Cooperative computer system for genome sequence analysis

Médigue C.[1,2], Vermat T.[3], Bisson G.[4], Viari A.[2] and Danchin A.[1]

[1] Institut Pasteur CNRS URA 1129 Unité de Régulation de l'Expression Génétique
28, rue du docteur Roux, F-75724 Paris Cédex 15

[2] Institut Curie CNRS URA 448 Atelier de BioInformatique
11, rue Pierre et Marie Curie F-75231 Paris Cédex 05
E-mail : medigue@radium.jussieu.fr

[3] Laboratoire de Biométrie, Génétique et Biologie des Populations CNRS URA 243
43, Bd du 11 Novembre 1918 F-69622 Villeurbanne Cédex

[4] INRIA Rhône-Alpes — LIFIA
46, av. Félix Viallet F-38031 Grenoble Cédex

Abstract

Analysis of the huge volumes of data generated by large scale sequencing projects clearly requires the construction of new sophisticated computer systems. These systems should be able to handle the biological data as well as the results of the analysis of this data. They should also help the user to choose the most appropriate method for a simple task and to string together the methods needed to solve a global analysis task. In this paper we present the prototype of a software system that provides an environment for the analysis of large-scale sequence data. In a first approach this environment has been put to the test within the *B. subtilis* sequencing project. This system integrates both a descriptive knowledge of the entities involved (genes, regulatory signals etc.) and the methodological knowledge concerning an extendable set of analytical methods (i.e. how to solve a sequence analysis problem through task decomposition and method selection). A knowledge representation based on two existing object-oriented models, named Shirka and SCARP, is used to implement this integrated system. In addition, the present prototype provides a suitable user interface for both displaying the results generated by several methods and interacting with the objects. We present in this paper an overview of the knowledge-based models used to build this integrated system, and a description of the way in which biological entities and sequence analysis tasks are represented. We give illustrations of the co-operation between user and system during the problem solving process. Such a system constitutes a computer workbench for molecular biologists studying the genetic programs of living organisms.

Introduction[1]

As research programs involving large-scale genome sequencing develop, vast amounts of new genomic sequences are produced and the need for more reliable data management systems as well as for powerful sequence analysis tools becomes obvious. Sequences have been stored and organized in general data bases, in specialized data bases and more recently in object-oriented data bases. In parallel, various sequence analysis methods, gathered in software packages, have made it possible to detect regularities in sequences and to predict gene functions. The overall organization of such systems and methods which is essential to the progress of the sequencing projects remains however inappropriate.

In a preliminary step, each new genomic sequence should be analyzed automatically using a pre-defined combination of methods. As an example, searching for non ambiguous signals or for coding sequences, or making translation and comparison with protein data banks, are operations which can be completely automated. Further analyses require however a tight collaboration between system and user. Indeed, depending on the biological questions the biologist wishes to address, it is necessary to employ one particular method over all others. Algorithms devoted to high level methods are getting ever more powerful as time passes but they require to be used in a skilled way, that takes into account an explicit description of their specific problem-solving strategy (i.e., that considers whether a method is well adapted and appropriately used) and is able to interact with an user-friendly environment. Using these methods thus remains difficult for the beginner. In consequence he tends to use a small number of available softwares rather than elaborate methods thus losing important aspects of the information content of a genome sequence. Finally, the lack of appropriate links between the management of biological entities (such as sequences, genes, regulatory signals etc.) and the methods that use and produce these entities severely limits the evolution of the data bases and of the corresponding knowledge bases. In the context of the genome sequencing projects, computers should now be used as *experimental* tools, producing a new source of investigation of living organisms, their study *in silico* .

This paper describes the first prototype of a co-operative knowledge-based environment dedicated to genomic sequence analysis. It allows one to model and manipulate the descriptive knowledge obtained from a genome sequencing project, to help the user in solving his sequence analysis problems through task decomposition and method selection, and finally to display and manage the set of new objects this analysis creates. Our system integrates *descriptive knowledge* of the biological entities together with *methodological knowledge* of an extendable set of

[1] This research is being supported by the French Ministère de l'Enseignement et de la Recherche, by CNRS (Centre National de la Recherche Scientifique GDR 1029) and GIP GREG (Groupement de Recherche sur l'Etude des Génomes 2 93 G404 00 71201 21).

analysis methods. It is currently developed using a class-based knowledge model named Shirka (Rechenmann & Uvietta 1991). This system offers high level descriptive capabilities and enables to characterize the entities using inference and classification mechanisms. Based on Shirka, a second system, called SCARP (Willamowski 1994; Médigue et al. 1993) allows the representation of the methodological knowledge. In this system, the problem solving process is based on successive decompositions of the complex problem (here the central concept is that of a *task*). Finally a graphic interface called APIC (Bisson & Garreau 1995) has been developed in order to display, modify, save or delete the entities. The first section of this paper describes the general features of Shirka, SCARP and APIC. Several biological examples extracted from our prototype are used to illustrate this section. The second section presents our prototype system : we show how biological entities and tasks are represented and how we integrate each "external" procedure. At this stage we put emphasis on the methods allowing for the characterization of genes in an unannotated DNA sequence: searching for similar proteins in data banks, searching for unambiguous signals and for putative coding sequences. Several examples illustrate the cooperation between user and system during the problem solving process. Finally, we discuss improvements of previously described strategies, further developments of new ones and future enhancements of the prototype.

Methodological Background

In order to build a system that integrates descriptive and methodological knowledge, we used the object-based knowledge model Shirka (Rechenmann & Uvietta 1991), implemented and embedded into an interactive environment using ILOG tools (ILOG, 1991). Shirka has already been tested for the representation of genomic organization of *E. coli* (Perrière & Gautier 1993) and for mammal genomic mapping (Perrière et al. 1993). Extensions of this system to the representation of methodological strategies have been incorporated in a co-operative problem solving environment, called SCARP (Willamowski 1994). An example of application is the SLOT system which is devoted to data analysis in the field of ecology (Chevenet & Willamowski 1993). The following section gives an overview of these two systems and of their associated graphical interface functionalities.

Object-based knowledge representation

Shirka (Rechenmann & Uvietta 1991) rests on the *class/instance shift*, where a class describes a set of potential instances. Classes allow a formal definition of the model, and instances result from instantiation of the classes. Both are represented by a structure called a scheme, made up of a set of named *slots* which are themselves specified by constraining *facets* (Fig. 1A). Facets allow for both slot valuation and association with inference mechanisms such as default value, pattern-matching or procedural attachment. When instantiating a class, a slot may either receive a value which must satisfy the constraints that have been defined in the class, or its corresponding value may be missing (incomplete instance). A *complex instance* is an instance in which some slot values refer to other instances (Fig. 1B). Complex objects thus premits recursive linking.

A.

```
{ gene
   a-kind-of  =  simple-biological-object ;
   name          $a          string;
   keyword       $list-of  string;
   phenotype     $a          string;
   type          $a          symbol
                 $domain    CDS RNA rRNA tRNA
}
{ protein-gene
    a-kind-of =  gene;
    type         $domain CDS;
    codon-usage  $a       integer;
                 $if-needed
                  { class-cds
                      ident $var<- name;
                      class $var-> codon-usage };
    translation  $a protein
}
```

B.

```
{ aceE
   is-a          =  protein-gene;
   keyword       =  "acetate";
   type          =  CDS;
   translation    =  prot-aceE
}
```

Figure 1 — Structure of classes and instances as knowledge description units : **A.** The two classes named **gene** and **protein-gene**, are made of a list of slots (keyword, type...). Each slot is described by a list of facets ($a, $domain...) together with their associated values. The $if-needed facet allows association of a slot valuation with a method. The displayed method is described by a class (class-cds) in which a particular slot is valued with the name of a function. The **protein-gene** class is a sub-class of the **gene** class : the inherited slot type is constrained to the value CDS, and the structure of this sub-class is defined with two new slots, codon-usage and translation. **B.** One instance of the class protein-gene, named **aceE**, is a complex instance; its slot translation is valued by an instance of the class **protein** according to the description of the class.

Scheme structures allow one to describe the position of a class in a specialization hierarchy. Each hierarchy forms a *concept* (Fig. 2). A class inherits the knowledge defined in its super-classes. It can further define constraints on inherited slots or simply add new specific slots. Since classes are organized in hierarchies, once an instance has been attached to a class, a natural mechanism consists in looking for its possible locations in the corresponding sub-classes. This is achieved through the classification mechanism. The algorithm attempts to match the instance to classify with every direct sub-classes of the instantiation class by checking if the slot values still satisfy the constraints of the chosen sub-class. The sub-class will be labeled *sure* if the instance satisfies all the constraints on all the slots; it is labeled as *possible* if no constraint in the sub-class is violated; it is *impossible* if at least one constraint is not satisfied. The algorithm then iterates on every sure or possible sub-class and ends at the leaves of the hierarchy.

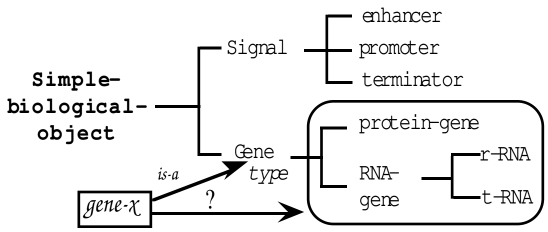

Figure 2 — Hierarchical structure of the **simple-biological-object** concept extracted from our knowledge base. This concept gathers elementary information units like protein and tRNA genes and the signals involved in their regulation (promoters, terminators, …). From an initial instance of the Gene class (named *gene-x*), the classification mechanism allows to determine the sub-classes to which this instance may be attached. Unknown values of the slots are inferred or provided by the user, and then compared with local constraints attached to the slots in the sub-classes. This classification mechanism defines either *sure*, *impossible* or *possible* classes. For example if the *type* value is RNA, a sure class is RNA-gene, possible classes are tRNA and rRNA, and an impossible class is protein-gene.

Representing methodological knowledge

Methodological knowledge includes the description of problems and their respective solving strategies (i.e., given a problem, how to select the appropriate (liking of multiple) processing method(s)). Different approaches to task-oriented representation are discussed in (Chandrasekaran et al. 1992).

The SCARP system (Willamowski 1994) is based on the Shirka model. In SCARP, a task is a model of a problem described by a specific set of input and output entities. Like descriptive knowledge, tasks are represented by classes (Fig. 3). Within SCARP, the following types of slots are defined for a task : global and strategic input, output entities, pre-tasks and post-tasks, visualization tasks for input and output, preconditions and postconditions. One problem may be defined at different abstraction levels by different, more or less specific task descriptions. At a low abstraction level, the problem defined by a task is precise enough so that a problem solving **strategy** can be associated with the task description (this is the case of **cds-prediction-using-codon-freq** in Fig. 3). The problem solving strategy describes how a task can be decomposed into more elementary subtasks or methods. Different operators are available to combine subtasks : sequence, parallel, iteration, user-choice etc. Finally, for each subtask a specific execution **context** describes how its execution is integrated into the execution of the complex task : data flow, whether the user has to validate input or output, etc. (Fig. 3). The resolution of each particular problem thus results in a very context-specific linking of methods.

```
{ cds-prediction-using-knowledge
  a-kind-of    =  cds-prediction ;
  &input  seq        $a      experimental-data;
  &input  window     $a      integer;
  &input  shift      $a      integer
       $if-needed     {...};
  &input  table      $a      data-file ;
  &output list-CDS   $list-of  CDS   }

{ cds-prediction-using-codon-freq
  a-kind-of =  cds-prediction-using-knowledge;
  &input  table      $to-verify  {...};
  strategy  $value  (user-choice   CODONPREFERENCE
                                   GENMARK  ) ;
 &stache CODONPREFERENCE
      $a { context
         exec    $a   CODONPREF-ANALYSIS ;
         data-flow $value
            (.seq       >  exec.seq )
            (.wind      >  exec.wind)
              ...
            (exec.res  >  .list-cds) };
}
```

Figure 3 – Examples of classes representation partially illustrating the **cds-prediction-using-knowledge** and **cds-prediction-using-codon-freq** tasks. The global input entity (seq) must be an instance of the **experimental-data** class and the global output (list-CDS) are instances of the coding-sequence class (**CDS**). The operator *user-choice* used in the strategy of **cds-prediction-using-codon-freq** task, allows one to combine the subtasks CODONPREFERENCE and GENMARK. We show here the specific execution context of the CODONPREFERENCE subtask: the elementary task CODONPREF-ANALYSIS is executed using the data flow indicated in parentheses.

As tasks are represented by object classes, they are organized in class hierarchies (Fig. 4). Inside a hierarchy, each class models a task at different abstraction levels, defining the overall problem solving strategy according to the specific problem context. This structure can then be used to support the choices among the available subtasks during the solving process. This is done using the classification mechanism.

To automatically handle the problem-solving process, an appropriate mechanism, the task engine, has been constructed. Four essential steps are distinguished in task or method execution : in the first one the task is instantiated and the input slots of the instance are filled in; in the second step the task is fitted to a given context and attached to the most adapted sub-class (this is performed through the classification mechanism, see Fig. 4); the third step consists in the decomposition of the task into its subtasks (each subtask in turn may refer to either a complex task or a method); in the last step the results are determined.

Via the task engine the system is able to automatically solve, on its own, complex problems. However, specific user-system interactions may also be required. In SCARP, every communication between the task engine and the user is handled by a dialog manager via a user interface. Whenever the task engine needs information from the user, it activates the dialog manager (for example, to get parameter values). While the task engine is working, cooperative dialog manager can also be activated by the user. The user interface of SCARP graphically displays the

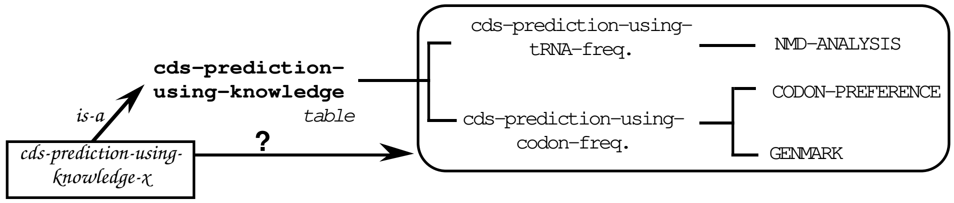

Figure 4 – Partial hierarchy of task classes representing the family of CDS prediction tasks. At the highest abstraction level, the task actually describes a very general goal (**`cds-prediction-using-knowledge`**). No specific solving strategy is associated with this task description : then using the classification mechanism, an instance of this task class (named *cds-prediction-using-knowledge-x*) has to be specialized. Here, this specialization is driven by the value of the *`table`* slot (see also Fig. 3). By contrast, at a lower abstraction level (for example the **`cds-prediction-using-codon-freq`** subtask) the problem is precise enough so that a decomposition scheme can be directly associated with the task description (see also Fig. 3).

current state of the solving process, i.e. the current task decomposition tree. As a consequence, the user may at any time modify the characteristics of the task, either parameter values or strategic choices. For each modification, a new version of the current solving process is created, saving unmodified parts of the precedent version, and the task engine then re-executes the parts of the problem related to the modification. Finally, the interface enables the user to define new problems and strategies. He can define new tasks either from scratch or by modifying existing tasks. This process for creating or modifying tasks concerns both their input/output and their decomposition strategy. Tasks may therefore be integrated into the knowledge base and executed at a later stage.

The graphical interface

An analysis task can produce a great variety of results, such as new biological entities, drawings etc. For this reason we developed a generic interface called APIC (Bisson & Garreau 1995). The word generic has here two different meanings. First, this interface is able to display all kinds of maps (genetic map, physical map, etc). Second, APIC is not limited to the display of biological entities but is also able to display graphical objects such as plots. The advantage of using such a generic approach is two-fold. On the one hand, APIC brings a standard interface displaying the information in an homogeneous way; therefore the comparison of the results is made easier. On the other hand, the interaction with the user is made simpler because he has only to manipulate a single general interface instead of several dedicated interfaces (remember the problems generated by the piling up of a lot of windows !).

A precise description of the interface's functionalities is out of the scope of this paper. Here are however a few relevant aspects. To compare and explore several displayed maps, APIC provides several classical functionalities such as selecting information to display, zooming, accessing the knowledge associated with an object by clicking on it (an example is shown in the next section, Fig. 11). It also provides more sophisticated functions such as synchronized scrolling, which allows to simultaneously scroll two or several maps having different scales and units (for instance a genetic and a physical map) while automatically keeping the relation between these maps. As in the SIGMA interface (Cinkosky & Fickett 1992), the user can also organize the objects into different layers. Moreover, in a given layer it is possible to superimpose several kinds of information, for example a plot with a physical map.

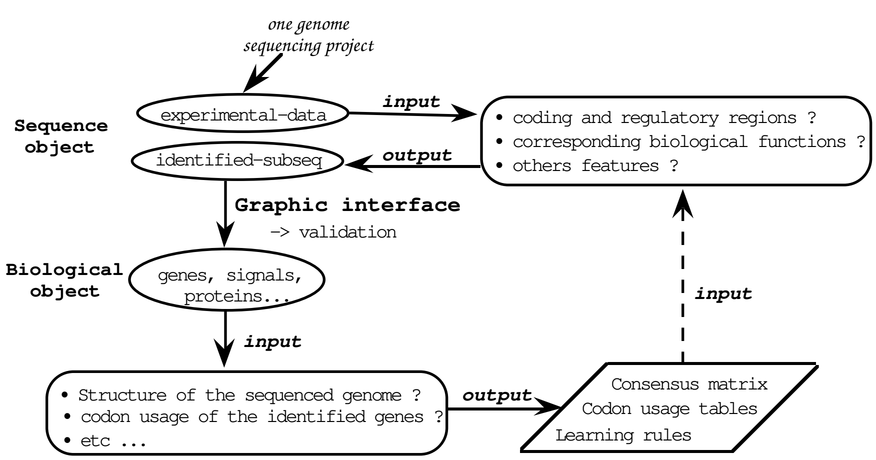

Figure 5 – Contribution of the prototype system in the context of the sequencing projects. New DNA sequences are submitted to a first series of treatment for identification of basic signals and comparison of putative genes product to the protein data banks. The entities thus identified are submitted to the agreement of the biologist through an interface that permits to visualize results, localize possible sequencing mistakes, etc. Others methods create "descriptors" of the studied genome (consensus matrices, codon bias...). These new descriptors are used in a second step to identify the objects which have not been characterized during the first step.

Finally, many aspects of the interface can be customized by the user without programming. In this way, he can modify the shape of the objects, their color, their size etc.; all these settings can then be saved and subsequently reused.

Results

The first prototype of a co-operative sequence analysis environment has been developed having as base the systems Shirka and SCARP. Since our laboratory is producing large amounts of DNA sequences in the context of the *B. subtilis* genome sequencing project (Glaser et al. 1993; Kunst et al. 1995; Ogasawara et al. 1995), we used these data to test the prototype. With such a simple prokaryotic model, the analysis of new sequences can be organized in *at least* two steps (Fig. 5). The first one is devoted to the detection of known structures such as long open reading frames and strict DNA signals. This step can be fully automated. It generates entities that can be compared to sequences present in genomic and protein data banks and then submitted for validation to the user. A second step involves more sophisticated analysis tools (multivariate analysis on codon usage, learning methods on signals etc) that study less known objects wich are more difficult to characterize.

To build a co-operative sequence analysis environment, we first establish a representation of the biological knowledge, i.e. of the object classes used and produced by the methods. Then SCARP is used to represent the sequence analysis methods. In the next section, we present an example of a sequence of methods employed to analyse an unannotated DNA sequence. We show how the results are compared with each other in the graphic interface. Finally, we describe how the external procedures are executed.

Representing biological knowledge

Biological entities have been represented in two main specialization hierarchies (Fig. 6). The first one forms the "sequence object" concept; it gathers object classes describing nucleotide and amino acid sequences together with their corresponding features. The DNA sequences come from data banks or individual laboratories. When sequences overlap with two or more neighbouring sequences, we generate longer DNA sequences ("contigs") stored in the "experiment>1" class (Fig. 6A). The DNA features are either features coming from data bank annotations or identified from subsequences produced by a previous sequence analysis. Starting from the ColiGene knowledge base (Perrière & Gautier 1993), we define a "biological object" concept that describes the common properties of a set of biological structures (genes, regulatory signals, operons etc. Fig. 6B). Object classes of this specialization hierarchy refer to validated entities of the "sequence object" concept.

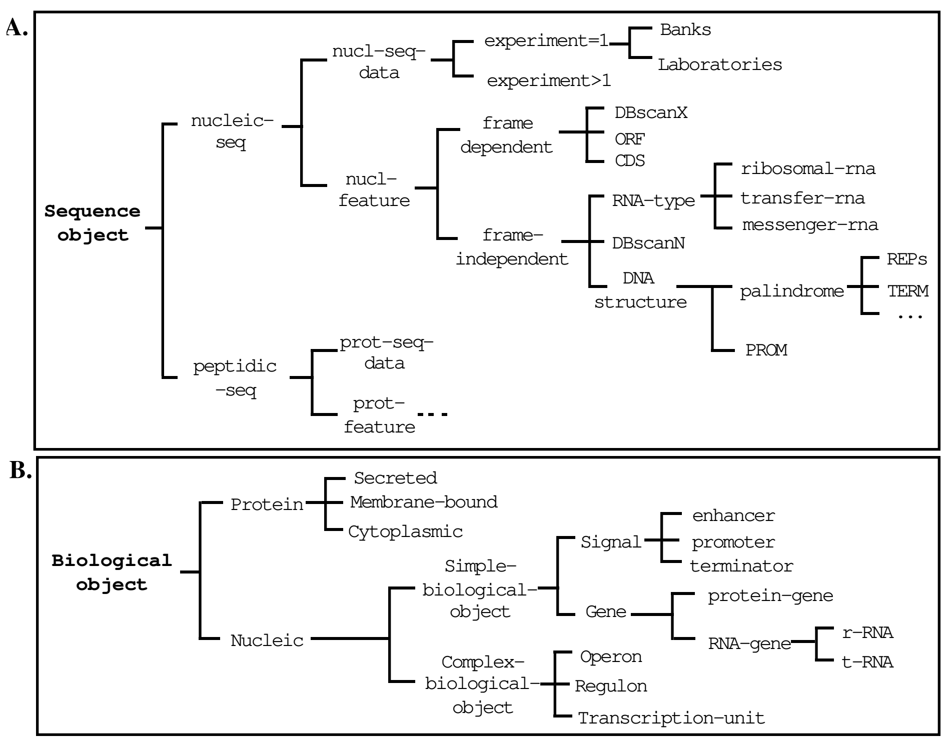

Figure 6 – Part of the hierarchical structure of the knowledge base developed with Shirka. Part A displays the hierarchy of the classes corresponding to the `Sequence-object` concept. It contains nucleic sequences and amino acid sequences translated from coding regions. In B, the `Biological-object` concept gathers elementary information units like protein, genes and the signals involved in their regulation, and finally classes for complex objects like operons and regulons.

In a previous work, we developed a specialized data base for the *E.coli* genome (Médigue et al. 1993). There, we defined the organization of the data, the structure of the data base and the environment of consultation and interrogation. From that model, another specialized data base for the *Bacillus subtilis* genome, called SubtiList, was built (Moszer et al. 1995). Data present in these bases permit us to extract clean, consistent and non redundant sequences to instantiate the object classes of our biological model. This provides us with the best conditions to extract information from sequence analysis.

Representing sequence analysis methods

To construct a model of the methodological knowledge contained in sequence analysis programs, task classes have been defined for the analysis of a single sequence on the one hand, and for the analysis of several sequences on the other hand (Fig. 7). Currently the first set of tasks brings together database scanning (Blast and Fast algorithms), basic statistics tools and pattern matching techniques including consensus and learning methods. CDS prediction analysis is specialized depending on whether a reference system (such as codon usage table) is required or not. The **n-sequences-analysis** task hierarchy includes tools used to build "descriptors" (consensus matrices, learning rules etc.) which can subsequently be used by the former tasks. We are also working on the integration of 2 or n sequences alignment methods with algorithms developed in our laboratory (Sagot et al. 1995). Combining two main groups of tasks of the latter with a toolbox containing elementary methods, we defined another specialization hierarchy allowing one to construct a representation of more specific strategies (Fig. 7). A combination of tasks permitting the analysis of an unannotated DNA sequence is then described; it is meant to identify signals such as regions coding for proteins and their corresponding translation initiation site. This latter specific analysis can be done either automatically or interactively. In the first case, at the beginning of the procedure, a subtask asks for all the parameters values necessary to the linked methods. To illustrate some specific user interactions developed in SCARP, each step of this analysis is now detailed.

One of the first questions that arises once a long newly sequenced stretch of DNA has been obtained is whether it contains translated coding regions identical or similar to already known proteins. The data bank scanning program BlastX (Altschul et al. 1990) has been integrated in our knowledge base to answer this question. The problem solving strategy associated to the **blastx-analysis** task consists first in executing an external procedure (see below) on the network or on the local workstation (Fig. 8). The user's choice is guided by an estimation of the BlastX running time. The results of the first step are then filtered on the basis of the percentage of identities between the query and the selected proteins. This is done by a subtask which is executed until the user is satisfied. To help the user in his decision, the instances which are created in the DBscanX class (see Fig. 6) are represented in a graph. Finally, stop codons are searched for and the final graph represents for the 6 reading frames the sub-sequences of the query which give translation products similar to proteins in data banks.

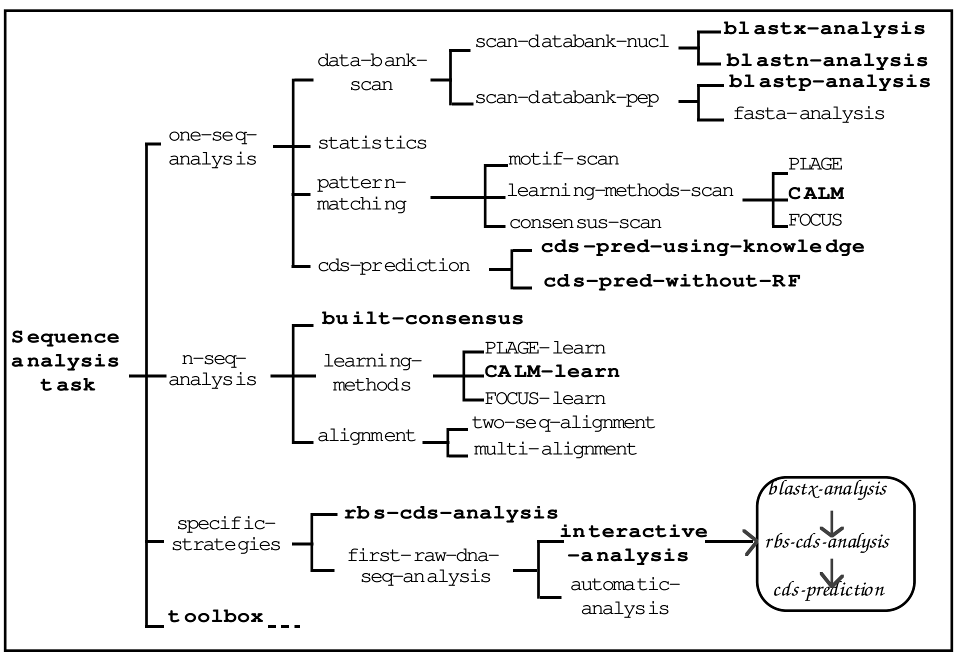

Figure 7 – Part of the hierarchical structure of the knowledge base developed with SCARP. The first set of tasks gathers analysis methods for a single sequence (`one-seq-analysis`) and the second set gathers analysis methods for several sequences (`n-seq-analysis`). We then define specific strategies allowing one to study an unannotated DNA sequence (see text). The problem solving strategy of the `interactive-analysis` task is circled. Subtasks of the `toolbox` task are elementary methods such as reverse-dna, translate-cds etc. Currently, only the tasks indicated in bold are fully functional.

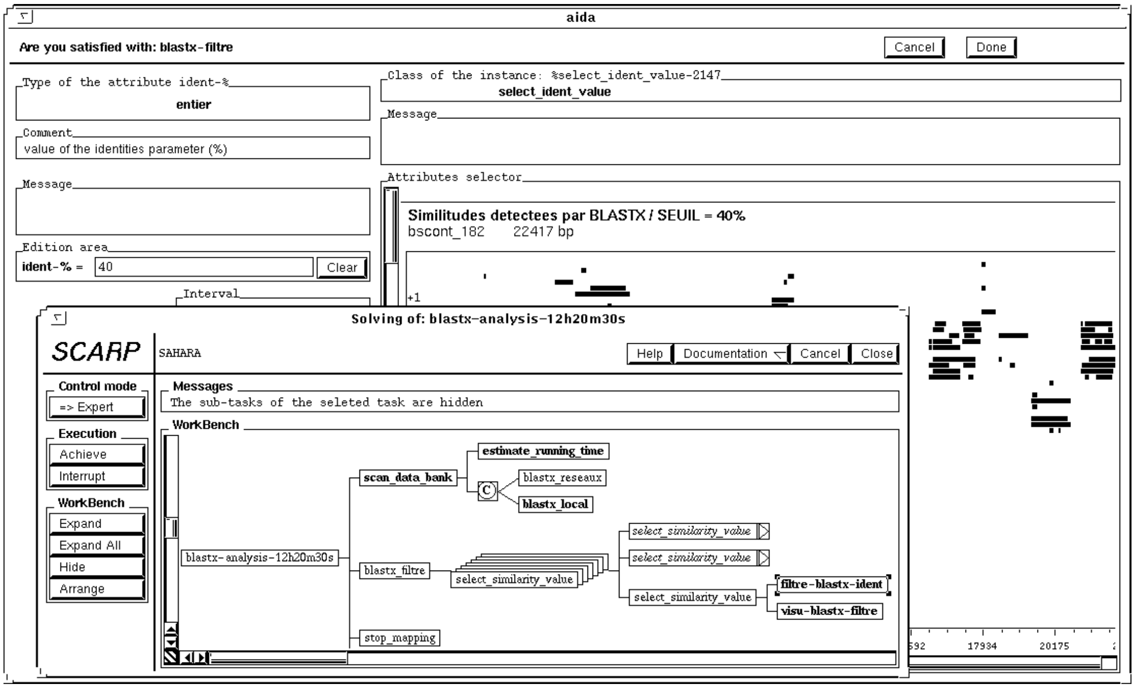

Figure 8 – Execution of the **`Blastx-analysis`** task with a *B. subtilis* contig (bscont_182). The frontmost window shows the current state of the solving process. The task is decomposed into several subtasks (see text) and the one beeing actually executed is `select-similarity-value`. The results presented in the other window were obtained with a value of `%ident` equal to 40. The system is now waiting for the user decision : either he keeps these results or changes again the `%ident` value. In this latter case, the resolution of the `select-similarity-value` task is considered to have failed (indicated in italics) and another instance of this task is created and executed.

Sequences coding for proteins are defined by their specific codon usage and by the presence, upstream of the start codon, of a Ribosome Binding Site (RBS) (Shine & Dalgarno 1974). This was the first example of a "consensus" sequence for a signal that has been clearly demonstrated as relevant to translation initiation. In the particular case of *B. subtilis*, the RBS are very well conserved regions. A simple consensus sequence can therefore be used for identifying CDSs in bacterial genomes, at least during the first round of gene identification. The first subtask of the **rbs-cds-analysis** task either searches for open reading frames (ORF, i.e a region of 3n nucleotides limited by a translation termination codon, UAA, UAG or UGA) or extracts ORFs from the knowledge base if instances of this class already exist for the studied sequence (Fig. 9). Then identification of translation initiation sites requires comparison with a pattern or a consensus matrix specific for this signal. In the latter case, the **build-consensus** task should be executed. Its purpose is to extract from all the validated instances of the RBS class a consensus matrix. Finally, newly created instances of the CDS class are visualized on a graph. As shown in Figure 9 navigation facilities are also available.

To finish this unannotated DNA sequence analysis, the execution of one or more CDS prediction methods is required. In spite of the fact that many of them have been published, there is no unique answer to this problem yet. An instance of the **cds-prediction** task must be specialized depending on whether a reference system to identify coding regions is available or not (Fig. 4 and Fig. 7). As an example of methods that do not use knowledge on the sequenced genome are the one by Fichant and Gautier which is based on Factorial Correspondence Analysis (Fichant & Gautier 1987) and the one by Ollivier et al. (1990) which is based on Fourier analysis. In the case of prokaryotic genomes, the problem is simplified since the codon usage derived from a set of known genes has been found to be strongly biased. However, the CDS prediction methods which require codon frequency tables should be used with caution as we found in a previous work that there is at least one class of genes in *E. coli* which escapes such identification, namely the class of genes exchanged by horizontal transfer (Médigue et al. 1991). The GenMark software based on Markov chain models (Borodovsky & McIninch 1993) gives efficient CDS prediction provided that genes have first been clustered into significant classes used as training sets (Borodovsky et al. submitted). We therefore integrated this method in the specialization hierarchy of the CDS prediction task (Fig. 4). The first part of the GenMark analysis task uses information about the sequence to choose the most appropriate matrix. The method is then applied on a sliding window and the results are plotted on a graph. An example of these results is given in the next section.

Comparison of the results : the graphical interface

We developed several specialized interfaces to visualize the tasks outputs (Fig. 8 and 9). However, as analysis methods accumulate in the knowledge base it becomes

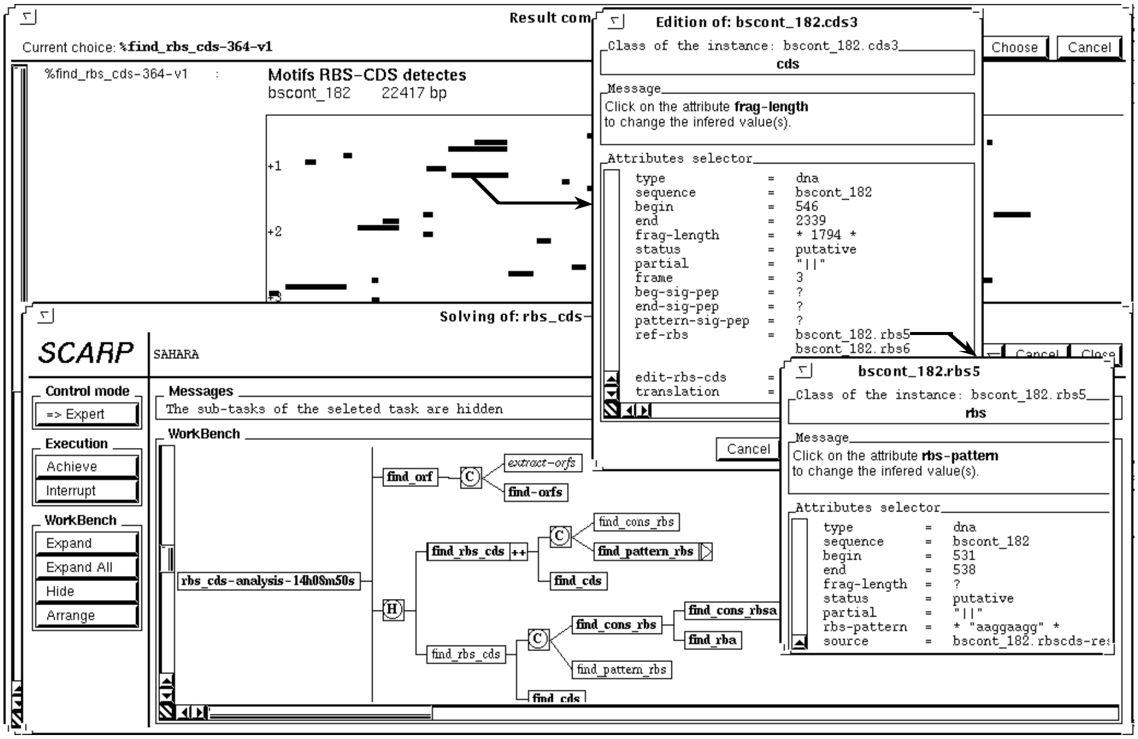

Figure 9 – Execution of the **`rbs-cds-analysis`** task with a *B. subtilis* contig (bscont_182). Here we show how the user can modify a previously made stategic choice (`find_pattern_rbs`). The dialog manager creates a new version of the current solving process (indicated by a ++ symbol and a H (for Hypothesis) in the resolution tree) and the task engine re-executes the part of the problem solving process which is related to the modification (`find_cons_rbs`). The system is now waiting until the user chooses between the two versions (`Result comparison` window). This choice is guided by a graphical representation of the CDS class instances which have been created on the 6 reading frames. By clicking on a box of this graph, the user can see the corresponding instance (`bscont_182.cds3`) and can use the navigation facilities to get the related information (bscont_182.rb5).

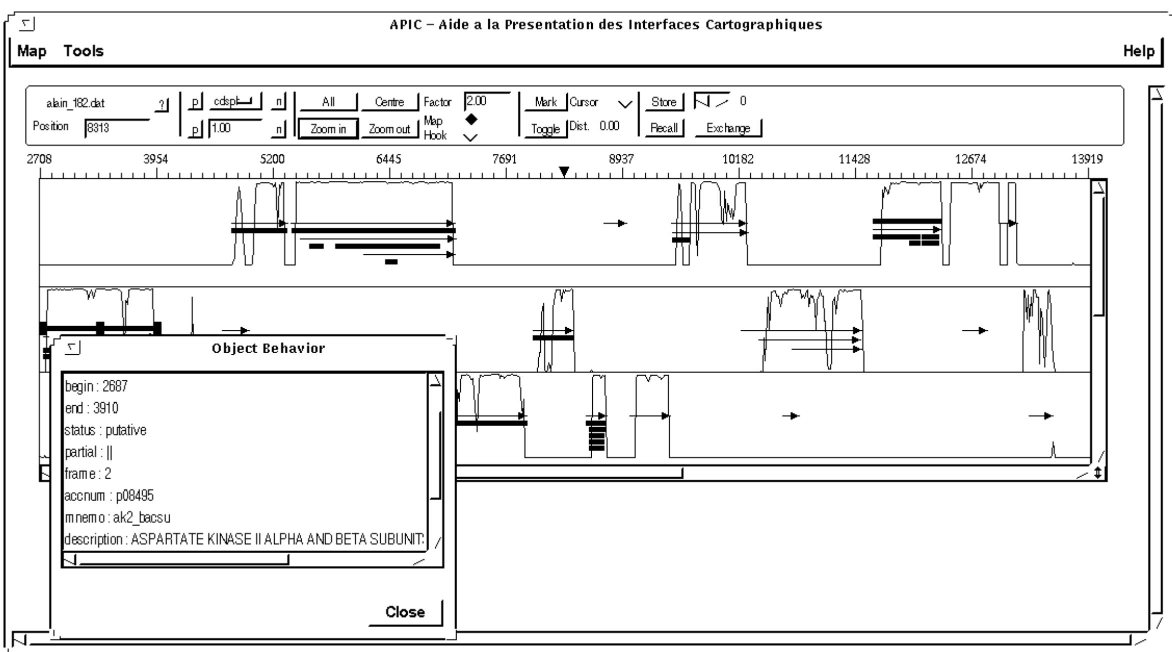

Figure 10 – Integration, in the graphical interface APIC, of the blastx-analysis results (black rectangles), of the rbs-cds-analysis results (arrows) and of the cds-prediction results (curves obtained with GenMark) on 3 reading frames of the bscont_182 contig. This representation shows that some significant BlastX similarities are missing in regions where coding regions and "strong" translation initiation sites are nevertheless found.

crucial to provide a more generic interface to display and compare the results.

The generic interface APIC (Bisson & Garreau 1995) has been developed for this purpose. We show in Figure 10 how the results of the different tasks can be superimposed. on each other. In this example, the map represents the entities found by a blastx-analysis task, a rbs-cds-analysis task and a cds-prediction task. In this map, we defined three different layers associated with the three reading frames where the rbs, cds and blastx entities were found.

Calling external procedures

As shown in the previous examples, several procedures are organized within a complex task to achieve a goal. The task description is written in the task engine's own language (currently in LISP). The final procedures however can be written in any language. This allows for a very easy and rapid integration of any existing software into the system. As it is explained below, no modification of the original software is needed and the integration can be performed even if the original sources are not available (this was the case for GenMark for instance).

The procedure is first launched via a server using the Unix RPC (Remote Procedure Call) protocol. The external program (foreign program) is not called directly but is strictly isolated by an encapsulating shell (procedure shell) which is responsible for handling the specific calling conventions and data conversions of the program. To ensure the independence between the program input format and the data representation within the database and to rationalize the data exchange format, the objects holding the data necessary to the external program are dumped into a "mailbox" in a normalized (albeit simple) form (Tina1).

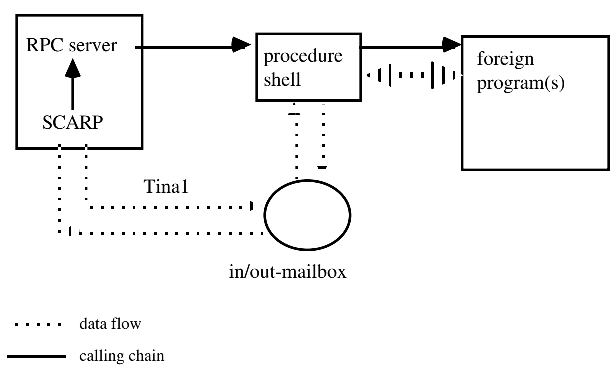

Figure 11 – The external calling procedure mechanism.

The procedure shell then performs the following steps :

- -1- read data from input mailbox (Tina1 format)
- -2- filter the components of the objects which are needed for the program execution and put these data in the required format.
- -3- run the foreign program(s)
- -4- reformat program(s) output and write the result in the output mailbox.

The whole process can be run synchronously (the whole SCARP task is then suspended, waiting for the procedure completion) or in background (the task continues and will be notified, through the RPC server, of the procedure completion). This is particularly useful when the actual subtask consists in sending a request to a foreign host through the network.

Conclusions

Sequence analysis is becoming an important activity in the efforts of the international research community to describe and understand the structural, functional and evolutionary relationships of model organisms. In this context, our goal is to design and implement a sophisticated computing system which incorporates advanced knowledge representation facilities and a comprehensive set of sequence analysis methods. The prototype presented in this paper already presents some original functionalities that can handle the genetic and biological knowledge produced by a complete genome sequencing project. We showed how tasks allow one to represent methodological and procedural knowledge. The resulting knowledge bases are easy to read and understand, and therefore easy to maintain. Moreover, as the problem solving processes are presented in a synthetic form, a real cooperation can be established between the user and the system.

Our system will include in the near future a great variety of sequence analysis methods including original ones, together with their strategies of use. Our immediate goal is to improve some of the previously described strategies. For example, in the solving strategy of the blastx-analysis task, we shall include some additional filtering steps *before* scanning the database (Claverie & States 1993) and *after* the process. Sonnhammer et al. (1994) and Scharf et al. (1994) have described interesting solutions to reduce the tedious browsing of large quantities of protein similarities. However, none of these approaches is integrated into a cooperative environment. Other tools, such as promoter prediction, are also needed. Promoters are defined as the binding sites of an RNA polymerase, active for the transcription initiation. Automatic identification of promoters usually involves two consensus sequences located at -10 and -35 regions upstream of the RNA start site. In reality, the spatial structure of the DNA/RNA molecule is probably involved in the recognition of a promoter by an RNA polymerase. Therefore, special methods must be designed for predicting promoters as well as other DNA/RNA structures recognized by appropriate factors. In this context, we are currently working on the integration in our system of methods allowing to characterize DNA/RNA secondary structures such as RNA genes (Fichant & Burks 1991), terminator sites of the transcription (d'Aubenton et al. 1990) and others palindromic structures.

The final aim of this work is a complete ready-to-use and extendable co-operative sequence analysis environment. The system is very modular and its high level knowledge representation language will provide extension facilities that will allows for the addition of new methods or for the modification of the solving strategy of a complex analysis task.

References

Altschul S. F., Gish W., Miller W., Myers E. W., and Lipman D. J. 1990. Basic local alignment search tool. *J. Mol. Biol.*, 215 : 403-410.

Aubenton C. Y., Brody E., and Thermes C. 1990. Prediction of rho-independent *E. coli* transcription terminators. A statistical analysis of their RNA stem-loop structures. *J. Mol. Biol.* 216: 835-858.

Bisson G. and Garreau A. 1995. APIC : a generic interface for sequencing projects. These proceedings.

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

Borodovsky M., McIninch J., Médigue C., Rudd K. and Danchin A. Detection of new genes in the bacterial genome using Markov models for three gene classes (submitted to NAR).

Chandrasekaran B., Johnson T. R., and Smith J. W. 1992. Task Structure Analysis for Knowledge Representation. *Communications of the ACM*, p. 124-137.

Chevenet F. and Willamowski J. 1993. Modelling methodological knowledge in data analysis. 13th International conference on *Artificial Intelligence, Expert systems, Natural language*, Avignon.

Cinkosky M. J. and Fickett J. W. 1992. SIGMA: System for Integrated Genome Map Assembly, User's Manual. Los Alamos, NM 87545.

Claverie J. M., and States D. J. 1993. Information enhancement methods for large scale sequence analysis. *Computers Chem.* 17 : 191-201.

Fichant G., and Gautier C. 1987. Statistical method for predicting protein coding regions in nucleic acid sequences. *Computer Application in BioSciences,* 3 : 287-295.

Fichant G., and Burks C. 1991. Identifying potential tRNA genes in genomic DNA sequences. *J. Mol. Biol.* 220: 659-671.

Glaser P., Kunst F., Arnaud M., Coudart M. P., Gonzales W., Hullo M. F., Ionescu M., Lubochinsky B., Marcelino L., Moszer I., Presecan E., Santana M., Schneider E., Schweizer J., Vertes A., Rapoport G., and Danchin A. 1993. *Bacillus subtilis* genome project: cloning and sequencing of the 97 kb region from 325° to 333°. *Mol. Microbiol.*, 10 : 371-384.

ILOG. 1991. AIDA: an environment for the development of graphical interfaces, User's Manual. ILOG ed, Gentilly, France.

Kunst F., Vassarotti A. and Danchin A. 1995. Organization of the European *Bacillus subtilis* genome sequencing project. *Microbiology,* 141 : 261-268.

Médigue C., Rouxel T., Vigier P., Hénaut A., and Danchin A. 1991. Evidence for horizontal gene transfer in *Escherichia coli* speciation. *J. Mol. Biol.* 222 : 851-856.

Médigue C., Viari A., Henaut A., and Danchin A. 1993. Colibri : a functional database for the *Escherichia coli* genome. *Microbiol. Rev.*, 57 : 623-654.

Médigue C., Willamowski J., Chevenet F. and Rechenmann F. 1993. Representation tasks for problem solving in molecular biology. Proceedings of the IJCAI'93 Workshop *Artificial intelligence and genome*. pp 67-76.

Moszer I., Glaser P., and Danchin A. 1995. SubtiList: a relational database for the *Bacillus subtilis* genome. *Microbiology.*, 141 : 261-268.

Ogasawara N., Fujita Y., Kobayashi Y., Sadaie Y., Tanaka T., Takahashi H., Yamane K., and Yoshikawa H. 1995. Systematic sequencing of the *Bacillus subtilis* genome: progress report of the Japanese group. *Microbiol.ogy,* 41 : 257-259.

Ollivier E. 1990. Analyse de séquences biologiques : développement d'une approche indépendante d'un système de référence par traitement du signal. Ph.D. *Thesis*, Université Paris 6.

Perrière G. and Gautier C. 1993. ColiGene: Object-centered representation for the study of *E. coli* gene expressivity by sequence analysis. *Biochimie*, 75 : 415-422.

Perrière G., Dorkeld F., Rechenmann F., and Gautier C. 1993. Object-oriented knowledge bases for the analysis of prokaryotic and eukaryotic genomes. Proceedings of the First International Conference on *Intelligent Systems for Molecular Biology*. pp. 319-327.

Rechenmann F., and Uvietta P. 1991. Shirka - An object-centered knowledge base management system, in *Artificial Intelligence in Numerical and Symbolic Simulation*, ALÉAS Publ, Lyon.

Sagot M. F., Escalier V., Viari A., and Soldano H. 1995. Searching for repeated words in a text allowing for mismatches and gaps. Second South American Workshop on String processing, Vinas del Mar, Chili, pp. 87-100.

Scharf M., Schneider R., Casari G., Bork P., Valencia A., Ouzounis C., and Sander C. 1994. GeneQuiz: a workbench for sequence analysis. Proceedings of the Second International Conference on *Intelligent Systems for Molecular Biology*. pp. 348-353.

Shine J., and Dalgarno L. 1974. The 3'-terminal sequences of *Escherichia coli* 16S ribosomal RNA complementary to nonsense triplets and ribosome binding sites. *Proc. Natl. Acad. Sci. USA* 71 : 1342-1346.

Sonnhammer E. L., and Durbin R. 1994. A workbench for large-scale sequence homology analysis. *Computer Application in BioSciences,* 10 : 301-307.

Willamowski J. 1994. *Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur.* Ph.D. Thesis, Université Joseph Fourier, Grenoble 1.