11th European Knowledge Aquisition Worrkshop 1999

Distributed Problem Solving Environment Dedicated to DNA Sequence Annotation

Thibault Parmentier[1], Danièle Ziébelin[1]

[1] Projet Sherpa, INRIA Rhône-Alpes, 655 Avenue de l'Europe, F-38330 MONTBONNOT - FRANCE.
{Thibault.Parmentier, Daniele.Ziebelin}@inrialpes.fr
http://www.inrialpes.fr/sherpa/

Abstract. Genomic sequence analysis is a task using techniques coming from different fields in order to extract biologically relevant objects (genes, regulatory signals...) from rough DNA sequences. Analysis methods, coming from domains like statistics, sequence alignment or pattern matching, have been developed and regrouped into program libraries. These libraries aim at helping biologists to manipulate data. However their use revealed itself being too difficult as it requires the user to have background knowledge to be handled efficiently. In order to tackle this problem, the ImaGene system has been proposed. ImaGene is a system built upon a generic task model allowing to model methods of DNA sequence analysis and to execute them thanks to shell scripts, binaries and specific libraries already developed. This system makes it possible to manipulate these methods and present a synthesis of the obtained results in a cartographic interface allowing the biologist to evaluate the biological pertinence of the results and to annotate DNA sequences. In order to profit from analysis methods and specialized libraries and to simplify their accesses, we have provided ImaGene with the possibility of using distributed methods and binaries. The solution chosen consists in transferring task code from server sites to client sites; but, unlike the solution adopted in Java, the data is computed at the server site if necessary. This distribution of the system has highlighted some problems like managing different versions of methods and dealing with tasks that have been recently decomposed into new sub-tasks.

1 Introduction

The use of specialized libraries of programs is not always easy: to choose, connect and carry out the suitable modules during the resolution of a problem often requires, the user to have an excellent control of the field concerned and a good knowledge of the organization of the libraries. This is for example true in domains like automobile or plane design, in which competences from several disciplines are requested. It is also true in genomic sequence analysis where computer science techniques used to manipulate genomic sequences meet techniques from statistics and pattern matching used to locate and extract biologically relevant objects. These techniques and methods are regrouped in software libraries or binaries. Though biologists are able to set

T. Parmentier, D. Ziébelin, Distributed Problem Solving Environment Dedicated to the DNA Sequence Annotation, *Proceedings of the 11th European Workshop (EKAW'99)*, Dagstuhl Castle, Germany, May 1999.

parameters of the distinct methods selected from the libraries, they need more competences to select the right modules of a library or to link them up.
In order to help users with these difficulties, Problem Solving Environments (PSEs) have been developed which offer automatic supervision of software libraries and binaries. To do that, the PSEs are based on solving strategies (methodological knowledge) easily comprehensible by users. The PSEs use this methodological knowledge to identify, specialize and execute the modules best adapted to the current problem. Users can control the solution process by means of graphical interfaces allowing them to visualize all stages of the execution of a strategy and the results.
The objective of designing a PSE is to provide an integrated environment and to supervise programs [18] coming from different specialized domains. A PSE proposes a model of each domain through knowledge bases representing [19]:

- on the one hand, domain entities (cf. Fig. 1) which we will call them domain knowledge. They represent the concepts being handled (variable, domain of definition, scatter plot, grid,...),
- on the other hand, problem solving strategies modeling different ways to solve problems (cf. Fig. 1). This methodological knowledge expressed thanks to strategies allows an evaluation of the problem to be solved. They lead to the selection, the sequence and the execution of the modules or programs of a library in the most relevant way [19][22][14].

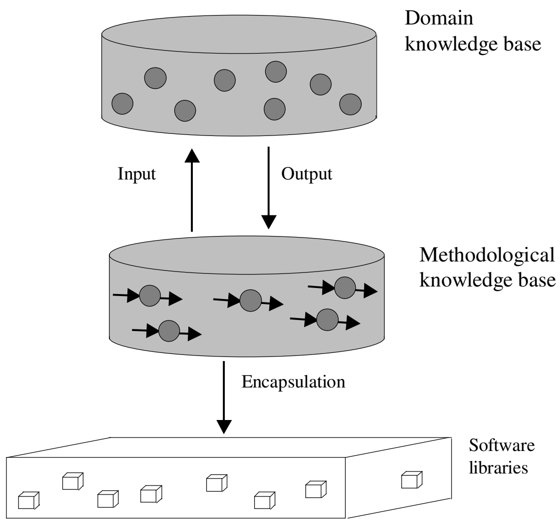

Fig. 1. The PSE is built around two kinds of knowledge which are used to pilot, in a suitable way, software libraries: the methodological knowledge contains problem solving strategy, and the domain knowledge base contains the input needed by the tasks and the output produced during the resolution processes (programs are encapsulated and controlled by the PSE thanks to the methodological knowledge).

The Sherpa project at INRIA Rhone-Alpes developed in collaboration with the company Ilog a generic PSE PowerTasks [17] whose knowledge base is built on an objects model. Domain knowledge is represented by means of classes and instances, methodological knowledge is described using particular classes and instances: tasks [4][5]. A task allows a compact and structured representation of a problem and knowledge necessary to its resolution with regard to the description of its data and results, as well as a strategy for solving the problem (cf. section 2).
PowerTasks was used for the task of genome annotation, within a framework involving several biology laboratories. This work was financed by a French organism, the G.R.E.G. ("Groupement de Recherche et d'Études sur les Génomes"). The system obtained, ImaGene, allows the analysis and the annotation of complete bacterial genoms [15][16]. It gives a biologist the possibility of exploiting and carrying out in a transparent way methods for DNA sequence analysis obtained from several libraries offering complementary methods. These methods seek and identify, on a DNA sequence, biological objects of interest, such as genes, signals of regulation... To this end, ImaGene uses three distinct windows allowing to manage the biological objects, the solution strategies and the solution processes (for the first two ones) and to visualize the DNA map including the biological objects resulting from the different solution processes (in the last window).
The installation and use of a PSE provides a satisfactory response to the problem of managing programs libraries. Nevertheless, some difficulties remain, like for example the installation and the updates of libraries. The code of these libraries used to be not portable, and difficult to install (split into several parts, calls to other tool libraries, incompatibilities of compilers versions...) and often involves the purchase of hardware or specialized software (required computing power, uses of specific databases). To illustrate this last difficulty, we briefly present the case of two strategies included in ImaGene, which, in its current version, encapsulates twelve libraries, eight of which require specialized resources.
Before explaining the way this strategy works, the biological background of DNA sequence analysis needs to be briefly reviewed/explained. A DNA sequence is a sequence built from four letters A, T, C, G which represent a nucleotide acid or nucleotide. A group of three nucleotides can be translated into an amino acid thanks to a translation table. In the case of bacteria, a DNA sequence is about 10^6 nucleotides long. But the major part of the process of nucleotides is never translated into amino acid: they do not code for proteins. The DNA sequence annotation consists in finding the coding sequences. There are different ways of selecting the interesting part of a DNA sequence.
One of these consists in comparing sequences coming from similar organisms starting from the position of the relevant biological objects on an analyzed sequence of a similar organism. This method is contained in a library that uses a database with sequences of several partially or completely analyzed organisms. During the installation of this library, it is necessary to have a copy of the complete database which must be upgraded regularly (every week). Another solution is to access the initial database remotely.
Another method is based on statistical search. The corresponding library uses the frequency of appearance of nucleotides in genes or other relevant biological objects. The installation of this method is thus linked to the installation, at the same time, of

statistical programs and to the data relating to partially or completely analyzed organisms, and finally to the procedures of error tolerance in sequence analysis. The installation of the three parts of this library, and the updating of one of them independently from the others two, is not go without difficulties, in particular for non-specialists in computer science.
In order to remove these difficulties of installation and maintenance [13], we propose an extension of the PSE PowerTasks towards a distributed problem solving environment (DPSE). Thanks to this distributed version of PowerTasks, the sites using this DPSE, on the one hand, become customers of competences of others sites and, on the other hand, carry out the modules and library programs. In this way, each site is server of its own competencies, ensures the maintenance of the modules and libraries which are under its control, gives total or partial access to them, and allows other sites to use its specialized software and hardware with its own licenses of use.
The distribution of domain and methodological knowledge bases across several sites presents many advantages and provides a solution to the problems mentioned in the preceding paragraph. In section 3, we will present the architectural choices of the DPSE PowerTasks, and particularly the technological alternatives that have guided us in this realization (cf. sections 3.1 and 3.2). The development of such a distributed architecture required an extension of the tasks model (cf. section 2), in the sense that new attributes were introduced and various communication protocols added (cf. section 4). Lastly, we present a way to adapt solution strategies allowing themselves to use tasks developed later in the distributed environment (cf. section 5).

2 Expressing Methodological Knowledge

In order to allow a representation of knowledge with a high level of abstraction [4], to give an easy access to this knowledge and to facilitate its maintenance, we chose to represent knowledge by means of object-oriented models. This choice makes it possible to describe domain knowledge and methodological knowledge in a similar way.
The knowledge representation within the PowerTasks environment is realized out using two different types of entities:

- the domain description is expressed in terms of classes and instances; (cf. Fig. 2)
- the solution strategies are built over programs libraries using tasks in order to allow a suitable use of those.

As defined in section 1, a task associates the description of a problem and its solving strategy. This strategy is represented, in our task model, by means of two graphs representing the specialization or decomposition of a task into more elementary subtasks. At the lower level, the description of an elementary problem is reduced to a task and its solving strategy to a call for the execution of a library program. The PSE methodological knowledge base thus consists in tasks hierarchies. They are constructed in declarative way using a graphic editor, with a explicit links of specialization and decomposition between tasks and sub-tasks.

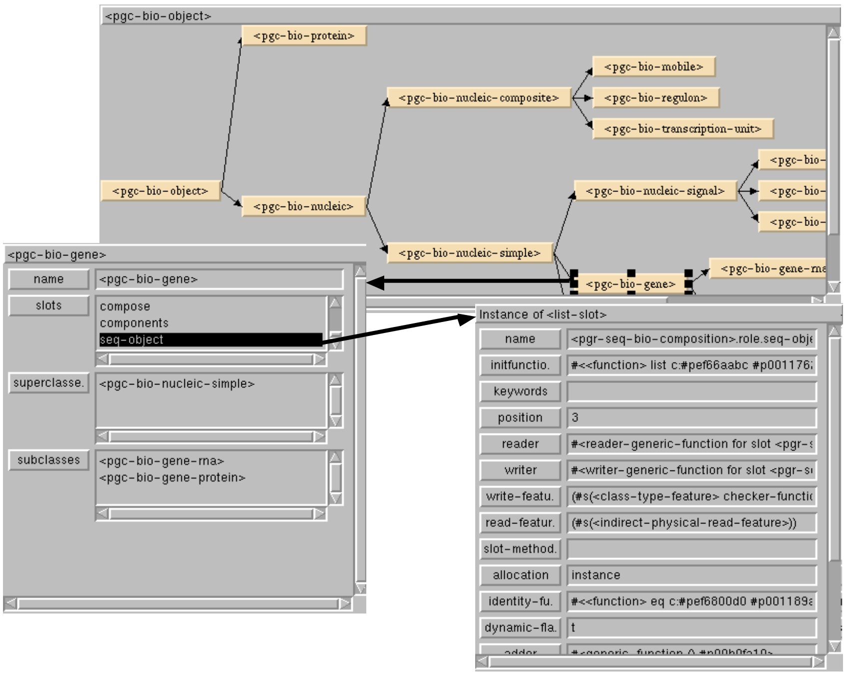

Fig. 2. In ImaGene, the domain knowledge base contains biological objects described by means of a hierarchy of classes. The editor at the left allows the user to visualize the class "gene" and the editor at the right-hand side allows the user to visualize the attribute "sequence" representing the DNA sequence associated with this gene. For a given instance of a gene, the values of the attributes are specified. In particular, the sequence is explicitly described by means of an alphabet of 4 symbols (A, T, G, C).

We can distinguish two categories of tasks: complex tasks which break up into sub-tasks and elementary tasks which refer to a program of library. Complex tasks allow users to approach problems via their strategies, i.e. specialization, sequence or iteration of sub-tasks (cf. Fig. 3). On the opposite, elementary tasks play the role of "black box". Their role is limited to encapsulating a library program and providing access to its functionality. An elementary task thus contains the most powerful up-to-date version of the encapsulated program.

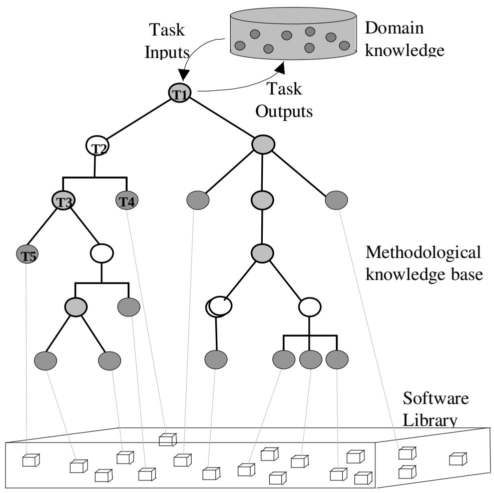

Fig. 3. Detailed description of links between the three main components of PowerTasks (methodological knowledge, domain knowledge and programs). For example, during a resolution based on the task T1, the T1 solution process uses inputs provided by the domain knowledge base. T1 can be specialized into two sub-tasks. If one of these subtasks T2 is selected, inputs necessary to the execution of T2 are transmitted to T2 via T1; T2 breaks up into two other tasks: an elementary task T4, which encapsulates a program and a complex task T3 which can be further specialized.

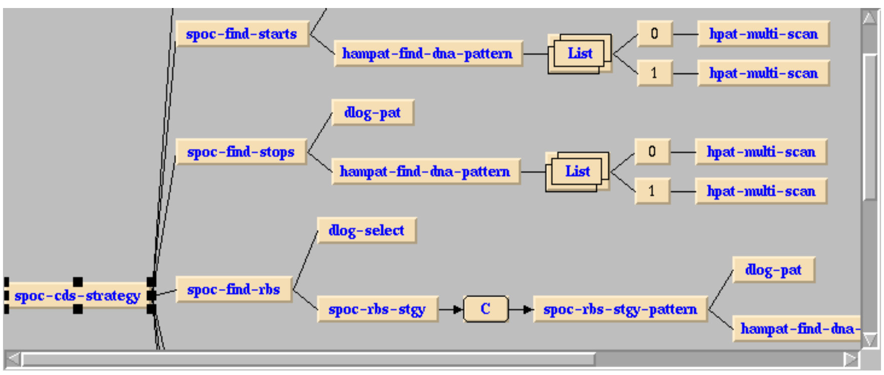

Fig. 4. In this graphic editor, we visualize the result of the execution of the "spoc-cds-stategy" which is a strategy included in ImaGene. "spoc-cds-stategy" allows the user to identify interesting patterns on genome sequences starting from patterns supplied by users. The task "spoc-cds-stategy" is broken up into three tasks "spoc-find-starts, spoc-find-stop, spoc-find-rbs", and these sub-tasks are decomposed into loops on lists ("List"), sequences or specialization ("C").

The internal task representation (cf. Fig. 5) is a class made up of two sets of attributes which describe:

– the problem by means of the data manipulated and the expected results. Each data element or result corresponds to an attribute, whose type can be simple (real, integer, chain...) or refer to a class of the knowledge base;
– the solution strategy by means of sub-tasks (cf. Fig. 4). A task is associated with a decomposition and/or specialization. Decomposition uses a list of sub-tasks and operators (sequence, choice, iteration) which will be interpreted during the solution process. Specialization consists in specifying inputs and the task application conditions in order to particularize the solving strategy according to these conditions.

```
(defctlgoal spoc-rbs-stgy (<ctl-task>)
  context: ((lst-adr-seq set-of: <address>)
            (sym-mode type: <symbol>
                      domain: ('pattern 'consensus)))
  goal-state: ((cons-parm type: <list>)
               (lst-rbs   set-of: <list>)))
```

Fig. 5. Code of the task spoc-cds-stgy (from ImaGene). The attribute "context" contains task inputs, "goal state" outputs. This task will solve thanks to a specialization, it is thus defined as a goal (defctlgoal).

During the problem resolution process, PowerTasks inferences engine chooses the solving strategy best adapted to the context. In order to achieve this, task execution is carried out two phases:
– the specialization phase uses a mechanism which classifies a task according to its data and to its conditions describing the context. This classification makes it possible to select the sub-task that suits best to the characteristics of the current problem;
– the decomposition phase uses choice, sequence or iteration operators. This makes it possible to decompose a task into simpler sub-tasks.

The alternation of these two phases (specialization and decomposition) results in an opportunistic resolution approach since the sequence of tasks is chosen only during execution [14] according to problem characteristics. For each resolution, the system selects the most suitable strategy. (cf. Fig. 6)

The task model of PowerTasks has the advantage of allowing a declarative expression of the problem solving strategies [10]. This characteristic differentiates it from the approaches adopted in the knowledge acquisition domain [5] [8], and is closely related to the way tasks are used in program supervision [7] [9] [23] [16]. This knowledge representation by means of objects and tasks is based upon several tasks models and PSEs [3] [19] [24] developed within the Sherpa project in the last five years. It gave rise to a number of collaborations for programs supervision in several domains like data analysis [2], signal processing and genomic sequences analysis [15].

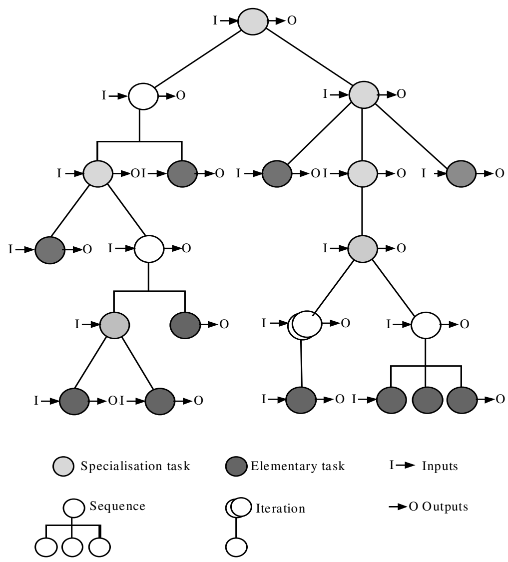

Fig. 6. problem solving strategy built above a specialization. The inference engine will start by specializing the problem according to its characteristics. Then, it will alternate between specialization and decomposition phases until it reaches the appropriates elementary tasks which are executed...

3 Distributed Environment

The PSE make it possible to use specialized libraries in an optimal way. The system allows the user to develop various strategies based on several distinct domains, creating in this way a list of problem solving strategies. For ImaGene, the application of PowerTask to the problem of DNA sequence annotation, a lot of strategies have been developed and integrated into the environment. However, their maintenance remains a problem because each strategy can be modified to take into account new discoveries like of course annotation of new genes, i.e. update of databases; but also some modifications more important. On the other hand, the installation of libraries themselves is a demanding problem, both in time and competence. These libraries require in addition resources which are not always available, like special compilers or expensive hardware configurations. In addition, the sites which use PowerTasks create methodological knowledge bases and therefore develop a private competence which could be put at the disposal of the community. We have thus extended the environment so as to allow a distributed use of methodological knowledge bases and specialized libraries. This choice enables one to reduce the costs of installation and maintenance of libraries to a minimum: only one server site is responsible for each strategy. We thus provide access to these libraries for sites having insufficient hardware or software configurations and propose using distant knowledge bases to client sites.

The options chosen for the distribution of PowerTask can be separated into two points: on the one hand the network configuration and on the other hand the code mobility. These choices were made to solve the different problems expressed above.

3.1 Network Configuration.

In biology like in several other experimental domains, research is carried out in different laboratories: each one working on a specific part of a global problem and proposing solutions based on its own speciality. Thus the network configuration most adapted to this collaborative way of working is made up of several server sites providing their own competencies. Each site maintains the libraries which are local for him, and the strategies of resolution which it created. The DPSE allows the code of each module to remain in its original environment with specialized software/hardware and required databases or knowledge bases. This configuration avoids rewriting some code, having to upgrade every strategy or library coming from a distinct domain, and problems of software/hardware compatibility. This choice of distributed architecture thus authorizes the coexistence of different hardware and software configurations.

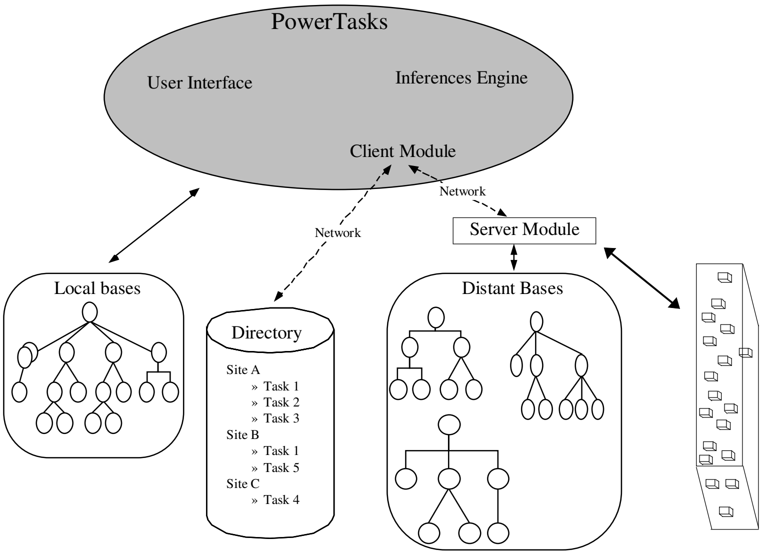

Fig. 7. Communications between elements of the DPSE. Network communications are indicated by arrows in dotted line. PowerTasks is localized at each user site. There is a server module at each site containing public tasks. This server also provide access to programs encapsulated in elementary public tasks.

Further, it allows the availability of all solution strategies whatever their localizations. Each server site thus contributes, according to its competence field, to design a single methodological knowledge base, multi localized. To do that, it develops single or multi-localized strategies and allows other sites to access these strategies while

making them public. Just as the WWW makes it possible to see a multitude of distant documents like a single hyper-document, the goal here is to see a multitude of strategies developed and maintained at various sites as constituting a single methodological base.
Accessing distant methodological knowledge bases is allowed by server modules. At the opposite, using local methodological knowledge bases remains directly controlled by PowerTasks (cf. Fig. 7).

3.2 **Code Mobility.**

Few years ago, Java proposed a new approach towards compilation and distributed architectures. The code is semi-compiled into byte code and this byte code is then executed on each hardware configuration thanks to the Java Virtual Machine. Our vision of the distribution of PowerTasks has been nearly the same: the methodological knowledge, i.e. the tasks, are similar to the byte code in that they are valid in each architecture (comprising PowerTasks). Tasks can be easily transferred through networks from server sites to client sites and then be used to solve problems. The main difference comes from the execution of library modules used to solve elementary tasks: they are executed at the server site. Thus the solution with respect to code mobility adopted in PowerTasks is mixed.
Using distant software library code can be done in three different ways:

- the code is untransportable due to hardware or software incompatibility of environments. This case is the most frequent because a lot of libraries for scientific programming are in Fortran or C. Then executions are launched at the server site and results transferred to client sites as in the case of RPC (Remote Procedure Call);
- the code can migrate through the network and be used on a distant machine (mobile). In this case, the client can transfer the selected code and then execute it locally. This solution allows a significant number of simultaneous accesses, without slowing down servers responses. It is the case of a language like Java;
- the code is available for several hardware configuration. In this case, server sites contain different versions of the same modules depending on the hardware configuration; mobility policy consists in transferring to client sites a version adequate to its configuration. Executions are then, also, local. These possibilities are offered, for example, by ActiveX or Ilog products.

In terms of the above classification, code mobility in the DPSE PowerTasks depends on its nature (task/program/data):

- tasks code is mobile: tasks language is interpreted. On the other hand, an interpreter module is included in PowerTasks allowing each client site to understand and use distant tasks;
- on the other hand, software library modules used to solve elementary tasks are generally executed at their server site;
- data used or produced by external programs or library modules are moved from client to server before execution and from server to client after execution. Each site keeps its own data it uses to work on, i.e. DNA sequences in the biological application.

Tasks transfer makes it possible to solve problems locally thanks to classification and/or decomposition and thus allows one to preserve user interactions present in the single-site model. Users can thus intervene interactively during a resolution. This approach gives the impression to users that all problem solving resources (i.e., methodological knowledge, binaries, databases,...) are available on their own computers, but binaries, databases and specific resources are only virtual since they remain on their original site.
All these choices affect the task model and imply some modifications, in order to:

- define accessible competencies localization,
- allow interaction between distributed tasks,
- give access to all the tasks in an identical way, independently of their localization,
- propose new techniques to deal with strategy actualization,
- check coherence between both domain and methodological knowledge bases.

These various points involve a modification of the task model (cf. section 4), the definition of mechanisms concerning task use and modification, the creation of new entities facilitating the adaptability of strategies (cf. section 4.1).

4 Distributed Tasks Model

The tasks model described in section 2, supports well the distribution (cf. Fig. 8). The distribution has only a minor effect on the whole system, since the tasks are transferred. Once the answers are returned, resolution proceeds locally until calls to binaries or modules. Nevertheless, the evolution towards a distributed tasks language led us, on the one hand, to add a status attribute in order to characterize distant authorization access, and, on the other hand, to propose protocols to manage tasks versions.

4.1 Tasks Status and Availability.

To be accessed and used from a distant site, a task must be declared as being *public*. This publication generates a submission process to a directory which centralizes each available task (i.e. public). Another alternative to this publication exists. It consists in providing the client only the task functionality without visualization of the strategy, this kind of tasks is declared as *blind*. This status is used in two distinct cases: to protect the methodological knowledge because it contains secret into, or to hide this knowledge because it is too complex and without any interest for the client (e.g. a low level of tasks decomposition due to the software library configuration). In ImaGene, some annotation methods come from private laboratories which just give permission to execute them without explaining their working, e.g. the use of the method based on nucleotide appearance frequency is restricted to execution.

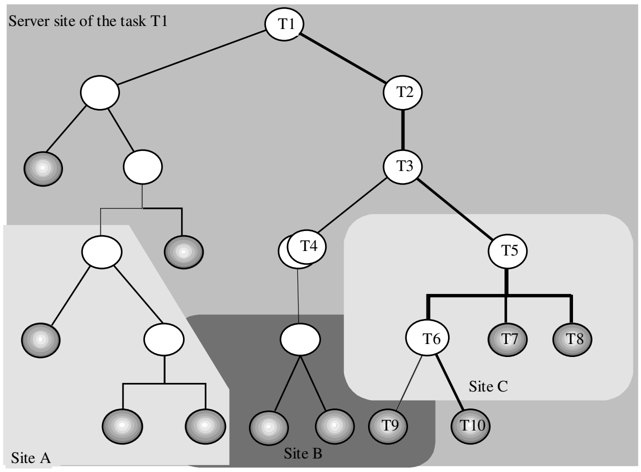

Fig. 8. Example of strategy localization. Strategy T1 is based upon strategies of site A, B and C. For example, during a resolution, task T1 specializes in T2, next in T3; T3 specialization is done towards a task of the site C (T5). This task T5 breaks up into three sub-tasks whose first (T6) specializes in T10, an elementary task (in gray) which is located at the first site.

Any site can publish a task to make it available to others. Doing this, it becomes server of the task with respect to the others sites, the strategy and the different resources used to solve this task (databases, binaries and calculation time). A site publishing or proposing public tasks remains client for distant tasks already used. Each site is thus potentially client and server. A site can even be server of a strategy whose elementary tasks are distant, it is then only server of tasks hierarchies. On the opposite, it is possible that the methodological base of a site is empty if the users just use public and blind tasks. The consequences of this status is that methodological knowledge bases of other sites are not seen in full, but only through their public tasks, sub-tasks of public tasks and blind tasks.

4.2 Publication Protocol.

To publish a task, a site must send to a directory (i) the name of the task, (ii) its signature (description of the inputs and outputs), (iii) its functional description and (iv) a number of version (identification of the last update). This publication is validated by the directory which checks that there is no name conflict. Setting a public status to a task leads to setting this status to all its sub-tasks except if they are already in a blind status. On the opposite, setting a blind status does not modify the status of the subtasks.

4.3 Modification Protocol.

Modifications on public/blind tasks are possible. They can be more or less complex depending on their type (cf. Fig. 9). These modifications may concern the signature (input/output description), the solution strategy or the encapsulated module used to solve the elementary task.

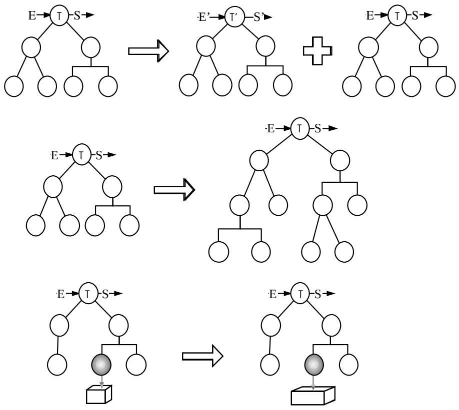

Fig. 9. Different kinds of modification: signature, strategy and programs.

Each of these modifications is briefly described in the paragraphs below.

- The task whose signature is changed must take another name. Indeed, there cannot be two different tasks having the same name and distinct signatures at the same site. This constraint is justified by the fact that the change of signature of a task, after its publication, would involve inconsistencies in the strategies using this task, making them thus unusable. Each server site publishing a task must ensure its persistence.
- The modification of a strategy which does not affect the signature of a public/blind task is considered as a change of version of this task. The strategies based on this task can use the old version or the new one. Forward planning, the old version will be replaced by the new one. But, making the change immediate would perturb users. Indeed, solving strategies of tasks act upon the way users apprehend the domain in question and an automatic change would disturb them even if the functionality of the new strategy is similar to the previous one.
- On the opposite, the modification of an elementary task solution resources is automatic. Indeed, the update of the binary/module encapsulated in an elementary task is totally transparent for users. The only difference is that the version number in the directory is incremented in order to be able to inform users of possible causes of an problem, if one occurs.

4.4 Use Protocol.

Client sites can consult the directory and use any of the described tasks. Public task use is similar to local task use: once the strategy has been transferred, the resolutions based on it are local, the only difference being that elementary tasks are solved at the server site (cf. Fig. 10). Blind task use is slightly different since they are not decomposed during the resolution. Thus, users can not make any difference between blind tasks and elementary tasks. But inside the system, exactly the same process of classification/decomposition phases is carried out.

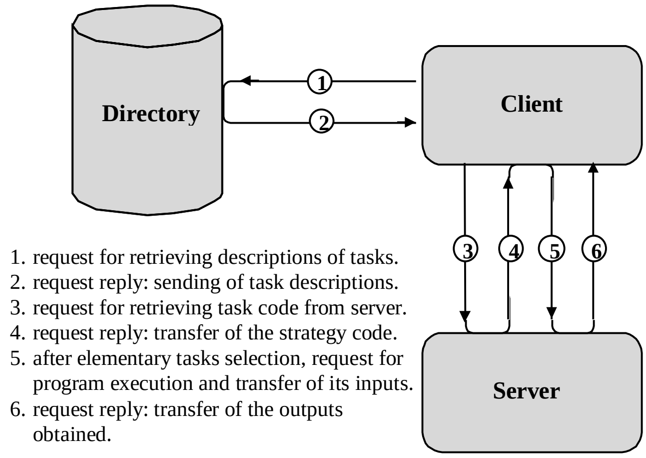

Fig. 10. Description of the communications during a resolution employing a distant task.

5 Perspectives: Adaptability of Solving Strategies

The new opportunities offered by the distribution of PowerTasks have highlighted an important problem: task solution strategy are fixed. There is no dynamic process to include new available tasks. This problem can be illustrated by means of a simple example in DNA sequence annotation. Suppose that a laboratory has a task consisting in launching several annotation methods one after another in order to compare results and find sequencing errors. Later on, a laboratory proposes a very interesting method through a public task. The first laboratory is not informed that a new method is available and it keeps using its old task. This is due to the fact that all the tasks available to solve a problem are explicitly declared as sub-tasks of the task containing the problem. To solve this problem, we propose to generate a *request* able to find in the directory the tasks that could solve the current task [21] [1]. This request would be based on the task functionality and its data signature. In order to include this request, the inference engine of PowerTasks will be modified: the request will be executed before specialization in order to find new possibilities to solve a task. The tasks obtained will be proposed to the user and eventually included in the task solving strategy.

T. Parmentier, D. Ziébelin, Distributed Problem Solving Environment Dedicated to the DNA Sequence Annotation, *Proceedings of the 11th European Workshop (EKAW'99)*, Dagstuhl Castle, Germany, May 1999.

6 Conclusion

PowerTasks is a generic problem solving system designed around an object model developed by Ilog software company and a task model resulting from our preceding experiments. It makes available through a single user interface:
- the description of tasks in a suitable language allowing the expression of a problem solving strategy,
- to control the execution of tasks, to control and set parameters to programs encapsulated in elementary tasks,
- to start the execution of these programs,
- to describe problems and to visualize resolution results,
- to intervene directly during the solving process by changing a value, adding a parameter or choosing a solving strategy.

The PSEs developed on top of PowerTasks include all these functionalities in a homogeneous environment. This one thus allows users a simple use of software libraries and an access to methodological knowledge without requiring them to be an expert in the different techniques used.

The PSEs permit experts to work on their specific domain while using knowledge developed by others. In addition, PowerTasks authorizes the use of distant problem solving strategies or programs. This contribution facilitates the diffusion of competences. Thus users of various domains can work in partnership through networks. This possibility represents a major asset in the resolution of problems, since it enables each task to be based on the last strategy developed by experts of the concerned domain and each elementary task to encapsulate the most recent and high-performing program. The strategies are written in a mobile code and are interpreted locally; this allows users to preserve the same interactions with the system as in a not distributed environment.

As the distributed model makes it possible to have several strategies solving the same problem, we are introducing rationality criteria into the choice of the tasks. These choices will not only be based on the input characteristics, but also on the geographical localization of the task, its execution time [11], and the quality of the solutions provided. Moreover, the possibility of launching parts of the resolution on a distant site allows one to launch them in parallel [6] [12]. Distributed problem solving environments thus generate new opportunities which are still to be investigated.

References

1. Bisson, G., Botraud, J-C.: A proposal to improve information retrieval on WEB sites, http://nangaparba.inrialpes.fr/, 1998.
2. Chevenet F., Jean-Marie, F., Willamowski, J.: SLOT: a cooperative problem-solving environment in explanatory data analysis. Proceedings 49th session of the International Statistical Institute, pp. 255-256, Firenza (IT), September 1993.
3. Chaillot M.: Une architecture de contrôle réactif pour la résolution coopérative de problèmes, PH.D Thesis, INPG, Grenoble (Fr), 1993.
4. Chandrasekaran B.: Generic tasks in knowledge based reasoning high level building blocks for expert system design, IEEE Expert, 1986.

5. Chandrasakaran, B., Johnson, T.R., Smith, J.W.: Task-structure analysis for knowledge modelling, Communication of the ACM, 35(9), pp. 124-137, 1992.
6. Charpillet F., Boyer, A.: Progress : un modèle d'agent pour la conception de systèmes multi-agents temps réel, Journées Francophones IA distribuée et Systèmes Multi-agents, 1997.
7. Clément V.: Raisonnements cognitifs appliqués au pilotage d'algorithmes de traitement d'images, PH.D Thesis, Université de Nice Sophia-Antipolis, Nice (Fr), 1990.
8. Delouis I.: LISA, un langage réflexif pour la modélisation du contrôle dans les systèmes à bases de connaissances. Application à la planification de réseaux électriques, PH.D Thesis, Université de Paris Sud centre d'Orsay, Paris (Fr), 1993.
9. Demazeau Y., Ferber, J.: Actes de la 1ère Journée Nationale sur les Systèmes Multi-Agents, ed. PRC-GDR IA MARCIA CRIN, France, LIFIA Publication, Grenoble, France, December 1992.
10. Erickson et al: Task modeling with reusable problem solving method, AI 2(79), p. 293-326, 1995.
11. Fink E., Statistical Selection Among Problem-Solving Methods, Research Report CMU-CS-97-101, 1997
12. Fujita S., V.R. Lesser, Centralized Task Distribution in the Presence of Uncertainly and Time Deadlines, Proceeding of ICAMS, Japan, 1996.
13. Gallopoulos E. et al.: Future Research Directions In Problem Solving Environments For computational Science, CSRD Report, N°1259, 1992.
14. KbuP'95, First international workshop on Knowledge-Based systems for the (re) Use of Program Libraries, INRIA, Sofia Antipolis, France, 1995.
15. Médigue, C., Vermat, T., Bisson, G., Viari, A., Danchin, A.: Cooperative computer system for genome sequence analysis, Proceedings of 3rd ISMB, Cambridge, United Kingdom, pp249-258, 1995.
16. Moszer I., Kunst, F., Dachin, A.: The European Bacillus subtilis genome sequencing project : current status and accessibility of the data from a new World Wide Web site, Microbiology 142, p. 261-268, 1996.
17. Rechenmann F.: Knowledge bases and computational molecular biology, In Nicolaas Mars (ed.), Towards very large knowledge bases (proceedings 2nd international conference on building and sharing very large-scale knowledge bases (KBKS), Enschede (NL), IOS press, Amsterdam, Pays Bas, pp. 7-12, 1995
18. Rice J. R., R. F. Boisvert, From Scientific Software Libraries to Problem-Solving Environnements ", IEEE Computational Science & Engineering, pp. 44-53, 1996.
19. Rousseau B., Vers un environnement de résolution de problèmes en biométrie, PH.D Thesis, Université C. Bernard, Lyon, France, 1988.
20. Schreider, A.Th., Wielinga, B.J., de Hoog, R., Akkermans, J.M., Van de Velde, W.: CommonKADS: A Comprehensive Methodology for Knowledge Based Systems Development, IEEE Expert, 9(6), pp. 28-37, 1994.
21. Thonnat, M., Clément, V., Van den Elst, J.: Supervision of perception tasks for autonomous systems: the OCAPI approach. J. of Information Science and Technology, Vol. 3(2), pp. 140-162, 1994.
22. Thonnat, M., Moisan, S.: Knowledge-based systems for program supervision, KbuP'95, First international workshop on Knowledge-Based systems for the (re) Use of Program Libraries, Sofia Antipolis, France, 1995.
23. Van Den Elst, J., Modélisation de connaissances pour le pilotage de programmes de traitement d'images, Ph.D. Thesis, Université de Nice Sophia Antipolis, Nice, France, 1996.
24. Willamowski J.: Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur, Ph.D. Thesis, Université Joseph Fourier, Grenoble, France, 1994.