possible de transformer, modifier, voire réécrire un logiciel écrit avec des objets. L'idée que l'on a le matin peut être testée dans l'après-midi. Cela ne veut pas dire qu'il faut programmer sans spécifier, mais simplement la programmation peut, elle aussi, servir à la spécification [Ferber90].

2.2. Les concepts élémentaires

Le chapitre 1 à travers les conclusions sur les cartes génomiques (Cf. §1.3.4), les logiciels de cartographie (Cf. §1.4.4) et l'énoncé des critères d'acceptation d'une interface (Cf. §1.5.2) a fait ressortir un nombre important d'objectifs et de contraintes à respecter.

Nous nous proposons maintenant de définir tous les concepts élémentaires qui sont utilisés dans l'interface cartographique, les problèmes qu'ils soulèvent et les solutions retenues.

Le lecteur pourra ensuite continuer à prendre connaissance de la façon dont ont été implémentés les différents concepts : à travers le chapitre 3 qui s'attache à décrire, plus précisément, le modèle des fenêtres de l'interface; et à travers la description de la bibliothèque de classes d'objets pour la construction d'interfaces cartographiques APIC, qui pour plus de clarté a été mise en annexes (Cf. Annexe 6).

Posons-nous alors la question "De quoi est-il question ?".

Il s'agit de représenter dans un *plan de travail,* différentes *cartes* adjointes d'informations spécifiques telles que le nom de la carte, son unité et des graduations. Une carte est composée d'*objets* que l'on aura classés comme appartenant à une famille (gène, etc.) et auxquels on aura attribué une position, et éventuellement un nom, une orientation, des incertitudes sur la position, une séquence, un descripteur permettant de décrire une courbe.

Ces objets devront posséder un comportement : par exemple, être sensible au clic de la souris. Un clic sur un de ces objets fera apparaître des informations relatives à cet objet.

Afin de permettre au biologiste de personnaliser sa représentation, il pourra organiser les objets qui composent une carte en *couches* (structuration horizontale qui peut par exemple, servir à la description des six phases de lecture d'une séquence) et en *plans* (qui autorisent le chevauchement d'objets et peuvent servir par exemple, à superposer les résultats de plusieurs méthodes sur une même carte).

Enfin, pour répondre aux besoins de confrontation de cartes d'origines, d'unités ou de résolutions différentes; le biologiste pourra définir des *liens* entre plusieurs cartes de manière global ou sur les objets communs que se partagent ces cartes.

2.2.1. Le plan de travail

Affichage multiple

Le problème à traiter ici est d'arriver à visualiser plusieurs cartes (dont le déplacement peut être synchronisé, nous le verrons plus tard) sans arriver à un fouillis de fenêtres. À la solution qui consiste à ouvrir une fenêtre par carte, on a préféré celle d'un plan de travail unique dans lequel sont empilées les cartes les unes à la suite des autres. En effet, cette solution offre la possibilité de structurer automatiquement l'environnement de travail du biologiste afin de lui éviter des manipulations redondantes et des recouvrements intempestifs. La comparaison synchronisée de cartes sera rendue plus facile si toutes les cartes ont été automatiquement alignées. Dans le même ordre d'idée, il est plus facile de concevoir une méthode générale de redimensionnement que d'avoir à redimensionner des cartes une à une. Un plan de travail unique, muni d'ascenseurs, dans lequel sont chargées toutes les cartes permet de gérer automatiquement les problèmes de recouvrement des cartes et de récupération de place (figure 13).

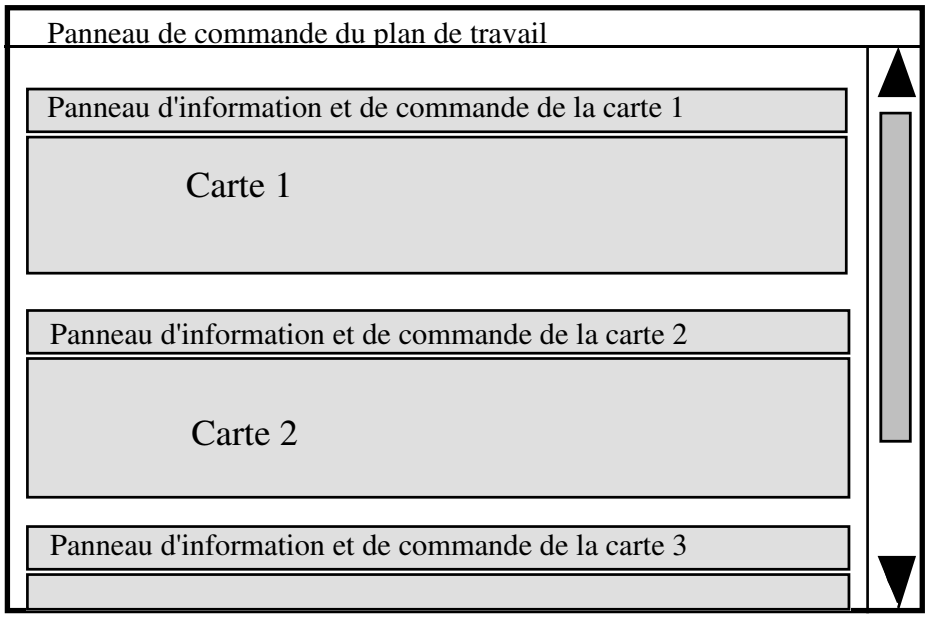


Figure 13 : Aspect général du plan de travail.
Le plan de travail comprend un panneau de commande qui permet de sélectionner des opérations concernant l'ensemble des cartes.

On peut alors imaginer un mécanisme qui autorise les réordonnacements de cartes à l'intérieur du plan de travail. Il sera possible de faire passer une carte au début ou en fin d'affichage, ou encore de faire progresser une carte d'un cran vers le haut ou vers le bas dans la liste des cartes du plan de travail.

Nous le verrons juste après, chaque carte dispose d'un panneau de commande pour contrôler les actions qu'il est possible de réaliser sur une carte (zoom, positionnement, etc.). Il paraît alors intéressant de pouvoir, dans le cas de cartes liées, retasser l'affichage en supprimant les panneaux de commandes intermédiaires devenus inutiles, ce qui permet d'accoler plus finement deux cartes.

2.2.2. La carte

Mode de représentation

Nous l'avons constaté lors de l'étude des logiciels de cartographie (Cf. §1.4), une carte peut être représentée verticalement ou horizontalement. On pourrait même imaginer une représentation spécifique pour les organismes procaryotes dont le chromosome est circulaire.

La représentation horizontale semble être a priori la plus naturelle, elle correspond effectivement à une lecture qui a lieu de gauche à droite à laquelle nous sommes plus habitués. Elle est de ce fait bien adaptée à la représentation des séquences et des courbes. Elle est également bien adaptée aux cartes où l'on trouve peu d'objets à la même position, ce qui est le cas généralement des cartes physiques (figure 14). Par contre elle est moins adaptée lorsqu'elle oblige une lecture verticale, ce qui est le cas des cartes génétiques où, par exemple, plusieurs gènes peuvent se trouver à la même position.

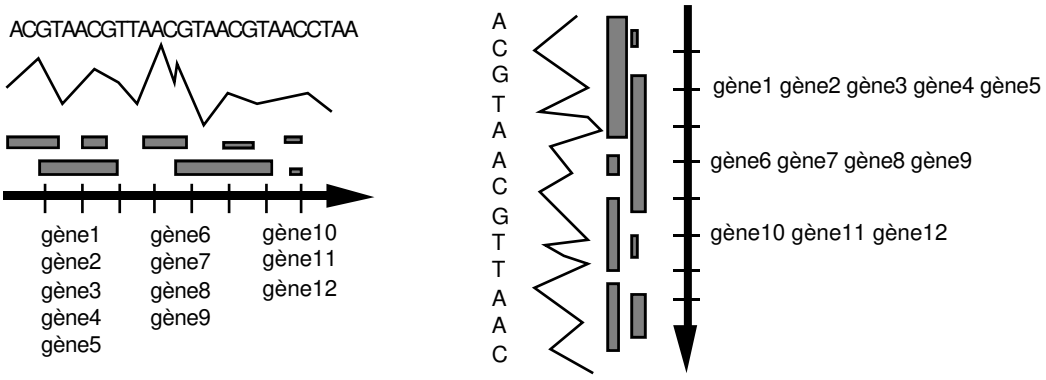


Figure 14 : Les représentations horizontale et verticale sont bien adaptées lorsqu'elles correspondent à une lecture qui a lieu de gauche à droite, à laquelle nous sommes plus habitués. Ainsi on représentera plutôt une carte physique par une représentation horizontale et une carte génétique par une représentation verticale.

La représentation horizontale nous apparaît donc être un bon compromis. Elle est, en effet, bien adaptée à la représentation des séquences, courbes et cartes physiques qui représentent le but privilégié des projets de séquençage. Mais il est évident qu'à terme, il faudrait pouvoir disposer des deux représentations.

La figure 15 montre l'aspect général d'une carte.

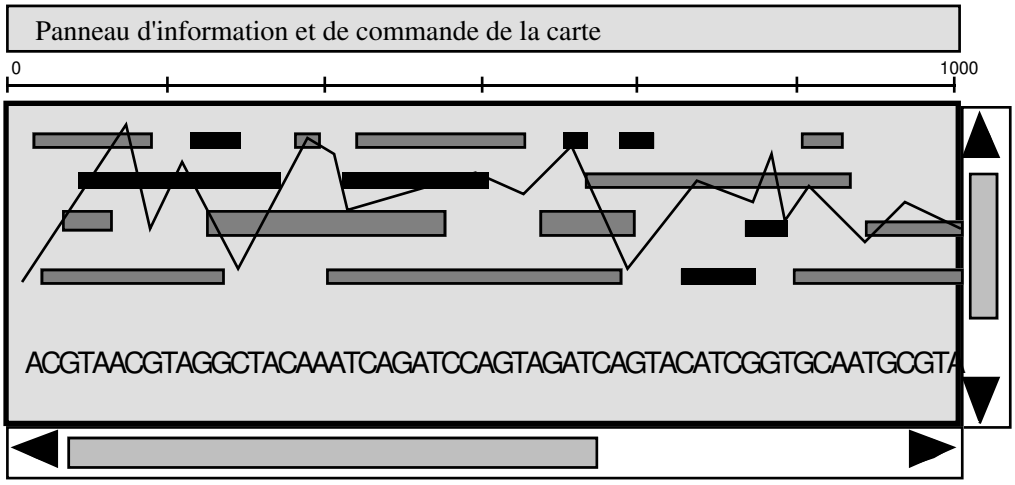


Figure 15 : Aspect général d'une carte.
Une carte comprend un panneau d'information et de commande qui permet de sélectionner des opérations (zoom, positionnement, etc.). Elle comporte une échelle graduée qui est l'axe horizontal sur lequel sont positionnés les objets de la carte. Deux ascenseurs permettent un déplacement vertical et horizontal à l'intérieur de la carte.

Navigation dans une carte

Plusieurs problèmes doivent être résolus en relation avec les possibilités de navigation dans une carte.

La résolution

Plutôt que de laisser à l'utilisateur la possibilité de donner directement un facteur d'échelle, c'est-à-dire le rapport entre une unité de la carte et le nombre de pixels représentant cette unité - rapport qui ne veut pas dire grand chose pour un biologiste - on a préféré offrir la possibilité de donner un coefficient multiplicateur qui s'applique à un facteur d'échelle interne.

Le volume d'une carte

Une carte peut avoir un volume important. Dans ces conditions, il faut pouvoir se déplacer dans la carte et ceci quelle que soit la résolution choisie. Si l'on prévoit un seul type de déplacement, par exemple le traditionnel outil de déplacement (scroller) fourni par la majorité des logiciels graphiques, la taille de la barre de défilement (scrollbar) - qui illustre le rapport entre la partie visible de la carte et la totalité de la carte - risque de devenir très vite non signifiante sur des cartes volumineuses.

On peut alors imaginer différents modes de déplacement :

- Standard : il est transparent pour l'utilisateur (cas de l'outil traditionnel de déplacement (scroller)).
- Absolu : il permet à l'utilisateur de donner exactement une position (dans l'unité de la carte).
- Relatif : il fournit la possibilité de choisir le sens du déplacement et la distance à parcourir. L'utilisateur peut alors donner un pas d'incrémentation (dans l'unité de la carte) ou le type de l'objet vers lequel il souhaite aller.

Efficacité de la perception

Une opération indispensable demandée à une carte est de pouvoir se dessiner entièrement.

D'autre part, afin de servir de mémoire externe au biologiste, il parait intéressant de pouvoir charger en mémoire des portions de carte visibles pour y revenir par la suite.

2.2.3. Les objets

Nous l'avons déjà évoqué dans le premier chapitre, les objets représentés sur une carte peuvent prendre des formes différentes : un segment, un point, une séquence sous la forme d'un texte ou une courbe.

Plutôt que de décrire autant de concepts différents, il nous est apparu plus générique de considérer un objet comme une somme d'informations à représenter : un moule dans lequel sont rangées des propriétés que l'on peut, si elles sont présentes, décider de montrer ou non.

La figure 16 montre la représentation des différentes propriétés qui peuvent être attachées à un objet.

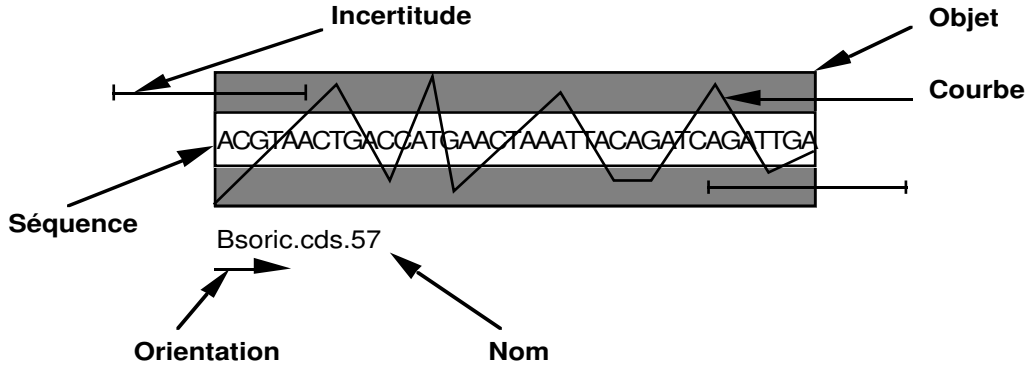


Figure 16 : Représentation des principales propriétés associées à un objet.

On peut attribuer à un objet, la liste des propriétés suivantes :

Un nom	:	c'est une chaîne de caractères qui correspond au nom biologique de l'objet. Cet propriété peut ne pas être donnée, certains objets peuvent ne pas avoir de nom.
Une orientation	:	cette propriété permet de préciser le sens de l'orientation de l'objet (3'->5' ou 5'->3').
Des positions	:	un objet peut avoir une dimension ou être un objet ponctuel. Ces propriétés précisent les positions de début et de fin de l'objet sur l'axe horizontal.
Des incertitudes	:	elles concernent les positions de début et de fin de l'objet.
Une séquence	:	cette propriété indique la séquence qui est associée à l'objet.
Une courbe	:	cette propriété permet de spécifier une courbe.

Une carte peut contenir ainsi plusieurs objets et dessiner pour chacun d'entre eux leurs nom, orientation, incertitudes, séquence et courbe. Ou alors une carte peut ne comporter qu'un seul objet qui représente par exemple la séquence totale ou une courbe de prédiction de parties codantes.

Un objet peut avoir un comportement qui sera déclenché s'il est sélectionné, par exemple afficher des informations textuelles.

Le critère de distinction qui a été retenu pour identifier la famille à laquelle appartient un objet est le *type* de l'objet. Il s'agit d'une propriété supplémentaire qui précise si l'objet est un gène, un codon Stop, etc.

Éfficacité de la perception

L'étude de la représentation d'un objet (et de ses propriétés) nous a conduit à spécifier des traitements supplémentaires autour du concept d'objet.

Représentation ponctuelle

Il est essentiel de pouvoir représenter un objet biologique de deux manières différentes : comme un segment ou comme un point.

Filtrage

Il est important de ne pas surcharger l'affichage et de pouvoir représenter l'essentiel, en choisissant parmi les propriétés de l'objet, celles qu'il est utile de montrer (par exemple, l'orientation sur une carte génétique n'a pas de sens). Il est également nécessaire de favoriser l'intuition en offrant à l'utilisateur la possibilité de montrer ou de cacher des familles d'objets.

Personnalisation de la représentation

L'interface doit permettre de personnaliser le graphisme utilisé pour chacune des propriétés à représenter. Ainsi, en fonction de ses préférences et du type de carte qu'il doit manipuler, l'utilisateur doit pouvoir aisément paramètrer l'affichage de manière à mettre en valeur les informations sur lesquelles il travaille.

A partir du type de l'objet on doit donc pouvoir changer les attributs des objets graphiques qui représentent les propriétés principales d'un objet :

- La forme (ponctuelle ou non), la couleur, la hauteur de l'objet graphique qui représente l'objet lui-même.
- La couleur et la police de caractères du label qui représente le nom de l'objet biologique.
- La couleur et la forme de l'objet graphique qui représente son orientation.
- La couleur et la forme des objets graphiques qui vont représenter les incertitudes sur les positions, de début ou de fin, de l'objet biologique.

Il faut noter ici que la courbe et la séquence associées à un objet sont représentées à l'intérieur de cet objet. Pour augmenter la hauteur d'une courbe il suffira donc simplement d'augmenter la hauteur attachée à la représentation du type de l'objet qui "encapsule" la courbe. Cette solution répond bien au problème lié à la superposition de courbes sur d'autres objets dont on ne connaît pas a priori la place qu'ils vont prendre à l'écran.

Affichage de la séquence

Un autre problème à traiter ici est celui de la visualisation de la séquence même lorsque l'échelle courante ne permet pas de dessiner les lettres qui représentent chaque base. Il est important de pouvoir repérer, très tôt, les objets dont on connaît la séquence. Ainsi après les avoir identifiés, l'utilisateur pourra augmenter la résolution pour visualiser plus finement la séquence. La solution que nous proposons ici consiste à distinguer trois cas de figure :

1. La résolution est trop faible (moins d'un pixel par base). La séquence sera représentée par un rectangle de couleur à l'intérieur de l'objet.

2. La résolution permet de représenter une information pour chaque base (au moins un pixel par base). Chaque base peut être codée par une couleur et sera représentée à l'écran par un trait de la couleur choisie. Ce mode de visualisation est très intéressant car il permet par exemple d'avoir une idée rapide de la composition des séquences. Par exemple, si l'on colore les bases G et C en noir et les bases A et T de la même couleur que l'objet, on peut comparer de façon visuelle les pourcentages en G et C de deux séquences différentes.

3. L'utilisateur souhaite voir la séquence au dessus d'une certaine résolution qu'il aura lui-même choisi (au moins au dessus de 5 pixels par base). Chaque base sera alors représentée par la lettre qui lui est associée et pourra, comme dans le deuxième cas de figure, être colorée.

2.2.4. Les couches et les plans

Il est essentiel de permettre à l'utilisateur de structurer dynamiquement l'affichage des différents types d'objets d'une carte. Cette structuration est utile lorsque l'on veut organiser les différentes informations de la carte. C'est par exemple le cas lorsque l'on veut afficher les codons STOP pour les six phases de lecture d'une séquence nucléotidique : chaque phase doit être positionnée dans une couche spécifique. Nous allons décrire ici la solution que nous avons retenue.

La structuration est effectuée au moyen de la notion de "couche" (*layer*) qui divise la carte en plusieurs bandes horizontales (figure 17). Chaque couche contient un certain nombre de types d'objets.

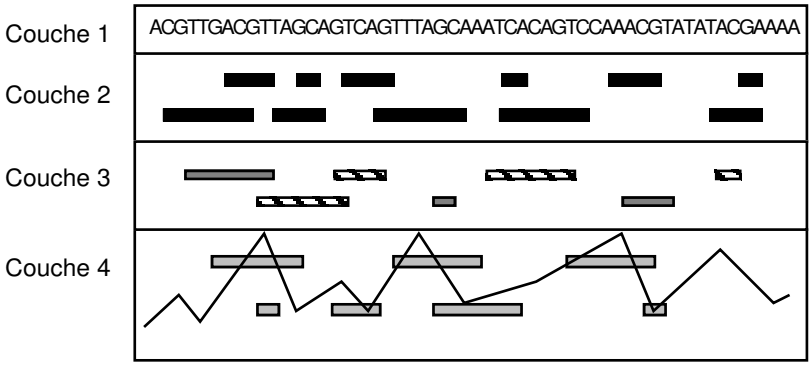


Figure 17 : Structuration des cartes en couches.
• Une couche est constituée par un ou plusieurs types d'objets.
• Un type d'objet n'appartient qu'à une seule couche.
• Les couches sont affichées par ordre croissant.

Au sein d'une couche, les objets biologiques ne sont définis que par leur emplacement sur la séquence. De ce fait, on peut avoir à l'affichage des objets qui apparaissent à une même position, soit parce qu'ils se chevauchent, soit parce que la résolution courante est trop faible pour pouvoir les séparer. Dans ces deux cas, il est souhaitable de les placer à des positions verticales différentes de manière à supprimer les chevauchements et ainsi optimiser la lisibilité des cartes. Le positionnement des objets dans une couche est effectué automatiquement à l'aide de la **Procédure de Positionnement des Objets** (PPO) qui est décrite en annexes (Cf. Annexe 4). Cette procédure consiste à décaler automatiquement sur l'axe vertical, les objets qui partagent partiellement ou totalement une même position dans la séquence.

Cependant, dans d'autres situations il est pertinent de pouvoir placer des objets de manière absolue sans passer par la PPO et ainsi avoir une superposition d'informations. C'est par exemple le cas lorsque l'on veut placer une courbe, caractérisant par exemple l'usage du code, sur un ensemble d'objets biologiques, ou encore, si l'on veut dessiner "en fond" de carte le dessin des bandes cytogénétiques d'un chromosome. Enfin, dans le cas des codons STOP évoqués ci dessus, comme ces objets sont de petite taille (3 bases), lorsque la résolution de la carte est faible il est clair que l'on aura des recouvrements qu'il est a priori inutile de vouloir éviter.

Ainsi, dans l'interface une carte est constituée par la superposition de trois plans qui sont gérés différemment (figure 18) : un plan central dans lequel les objets sont positionnés à l'aide de la PPO et deux autres plans dans lesquels le positionnement vertical se fait de manière absolue. Le positionnement vertical d'un objet peut être choisi grâce à son attribut position (Cf. Annexe 1); si cet attribut est absent, l'objet est par défaut mis à la position verticale 0.

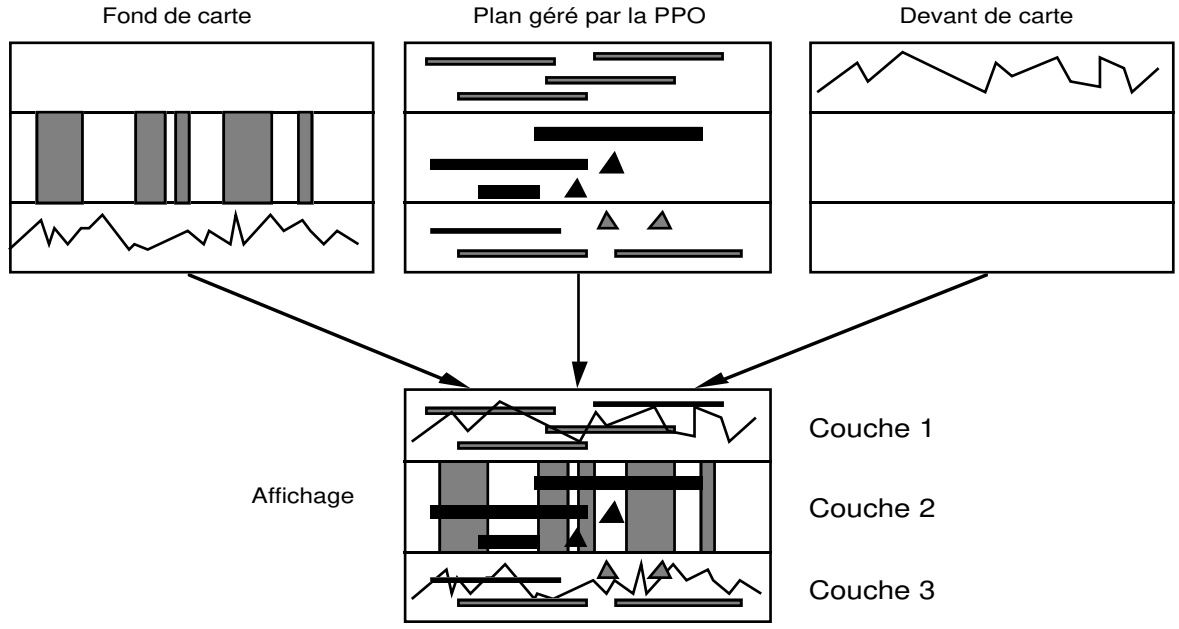


Figure 18 : Structuration en plans des cartes.
• Un plan est constitué par un ou plusieurs types d'objets.
• Un type d'objet n'appartient qu'à un seul plan.
• Les objets du premier plan (devant de carte) et du dernier plan (fond de carte) ne sont pas pris en compte par la PPO.

Soulignons que les couches sont toujours de taille identique dans les trois plans de travail, cette taille est calculée de manière dynamique en fonction des objets que l'on affiche dans la carte. Cependant, il faut noter que la structuration en couches et en plans est contrainte par l'ensemble des types existants transmis à l'interface. L'utilisateur ne peut pas décider de ranger les petits gènes (<1000 nucléotides) et les grands gènes (>1000 nucléotides) dans des couches différentes. Pour ce faire, il devra définir explicitement dans les données (au niveau de la base de connaissances ou des fichiers de données) ces deux nouveaux types afin de pouvoir les utiliser.

2.2.5. Liens entre cartes

La confrontation des résultats, nous l'avons identifié au chapitre 1 (Cf. §1.4.4), est un des objectifs principaux que doit remplir une interface cartographique.

Il s'agit donc de pouvoir lier des cartes entre elles et de permettre ainsi le déplacement synchrone à l'intérieur de ces cartes.

Le problème est complexe à gérer si les cartes ne sont pas à des échelles identiques. Le problème se complexifie encore si l'on accepte de placer des points d'accrochage entre les objets car le facteur d'échelle peut alors varier selon l'endroit où l'on se trouve dans la séquence : par exemple, si l'on accroche entre eux les gènes d'une carte physique et cytogénétique, l'échelle de la carte cytogénétique ne varie pas linéairement avec celle de la carte physique; pire, on peut même trouver des inversions de positions entre les gènes.

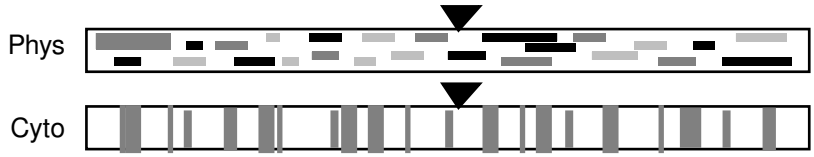
On peut dégager quelques principes et besoins généraux concernant les liens entre cartes :

- Les ensembles de cartes liées sont disjoints. En d'autres termes, une carte ne peut logiquement pas appartenir à plusieurs liens (*hook*) en même temps.

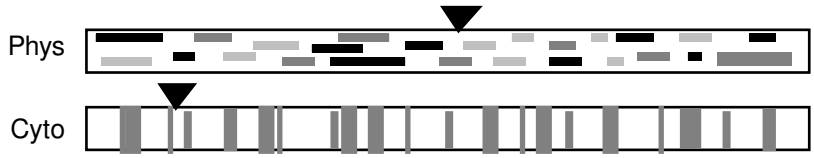
- Le déplacement effectué sur une carte, que l'on appelle alors carte "maître", sera répercuté de manière indépendante dans chacune des autres cartes liées en accord avec le Facteur d'Échelle (ou rapport d'unités) qui existe entre les cartes (noté FE dans la suite du texte). Le FE est calculé à partir des unités qui sont utilisées dans chacune des cartes liées.

- Il est nécessaire de montrer à tout moment la position relative des cartes liées afin de permettre à l'utilisateur de visualiser la correspondance entre les positions des différentes cartes. Cette visualisation s'effectue à l'aide d'un Curseur de Position (noté CP dans la suite du texte) qui est matérialisé par un triangle placé sous la barre d'échelle. Ce curseur est considéré comme étant sans dimension (1 point) afin de ne pas compliquer inutilement sa gestion.

- Par défaut les différentes cartes sont centrées, c'est-à-dire que les centres des cartes liées se correspondent. Ce choix est plus naturel pour l'utilisateur que celui de cadrer à gauche ou à droite. Lorsque les cartes ne sont pas à la même échelle, le centrage n'est plus possible sur les bords et l'on doit se contenter de déplacer le curseur de position.
  Centrage des cartes possible



Centrage impossible (on ne peut plus déplacer la représentation cytogénétique)



2.2.5.1. Lien global

Tout d'abord, nous allons examiner le cas du déplacement "simple", celui où les différentes cartes sont globalement liées et où le facteur d'échelle est connu et varie linéairement d'une carte à l'autre. Si N cartes sont liées, chaque carte C doit connaître son facteur d'échelle FE avec les N-1 autres cartes. Soulignons au passage que ce facteur d'échelle FE est bien sûr influencé par le facteur de zoom FZ qui est propre à chacune des cartes. Lorsque l'on déplace la carte "maître" M d'un certain nombre de pixels, l'algorithme général est le suivant :

```
Procédure Lien_Global
  Déplacer la carte maître
  Pour chacune des cartes Ci liées
    Lien_Global-1
```

```
Procédure Lien_Global_1
    Calculer le décalage à faire dans Ci en fonction de FE
    Selon le cas
      1) Le décalage de la carte est possible
           • On effectue le décalage
      2) Le décalage n'est que partiellement possible (bout de carte)
           • Décaler la carte autant que possible
           • Bouger le curseur de position CP de la quantité restante
      3) Le curseur CP n'est pas au centre
           • Bouger le curseur CP jusqu'à ce qu'il arrive au centre
           • Décaler la carte de la quantité restante
```

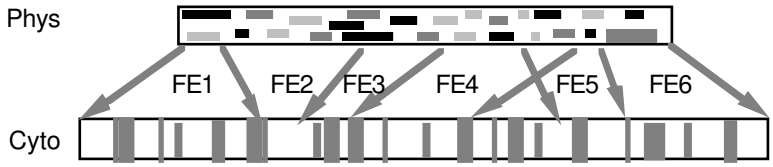
2.2.5.2. Liens entre objets

Plutôt que de faire un simple accrochage global entre les séquences, l'utilisateur peut vouloir effectuer cet accrochage par rapport aux objets communs entre les cartes. Cette possibilité est intéressante lorsque l'on souhaite synchroniser le déplacement de cartes qui possèdent des échelles qui sont non seulement différentes, mais qui ne sont, de surcroît, pas linéairement corrélées. Par exemple, c'est le cas pour les cartes physique et cytogénétique. Cette possibilité est également intéressante dans la comparaison de cartes de même unité mais avec des origines différentes : on peut alors imaginer de créer des objets communs fictifs pour "marquer" les inclusions entre cartes.

Dans ce cas, l'interface doit s'arranger pour que le déplacement se synchronise sur les objets de même type et de même nom. Dans le cas d'un lien global, le déplacement est relativement simple à appliquer car le facteur d'échelle FE entre deux cartes est constant. Par contre, dans le cas de liens par objets, ce facteur dépend de l'endroit dans lequel on est. Par exemple, entre une carte physique et cytogénétique, le facteur peut être de 1000 entre les gènes G1 et G2 et de 800 entre G2 et G3. De plus, à un objet ponctuel sur l'une des cartes peut correspondre un objet ayant une dimension sur l'autre. Enfin, il peut y avoir une inversion des positions entre les objets des deux cartes. Afin de résoudre ce problème nous proposons de faire varier continûment le facteur d'échelle avec si nécessaire des déplacements rétrogrades.

Facteur d'échelle unique pour le lien global

Facteur d'échelle variable pour les liens entre objets



Synchronisation continue

Lorsqu'on a deux cartes liées, il va y avoir autant de facteurs d'échelle FE qu'il y a d'objets communs entre les deux cartes et d'intervalles entre les objets communs (figure 19).

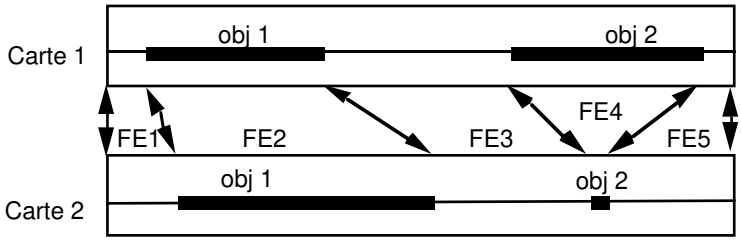


Figure 19 : Facteur d'échelle variable entre objets.

La solution consiste donc à définir, pour chaque couple de cartes Ci et Cj liées, une table dans laquelle chacune des cellules correspond à un interval commun, c'est-à-dire, soit un objet commun, soit un interval placé avant ou après un objet commun. Dans chacune des cellules de cette table on place le facteur d'échelle entre Ci et Cj correspondant au segment de séquence considéré. Cette table de correspondance doit être construite au moment où l'utilisateur déclare ou modifie les liens. Dans l'exemple précédent, la table de correspondance est la suivante :

Origine cartes	-	Début objet 1	FE 1
Début objet 1	-	Fin objet 1	FE 2
Fin objet 1	-	Début objet 2	FE 3
Début objet 2	-	Fin objet 2	FE 4
Fin objet 2	-	Fin cartes	FE 5

Dès lors que l'on dispose de cette table, l'algorithme de déplacement correspond à une généralisation de l'algorithme donné précédemment pour le lien global.

```
Procédure Lien_Objets_1
  Déplacer la carte maître M
  Pour chacune des cartes Ci liées
    Soit T la table contenant les facteurs d'échelle entre M et Ci
    • Si le décalage effectué dans M ne fait pas sortir de
      l'intervalle de T dans lequel on se trouvait avant le
      déplacement (dans ce cas FE est constant)
        Alors utiliser Lien_Global_1 avec le FE(T) entre M et Ci
    • Si le décalage effectué dans M fait changer d'intervalle dans T
      (dans ce cas on passe a un autre FE)
        Alors
          • Positionner la carte Ci par rapport au début du nouvel
            intervalle courant (avec une gestion éventuelle du CP)
          • Utiliser Lien_Global_1 pour se déplacer dans
            l'intervalle
```

Notons qu'il est très simple dans une carte de trouver la position "à l'affichage" du début de l'intervalle courant (qui correspond au début ou à la fin d'un objet) puisque celle-ci ne dépend que de la position réelle de l'objet considéré et de l'échelle propre à la carte considérée.

Nous terminerons l'explication de cette solution en fixant deux limites :

- Lorsqu'une une carte comporte beaucoup d'inversions de position, il devient alors difficile d'y suivre le déplacement.

- Lorsque plusieurs objets communs se trouvent à la même position, un choix arbitraire est réalisé pour déterminer celui sur lequel est effectué la synchronisation (en l'occurrence, l'objet qui en partant du bas de la carte se trouve à la position verticale la plus élevée).

2.3. Modification d'objets

Une des tâches principales d'un processus d'édition consiste à fournir la possibilité de modifier des objets interactivement puis à répercuter ces modifications dans une base de données. L'interface, qui est l'intermédiaire entre l'utilisateur et le système de gestion de bases de données, doit alors envoyer l'ordre de mise à jour à la base de données. Cette mise à jour peut alors nécessiter une validation préalable et inférer d'autres modifications importantes (remise en cause d'informations préalablement déduites, etc.)

Il s'agit là d'un problème non trivial à résoudre, c'est pourquoi nous ne l'avons pas considéré dans un premier temps, car nous voulions nous limiter à la fourniture d'une boîte à outils (*toolkit)* de visualisation et de confrontation de cartes génomiques.

Chapitre 3

APIC : Aide à la Présentation des Interfaces Cartographiques

Ce chapitre montre le résultat du travail qui a été réalisé. Il décrit l'architecture de la bibliothèque de classes d'objets qui a été créée pour répondre aux concepts qui ont été spécifiés dans le chapitre précédent.

La mise en œuvre de chacun des concepts est détaillée et illustrée sur des données réelles qui auront été présentées au préalable.

3.1. Une bibliothèque d'objets

À terme, l'interface cartographique doit se présenter sous trois formes (figure 20). Premièrement, celle d'une bibliothèque de fonctions de visualisation de séquences organisées sous la forme d'une API (Application Programming Interface) utilisables par les utilisateurs qui souhaitent construire une interface spécifique. Deuxièmement, celle d'une application autonome qui permettra de visualiser des informations contenues dans des fichiers externes décrits au format TINA-1 (Cf. Annexe 1). Troisièmement, dans le cadre du projet GREG, l'interface constitue le module permettant de visualiser la base de connaissances et d'interagir avec elle. Bien évidemment, ces deux dernières applications ne sont en fait qu'une mise en pratique des fonctions offertes par l'API.

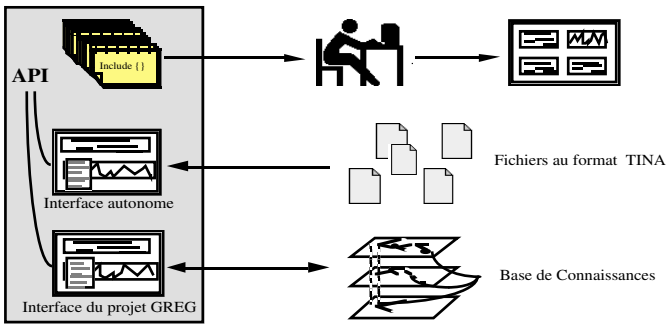


Figure 20 : les différentes déclinaisons d'APIC.

Actuellement, la seule version d'APIC implémentée est celle d'une application autonome qui permet de visualiser des informations contenues dans des fichiers décrits au format TINA-1.

Ces fichiers sont constitués d'une liste de couples descripteur/valeur dont la syntaxe est donnée en annexes (Cf. Annexe 1). De ce fait, lorsqu'une requête d'affichage est lancée par un outil d'analyse, cet outil doit générer un fichier décrivant les objets à représenter, puis ce fichier est interprété par l'interface graphique. Dans un deuxième temps, il faudra développer la version d'APIC qui doit être intégrée au projet GREG et pour cela, définir le protocole de communication entre l'interface et la base de connaissances.

3.2. Architecture

La conception d'une boîte à outils implique la prise en considération de plusieurs modèles [Voisard92] :

- *Le modèle de données du système qui gère les données* qui est le modèle du système de gestion de données sous-jacent.

- *Le modèle de données du domaine d'application* qui correspond à la description d'un domaine d'application particulier. Il peut s'agir par exemple du modèle qui est utilisé pour la gestion des connaissances sur la bactérie *Bacillus Subtilis* par l'Institut Pasteur. Ce modèle de l'application doit être traduit en utilisant le modèle du système de gestion des données sous-jacent. Par exemple, le modèle de l'Institut Pasteur est exprimé en utilisant le système de gestion de bases de connaissances SHIRKA de l'INRIA.

- *Le modèle d'édition* qui s'adresse au concepteur d'applications. Il comprend tous les objets nécessaires à l'affichage de cartes sur mesure (personnalisable), ainsi que des opérations permettant de les manipuler. Son implémentation nécessite l'utilisation d'outils de construction d'interface (dans notre cas la bibliothèque de classes d'objets graphiques ILOG Views de la société ILOG Software). A. Voisard, dans son étude d'un modèle d'édition de cartes vers une boîte à outils géographique, décompose le modèle d'édition en deux sous-modèles. Cette distinction permet de séparer les problèmes d'interface proprement dits des problèmes liés à la mise en correspondance entre les objets de la base et les objets de l'interface. Ces deux modèles sont :

    - *Le modèle des fenêtres* qui comprend les fenêtres dédiées à l'affichage de cartes qu'elle appelle MapWidgets ou MapGets.
    - *Le modèle spatial externe* qui englobe les objets qui sont affichés à l'intérieur des fenêtres. Ces objets proviennent d'une transformation des objets de la base de données.

La figure 21 résume les différents modèles et outils qui viennent d'être mentionnés, ainsi que leur connexion.

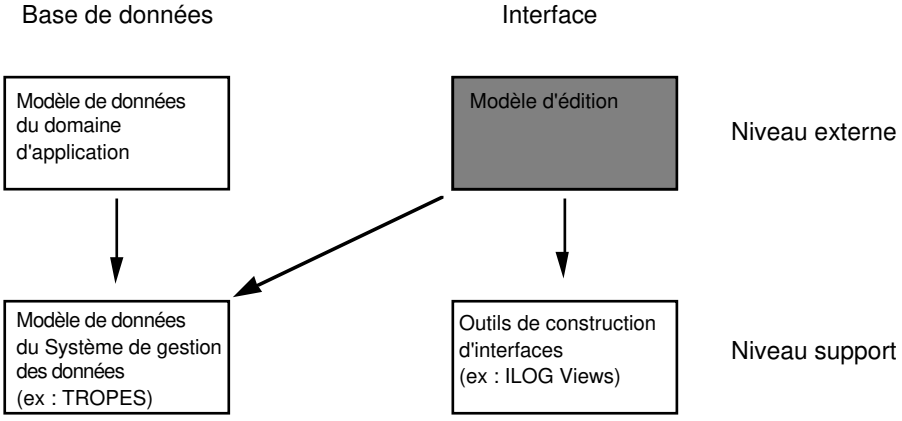


Figure 21 : Modèles et outils intervenants dans l'édition de cartes.

Modèle spatial externe

Compte tenu de la contrainte de concevoir une boîte à outils générique qui puisse être utilisée par différents projets de séquençage, il nous est apparu indispensable de définir un modèle spatial externe indépendant du modèle du domaine d'application.

La définition de ce modèle se réduit à deux classes d'objets : *ApicMap* et *ApicObject.*

La classe d'objets ApicMap est le moule dans lequel seront rangées les principales propriétés d'une carte (Cf. Annexe 1). On y trouvera entre autres : son nom, son unité, etc.

La classe d'objets ApicObject correspond, quant à elle, à la description du concept d'objet qui a été précédemment décrit dans le chapitre 2 (Cf. §2.2.3).

Correspondance avec les données externes

Une classe *ApicInterface* permet d'assurer la correspondance entre les objets externes (issus d'une base de données ou de simples fichiers) et les objets affichés dans une fenêtre. Ainsi, pour connecter l'interface à un autre système de gestion de données il suffit seulement de réimplémenter les opérations de cette classe.

Le modèle de fenêtres

Nous allons maintenant présenter le modèle de fenêtres défini pour l'interface APIC. C'est sur ce modèle que reposent les différentes fonctionnalités de l'interface (figure 22).

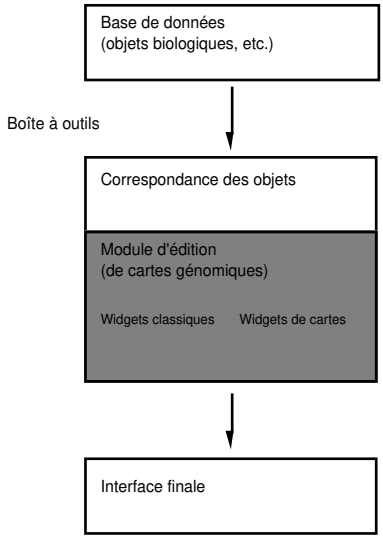


Figure 22 : Fonctionnalités de la boîte à outils.

Il est apparu naturel d'organiser l'interface de telle manière que chaque concept défini dans le chapitre 2 (plan de travail, carte ,couches, plans et liens) corresponde à une fenêtre et à une classe d'objets.

Le modèle de fenêtres de l'interface Apic se compose donc de plusieurs classes :

- ApicMultiMapget : c'est le *plan de travail* qui correspond au concept qui a été défini dans le chapitre 2 (Cf. §2.2.1).
- ApicMapGet : c'est la *carte* qui correspond au concept qui a été défini dans le chapitre 2 (Cf. §2.2.2).
- ApicArrange
ApicHook
ApicLayout
ApicDisplay
ApicSetting
ApicHelp : ce sont les *outils* qui correspondent aux concepts qui ont été décrits dans le chapitre 2.
- ApicMapBehavior : c'est une classe qui permet de décrire le *comportement d'une carte* (affichage des informations relatives à la carte).
- ApicObjectBehavior : c'est une classe qui permet de décrire le *comportement d'un objet* (affichage des informations relatives à l'objet).

Une description détaillée des classes de la bibliothèque de classes d'objets pour la construction d'interfaces cartographiques APIC est donnée en annexes (Cf. Annexe 6).

Limites

La bibliothèque de classes d'objets APIC a été conçue indépendamment de tout modèle de données d'un SGBD. Cette remarque conduit à fixer deux limites à son utilisation :

- Cette indépendance peut poser des problèmes de mise en correspondance d'objets appartenant à deux univers différents (ex : le langage Le-Lisp qui est la base du système de gestion de bases de connaissances SHIRKA de l'INRIA et le langage à objets C++).
- Elle est un frein à la mise en place d'un mécanisme de dépendances, qui pourrait ressembler au système MVC (Modèle-Vue-Controleur) proposé par le langage Smalltalk [Masini89], et être utilisé pour propager l'information de mise à jour d'un objet sur des vues différentes (dans notre cas des cartes) issues du même modèle.

3.3. Évaluation de l'interface

Avant de nous intéresser plus particulièrement aux fonctionnalités qu'offre la boîte à outils APIC, aux objets d'interaction et aux opérations sur ces objets; nous nous proposons, dans cette section, de présenter les données qui ont permis de tester ce qui a été réalisé.

Les données qui ont servi de jeu d'essai à la réalisation de cette interface sont issues de l'étude des bactéries *Bacillus Subtilis* et *Escherichia Coli*. Elles ont été fournies par l'institut Pasteur.

Ces données vont permettre d'illustrer deux objectifs importants dans le domaine de la cartographie génétique :

- Pour la bactérie *Bacillus Subtilis*, la confrontation de résultats obtenus par des méthodes différentes à partir d'un même fragment d'ADN issu du séquençage.
- Pour *Escherichia coli*, la confrontation de deux types de cartes : génétique et physique.

Nous pourrons alors vérifier que l'interface répond bien à ces besoins.

***Bacillus Subtilis*, la confrontation de résultats**

Les résultats qui seront montrés proviennent de trois méthodes d'analyse de séquences qui ont déjà été abordées dans la présentation du déroulement d'un projet de séquençage (Cf. §1.2), à savoir :

- La recherche d'homologies dans des banques de protéines : BlastX [Altschul90].
  Cette expérience a pour objectif de déterminer s'il existe dans un contig (grand fragment séquencé du génome obtenu par "raboutage" de plusieurs séquences) des régions déjà connues. En entrée le contig est donné, traduit dans les six phases. La méthode Blastx recherche alors dans une ou des banques de protéines la liste des protéines qui ressemblent le plus à certaines régions du contig. Les objets ainsi obtenus sont appelés des homologies. Cette méthode fournit éventuellement la fonction biologique identifiée de la protéine.

- La recherche de motifs.
  On va chercher à identifier des objets biologiques sur la séquence. Les objets biologiques qui concernent cette étape sont ceux qui ont été décrits dans le chapitre 1 (Cf. §1.1). On y retrouve donc les ORFs, les RBSs et les CDSs. Les ORFs (Open Reading Frame) sont des zones susceptibles de contenir un gène. Les RBSs (Ribosome Binding Site) ou site de fixation du ribosome sont des motifs proches de la séquence "AGGAGG". Les CDSs (CoDing Sequence) sont des régions de la séquence dont la lecture conduira à la fabrication de protéines. Pour cette expérience, les données en entrée sont la longueur minimale des ORFs, les motifs RBS et la distance entre le site de fixation du ribosome (RBS) et le signal du début de la synthèse de la protéine (codon START). Les données attendues sont bien sûr les ORFs, RBSs et CDSs.

- La prédiction de parties codantes.
  Il s'agira ici de montrer, sous forme de courbes, le résultat d'une des méthodes généralement employées dans la prédiction de parties codantes : GENMARK [Borodovsky93].

La comparaison de ces résultats différents doit permettre de faire ressortir des similarités ou des incohérences.

***Escherichia coli*, la confrontation de deux types de cartes : génétique et physique**

Il s'agira simplement de représenter des gènes issus du même organisme (*Escherichia Coli*) sous deux formes différentes : génétique et physique.

La carte physique donnera la position en Kilobases des gènes du chromosome de *E. Coli* (ceux qui sont précisément localisés sur le chromosome).

La carte génétique donnera, pour les mêmes gènes, la position en minutes. Dans ce cas, la détermination est expérimentale, et plusieurs gènes sont localisés sur la même minute.

Le chromosome de *E. Coli* est circulaire et de longueur 4720 KiloBases. Dans les expériences de génétique visant à déterminer la localisation des gènes, on interrompt un processus particulier d'échange de matériel génétique entre bactéries (la conjugaison) et on regarde quels sont les gènes qui sont passés entre les deux bactéries (une donneuse et une réceptrice). Ici c'est la notion de temps qui intervient et on donne la position en minute.

Le chromosome de *E. Coli*, dans cette unité va de 0 à 100 minutes, on a donc un rapport de 4720 KiloBases = 100 minutes.

On verra alors que l'on peut illustrer deux problèmes :

- Des erreurs de positionnement : des gènes ne sont pas placés au même endroit sur le chromosome dans les deux cartes, par exemple les gènes *ponB*, *hns*,et *ftsQ*.
- Des approximations : les positions génétiques sont sous-estimées entre 15 et 80 minutes.

3.4. Le plan de travail (MULTIMAP)

Lorsque l'on utilise l'interface APIC comme un outil autonome, la partie visible du système est le plan de travail (MultiMap) qui se présente comme une surface dans laquelle l'utilisateur peut charger une ou plusieurs cartes décrites dans des fichiers au format TINA-1. Lors du lancement d'APIC un plan de travail vierge est automatiquement créé.

Le plan de travail (figure 23), qui correspond au concept décrit dans le chapitre 2 (Cf. §2.2.1), comprend un panneau de commande qui permet de sélectionner des opérations concernant l'ensemble des cartes. On trouve les commandes suivantes :

Dans le menu MAP

- NEW : initialisation du plan de travail.
- ADD : chargement d'une carte (format TINA-1) dans le plan de travail.
- REMOVE : suppression d'une carte du plan de travail.

- ARRANGE : organisation des cartes dans le plan de travail.
- PRINT : imprime l'ensemble du plan de travail (non implémenté).
- POSTSCRIPT : sauve l'ensemble du plan de travail dans un fichier PostScript (non implémenté).
- QUIT : Permet de quitter le plan de travail (avec demande de confirmation)

Dans le menu TOOLS

- GOTO : énumération des objets et positionnement sur un objet particulier (non implémenté).
- FIND : recherche d'objets particuliers selon plusieurs critères (non implémenté).
- DISPLAY : sélection des informations à afficher pour les différents objets.
- SETTING : personnalisation de l'aspect des différents objets.
- LAYOUT : organisation en couches et en plans des différents objets.
- HOOK : synchronisation du défilement d'un ensemble de cartes.

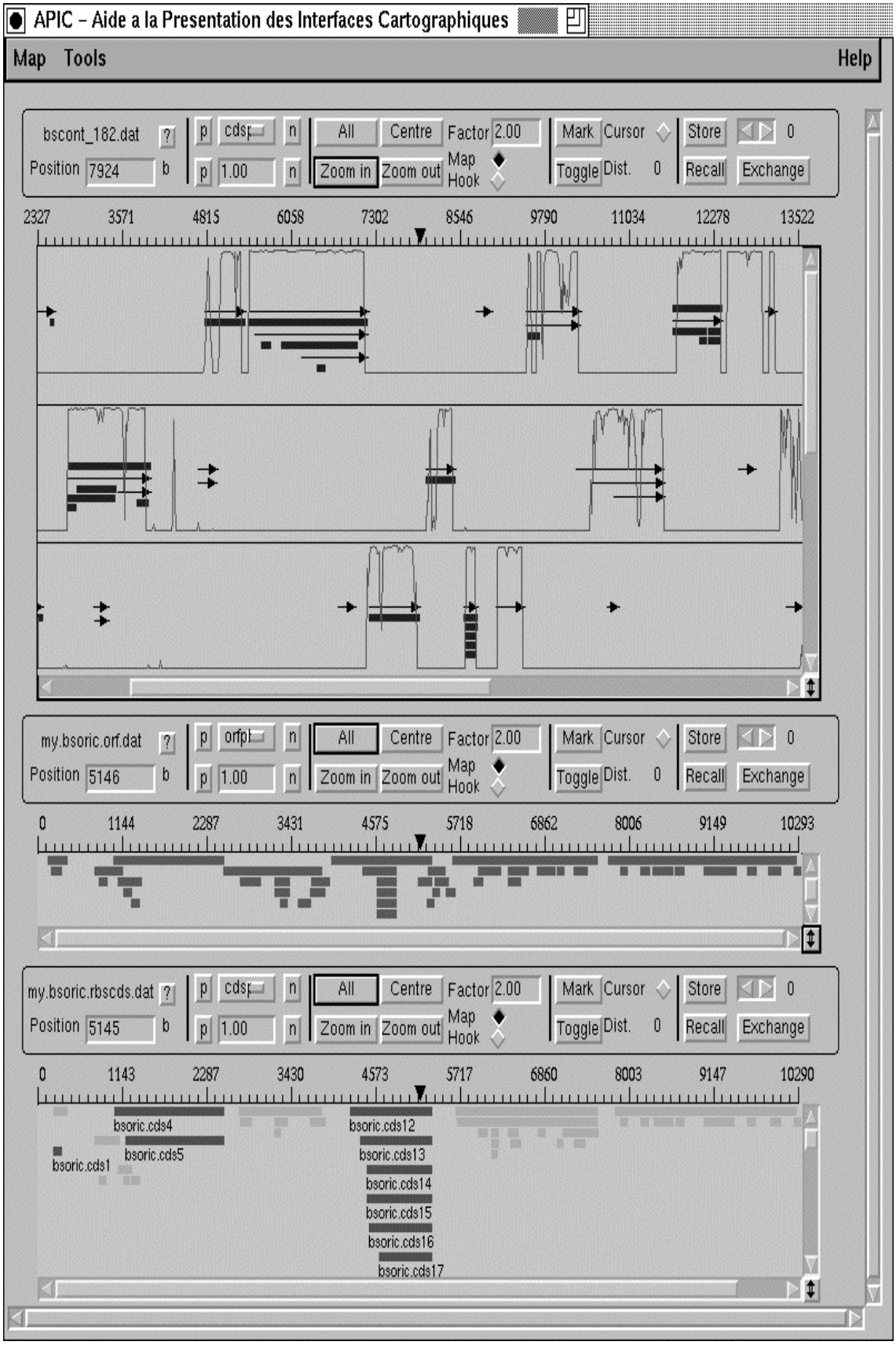
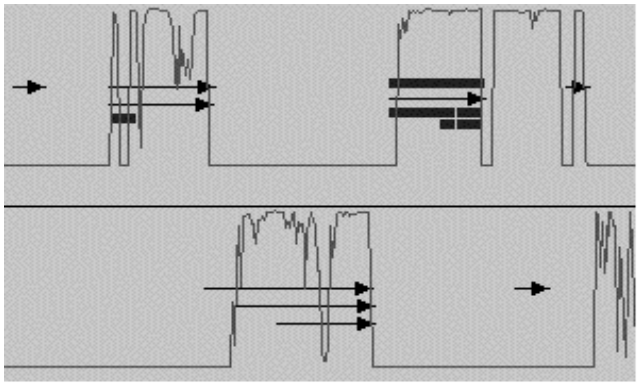
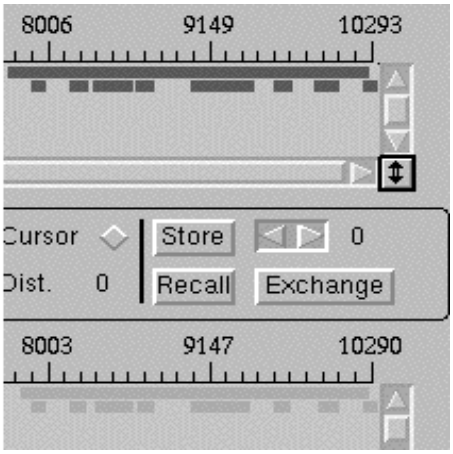


Figure 23 : Aspect général du plan de travail (MULTIMAP).

Nous pouvons, au passage, constater deux utilisations différentes de l'interface (figures 24-a et 24-b) :





24-a 24-b

Figures 24-a et 24-b : Deux utilisations différentes de l'interface.

Tout d'abord, dans la première carte qui résulte de la superposition de trois expériences (figure 24-a) :

- Les rectangles représentent les homologies trouvées dans les banques de données par la méthode BlastX.
- Les flèches représentent les CDS (CoDing Sequence).
- La courbe représente le résultat d'une méthode de prédiction de parties codantes : GENMARK.

Sur les deux phases montrées dans cette figure, on distingue bien qu'il manque des homologies BlastX dans des régions qui sont apparemment codantes (il y a des pics dans la courbe sans rectangle). Cette figure montre également d'étranges sites de début de traduction (flèches isolées).

Ensuite les deux dernières cartes (figure 24-b) qui résultent de l'étude d'un autre fragment séquencé sont en "pleine échelle". Elles montrent deux types de données qui intéressent particulièrement les biologistes :

- Les ORF (Open Reading Frame).
- Les couples RBS/CDS (Ribosome Binding Site et CoDing Sequence).

On distingue alors nettement dans cette figure, en bout de carte, une très grande similarité.

3.5. Organisation des cartes (ARRANGE)

Par ailleurs, chacune des cartes est composée de deux parties. D'une part, on trouve un panneau de commande (HEADER) permettant à la fois d'afficher des informations propres à la carte et de déclencher les fonctions courantes comme le zoom, ou le déplacement. D'autre part, on trouve la carte (MAP) proprement dite qui contient la représentation des séquences.

Les réglages de l'affichage et de la position d'une carte sur le plan de travail (MULTIMAP) sont modifiés à l'aide du menu MAP:ARRANGE. La boîte de dialogue de MAP:ARRANGE (figure 25) est composée d'un sélecteur de carte et de deux séries de boutons. Les boutons dans la zone POSITION permettent respectivement de faire passer la carte sélectionnée en première ou dernière position du plan de travail ou encore de faire progresser cette carte d'un cran vers le haut ou vers le bas dans la liste des cartes. Les composants dans la zone DISPLAY permettent à l'utilisateur de choisir les éléments de la carte (HEADER et MAP) qui sont visibles sur le plan de travail ainsi que le nombre de graduations (STEPS) qui doivent apparaître dans l'échelle placée au-dessus de la carte.

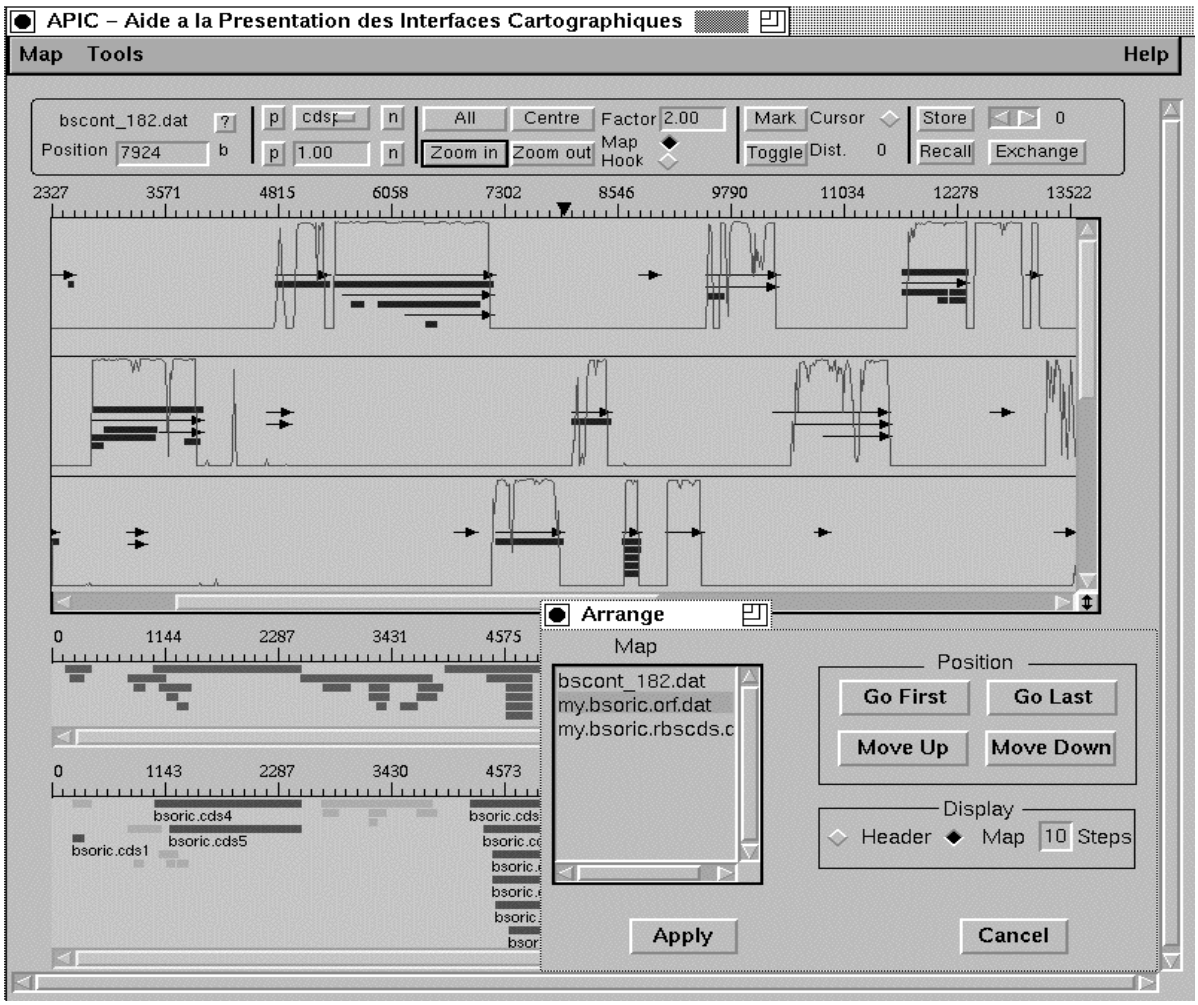


Figure 25 : Organisation des cartes dans le plan de travail (MultiMap).
Les panneaux de commande (HEADER) des deux dernières cartes ont été rendues invisibles.

Bien évidemment, lorsqu'un élément est rendu invisible on "tasse" toutes les fenêtres situées en dessous, ce qui permet de gagner de la place et facilite la comparaison des cartes entre elles. Du point de vue de l'interaction, on considère que la modification de la position d'une carte dans la boîte de dialogue ARRANGE a un impact immédiat sur la liste des cartes dans la boîte de dialogue, mais que les modifications de l'affichage sont prises en compte sur le plan de travail uniquement lors de la validation (bouton APPLY).

3.6. Description d'une carte (MAP)

Une carte permet de représenter les informations sur une séquence, ces informations étant transmises à l'interface APIC par l'intermédiaire d'un fichier au format TINA-1 (Cf. Annexe 1). La modification et la manipulation des objets d'une carte s'effectuent à l'aide des opérations (GOTO, FIND, DISPLAY, SETTING, LAYOUT, HOOK) qui sont présentes dans le menu TOOLS. Tous ces outils seront décrits dans les sections suivantes. Ainsi que nous l'avons déjà précisé, une carte est composée de deux parties différentes : un panneau de commande (HEADER) et la carte elle-même (MAP) que nous allons détailler.

3.6.1. Visualisation d'une carte

Une carte (figure 26), est dessinée dans une fenêtre munie d'ascenseurs verticaux et horizontaux. La hauteur d'une carte est redéfinissable par l'intermédiaire du bouton situé en bas à droite, à l'intersection des deux ascenseurs. La largeur d'une carte est fonction de la largeur du plan de travail (MULTIMAP). Au-dessus de la carte se trouve une échelle graduée (avec 10 jalons par défaut) qui indique la position de la fenêtre dans la séquence. Afin d'aider le biologiste à se positionner dans une carte, le curseur peut être représenté (Cf. §3.6.2) sous la forme d'une barre verticale. En outre, cette possibilité est tout à fait intéressante dans le cas de cartes synchronisées entre elles (Cf. §3.10) car ainsi, on peut visualiser plus aisément la correspondance entre les différentes cartes. La taille du tracé du curseur est constante (1 pixel) et indépendante du facteur de zoom. Si dans une carte le curseur est en dehors de la partie visible, le curseur devient invisible.

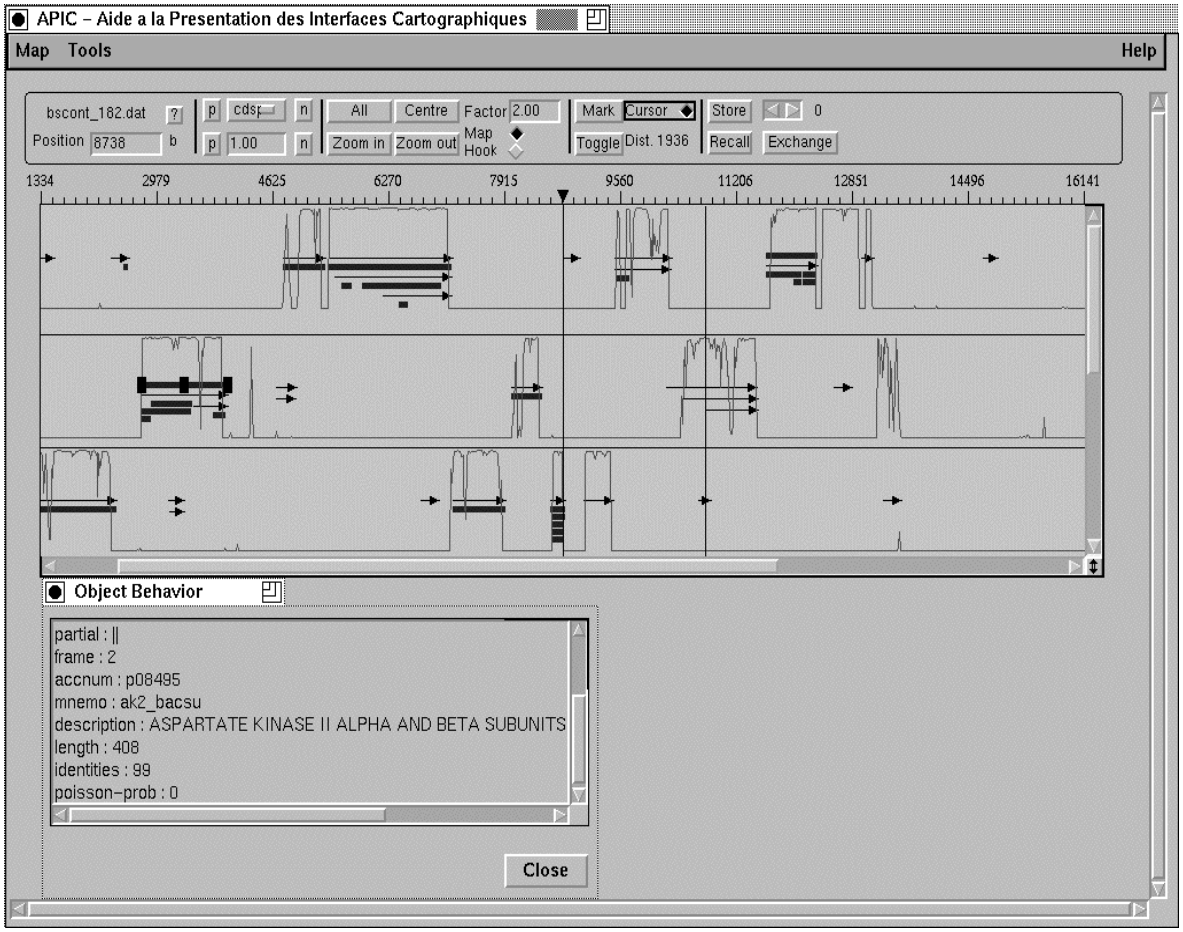


Figure 26 : Visualisation d'une carte.
Lorsque l'utilisateur clique sur un objet de la carte, une fenêtre ("Object Behavior") apparaît qui contient toutes les informations concernant cet objet. Ici l'objet sélectionné "ak2_bacsu" se trouve dans la deuxième phase.

Par ailleurs, l'utilisateur a la possibilité de poser une marque (MARK) dans une carte à une position spécifique à l'aide d'une barre verticale. Ainsi, lorsque l'utilisateur appuie sur la touche "M" à la position du curseur, une barre verticale permanente est dessinée dans la carte courante (et toutes les cartes liées à cette carte). Pour faire disparaître la marque de la carte courante (et des cartes liées) on utilisera la combinaison de touches "SHIFT + M". En corrélation avec la pose de ce marqueur, on affiche sur le panneau de commande et d'information de chacune des cartes (HEADER Cf. §3.6.2) la distance qui existe entre le marqueur et la position courante du curseur. Ainsi, l'utilisateur peut facilement mesurer la distance qui sépare deux objets.

Lorsque l'utilisateur clique sur un objet de la carte, une fenêtre apparaît qui contient toutes les informations concernant cet objet qui étaient présentes dans le fichier au format TINA-1 (Cf. Annexe 1). Par défaut, une fois une fenêtre ouverte, si l'utilisateur clique sur un autre objet de la carte, c'est la même fenêtre de visualisation qui est utilisée par le système. Si l'utilisateur souhaite ouvrir une seconde fenêtre, par exemple pour pouvoir comparer la définition de deux entités différentes, il doit appuyer sur la touche SHIFT avant de cliquer sur l'objet.

3.6.2. Contrôle d'une carte

Toutes les opérations courantes de manipulation, comme le zoom ou les déplacements, sont directement gérées à partir du panneau de commande (HEADER) qui est placée au-dessus de la carte. Le panneau de commande d'une carte (figure 27) est composé de cinq parties différentes.

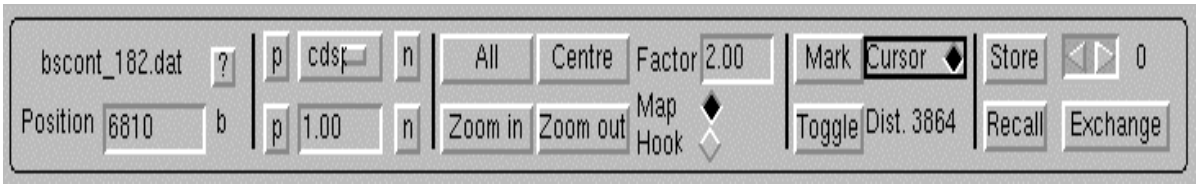


Figure 27 : Le bandeau de commande et d'information d'une carte.

La première partie du panneau de commande contient le nom de la carte ainsi qu'un bouton "?" qui permet de faire apparaître une fenêtre qui contient l'intégralité des informations qui sont présentes dans l'entête du fichier de description de la carte (Cf. Annexe 1). En-dessous du nom de la carte se trouve la position courante du curseur dans la carte exprimée dans l'unité de la carte. Il est important de noter que la *valeur de la position est éditable*; ainsi, en "cliquant" sur cette valeur l'utilisateur peut spécifier une autre position et, après l'avoir validée avec la touche RETURN, la carte courante est automatiquement centrée sur cette nouvelle position (il en est de même des cartes qui sont éventuellement liées à la carte courante).

La seconde partie du panneau de commande offre à l'utilisateur des possibilités de déplacement autres que celles, plus classiques, offertes par les ascenseurs de la fenêtre. La boîte centrale en haut correspond en fait à un menu qui contient l'ensemble des types d'objet qui apparaissent dans la carte courante. L'utilisateur peut ainsi sélectionner un type puis à l'aide des boutons ("p" pour previous et "n" pour next) situés de part et d'autre de ce menu, il peut se déplacer vers l'objet le plus proche qui appartient à ce type. Lors du déplacement, le centrage est effectué par rapport au début de l'objet si son orientation est directe (3'-5') et sur la fin de l'objet si elle est inverse (5'-3'). La boîte qui est placée sous le menu précédent est une zone éditable dans laquelle l'utilisateur peut écrire un nombre exprimant un pas de déplacement dans l'unité de la carte (bases, ...). Une fois ce pas déterminé, l'utilisateur peut déplacer la fenêtre vers la droite ou vers la gauche à l'aide des boutons situés de part et d'autre de la zone éditable.

La troisième partie du panneau de commande est celle qui permet de contrôler le facteur d'agrandissement appliqué aux cartes (ZOOM) et le centrage. Le bouton ALL met la carte en "pleine échelle", c'est-à-dire que le système calcule automatiquement le facteur de zoom qui permet de faire apparaître l'intégralité de la carte sur l'écran. Les boutons ZOOM IN et ZOOM OUT permettent respectivement de réduire et d'augmenter l'échelle de la carte. Enfin, le bouton CENTRE permet de centrer rapidement la carte sur une position donnée (au niveau de l'utilisation, le centrage peut être vue comme une sorte de zoom ayant un facteur de 1). Le ZOOM et le centrage fonctionnent de la manière suivante : dans un premier temps l'utilisateur "clique" sur un emplacement de la carte, puis il sélectionne l'action (ZOOM IN, OUT ou CENTRE) qu'il veut effectuer. Lors de l'exécution, le système centre alors la carte sur la position qui a été sélectionnée (Note : il serait possible d'étendre ce fonctionnement en donnant la possibilité à l'utilisateur de sélectionner une zone et d'effectuer le zoom sur celle-ci).

La position du “clic” souris est “oubliée” chaque fois que l’utilisateur sort de la carte courante (c’est-à-dire de la zone définie par la carte proprement dite et son panneau de commande). De ce fait, lors d’un ZOOM, si l’utilisateur n’a pas préalablement sélectionné de position, le ZOOM est automatiquement centré sur la position courante du Curseur de Position. La valeur de l’agrandissement ou de la réduction opéré par le ZOOM est déterminée par le facteur de modification d’échelle dont la valeur apparaît dans la zone éditable FACTOR. Lorsque l’utilisateur a modifié la valeur de FACTOR, celle-ci reste identique jusqu’à la modification suivante. Cette valeur correspond à un nombre réel. Du point de vue de l’implémentation, il est important de noter que le facteur d’échelle résultant est arrondi à un nombre entier. Dans le cas contraire on risquerait d’avoir des problèmes lors de l’affichage de séquences nucléotidiques. Par exemple, si l’on a un facteur de zoom qui est égal à 5.6 (1 base valant 5.6 pixels à l’écran), les lettres ne vont pas pouvoir être affichées avec un espacement régulier ; ce phénomène est encore plus gênant dans le cas de l’affichage coloré (Cf. §3.9) car il conduit à un affichage erroné. Par ailleurs, les deux boutons nommés MAP et HOOK permettent de spécifier le comportement du zoom. Lorsque l’option MAP est sélectionnée, le zoom ne porte que sur la carte courante; ainsi, même si la carte est synchronisée avec d’autres, l’affichage de ces dernières n’est pas modifié (au phénomène de recentrage des cartes près). Par contre, lorsque l’option HOOK est active, le système effectue un ZOOM sur l’ensemble des cartes liées à la carte courante. Le facteur de ZOOM qui est utilisé est celui indiqué dans la carte courante.

La quatrième partie du panneau de commande concerne la notion de marquage dans une carte (MARK) évoquée dans la section précédente §3.6.1. Une fois qu’une marque a été posée (à l’aide de la touche “M”) l’utilisateur peut centrer rapidement la carte sur cette marque en “cliquant” sur le bouton MARK. Lorsqu’il appuie sur le bouton TOGGLE, la carte est également positionnée sur la zone préalablement marquée, mais en plus, la marque est supprimée de sa position actuelle et repositionnée à l’endroit où était centrée la fenêtre courante avant que se produise le déplacement. Grâce à cette commande, l’utilisateur peut facilement et rapidement basculer entre deux positions sur une carte. Bien évidemment si la carte courante est synchronisée avec d’autres, la remise à jour des marques concerne l’ensemble des cartes synchronisées. La boîte de sélection étiquetée CURSOR permet de faire apparaître ou disparaître la barre verticale associée au curseur (Cf. §3.6.1). Enfin le champ marqué “Dist” donne la distance séparant la position actuelle du curseur de la position préalablement marquée (Cf. §3.6.1).

Enfin la dernière partie du panneau de commande permet à l’utilisateur de mémoriser des “vues” de la carte. Une vue est totalement définie par deux valeurs : d’une part, le facteur d’échelle, et d’autre part, la position dans la séquence. Ainsi, le bouton STORE permet de mémoriser ces deux valeurs dans la “mémoire” dont le numéro est affiché à droite. Le numéro d’une mémoire peut être modifié par l’utilisateur en utilisant les deux flèches ou en sélectionnant la zone et en y introduisant un chiffre compris entre 0 et 9. Lorsque des cartes sont liées, la position courante est mémorisée pour l’ensemble des cartes liées; par ailleurs, afin d’assurer la cohérence de la numérotation, le changement d’un numéro de mémoire est répercuté sur toutes les cartes liées. Lors du chargement d’une carte, la mémoire 0 est automatiquement initialisée avec le facteur d’échelle qui est indiqué dans le champ MAP-SCALE du fichier au format TINA-1 décrivant la carte et la position indiquée par le champ MAP-START (Cf. Annexe 1). Le bouton RECALL permet de réappliquer une vue précédemment mémorisée, c’est-à-dire de remettre la carte à la position et au facteur d’échelle contenu dans la mémoire.

Le bouton EXCHANGE fait de même, mais en plus le contenu de la mémoire est remplacé par la définition de la vue “avant modification”. Ainsi, comme pour la commande TOGGLE, précédemment décrite, l’utilisateur peut très facilement basculer d’une position vers une autre.

3.7. Gestion de l’affichage (DISPLAY)

Cette fonction permet à l’utilisateur de sélectionner de manière rapide à la fois les objets qui sont à afficher dans les cartes ainsi que les informations que l’on doit y attacher. La figure 28 représente la boîte de dialogue permettant de modifier les caractéristiques de l’affichage.

Tout d’abord, l’utilisateur peut sélectionner à l’aide du sélecteur de cartes (MAPS) de gauche le nom de la carte sur laquelle il souhaite effectuer des modifications. Toutefois, par l’intermédiaire des boutons situés dans la zone SPREAD, l’utilisateur peut modifier la “portée” des modifications qu’il va effectuer. Si l’utilisateur sélectionne le bouton ONE MAP, seule la carte qu’il vient de sélectionner est concernée. S’il sélectionne le bouton THE HOOK, toutes les cartes qui sont liées à la carte sélectionnée sont concernées. Enfin le bouton ALL THE MAPS permet de faire porter les modifications sur l’ensemble des cartes du plan de travail. Le sélecteur (TYPES) placé au centre de la boîte permet de sélectionner le type d’objet sur lequel l’utilisateur désire travailler ; cette liste contient tous les types d’objets de la carte sélectionnée.

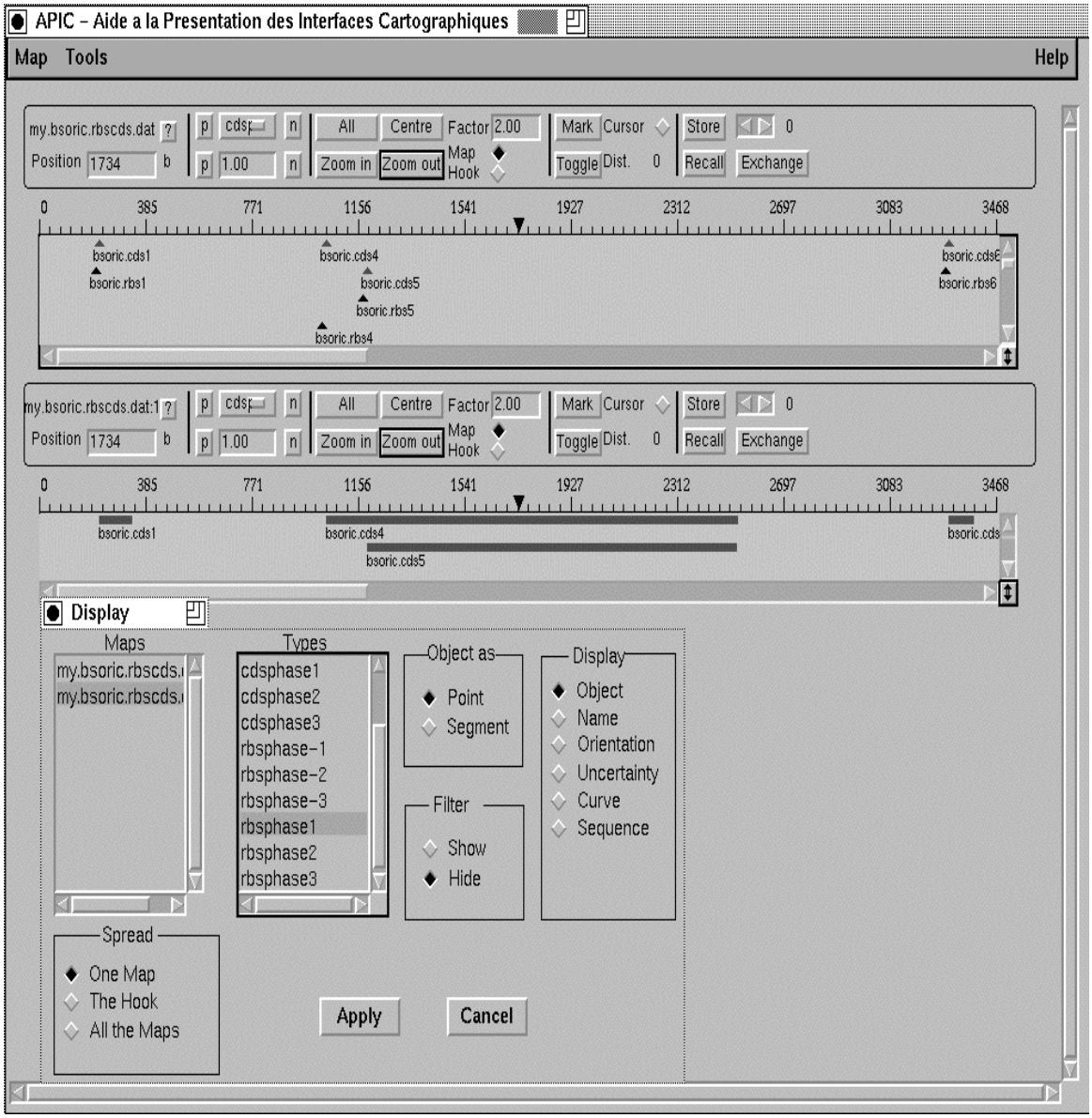


Figure 28: Modification de l'affichage des types d'objets.
La même carte est représentée selon deux vues différentes (partie supérieure de la figure). Dans la première, les CDSs de la première phase sont représentés de manière ponctuelle. Alors que dans la deuxième, ils sont représentés avec leur véritable dimension. De plus sur la deuxième vue, on a caché (Hide) les RBSs de la première phase.

La zone FILTER contient deux boutons. Lorsque l'on sélectionne le bouton HIDE tous les objets appartenant au type sélectionné sont cachés dans toutes les cartes qui ont été retenues par l'utilisateur. Précisons que cette opération consiste à ôter les objets de l'affichage sans pour autant provoquer une réorganisation des cartes. D'un point de vue technique, cette opération est implémentée à l'aide des couches (layers) de ILOG Views. Lorsque l'on sélectionne le bouton SHOW, l'opération inverse se produit et tous les objets précédemment cachés réapparaissent. Lors du chargement d'une carte tous les types d'objets sont par défaut visibles sauf si le fichier de ressources (Cf. §3.13) contient une indication contraire.

La zone SHOW AS contient deux boutons permettant de choisir la manière dont les types d'objets sont représentés sur la carte. L'option POINT signifie que l'objet va toujours être considéré comme étant ponctuel ; dès lors, on le représente par un graphique (par défaut un triangle) dont la taille ne varie ni en fonction de la taille réelle de l'objet, ni en fonction du facteur de zoom. L'option SEGMENT indique que l'objet doit être représenté comme une boîte dont la longueur est fonction de la taille de l'objet et du facteur de zoom qui est utilisé.

La zone DISPLAY permet de contrôler les informations qui doivent être attachées aux objets. Cette zone contient six boutons sélectionnables permettant respectivement de montrer et de cacher : les objets (OBJECT), le nom des objets (NAME), l'orientation des objets (ORIENTATION), les incertitudes de position (UNCERTAINTY), la courbe associée à l'objet (CURVE) et la séquence nucléotidique (SEQUENCE) si elle existe et que le facteur d'agrandissement permet de la distinguer (résolution supérieure à 5 pixels par base). Notons que la manière dont ces différentes informations sont affichées dépend des choix de personnalisation de l'affichage qui ont été effectués (Cf. §3.9). Si l'utilisateur n'active aucun de ces boutons, les types d'objets sélectionnés n'apparaîtront plus dans la carte. Cette disparition est de "nature" différente de celle provoquée par le champ FILTER : dans FILTER les objets sont momentanément "cachés" alors qu'ici ils disparaissent totalement de la carte.

Enfin, du point de vue de l'interaction, seules les commandes de masquage/visualisation de la zone DISPLAY ont une action qui est immédiate sur l'affichage des cartes. Pour les autres commandes les modifications ne sont prises en compte que lors de la commande APPLY.

3.8. Disposition des objets sur la carte (LAYOUT)

Ainsi que nous l'avons décrit dans la section 2.2.4, l'un des avantages offerts par APIC est de permettre de structurer l'affichage des divers objets dans la carte. Cette structuration est effectuée au moyen du concept de "couche" qui découpe la carte en plusieurs bandes horizontales et de "plan" qui définit la superposition des objets sur la carte. Une boîte de dialogue (figure 29) permet à l'utilisateur de configurer ces deux aspects simultanément.

La partie supérieure gauche de la boîte de dialogue fonctionne de manière identique à ce que nous avons vu dans la boîte de paramétrage de l'affichage (Cf. §3.7) des objets biologiques. L'utilisateur peut sélectionner une carte à l'aide d'un sélecteur (MAPS), puis définir à l'aide de la zone SPREAD quelles sont les cartes concernées par les modifications qu'il souhaite effectuer : la carte sélectionnée, les cartes qui sont liées à la carte sélectionnée, toutes les cartes du plan de travail. Le sélecteur TYPES contient les types de la carte sélectionnée.

L'utilisateur peut alors choisir un numéro de couche (LAYER) compris entre 1 et 9 ; ce numéro est modifiable en cliquant sur les flèches placées de part et d'autre ou en le tapant directement. Le sélecteur de droite contient l'ensemble des types qui apparaissent dans la couche sélectionnée (nous l'appellerons la couche courante). Comme dans le cas du sélecteur de types, le sélecteur de couches ne visualise que les types qui apparaissent dans la carte sélectionnée pour la couche courante. Lorsque l'utilisateur sélectionne un type dans le sélecteur de types (TYPES), puis appuie sur le bouton ADD, le type choisi est ajouté dans la couche courante (mais sans disparaître de TYPES). Comme un type ne peut apparaître que dans une seule couche à la fois, lorsque l'utilisateur veut changer un type de couche, il a juste à se positionner sur la nouvelle couche, sélectionner le type et cliquer sur le bouton ADD.

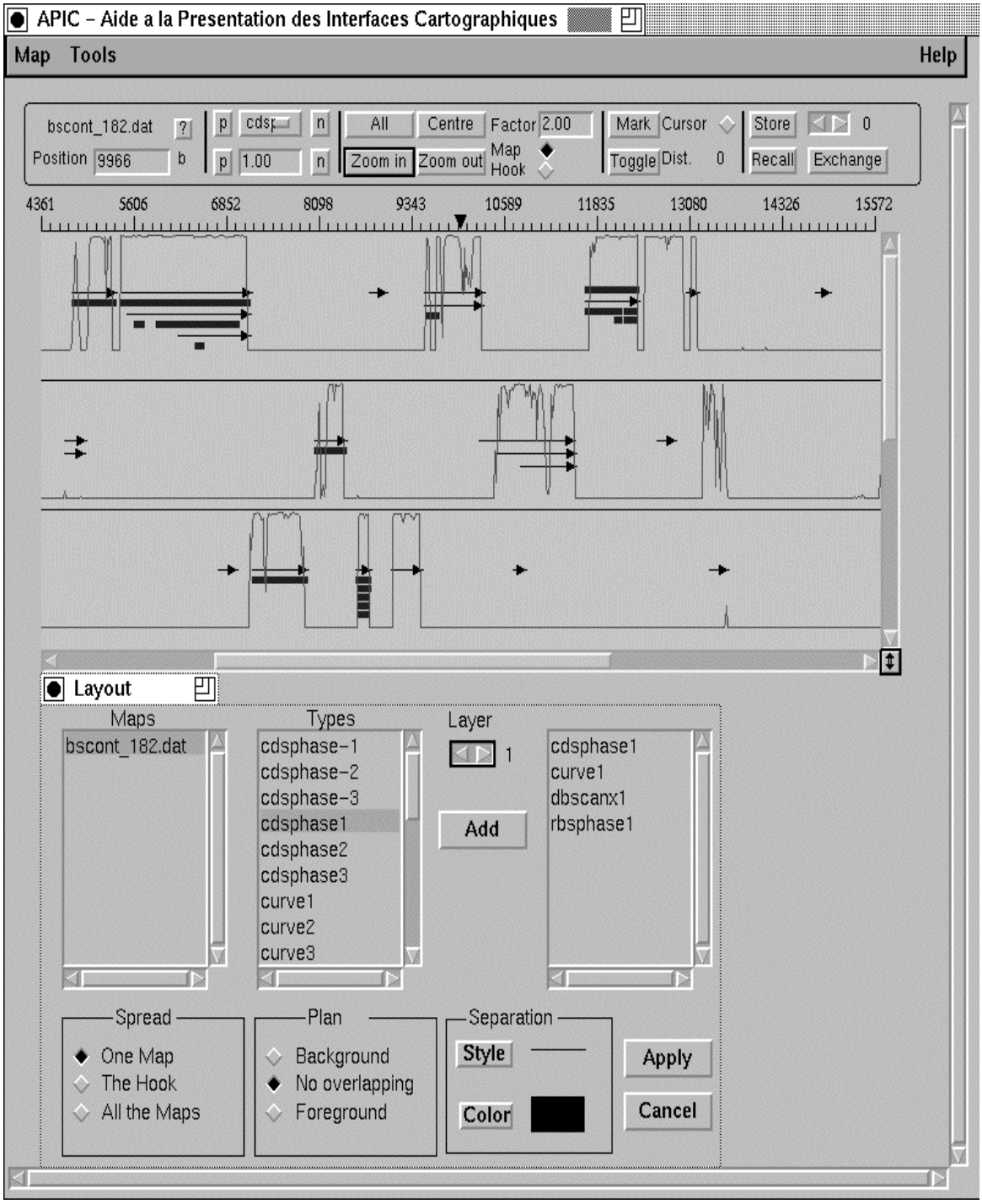


Figure 29 : Affectation des types d'objets aux différentes couches.
La boîte de dialogue LAYOUT.

Du fait du temps nécessaire pour réafficher les couches, la modification de l'affichage n'est prise en compte que lors de la validation par APPLY. Notons que l'interface dessine toutes les couches comprises entre 1 et la dernière contenant un type d'objet. Par exemple, si l'on a des types dans les couches 1 et 3, la couche 2 bien que vide est tout de même dessinée. Sa présence sera (éventuellement) matérialisée par un trait de séparation.

La zone PLAN permet de définir le plan auquel appartient un type d'objet. Les trois boutons (BACKGROUND, NO OVERLAPPING et FOREGROUND) permettent respectivement de placer le type d'objet sélectionné soit en fond de carte sans positionnement automatique (BACKGROUND), soit de le positionner à l'aide de la Procédure de Positionnement Automatique (NO OVERLAPPING), soit enfin de le mettre au premier plan de la carte sans positionnement automatique (FOREGROUND). Nous ne reviendrons pas sur les conséquences de ce choix qui ont été exposées dans la section 2.2.4.

Enfin, la zone SEPARATION placée en-dessous du multi-sélecteur de couche permet à l'utilisateur de séparer la couche courante (N) de la couche qui lui est supérieure (N+1) à l'aide d'un trait dont l'aspect et la couleur sont modifiables à l'aide de deux menus STYLE et COULEUR. Par exemple, on peut utiliser les styles suivant : None ------- ——— ═══.

3.9. Paramétrisation de l'affichage (SETTING)

APIC permet de personnaliser le graphisme utilisé pour afficher les différents éléments d'une carte. Ainsi, en fonction de ses préférences et du type de carte qu'il doit manipuler, l'utilisateur peut aisément paramètrer l'affichage de manière à mettre en valeur les informations sur lesquelles il travaille. Ce contrôle s'effectue à l'aide d'une boîte de dialogue (figure 30).

La partie gauche (champs MAPS, SPREAD et TYPES) de la boîte de dialogue permet, comme dans le cas des commandes de gestion de l'affichage et la disposition des objets (Cf. §3.7 et §3.8), de sélectionner la carte et le type d'objet sur lesquels on travaille. La zone OBJECT contrôle l'aspect de l'objet graphique qui est associé à l'objet biologique. Ainsi que nous l'avons vu dans la section 3.7, un objet peut être représenté sous une forme ponctuelle ou sous la forme d'un segment. L'utilisateur peut sélectionner au moyen de deux menus (POINT, SEGMENT) le graphisme qui sera utilisé pour représenter ces deux formes ; par défaut, les objets ponctuels sont représentés par des triangles et les segments par des rectangles pleins. Le menu COLOR sert à définir la couleur utilisée pour dessiner l'objet. Le champ éditable HIGH permet de définir la hauteur par défaut d'un objet exprimée en nombre de pixels. Au cas où le type d'objet sélectionné représente une courbe, ce champ permet à l'utilisateur d'agrandir l'échelle verticale de cette courbe. Enfin, le champ éditable OFFSET permet de contrôler l'écartement vertical entre les objets.

Les zones NAME, ORIENTATION, UNCERTAINTY et CURVE permettent de personnaliser la façon dont ces différents éléments sont affichés à l'aide de menus. Ainsi, l'utilisateur peut modifier la couleur ainsi que le graphique (ou la police de caractères dans le cas du nom) qui représente l'information.

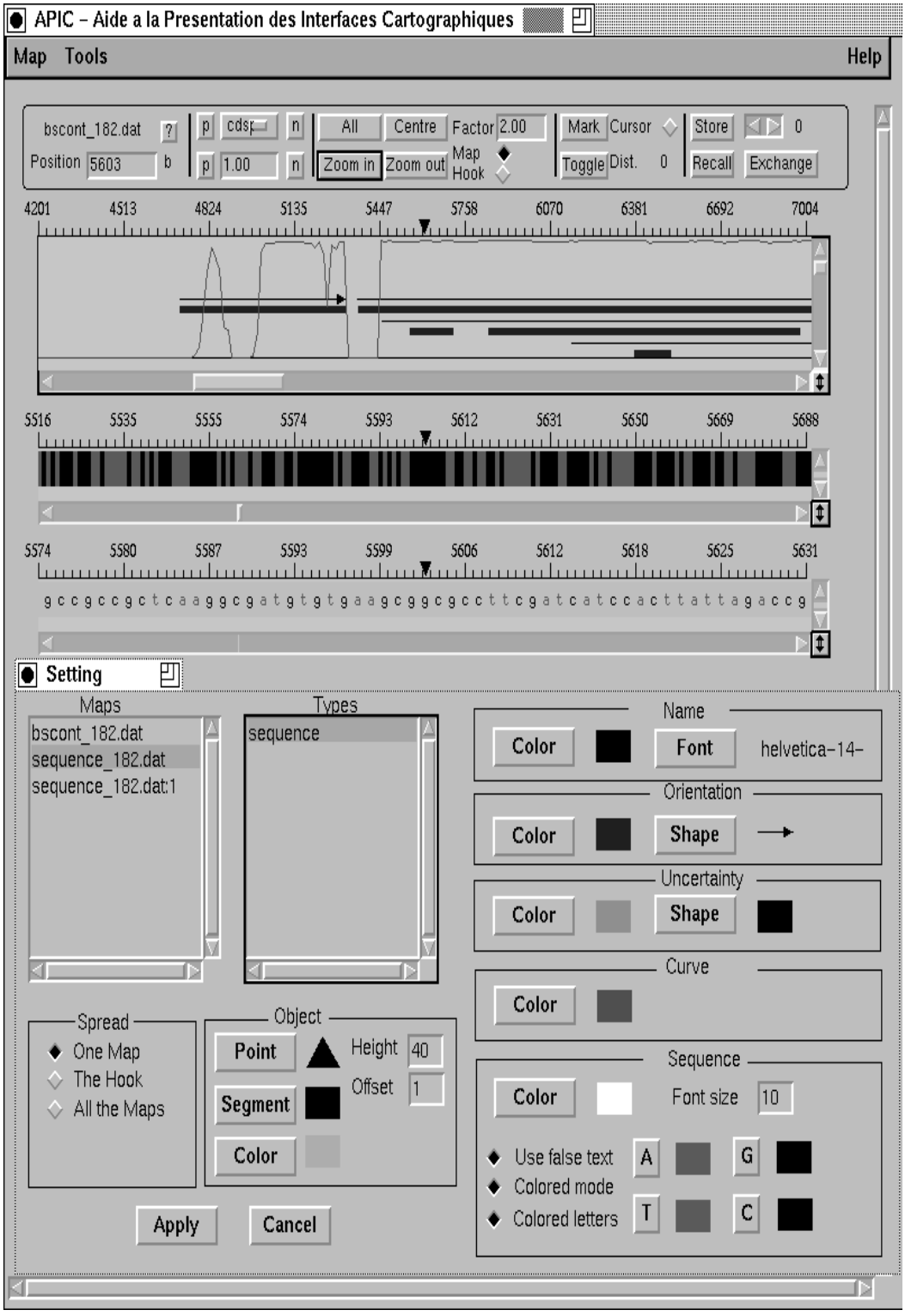


Figure 30 : Personnalisation de l'affichage des objets.
Les deux dernières cartes représentent une séquence. Dans la première de ces deux cartes, les bases sont représentées par des traits de couleur qui font ressortir distinctement deux groupes de bases : les bases G et C en noir et les bases A et T dans une couleur plus claire. Dans la dernière carte, on visualise les bases en toutes lettres et en couleur. Les trois cartes de cette figure ont été "accrochées" (Cf. §2.2.5), ainsi les centres de ces cartes se correspondent et leur déplacement est synchronisé.

Enfin, grâce à la zone SEQUENCE l'utilisateur peut personnaliser la façon dont les séquences sont affichées en fonction du degré d'agrandissement de la carte. Toutefois, lorsqu'une séquence est associée à un objet, il est clair que les nucléotides qui la composent ne peuvent être visualisés que si l'échelle de la carte le permet. La valeur contenue dans le champ éditable FONT SIZE indique la taille minimale de la police représentant les nucléotides (par défaut 5 points). En d'autres termes, les nucléotides qui composent la séquence ne seront affichés que lorsque la résolution sera supérieure ou égale à la taille minimale de la police. Ainsi, on n'affiche en clair les séquences que lorsque chaque base représente au moins 5 pixels sur l'écran. Il est nécessaire que le zoom arrondisse toujours le facteur d'échelle à l'entier le plus proche afin d'éviter les tailles de police fractionnaires qui introduisent des décalages gênants lors de l'affichage.

Par ailleurs, afin de pouvoir visualiser les nucléotides même lorsque l'échelle courante ne permet pas de dessiner les lettres, APIC propose un mode de visualisation original dans lequel les quatre nucléotides sont codés par des couleurs qui sont modifiables à l'aide de quatre menus respectivement notés A, T, G, C. Dans ce mode, chaque base est représentée à l'écran par un trait de la couleur choisi. Bien évidemment, ce mode de visualisation n'est applicable que si la résolution courante est d'au moins 1 pixel par base. Ce mode de visualisation est très intéressant car il permet d'avoir une idée rapide de la composition des séquences : par exemple, si l'on colore les nucléotides G et C en noir et les nucléotides A et T de la même couleur que l'objet, on peut comparer de façon visuelle les pourcentages en G et C de deux séquences différentes. En dessous d'une résolution de 1 pixel par base, la présence d'une séquence est représentée à l'aide d'un simple graphique.

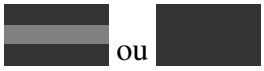
	USE FALSE TEXT Résol. [0..1[Pixel	COLORED MODE Résol. [1..5[Pixels	COLORED LETTERS Réso. [5..∞[Pixels
Case active			
Case inactive			

Figure 31 : Personnalisation de l'affichage des séquences.

La sélection des différents modes d'affichage de la séquence en fonction de la résolution est contrôlée à l'aide de trois cases sélectionnables qui sont placées à gauche de la zone SEQUENCE. La figure 31 montre les résultats obtenus en fonction des options d'affichage qui ont été choisies. La case USE FALSE TEXT permet de choisir si la présence d'une séquence dans un objet doit être visualisée ou non sous forme d'un mince rectangle lorsque la résolution de l'écran est inférieure à 1 pixel par base. La couleur de ce rectangle est définie par la valeur du champ COLOR. La case COLORED MODE indique si l'interface doit utiliser l'affichage des séquences sous la forme du code de couleurs lorsque la résolution est comprise entre 1 pixel par base et celle à partir de laquelle on représente les nucléotides sous la forme de lettres (5 pixels par base). Enfin la case COLORED LETTERS permet d'indiquer au système s'il doit conserver pour afficher les lettres correspondant aux nucléotides le code de couleurs précédemment défini ou si, a contrario, il doit utiliser pour toutes les lettres la couleur indiquée dans le champ COLOR.

3.10. Synchronisation des cartes (HOOK)

L'une des fonctions importantes qu'offre l'interface APIC est la possibilité de synchroniser le décalage horizontal entre les cartes à l'aide de la fonction "d'accrochage" des cartes (HOOK). Cette fonction permet le déplacement synchrone de plusieurs fenêtres de visualisation sur la séquence à l'aide de l'ascenseur ou des boutons situés dans le panneau de commande (HEADER) de la carte.

La déclaration d'un lien s'effectue à l'aide d'une boîte de dialogue (figure 32). L'utilisateur doit sélectionner deux cartes ou plus à l'aide du multi-sélecteur de droite (ALL MAP) et choisit un numéro de lien (entre 1 et 9). Lorsque l'utilisateur clique sur le bouton ← les cartes apparaissent dans la liste (non éditable) de gauche mais sans disparaître du sélecteur de droite. Comme une carte ne peut appartenir à deux liens en même temps, le fait de placer une carte dans un lien la supprime du lien dont elle faisait éventuellement partie. Lorsque l'utilisateur veut supprimer le lien existant entre plusieurs cartes, il a juste à sélectionner le numéro du lien et cliquer sur le bouton →. Enfin, le bouton sélectionnable OBJECT SYNCHRONIZING permet de signaler à l'interface que les déplacements dans le lien courant doivent se synchroniser sur les objets qui sont communs à l'ensemble des cartes qui participent au lien.

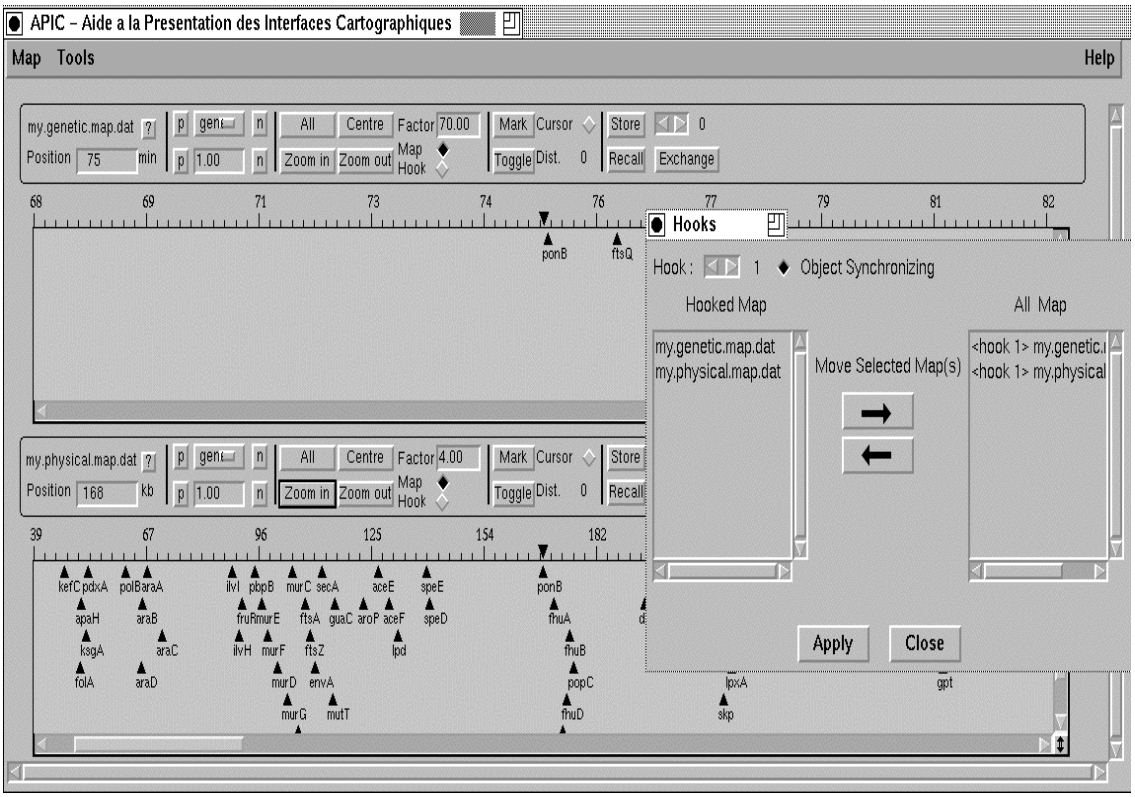


Figure 32 : Lien entre une carte génétique et une carte physique sur les objets communs.
Le déplacement dans l'une des cartes sur le gène ponB a resynchronisé l'autre carte sur le même gène. On constate alors une erreur de positionnement sur la carte génétique. En effet la carte génétique indique que le gène ponB se situe à 75 minutes alors que la carte physique le situe à 166 Kilobases et qu'il devrait donc se situer comme l'indique la figure 33 entre 3 et 4 minutes.

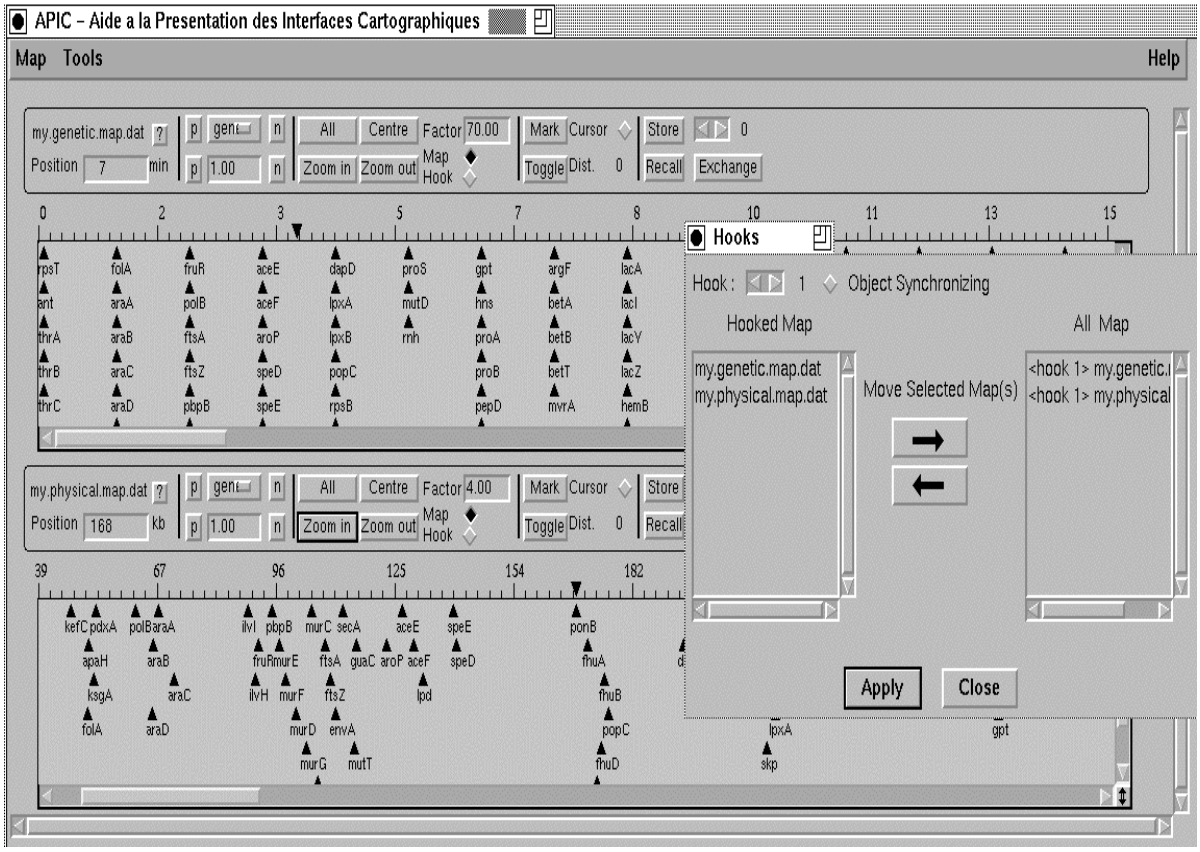


Figure 33 : Lien global entre une carte génétique et une carte physique. Le déplacement dans ces deux cartes est synchronisé en fonction du rapport qui existe entre les unités de ces cartes. Le chromosome de *E. Coli* est circulaire et de longueur 4720 KiloBases. La carte génétique va de 0 à 100 minutes, on a donc un rapport de 4720 KiloBases = 100 minutes.

3.11. Positionnement sur un objet (GOTO)

Cet outil n'est pas implémenté, mais nous en donnons ici une description qui pourra servir de base à une implémentation future.

La fonction de positionnement permet à l'utilisateur de positionner une carte (ou plusieurs cartes si l'on travaille avec des cartes liées) sur un objet de la séquence ayant un nom précis. En outre, l'utilisateur peut se servir de cette fonction simplement pour recenser quels sont les objets d'un type donné. Pour ce faire, le système propose à l'utilisateur la boîte de dialogue suivante (figure 34).

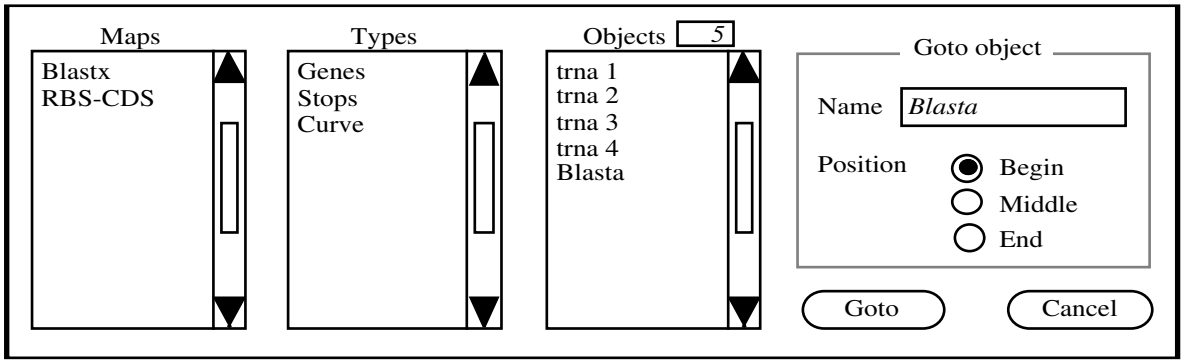


Figure 34 : Positionnement sur un objet.

Dans cette boîte, l'utilisateur sélectionne tout d'abord la carte sur laquelle il veut faire le positionnement à l'aide du sélecteur Maps. Il sélectionne ensuite, toujours à l'aide d'un sélecteur, le type d'objet sur lequel il veut travailler; les types qui apparaissent dans cette liste sont ceux de la carte sélectionnée. Le dernier sélecteur contient le nom de tous les objets qui apparaissent dans la carte sélectionnée pour le type d'objet sélectionné. Notons qu'en haut de ce sélecteur un champ non éditable indique le nombre d'objets dans la liste. L'utilisation de deux sélecteurs indépendants pour le type et le nom des objets permet d'avoir des sélecteurs contenant moins d'éléments.

Le champ NAME sur la droite est éditable et permet à l'utilisateur d'introduire directement le nom de l'objet sur lequel il souhaite se positionner plutôt que le choisir dans le sélecteur de droite. Si après validation par la touche RETURN ou sélection du bouton GOTO le nom donné par l'utilisateur n'est pas correct (c'est-à-dire qu'il n'apparaît pas dans la liste du sélecteur d'objets), le système manifeste l'erreur en faisant passer le nom en inverse vidéo. Enfin, l'utilisateur peut choisir si le centrage doit s'effectuer par rapport au début, au milieu ou à la fin de l'objet sélectionné grâce à une série de boutons.

3.12. Recherche d'un objet (FIND)

Cet outil n'est pas implémenté, mais nous en donnons également une description.

Cette commande permet de rechercher dans une carte un ou plusieurs objets, vérifiant une série de critères (position de l'objet, taille de l'objet, séquence nucléotidique,...). Contrairement à la fonction précédente GOTO, la recherche ne repose donc pas uniquement sur le nom de l'objet. L'interaction s'effectue à l'aide de la boîte de dialogue suivante (figure 35).

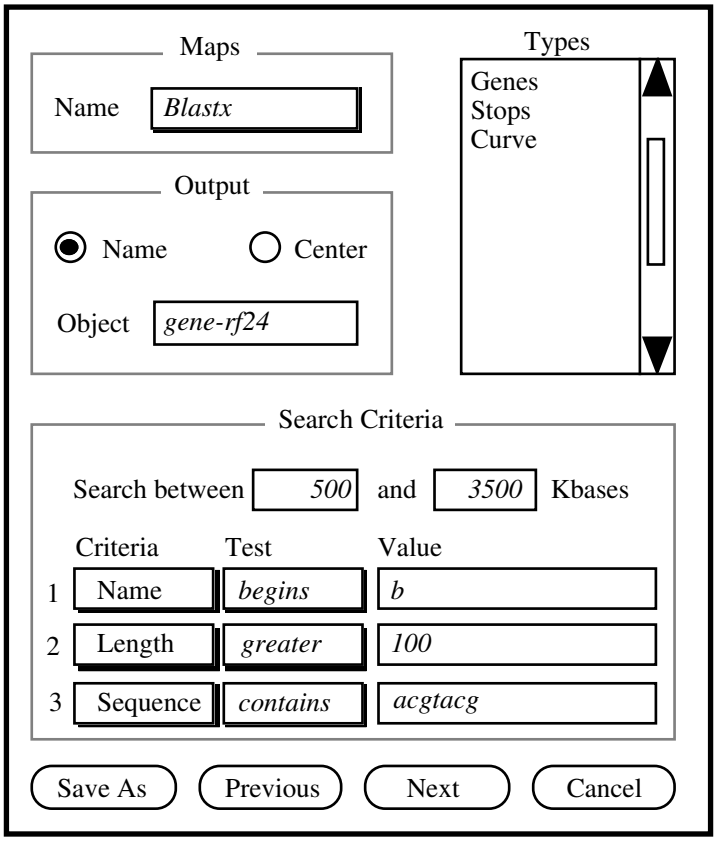


Figure 35 : Recherche d'un objet dans une carte.

Dans un premier temps, l'utilisateur sélectionne à l'aide d'un menu (MAPS) la carte dans laquelle il veut effectuer la recherche, puis le ou les types d'objet sur lesquels porte cette recherche. Il lui faut alors définir les critères qui l'intéressent. Grâce aux deux champs éditables SEARCH BETWEEN et AND, l'utilisateur peut préciser la zone de la séquence, exprimée dans l'unité de la carte sélectionnée, dans laquelle doit être effectuée la recherche. Par défaut, le système donne initialement comme bornes de la recherche les valeurs minimale et maximale de la séquence. L'utilisateur a ensuite la possibilité de définir jusqu'à trois critères de recherche. Chaque critère est composé d'un triplet : nom du critère, test à effectuer, valeur à tester.

Le champ CRITERIA correspond à un menu qui contient les différents champs que l'on trouve dans le fichier de description TINA des objets (TYPE, NAME, BEGIN, END, SEQUENCE, ...), et en particulier les champs spécifiques qui ont été introduits par l'utilisateur et qui ont une signification spécifiquement biologique. Cependant, deux champs du fichier n'apparaissent pas dans ce menu car ils reflètent des critères propres à l'interface, ce sont les champs : POSITION et CURVE. Par contre, une valeur supplémentaire est ajoutée, c'est la valeur NONE qui permet à l'utilisateur de signaler qu'il n'utilise pas ce critère.

Le second champ TEST est également un menu déroulant; il permet de sélectionner le type de test à effectuer. Les tests possibles sont les suivants : CONTAINS, BEGINS, EQUAL, GREATER, LESS ; les deux derniers tests ne concernent que les champs contenant une valeur numérique; toutefois comme le système n'a pas la possibilité de connaître le type des champs, les valeurs de test sont toujours proposées et le contrôle de type n'est effectué que lors de l'exécution de la requête. Enfin, le dernier champ est une zone éditable dans laquelle l'utilisateur peut introduire la valeur de référence.

Une fois que les critères de recherche sont définis, l'utilisateur peut sélectionner le type de sortie qu'il souhaite obtenir à l'aide des boutons NAME et CENTER. S'il sélectionne NAME, chaque fois qu'un objet satisfaisant les critères indiqués est trouvé, son nom est affiché dans le champ OBJECT de la zone OUTPUT. Par contre s'il sélectionne le bouton CENTER, chaque fois qu'un objet est trouvé, on centre automatiquement la carte dans laquelle se trouve cet objet (ainsi que les cartes qui sont éventuellement liées à cette carte).

En bas de la fenêtre, les boutons PREVIOUS et NEXT permettent respectivement de se déplacer d'objet en objet lors de la recherche. Notons, pour des raisons de cohérence, que la recherche reprend automatiquement à partir du début de la séquence à chaque fois qu'un critère de recherche est modifié. Le bouton SAVE AS permet de sauver dans un fichier au format TINA-1 l'ensemble des objets qui satisfont les critères de recherche. Dans ce fichier, MAP-DESCRIPTION devra contenir l'énoncé des critères de recherche en plus des informations qui s'y trouvaient déjà lors du chargement de la carte. Cette possibilité de sauver le résultat d'une recherche est tout à fait pratique car ainsi, le biologiste peut facilement extraire les informations qui l'intéressent et réafficher la carte ainsi "simplifiée" dans une autre fenêtre de APIC.

3.13. Gestion des ressources

Lorsque l'on est amené à travailler régulièrement sur les mêmes types de cartes, il est utile de pouvoir sauver et recharger les configurations d'affichage (appelées "Ressources") qui ont été établies à l'aide des fonctions dans DISPLAY, LAYOUT et SETTING (sections §3.7 §3.8 §3.9). Par exemple, si l'on utilise une méthode comme BLASTX, il serait extrêmement fastidieux de reconstruire régulièrement les 6 couches correspondant aux 6 phases de lecture.

Dans APIC, chaque carte possède son propre formatage des types d'objets. Les différents paramètres d'affichage attachés à un type d'objet peuvent être repérés à l'aide d'une suite de mots clés de la forme : FONCTION.ZONE.ATTRIBUT.TYPE.

- La FONCTION correspond à la famille d'opérations qui est concernée parmi la liste (DISPLAY, LAYOUT, SETTING).
- La ZONE correspond au nom de la zone dans la boîte de dialogue correspondant à la FONCTION. Par exemple pour la fonction DISPLAY le champ ZONE peut prendre ses valeurs dans la liste (OBJECT AS, FILTER, DISPLAY).
- Le champ ATTRIBUT correspond à la liste des paramètres que l'on peut modifier pour une FONCTION et une ZONE données. Ainsi avec les fonction et zone DISPLAY.FILTER, l'attribut peut prendre les valeurs suivantes (SHOW, HIDE).

- Enfin, le champ TYPE correspond au type qui est concerné par le paramètre. Notons que le champ TYPE n'est pas forcément présent, c'est le cas pour les ressources LAYOUT.SEPARATION.STYLE et LAYOUT.SEPARATION.COLOR qui sont valables au niveau de la carte tout entière, indépendamment des styles que l'on y trouve.

Grâce à ce format il devient très simple de sauver les ressources d'affichage sous la forme d'un fichier au format TINA-1 composé de couples "Ressource{valeur}". L'annexe 3 donne la liste complète des ressources que l'on peut sauver. Notons que l'ordre de sauvegarde importe peu puisque le nom des ressources est non ambigu.

Pour l'instant, le chargement et la sauvegarde des ressources est automatique. A terme ces deux opérations devraient pouvoir s'effectuer à la demande et ainsi permettre interactivement l'affectation d'un même fichier de ressources à des cartes différentes.

Les ressources concernant des types absents dans une carte ne sont pas prises en compte ; à l'inverse, une valeur par défaut est prévue pour les ressources qui ne sont pas décrites dans le fichier de ressources.

Conclusion

Cette conclusion dresse le bilan de l'étude qui a été réalisée, décrit l'expérience qui a été accumulée et fait une analyse rétrospective de l'année consacrée à ce travail.

Bilan de l'étude

Le sujet de ce mémoire avait pour but la conception et la réalisation d'une interface cartographique d'aide à l'analyse des séquences génomiques.

Très tôt, devant la diversité des représentations associées à une séquence d'ADN, nous avons cherché à généraliser cette interface et considérer un générateur d'interfaces de ce type. Nous avons donc considéré le problème en amont et notre but final a été d'offrir *une boîte à outils permettant de construire des interfaces spécifiques.*

Selon les méthodes d'analyses qui sont utilisées dans les projets de séquençage, les résultats obtenus peuvent se présenter sous des formes extrêmement variables : cartes génétiques, physiques, séquences, courbes, etc. De ce fait pour réaliser une interface qui soit réellement générique, il nous est apparu essentiel dans un premier temps de *s'extraire de la vision "biologique" du problème à résoudre pour n'en garder qu'une vision purement graphique.* On a donc décidé de traiter le cas de la représentation sur un axe orienté et gradué (vertical ou horizontal) d'une succession d'objets qui peuvent être des points, des segments ou des courbes. À ces objets seront fixés des annotations diverses comme le nom de l'objet, la séquence nucléotidique ou encore l'orientation.

L'interface devait permettre un affichage multiple de cartes en limitant l'explosion de fenêtres. Cet objectif a été résolu en introduisant la notion de *plan de travail* dans lequel sont empilées les cartes les unes à la suite des autres. On a vu alors que cette notion permet de structurer automatiquement l'environnement de travail de l'utilisateur en réglant les problèmes d'ordonnancement, d'alignement et de redimensionnement de cartes.

Le volume des informations à représenter peut être important. Il était donc nécessaire de fournir les moyens à l'utilisateur, de conserver une perception efficace de la représentation et de pouvoir "naviguer" facilement à l'intérieur des cartes et ceci quelle que soit leur grandeur. Nous avons alors étendu les opérations d'une *carte* en développant des méthodes courantes de manipulation, comme le zoom ou les déplacements.

Les *objets* représentés sur une carte peuvent prendre des formes différentes : un segment, un point, une séquence sous la forme d'un texte ou une courbe. Plutôt que de décrire autant de concepts différents, il nous est apparu plus générique de considérer un objet comme une somme d'informations à représenter : un moule dans lequel sont rangées des propriétés (un nom, une orientation, des incertitudes sur la position, une séquence, une courbe) que l'on peut, si elles sont présentes, décider de montrer ou non.

L'objectif principal d'une interface cartographique est qu'elle doit permettre, par des visualisations diverses, d'avoir un stimulus à l'imagination. Nous avons alors détaillé différents concepts *couches*, *plans* et *liens* permettant d'organiser la représentation des cartes (disposition de l'information) et de relier.

Il est donc possible de structurer dynamiquement l'affichage grâce aux notions de *couches* qui divise la carte en plusieurs bandes horizontales et de *plans* qui permettent la superposition d'objets.

Devant l'accroissement des données issues des projets de séquençage, il est indispensable d'avoir un moyen de comparer rapidement des cartes et ceci qu'elles soient ou non issues d'organismes différents et quelles que soient leurs origines, unités ou résolutions. Pour répondre à ce besoin, nous avons défini des principes de liens (hook) entre cartes. Ces principes permettent le déplacement synchrone à l'intérieur de ces cartes. Un *lien global* permet de synchroniser le déplacement entre des cartes dont le facteur d'échelle est connu et varie linéairement, alors qu'un *lien entre objets* permet d'effectuer cet accrochage par rapport aux objets communs que se partagent les cartes. Cette possibilité est intéressante lorsque l'on souhaite synchroniser le déplacement de cartes qui possèdent des échelles qui sont non seulement différentes, mais qui ne sont, de surcroît, pas linéairement dépendantes. Cette alternative peut s'avérer également originale pour "marquer" des inclusions entre cartes en définissant des objets communs fictifs pour synchroniser les origines des cartes.

Une interface cartographique doit être suffisamment conviviale et générique pour pouvoir être acceptée par des projets différents. Nous avons souhaité que, sans programmation, la boîte à outils pour l'édition de cartes génomiques soit utilisable par des types d'utilisateurs différents. La recherche de cette adaptabilité nous a conduit à définir des outils supplémentaires pour pouvoir personnaliser la représentation de l'utilisateur.

Ainsi, afin de ne pas surcharger l'affichage, l'utilisateur peut choisir de représenter l'essentiel en retenant uniquement les propriétés et les types d'objets qu'il est utile de montrer. De plus, en fonction de ses préférences et du type de carte qu'il doit manipuler, l'utilisateur peut personnaliser le graphisme (forme, couleur et police de caractères) utilisé pour représenter les différentes propriétés de l'objet à afficher (l'objet lui-même, son nom, son orientation, etc.). Enfin, dans le but de pouvoir représenter même à une très petite échelle la séquence d'un objet, nous avons été amenés à distinguer plusieurs cas de figure parmi lesquels on peut noter surtout le moyen de montrer les bases qui composent la séquence sous la forme de traits de couleur, ce qui peut faciliter la recherche de régularités dans une séquence.

Expérience accumulée

Cette année a été pour moi le moyen d'acquérir beaucoup de nouvelles connaissances. En biologie moléculaire tout d'abord, et à travers les travaux du projet SHERPA dirigée par F. Rechenmann, où j'ai pu constater combien le vivant est une source d'inspiration importante pour le développement des sciences. J'ai assimilé les bases du système UNIX. J'ai pu mettre en pratique des connaissances théoriques sur les langages à objets à travers l'utilisation du langage C++. J'ai découvert le monde des interfaces graphiques à travers l'utilisation de la bibliothèque de classes d'objets graphiques ILOG Views de la société ILOG Software. ILOG Views est un produit récent (la première version est sortie en mars 1994) et j'ai donc dû faire preuve de beaucoup d'autonomie pour arriver à réaliser rapidement un premier prototype d'interface.

Analyse rétrospective

À l'exception de la représentation d'une séquence, où les lettres qui représentent les bases peuvent être associées explicitement à des couleurs, la bibliothèque de classes d'objets pour l'édition de cartes génomiques APIC possède un haut niveau de généricité. Elle aborde de manière générale les problèmes de représentation d'objets sur un axe ordonné. Elle pourrait convenir à d'autres types de représentation, par exemple dans le suivi de tâches sur un axe temporel. L'exception du traitement de la séquence pourrait être résolu en offrant la possibilité de coder n'importe quel caractère d'un texte (donc d'une séquence) par une couleur.

La réalisation d'un prototype est une phase exploratoire nécessaire à l'élaboration d'une interface. Elle permet de valider les choix de l'utilisateur et amène souvent des remarques qui n'avaient pas été décelées au départ. Mais il faut alors se méfier de la tendance qui peut être donnée au détail. Nous avons constaté qu'il était essentiel de savoir ne pas aller trop loin dans le prototypage et assez loin pour montrer et obtenir des réactions. Il n'est pas nécessaire par exemple d'envisager, dès le départ, tous les cas de figure.

Nous terminerons ce mémoire en précisant qu'actuellement, l'Institut Curie à Paris utilise l'interface cartographique construite à partir de la bibliothèque de classes d'objets APIC pour l'étude de la bactérie *Bacillus Subtilis* et que deux articles qui abordent l'interface ont été acceptés pour publication à la troisième conférence internationale ISMB, Intelligent Systems for Molecular Biology qui aura lieu en juillet 1995 en Angleterre à Cambridge [Bisson95][Médigue95].

[Altschul90]
S. F. Altschul, W. Gish, E. W. Myers, D. J. Lipman, Basic local alignement search tool, J. Mol. Biol., 215 : pp. 403-410, (1990)

[Beilvert93]
J. F. Beilvert, Conception d'une base de données dédiée à l'analyse du génome, réalisée avec le SGBD orienté objets O2, Mémoire, Conservatoire National des Arts et Métiers, Paris, (1993)

[Bisson94]
G. Bisson, A. Garreau, Spécifications de l'interface APIC, Document interne, Grenoble, (1994)

[Bisson95]
G. Bisson, A. Garreau, APIC : a generic interface for sequencing projects, article à paraître, ISMB, Cambridge (Juillet 1995)

[Borodovsky93]
M. Borodovsky, J. D. McIninch, GENMARK : Parallel gene recognition for both DNA strands, Computers & Chemistry, 17(2) : pp. 123-133, (1993)

[Coutaz90]
J. Coutaz, Interfaces homme-ordinateur, Conception et réalisation, Dunod Informatique, (1990)

[De Paoli94]
H. De Paoli, Système d'aide à l'analyse de séquences génomiques avec un SGBD orienté objets et un système à base de connaissance, Mémoire, Conservatoire National des Arts et Métiers, Paris, (1994)

[Dorkeld94]
F. Dorkeld, MultiMap : un modèle objet dédié à la cartographie comparée des génomes des mammifères, Thèse de doctorat, Université Claude Bernard, Lyon 1, (1994)

[Durbin92]
E. Durbin, J. Thierry-Mieg, ACeDB, A *C. Elegans* Database, Users' guide & System Manual, (1992)

[Eboueya92]
M. Eboueya, Les interfaces graphiques dans la communication homme-ordinateur, Cours INRIA : Les interfaces intelligentes dans l'information scientifique et technique, Klingenthal (Bas-Rhin), pp. 43-73, (18-22 Mai 1992)

[Ferber90]
J. Ferber, Conception et programmation par objets, Hermes, (1990)

[Greg93]
Groupement de Recherches et d'Études sur les Génomes, Système coopératif pour l'analyse de séquences génomiques, Fiche de présentation du projet, Responsable scientifique du projet : F. Rechenmann, Gif-sur-Yvette (Essonne), (1993)

[Ilog94]
Ilog Software, Ilog Views, Manuel de référence, Version 1.3 et 2.0, (Mars et Juin 1994)

[Jordan89]
B. Jordan, Les cartes du génome humain, La recherche, 216, (Décembre 1989)

[Khalil91]
G. Khalil, C, C++ et UNIX, initiation aux langages et environnement, Masson, (1991)

[Masini89]
G. Masini, A. Napoli, D. Colnet, D. Leonard, K. Tombre, Les langages à objets, langages de classes, langages de frames, langages d'acteurs, InterEditions, (1989)

[Medigue91]
C. Medigue, Conception et exploitation d'une base de données spécialisée dédiée à l'analyse du génome d'*Escherichia coli*, Thèse de doctorat, Université Paris 6, (1991)

[Medigue95]
C. Medigue, T. Vermat, G. Bisson, A. Viari, A. Danchin, Cooperative computer system for genome sequence analysis, article à paraître, ISMB, Cambridge (Juillet 1995)

[Mollaret89]
M. Mollaret, C/C++ et programmation objet, Armand Colin, (1989)

[Pitaval 94]
C. Pitaval, Les logiciels de cartographie génomique, Examen probatoire CNAM, CUEFA, Grenoble, (Mars 1994)

[Schmeltzer95]
O. Schmeltzer, Modélisation de cartes génomiques, Une formalisation et un algorithme de construction fondé sur le raisonnement temporel, Thèse de doctorat, Université Joseph Fourier, Grenoble 1, (1995)

[Sigma92]
Los Alamos National Laboratory, SIGMA, System for Integrated Genome Map Assembly, Disponible par ftp anonyme sur atlas.lanl.gov, (1992)

[Stroustrup92]
B. Stroustrup, Le langage C++, 2ème Edition, Addisson-Wesley (1992)

[Voisard92]
A. Voisard, Un modèle d'édition de cartes : vers une boîte à outils géographique, VIIIèmes journées Bases de données avancées, Tregastel, pp. 64-84, (15-18 Septembre 1992)

Annexe 1 : Description d'une carte dans le format TINA-1

Les caractéristiques d'une carte et la liste des entités qui apparaissent dans une carte sont transmises à l'interface APIC en utilisant le format TINA-1 utilisé à l'Institut PASTEUR [De Paoli94]. Dans ce format chaque entité à afficher est décrite par une liste de couples descripteur/valeur "à plat".

```
attribut1 { valeurs de cet attribut }
attribut2 { valeurs de cet attribut } ....
```

Les descripteurs peuvent être écrits avec une combinaison quelconque de caractères sauf l'accolade et l'espace. Les *majuscules ne sont pas signifiantes*, ainsi les attributs `Date` et `DATE` sont identiques. Un fichier de description se compose d'un entête et d'une liste d'entités.

En-tête d'un fichier

L'entête sert à préciser les caractéristiques globales de la carte. Certaines de ces caractéristiques ne sont présentes qu'à des fins d'identification informatique (la date) ou biologique (la provenance) du fichier. D'autres correspondent à des informations qui vont être utilisées par l'interface pour afficher la carte, c'est par exemple le cas pour le nom de l'unité de mesure qui est nécessaire à la bonne gestion des zooms. Enfin, par l'intermédiaire des fichiers de ressources cet entête permet de définir le mode de représentation qui est souhaité par l'utilisateur. Les champs notés en **gras** sont nécessaires, les autres sont optionnels.

`File-format` `{chaîne de caractères}`
Contient la version du format du fichier. Actuellement la valeur de ce champ doit toujours être TINA-1.
`File-format {TINA-1}`

`File-date` `{chaîne de caractères}`
Indique la date et l'heure de création du fichier
`File-date {10/12/94 9h32}`

`Map-name` `{chaîne de caractères}`
Nom de la carte qui est décrite dans le fichier
`Map-name {carte E.Coli}`

`Map-origin` `{chaîne de caractères}`
Identifie la provenance de la carte (résultats, banque de données, etc.)
`Map-origin {résultat de BLASTX}`

`Map-description` `{chaîne de caractères}`
Donne une description des caractéristiques de la carte
`Map-description{partie intéressante entre 10 et 15 kB}`

`Map-unit` `{chaîne de caractères}`
Indique l'unité dans laquelle la carte est exprimée (CentiMorgan, kb, ..). On considère que la table de conversion entre les différentes unités est décrite dans un fichier externe (Cf. Annexe 2)

```
Map-unit {centi-morgan}
```

`Map-accuracy` `{réel positif}`
Nombre positif représentant le pas de résolution minimal de la carte. Par défaut cette valeur vaut 1. Ce champ est nécessaire pour connaître le nombre de chiffres significatifs lorsque l'on travaille avec des séquences dans lesquelles les coordonnées des objets sont des réels (par exemple, c'est le cas pour une carte génétique exprimée en minutes). Si la différence entre deux coordonnées est plus petite que ce seuil les deux valeurs sont considérées comme désignant la même position.

```
Map-accuracy {.01}
```

`Map-scale` `{réel positif}`
Indique le facteur d'échelle (rapport entre l'unité de mesure et le nombre de pixels sur l'écran) qui est utilisé pour l'affichage initial de la carte. Par défaut ce facteur vaut 1 : une unité est égale à un pixel.

```
Map-scale {5.5}
```

`Map-start` `{entier}`
Indique la position sur laquelle la partie gauche de l'affichage initial doit être calé. Par défaut on positionne la carte sur l'objet de la séquence ayant la plus petite coordonnée sur l'axe horizontal.

```
Map-start {1000}
```

`Map-RSC-file` `{nom de fichier}`
Indique le fichier de ressource (Cf. §3.13) qui doit être utilisé lors de l'affichage de la carte. Ce fichier permet de choisir comment les entités sont représentées et organisées dans la carte.

```
Map-RSC-file {format_sortie_BLASTX}
```

`Other_fields 1` `{contenu quelconque}`
`...` `...`
`Other_fields n` `{contenu quelconque}`
Permet d'indiquer différentes informations biologiques qui ne sont pas utilisées par le système, mais qui sont affichées lorsque l'utilisateur "clique" sur une carte pour en connaître la description.

Liste des entités de la carte

Les entités peuvent être décrites dans un ordre quelconque vis-à-vis de leur apparition dans la séquence. Toutefois les différents descripteurs d'une entité doivent être mis *consécutivement*, le premier descripteur étant toujours le descripteur TYPE qui indique le type de l'objet. La façon dont un type d'objet est affiché est contrôlée à l'aide des fonctions de DISPLAY, LAYOUT, SETTING ainsi que par l'intermédiaire des fichiers de ressources (sections §3.7 à §3.9).

`Type` `{chaîne de caractères}`
Chaîne identifiant le type de l'objet (RBS, Gène, Stop, ...)
`Type {Stop}`

`Name` `{chaîne de caractères}`
Chaîne indiquant le nom biologique de l'objet. Ce champ est optionnel car certains objets peuvent ne pas avoir de nom. Bien évidemment, dans l'interface les objets graphiques ne sont pas identifiés par ce nom, mais (éventuellement) par un identificateur généré lors du chargement.
`Name {Blasta}`

`Begin` `{réel}`
Nombre indiquant la position sur l'axe horizontal du début de l'objet dans la séquence.
`Begin {3250}`

`End` `{réel}`
Nombre indiquant la position finale de l'objet dans la séquence.
`End {3320}`

`Oriented` `{+ | -}`
Ce descripteur permet de préciser le sens de l'orientation de l'objet. A priori, ce descripteur prend soit la valeur "+" (sens 3'→5') soit la valeur "-" (sens 3'←5'). Par défaut, l'orientation est de type "+".
`Oriented {+}`

`Delta-begin-min` `{réel positif}`
`Delta-begin-max` `{réel positif}`
Ces deux descripteurs expriment respectivement les bornes gauche et droite (et inversement si l'orientation est -) de l'incertitude par rapport à la position de début de l'objet sur l'axe horizontal. Par défaut ces valeurs valent 0.
`Delta-begin-min {2}`
`Delta-begin-max {25}`

`Delta-end-min` `{réel positif}`
`Delta-end-max` `{réel positif}`

Ces deux descripteurs expriment respectivement les bornes gauche et droite (et inversement si l'orientation est -) de l'incertitude par rapport à la position de fin de l'objet sur l'axe horizontal. Par défaut ces valeurs valent 0.

```
Delta-end-min {10}
Delta-end-max {20}
```

`Position` `{entier}`

Permet de fixer la position verticale (par rapport à la couche courante) de l'entité décrite. Lorsque ce descripteur n'est pas présent, l'interface APIC positionne automatiquement l'objet à l'aide de la Procédure de Positionnement des Objets (Cf.§ 2.2.4) qui gère les recouvrements. Par contre, lorsque `Position` est donnée, l'interface place directement l'objet aux coordonnées indiquées sans gérer les recouvrements. Cette possibilité permet d'utiliser ponctuellement l'interface APIC comme d'une surface de dessin classique, par exemple, lorsque l'utilisateur veut placer une courbe en surimpression des objets, ou faire apparaître le dessin des bandes cytogénétiques en fond de carte. Par défaut, les objets sont positionnés en fond de carte, mais ceci peut être modifié à l'aide des options de modification de l'affichage (Cf. §3.8).

```
Position {100}
```

`Sequence` `{chaîne de caractères}`

Indique la séquence qui est associée à l'objet courant.

```
sequence {ACGACCTAGGACGTTCAGG}
```

`Curve` `{x0 y0 x1 y1 .... xn yn}`

Ce descripteur permet de spécifier une courbe qui sera associée à l'entité courante. La description est effectuée en donnant la liste des coordonnées horizontale et verticale (X, Y) qui la composent. Lors de l'affichage ces points sont reliés entre eux par des segments de droite. La couleur d'une courbe peut être modifiée à l'aide de l'éditeur de ressources (Cf. §3.9).

```
Curve {0 0 10.2 25 40.5 100 60 80.0 100 0}
```

`Other_fields 1` `{contenu quelconque}`
`...` `...`
`Other_fields n` `{contenu quelconque}`

Permet d'indiquer différentes informations biologiques qui ne sont pas utilisées par le système, mais qui sont affichées lorsque l'utilisateur "clique" sur un objet pour en connaître la description.

```
status-objet {putatif}
probabilité-de-poisson {0}
```

Annexe 2 : Description des unités dans le format TINA-1

Les unités, dans lesquelles peuvent être exprimées les valeurs des positions des objets dans une carte, sont stockées selon le format TINA-1 : attribut {valeur}.

La description retenue pour représenter une unité et ses relations avec d'autres unités est la suivante :

Unité {u1}

Où u1 est une chaîne de caractères qui représente une unité, par exemple : "b" pour base.

u2 {fu1u2}

Où u2 est une chaîne qui représente une unité (par exemple, "cM" pour centiMorgan) pour laquelle on peut donner un facteur d'échelle fu1u2 qui permet de convertir une position dans l'unité u1 en position dans l'unité u2.

Ce qui donne pour traduire les relations entre les unités "b" pour base, "Kb" pour Kilobases et "cM" pour centiMorgan la liste suivante :

```
Unité {b}
Kb {0.001}
cM {0.000001}
Unité {Kb}
b {1000}
cM {0.001}
Unité {cM}
b {1000000}
Kb {1000}
```

Annexe 3 : Description des ressources dans le format TINA-1

Les ressources sont stockées selon le format TINA-1 : attribut{valeur}.

Comme décrit dans la section 3.13, les attributs utilisent le format suivant : FONCTION.ZONE.ATTRIBUT.TYPE. On distingue deux grands types de valeur en fonction de l'interaction que l'on trouve dans les boîtes de dialogue. Pour ce qui concerne les ressources qui sont modifiables à l'aide d'un bouton, la valeur correspond à un booléen. Pour ce qui concerne les ressources qui sont modifiables à l'aide d'un champ éditable ou d'un menu, la valeur correspond à la valeur introduite par l'utilisateur ou à l'option qui a été sélectionnée.

Opérations DISPLAY

Display.Object as.Point	{booléen}	
Display.Object as.Segment	{booléen}	
Display.Filter.Show	{booléen}	
Display.Filter.Hide	{booléen}	
Display.Display.Object	{booléen}	
Display.Display.Name	{booléen}	
Display.Display.Orientation	{booléen}	
Display.Display.Uncertainty	{booléen}	
Display.Display.Curve	{booléen}	
Display.Display.Sequence	{booléen}	

Opérations LAYOUT

Layout.Layer.Number	{entier 1..9}	
Layout.Plan.Background	{booléen}	
Layout.Plan.Nooverlapping	{booléen}	
Layout.Plan.Foreground	{booléen}	
Layout.Separation.Style	{None, Dot, Single, Bold, Double}	*Note : pas de type*
Layout.Separation.Color	{code de couleur}	*Note : pas de type*

Opérations SETTING

Setting.Object.Point	{nom de forme}
Setting.Object.Segment	{nom de forme}
Setting.Object.Color	{code de couleur}
Setting.Object.High	{entier}
Setting.Object.Offset	{entier}
Setting.Label.Color	{code de couleur}
Setting.Label.Font	{nom de police et taille}

Setting.Orientation.Color	{code de couleur}
Setting.Orientation.Shape	{nom de forme}
Setting.Uncertainty.Color	{code de couleur}
Setting.Uncertainty.Shape	{nom de forme}
Setting.Sequence.Color	{code de couleur}
Setting.Sequence.Font size	{entier}
Setting.Sequence.A	{code de couleur}
Setting.Sequence.T	{code de couleur}
Setting.Sequence.G	{code de couleur}
Setting.Sequence.C	{code de couleur}
Setting.Sequence.Usefalsetext	{booléen}
Setting.Sequence.Coloredmode	{booléen}
Setting.Sequence.Coloredletters	{booléen}

Annexe 4 : Procédure de Positionnement des Objets

L'algorithme de positionnement des objets dans les couches consiste à décaler automatiquement sur l'axe vertical les objets qui partagent, partiellement ou totalement, une même position dans la séquence. Pour simplifier la gestion, les objets une fois dessinés sont assimilés à des boîtes rectangulaires (figure 36). La valeur OFFSET est définissable par l'utilisateur (Cf. §3.9) et permet de régler l'espacement vertical.

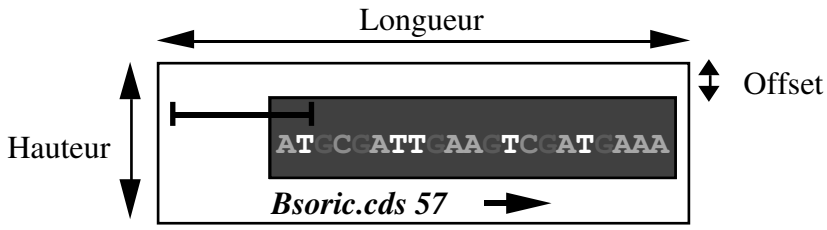


Figure 36 : Gestion des objets comme une boîte.

Dès lors, le principe de l'algorithme est très simple : il consiste à placer un par un les objets dans l'affichage en contrôlant à chaque fois si l'objet "intersecte" ou non avec un autre. En cas de collision, l'objet courant est automatiquement décalé vers le bas. La procédure est utilisée pour chacune des couches et renvoie la hauteur maximale de cette couche. Enfin, les objets sont placés par taille croissante afin de limiter l'apparition de "trous" dans l'affichage.

```
Entrées   : Liste LO des objets à placer
Sortie    : Taille maximale "TailleMax" de la couche

Procédure  Placement_Objets
  TailleMax = 0
  Trier les objets par longueur décroissante
  Pour chacun des objets Oi de la liste LO
    Calculer la liste LI des objets qui intersectent avec Oi
    Si LI = Ø
      Alors
        Ymax = 0
      Sinon
        Ymax = Maximum (Hauteur des objets de LI)
    Afficher Oi à la position Ymax
  TailleMax = Maximum (Taillemax, Ymax + Hauteur(Oi))
```

Annexe 5 : Exemple de classe et d'utilisation de la bibliothèque de classes d'objets graphiques ILOG Views

Un exemple de classe ILOG Views, définition de la classe *IlvLabel* [Ilog94] :

```
class ILVEXPORTED IlvLabel
: public IlvSimpleGraphic {
protected:
   IlvPoint _center;
   char*    _label;
public:
   IlvLabel(IlvDisplay*     display,
        const IlvPoint& position,
        const char*          label,
        IlvPalette*     palette=0);
   IlvLabel(IlvDisplay*     display,
        IlvPos          x, IlvPos y,
        const char*          label,
        IlvPalette*     palette=0);
   ~IlvLabel ();
   // ____________________________________________________________
   virtual IlvBoolean zoomable() const;
   const char* getLabel() const { return _label; }
   virtual void  setLabel(const char* s);
   virtual void boundingBox(IlvRect& rect, const IlvTransformer* t = 0) const;
   virtual void applyTransform(const IlvTransformer* t);
   IlvBoolean contains(const IlvPoint& p, const IlvPoint& tp,
            const IlvTransformer* t = 0) const;
   virtual void draw(IlvPort*, const IlvTransformer* t = 0,
             const IlvRegion* = 0) const;
   IlvBoolean intersects(const IlvRegion& tregion,
              const IlvTransformer* t = 0) const;
   IlvBoolean inside(const IlvRegion& tregion,
             const IlvTransformer* t = 0) const;
   DeclareTypeInfo();
   virtual void print(ostream&, int level = 0) const;
   void setc.enter(const IlvPoint& p) { _center = p; }
};
```

Cette classe décrit un objet graphique particulier : un label. Elle hérite de la classe IlvSimpleGraphic *: public IlvSimpleGraphic*. Cette indication permet de préciser que l'ensemble de classe public de la classe IlvSimpleGraphic fait également partie des membres publics de la classe dérivée IlvLabel. L'indication *protected*, qui fait partie de la définition de la classe, limite l'accès des attributs (ici _label et _center) ou méthodes qui la suivent aux méthodes de la classe elle même et aux méthodes des classes dérivées de cette classe.

L'accès à un membre protégé d'une classe est en fait identique à celui d'un membre public à part qu'il est restreint aux seules classes dérivées.

La méthode *set Label(const char*)* permet d'associer une chaîne de caractères à une instance de la classe IlvLabel, tandis qu'inversement la méthode *const char* getLabel()* permet de récupérer cette chaîne de caractères.

Un exemple d'utilisation de la bibliothèque ILOG Views [Ilog94]:

```
include <ilviews/contain.h>
include <ilviews/label.h>
include <ilviews/contint.h>

main(int argc, char** argv)
(
        IlvDisplay* display = new IlvDisplay("Exemple", " ", argc, argv);
        if   (display->isBad()) (
            IlvFatalError("Couldn't initialize display");
            delete display;
            exit(-1);
        )
        IlvRect size(0, 0, 200, 200);
        IlvTopView* topview = new IlvTopView(display, "Exemple", "Exemple", size);
        IlvContainer* view = new IlvContainer(topview, size);

        IlvLabel* label = new IlvLabel(display, IlvPoint(100, 100), "Deplacer moi");

        view->addObject(label);
        view->setObjectInteractor(label, new IlvViewMoveInteractor());

        IlvMainLoop();
)
```

Analyse de cet exemple :

```
main(int argc, char** argv)
```

La fonction principale en C++.

```
IlvDisplay* display = new IlvDisplay("Exemple", " ", argc, argv);
if   (display->isBad()) (
     IlvFatalError("Couldn't initialize display");
     delete display;
     exit(-1);
)
```

Le constructeur de base de la classe IlvDisplay autorise l'entrée dans le domaine ILOG Views.

La méthode isBad() teste si ILOG Views peut être lancé correctement.

```
IlvRect size(0, 0, 200, 200);
IlvTopView* topview = new IlvTopView(display, "Exemple", "Exemple", size);
```

Création d'une nouvelle fenêtre de la classe IlvTopView appelée topview.

```
IlvContainer* view = new IlvContainer(topview, size);
```

L'objet view qui est créé est une instance de la classe IlvContainer. Cette classe est une sorte de boîte dans laquelle des objets peuvent être rangés et réagir à des événements (ex : un clic souris).

```
IlvLabel* label = new IlvLabel(display, IlvPoint(100, 100), "Déplacer moi");
```

La classe IlvLabel permet de définir des objets graphiques qui sont des labels, en précisant leur contenu et leur place.

```
view->addObject(label);
```

L'objet label est inséré dans le container qui vient juste d'être créé.

```
view->setObjectInteractor(label, new IlvViewMoveInteractor());
```

Un autre concept apparaît ici : les interacteurs. Créer un objet de type interacteur et associer cet interacteur avec un objet graphique, comme un label, permet de définir un comportement pour cet objet. Dans cet exemple, l'utilisateur peut déplacer le texte "Déplacer moi" dans le container en pressant le bouton gauche de la souris sur le label et en relâchant le bouton à la position désirée.

```
IlvMainLoop();
```

Le programme entre en action en appelant la fonction IlvMainLoop.

Annexe 6 : La bibliothèque de classes d'objets pour la construction d'interfaces cartographiques APIC.

La description qui suit ne considère que l'ensemble minimal des classes et des méthodes à connaître pour construire des interfaces cartographiques à partir de la bibliothèque de classes d'objets APIC. Les méthodes d'interaction spécifiques employées lorsque la bibliothèque APIC est utilisée de manière autonome pour fournir une interface standard ne sont pas détaillées.

ApicMultiMapGet

C'est le plan de travail dans lequel sont rangées les cartes que l'utilisateur souhaite visualiser.

Description de la classe

```
class  ApicMultiMapGet
{
public:
    ApicMultiMapGet(IlvDisplay*                 display,
                    IlvCompositeAbstractView*   parent=0,
                    IlvBoolean                  visiblemenubar=IlvTrue);
    ApicMapGet*     addMapGet(const char* path);
    IlvBoolean      removeMapGet(ApicMapGet* mapget);
    ApicMapGet**    getMapGets();
    void            Draw();
    void            moveResize(const IlvRect& size);
    void            clear();
    void            boundingBox(IlvRect& rect);
    IlvDouble       getconversionfactor(const char* unit1,const char* unit2);
    ApicHook*       getHook();
    ApicArrange*    getArrange();
    ApicLayout*     getLayout();
    ApicDisplay*    getDisplay();
    ApicSetting*    getSetting();
    ApicHelp*       getHelp();
};
```

- Le constructeur ApicMultiMapGet initialise un plan de travail qui peut être inséré dans une fenêtre <parent>. Le bandeau de commande du plan de travail ne sera pas visible si <visiblemenubar> a la valeur IlvFalse.

- La méthode `addMapGet` ajoute la carte, dont la description est contenue dans un fichier au format TINA-1 `<path>`, dans le plan de travail. Cette méthode retourne un pointeur sur la carte qui a été créée.

- La méthode `removeMapGet` retire une carte du plan de travail.

- La méthode `getMapGets` retourne, sous la forme d'un tableau, l'ensemble des cartes contenues dans le plan de travail.

- La méthode `Draw` affiche le plan de travail.

- La méthode `moveResize` déplace et redimensionne le plan de travail selon le rectangle de référence `<size>` qui est donné.

- La méthode `clear` réinitialise le plan de travail. Toutes les cartes qui étaient contenues dans le plan de travail sont alors retirées du plan de travail.

- La méthode `boundingBox` initialise le rectangle de référence `<rect>` avec les dimensions du plan de travail.

- La méthode `getconversionfactor` retourne le facteur de conversion qui permet de relier les deux unités `<unit1>` et `<unit2>`.

- Les méthodes `getHook` à `getHelp` retournent des pointeurs sur des outils. Il sera alors possible d'utiliser ces outils et de déclencher les opérations qu'ils autorisent.

ApicMapGet

C'est la carte, objet d'interaction, qui est contenue dans le plan de travail.

Description de la classe

```
class ApicMapGet
{
public:
    ApicMapBehavior*      getMapBehavior();
    ApicObjectBehavior*   getObjectBehavior();
    void                  goToposition(IlvDouble pos);
    void                  settypeincrement(const char* type);
    void                  goTopreviousstypeincrement();
    void                  goTonexttypeincrement();
    void                  setunitincrement(IlvDouble unitincrement);
    void                  goTopreviousunitincrement();
    void                  goTonextunitincrement();
    void                  all();
    void                  centre(IlvDouble center);
    void                  zoomin(IlvDouble pos);
    void                  zoomin();
    void                  zoomout(IlvDouble pos);
    void                  zoomout();
    void                  setscalefactorincrement(IlvDouble scalefactorincrement);
    void                  setscopetomap();
    void                  setscopetohook();
```

```
    void            makeMark(IlvDouble position);
    void            eraseMark();
    void            mark();
    void            toggle();
    void            setcursorbar(IlvBoolean cursorbar);
    IlvDouble       computeDeltaMark();
    void            storememory(IlvInt memorynumber);
    void            recallmemory(IlvInt memorynumber);
    void            exchangememory(IlvInt memorynumber);
    IlvDouble       getPosition();
    IlvDouble       getCenter();
    IlvDouble       getMark();
    ApicMap*        getMap();
    ApicObject**    getObjects();
    void            boundingBox(IlvRect& rect);
    void            resize(IlvDim h);

};
```

- La méthode `getMapBehavior` retourne un pointeur sur l'objet d'interaction qui est associé au comportement d'une carte.

- La méthode `getObjectBehavior` retourne un pointeur sur l'objet d'interaction qui est associé au comportement d'un objet.

- La méthode `goToposition` positionne le début de la carte à la position `<pos>`.

- La méthode `settypeincrement` donne le type d'objet vers lequel on souhaite se déplacer `<type>`.

- Les méthodes `goTopreviousstypeincrement` et `goTonexttypeincrement` recentrent la carte sur l'objet le plus proche du centre dans le type choisi.

- La méthode `setunitincrement` donne un pas d'incrémentation `<unitincrement>` dans l'unité de la carte.

- Les méthodes `goTopreviousunitincrement` et `goTonextunitincrement` déplacent la carte du pas d'incrémentation choisi.

- La méthode `all` calcule automatiquement le facteur d'échelle interne afin que la totalité de la carte soit visible.

- La méthode `centre` centre la carte sur la position `<center>`.

- Les méthodes `zoomin` et `zoomout` multiplient ou divisent le facteur d'échelle interne par le coefficient courant et demandent à la carte de se redessiner, centrée à la position `<pos>`.

- La méthode `setscalefactorincrement` change le coefficient (multiplicateur ou diviseur) qui sera appliqué sur le facteur d'échelle interne en cas de zoom.

- Les méthodes `setscoptomap` et `setscopetohook` précisent si le changement de résolution doit être appliqué à la carte seule ou à toutes les cartes qui sont liées à cette carte.

- La méthode `makeMark` pose une marque à la position `<position>`.

- La méthode `mark` centre la carte sur la marque courante.

- La méthode `toggle` centre la carte sur la marque courante, supprime cette marque et repose une marque à l'endroit où était centrée la carte avant que se produise le déplacement.

- La méthode `setcursorbar` permet de faire apparaître ou disparaître la barre verticale associée au curseur.

- Le méthode `computeDeltaMark` calcule la distance qui sépare la marque du centre de la carte.

- La méthode `storememory` range en mémoire la partie visible de la carte.

- La méthode `recallmemory` rappelle une vue qui a été mémorisée.

- La méthode `exchangememory` rappelle une vue qui a été mémorisée et remplace le contenu de la mémoire par la définition de la vue avant modification.

- Les méthodes `getPosition`, `getCenter` et `getMark` retournent des valeurs de position.

- La méthode `getMap` retourne un pointeur sur le "moule" qui contient toutes les propriétés qui sont attachées à une carte (Cf. Annexe 1).

- La méthode `getObjects` retourne, sous la forme d'un tableau, l'ensemble des objets contenus dans une carte.

- La méthode `boundingBox` initialise le rectangle de référence `<rect>` avec les dimensions de la carte.

- La méthode `resize` redimensionne la hauteur de la carte.

ApicArrange

C'est l'outil qui permet de gérer l'ordre des cartes dans le plan de travail.

Description de la classe

```
class ApicArrange :
{
public:
    void            GoFirst(ApicMapGet* mapget);
    void            GoLast(ApicMapGet* mapget);
    void            MoveUp(ApicMapGet* mapget);
    void            MoveDown(ApicMapGet* mapget);
    void            setHeader(ApicMapGet* mapget,IlvBoolean header);
    void            setMap(ApicMapGet* mapget,IlvBoolean map);
    void            setSteps(ApicMapGet* mapget,IlvInt steps);
    IlvBoolean      isVisibleHeader(ApicMapGet* mapget);
    IlvBoolean      isVisibleMap(ApicMapGet* mapget);
    IlvInt          getSteps(ApicMapGet* mapget);
};
```

- Les méthodes `Gofirst` et `GoLast` permettent respectivement de faire passer la carte `<mapget>` en première ou en dernière position du plan de travail.

- Les méthodes `MoveUp` et `MoveDown` permettent de faire progresser une carte d'un cran vers le haut ou vers le bas dans la liste des cartes.

- Les méthodes `setHeader` et `setMap` permettent de choisir si le panneau de commande de la carte et/ou la carte elle même sont visibles sur le plan de travail.

- La méthode `setSteps` permet de choisir le nombre de graduations qui doivent apparaître dans l'échelle placée au-dessus de la carte.

- Les méthodes `isVisibleHeader` et `isVisibleMap` indiquent si le panneau de commande de la carte et la carte sont visibles.

- La méthode `getSteps` retourne le nombre de graduations de l'échelle.

ApicHook

C'est l'outil qui permet de lier plusieurs cartes. Le déplacement, à l'intérieur de ces cartes, sera alors synchronisé; de façon globale ou sur les objets communs que se partagent ces cartes.

Description de la classe

```
class ApicHook :
{
public:
    IlvInt          ishooked(ApicMapGet* mapget);
    void            hookMapGet(ApicMapGet* mapget, IlvInt hook);
    void            unhookMapGet(ApicMapGet*);
    void            setobjectsynchronizing(IlvInt hook,IlvBoolean objectsynchronizing=IlvTrue);
    ApicMapGet**    gethookedMapGets(IlvInt hook,IlvUInt& count);
};
```

- La méthode `ishooked` retourne le numéro du lien auquel participe la carte.
- La méthode `hookMapGet` associe une carte à un lien.
- La méthode `unhookMapGet` libère la carte de tous liens.
- La méthode `setobjectsynchronizing` change le mode de synchronisation. Par défaut, la synchronisation entre plusieurs cartes est réalisée globalement en fonction des rapports qui existent entre les unités de ces cartes (s'ils existent). Cette opération demande à ce que la synchronisation soit non plus réalisée de cette manière mais sur les objets communs que se partagent les cartes (si elles ont des objets communs).
- La méthode `gethookedMapGets` retourne sous la forme d'un tableau, l'ensemble des cartes qui participent à un lien `<hook>`. Le nombre de ces cartes est mis à jour dans la variable `<count>`.

ApicLayout

C'est l'outil qui permet d'organiser la représentation des objets dans une carte. Il permet de définir des couches, qui correspondent à une structuration horizontale, et des plans, qui permettent la superposition d'objets.

Description de la classe

```
class ApicLayout :
{
public:
    void            setPlanBackground(ApicMapGet* mapget,const char* type);
    void            setPlanNooverlapping(ApicMapGet* mapget,const char* type);
    void            setPlanForeground(ApicMapGet* mapget,const char* type);
    IlvBoolean      isPlanBackground(ApicMapGet* mapget,const char* type);
    IlvBoolean      isPlanNooverlapping(ApicMapGet* mapget,const char* type);
    IlvBoolean      isPlanForeground(ApicMapGet* mapget,const char* type);
    IlvInt          getLayer(ApicMapGet* mapget,const char* type);
```

```
    void        Add(ApicMapGet* mapget,char* type,IlvInt layer);
    void        setSeparation(ApicMapGet* mapget,const char* name);
    void        setSeparationColor(ApicMapGet* mapget,const char* name);
};
```

- Les méthodes `setPlanBackground`, `setPlanNooverlapping` et `setPlanForeground` définissent le plan (devant de carte, plan intermédiaire, fond de carte) dans lequel seront affichés les objets d'un type donné `<type>` dans une carte donnée `<mapget>`.

- Les méthodes `isPlanBackground`, `isPlanNooverlapping` et `isPlanForeground` indiquent quel est le plan retenu pour un type d'objet dans une carte.

- La méthode `getLayer` retourne le numéro de la couche (horizontale) dans laquelle sont rangés les objets d'un type donné dans une carte.

- La méthode `Add` ajoute dans une carte `<mapget>` et dans une couche `<layer>` les objets d'un type donné.

- Les méthodes `setSeparation` et `setSeparationColor` définissent la forme et la couleur de l'objet graphique (généralement une ligne) qui va séparer deux couches horizontales.

ApicDisplay

C'est l'outil qui permet de sélectionner parmi les propriétés d'un objet, celles que l'on veut montrer.

Description de la classe

```
class ApicDisplay :
{
public:
    void        setObjectasPoint(ApicMapGet* mapget,const char* type);
    void        setObjectasSegment(ApicMapGet* mapget,const char* type);
    IlvBoolean  isObjectasPoint(ApicMapGet* mapget,const char* type);
    IlvBoolean  isObjectasSegment(ApicMapGet* mapget,const char* type);
    void        setDisplayObject(ApicMapGet* mapget,const char* type,IlvBoolean
display=IlvTrue);
    void        setDisplayName(ApicMapGet* mapget,const char* type,IlvBoolean
display=IlvTrue);
    void        setDisplayOrientation(ApicMapGet* mapget,const char* type,IlvBoolean
display);
    void        setDisplayUncertainty(ApicMapGet* mapget,const char* type,IlvBoolean
display=IlvTrue);
```

```
        void            setDisplayCurve(ApicMapGet* mapget,const char* type,IlvBoolean
display=IlvTrue);
        void            setDisplaySequence(ApicMapGet* mapget,const char* type,IlvBoolean
display=IlvTrue);
        IlvBoolean      isDisplayObject(ApicMapGet* mapget,const char* type);
        IlvBoolean      isDisplayName(ApicMapGet* mapget,const char* type);
        IlvBoolean      isDisplayOrientation(ApicMapGet* mapget,const char* type);
        IlvBoolean      isDisplayUncertainty(ApicMapGet* mapget,const char* type);
        IlvBoolean      isDisplayCurve(ApicMapGet* mapget,const char* type);
        IlvBoolean      isDisplaySequence(ApicMapGet* mapget,const char* type);
        void            Show(ApicMapGet* mapget,const char* type);
        void            Hide(ApicMapGet* mapget,const char* type);
        lvBoolean       isVisible(ApicMapGet* mapget,const char* type);
    };
```

- Les méthodes `setObjectasPoint` et `setObjectasSegment` représentent les objets du type donné `<type>` comme des points ou des segments.

- Les méthodes `isObjectasPoint` et `isObjectasSegment` indiquent si les objets du type donné sont représentés comme des points ou des segments.

- Les méthodes `setDisplayObject`, `setDisplayName`, `setDisplayOrientation`, `setDisplayUncertainty`, `setDisplayCurve` et `setDisplaySequence` indiquent les propriétés (nom, orientation, etc.) à afficher pour un type d'objet dans une carte.

- Les méthodes `isDisplayObject`, `isDisplayName`, `isDisplayOrientation`, `isDisplayUncertainty`, `isDisplayCurve` et `isDisplaySequence` indiquent quelles sont les propriétés qui sont affichées.

- La méthode `Show` indique que les objets du type donné `<type>` dans la carte donnée `<mapget>` doivent être montrés.

- La méthode `Hide` indique que les objets du type donné `<type>` dans la carte donnée `<mapget>` doivent être cachés.

- La méthode `isVisible` indique si les objets du type donné dans la carte donnée sont visibles.

ApicSetting

C'est l'outil qui permet de personnaliser la représentation des propriétés d'un objet.

Description de la classe

```
class ApicSetting :
{
public:
    void        setObjectPoint(ApicMapGet* mapget,const char* type,const char* name);
    void        setObjectSegment(ApicMapGet* mapget,const char* type,const char* name);
    void        setObjectColor(ApicMapGet* mapget,const char* type,const char* name);
    void        setObjectHeight(ApicMapGet* mapget,const char* type,IlvInt height);
    void        setObjectOffset(ApicMapGet* mapget,const char* type,IlvInt offset);
    void        setNameColor(ApicMapGet* mapget,const char* type,const char* name);
    void        setNameFont(ApicMapGet* mapget,const char* type,const char* name);
    void        setOrientationColor(ApicMapGet* mapget,const char* type,const char* name);
    void        setOrientation(ApicMapGet* mapget,const char* type,const char* name);
    void        setUncertaintyColor(ApicMapGet* mapget,const char* type,const char* name);
    void        setUncertainty(ApicMapGet* mapget,const char* type,const char* name);
    void        setCurveColor(ApicMapGet* mapget,const char* type,const char* name);
    void        setSequenceColor(ApicMapGet* mapget,const char* type,const char* name);
    void        setSequenceFontSize(ApicMapGet* mapget,const char* type,IlvInt size);
    void        setUseFalseText(ApicMapGet* mapget,const char* type,IlvBoolean state);
    void        setColoredMode(ApicMapGet* mapget,const char* type,IlvBoolean state);
    void        setColoredLetters(ApicMapGet* mapget,const char* type,IlvBoolean state);
    void        setAColor(ApicMapGet* mapget,const char* type,const char* name);
    void        setCColor(ApicMapGet* mapget,const char* type,const char* name);
    void        setGColor(ApicMapGet* mapget,const char* type,const char* name);
    void        setTColor(ApicMapGet* mapget,const char* type,const char* name);
};
```

- Les méthodes `setObjectPoint` et `setObjectSegment` permettent de choisir les objets graphiques qui vont représenter, comme un point ou comme un segment, les objets biologiques d'un type donné dans une carte.

- La méthode `setObjectColor` permet de choisir la couleur associée à un type d'objet dans une carte.

- La méthode `setObjectHeight` donne la hauteur associée à un type d'objet dans une carte.

- La méthode `setObjectOffset` permet de contrôler l'écartement vertical pour un type d'objet dans une carte.

- Les autres méthodes permettent de choisir la forme, la couleur ou la police de caractères associée à chaque propriété (nom, orientation, incertitude, courbe, séquence) d'un type d'objet.

Remarque : toutes les formes, couleurs et polices de caractères sont contenues dans des fichiers de ressources ILOG Views (ApicShape.ilv, ApicColor.ilv et ApicFont.ilv). Ces fichiers peuvent être modifiés et visualisés avec l'éditeur de dessin *ilvedit* d'ILOG Views. Il est alors facile de compléter ces différentes "palettes". Chaque forme, couleur et police de caractères est associée à un nom qui permet son utilisation dans l'interface.

ApicMap

C'est le moule dans lequel sont rangées toutes les caractéristiques d'un carte (Cf. Annexe 1).

Description de la classe

```
class ApicMap
{
public:
    const char*     format() {return _format;}
    const char*     date() {return _date;}
    const char*     name() {return _name;}
    const char*     origin() {return _origin;}
    const char*     description() {return _description;}
    const char*     unit() {return _unit;}
    lvDouble        scale() {return _scale;}
    IlvBoolean      isstarted() {return _started;}
    IlvDouble       start() {return _start;}
    const char*     resourcefile() {return _resourcefile;}
    IlvAList*       allfields() {return _allfields;}
    IlvDouble       accuracy() {return _accuracy;}
};
```

- Ces méthodes restituent les propriétés, relatives à la description générale d'une carte, qui sont définies dans l'annexe 1.

ApicObject

C'est le moule dans lequel sont rangées toutes les caractéristiques d'un objet (Cf. Annexe 1).

Description de la classe

```
class ApicObject
{
public:
    const char*     type() {return _type;}
    const char*     name() {return _name;}
    IlvDouble       begin() {return _begin;}
    IlvDouble       end() {return _end;}
```

```
    IlvBoolean      orientation() {return _orientation;}
    IlvDouble       deltabeginmin() {return _deltabeginmin;}
    IlvDouble       deltabeginmax() {return _deltabeginmax;}
    IlvDouble       deltaendmin() {return _deltaendmin;}
    IlvDouble       deltaendmax() {return _deltaendmax;}
    IlvPos          fixedposition() {return _fixedposition;}
    const char*     sequence() {return _sequence;}
    const char*     curve() {return _curve;}
    IlvAList*       allfields() {return _allfields;}
};
```

- Ces méthodes restituent les propriétés, relatives à la description des objets d'une carte, qui sont définies dans l'annexe 1.

ApicInterface

C'est la classe qui permet d'assurer la correspondance entre les objets externes (dans notre cas issus de fichiers au format TINA-1) et les objets affichés dans une fenêtre.

Description de la classe

```
class ApicInterface
{
public:
    ApicMap*        getMap(const char* path);
    ApicObject**    getObjects(const char* path);
    IlvHashtable*   getUnits();
    IlvHashTable*   getResources(const char* path);
    IlvBoolean      setResources(const char* path, IlvHashTable* resources);
};
```

- La méthode `getMap` récupère les principales propriétés associées à une carte (name, unit, etc.) et crée une instance de la classe ApicMap. Elle retourne un pointeur sur l'instance qui a été créée.

- La méthode `getObjects` récupère les principales propriétés associées aux objets (name, type, begin, end, etc.) qui composent une carte et crée des instances de la classe ApicObject. Elle retourne un tableau de pointeurs sur les instances qui ont été créées.

- La méthode `getUnits` récupère les différentes unités ainsi que les facteurs de conversion entre ces unités. Elle retourne un pointeur sur une table. IlvHashTable est une classe prédéfinie de la bibliothèque de classes d'objets ILOG Views pour gérer des tables de hash). Cette table a pour clé un couple d'unités (u1,u2) (sous une forme concaténée u1u2) et pour valeur le facteur de conversion qui lie ces deux unités.

- La méthode `getResources` récupère les différentes ressources définies pour une carte. Elle retourne un pointeur sur une table. Cette table a pour clé un attribut qui représente la ressource et pour valeur la valeur de cet attribut (Cf. Annexe 3).

- La méthode `setResources` sauve les ressources qui ont été définies pour une carte.