Software Design and Programming with Scientific Users

Catherine Letondal
letondal@pasteur.fr

Institut Pasteur
Pôle Informatique
28, rue du Dr. Roux
75724 - Paris Cedex 15, France

1 Introduction

In this document, we describe the approaches for computer systems development that we have used along several years in a biology research institute. We had to cope with highly evolutive and diverse needs, in a context where the frontier between computer scientists and ”end-users” is sometimes blurred. In order to develop more flexible systems and to empower these specific users, we have tried to develop a participatory culture, by organizing workshops, conducting interviews, as well as developing end-user programming tools. We first describe the context of these activities within the Institut Pasteur. We then report on characteristics of software development in bioinformatics. We finally devote the last part of this document to two examples of projects.

2 Context

The Institut Pasteur, in Paris, France, is a non-profit private foundation “dedicated to the prevention and treatment of diseases through biological research, education and public health activities”. Founded by Louis Pasteur in the 1880s, the Institut Pasteur is a world-renowned research laboratory, attracting researchers from approximately 60 countries. It is also a teaching institution, with over 200 doctoral students and visiting researchers who participate in 15 diploma courses.

The Information Technology (IT) department installs, develops and maintains scientific software for biologists, including support for scientific databases and network infrastructure. It also runs continuing-education courses on scientific software and programming for the biologists. In addition, some members of the IT department pursue their own research, in bioinformatics and even human-computer interaction . From the biologists' perspective, the IT department is primarily a service for dealing with local technical problems and email accounts. However, some also collaborate on specific IT projects. Because the center does not have sufficient resources to provide ”programmers-for-hire” for each biologist, the IT department seeks to create more general-purpose tools to support multiple biologists, valuing reuse and accessibility over individual solutions to particular problems.

3 Software Development in Biology

Having installed scientific software for 8 years at the Pasteur Institute, and having taught biologists how to use these scientific tools, I have observed that software development in the field of academic research in biology follows two main paths:

- large-scale projects like bioperl [37], and development in important bioinformatics centers such as Blast at NCBI [1];
- local development, by biologists who have learned basic programming but who are not professional developers. These biologists either have to deal with everyday tasks of managing data and analysis results, or have to model and test scientific ideas.

These two lines often merge since local developers distribute software they programmed for their own research via public repositories. Biological software development has the following characteristics:

- *Software activity is very dynamic:* the researcher cannot always wait before testing new scientific ideas for a professional software developer to produce a prototype or software adaptation.
- *The majority of programs are developed by domain experts:* professional developers having a computer science background rather participate to more technological projects.
- *Software production is not the goal:* numerous ready-made tools are available. In scientific research, software is often created only to test new ideas rapidly, thus its production does not always follow typical software engineering life-cycles.

- *Software is recycled:* biologists who program are often more likely to recycle existing program chunks rather than creating and re-using well-designed modules, even though this sometimes leads to carrying on out-of-date and over-sized data structures.

4 End-User Software Development

4.1 Typical Problems

In order to illustrate the need for end-user development [8] [24] in the context of biology and bioinformatics, let us show some typical examples (see also [7]). Below is a list of real programming situation examples, drawn from interviews with biologists, news forum, or technical desk. These situations happened when working with molecular sequences, i.e either DNA or protein sequences (a sequence is a molecule that is very often represented by a character string, composed of either DNA letters – A, C, T and G – or amino-acid letters – 20 letters).

- scripting: search for a sequence pattern, *then* retrieve all the corresponding secondary structures in a database
- parsing: search for the best match in a database similarity search report *but relative to each subsection*
- formatting: renumber the positions of a sequence from -3000 to +500 instead of 0 to 3500
- variation: search for patterns in a sequence, *except repeated ones*
- finer control on the computation: control of the order in which multiple sequences are compared and aligned
- simple operations: search in a DNA sequence for the characters other than A, C, T and G

As illustrated by these examples, unexpected problems may arise at any time. However, these scenarios involve rather simple programmatic manipulations, without any algorithmic difficulty or complex design. An important reason why programming is needed here is that the function, although easy to program, or even already implemented somewhere inside the program, has not been *explicitly* featured in the user interface of the tool.

4.2 End-User Programming in Biology Research

Scientific users often use spreadsheets and local easy to use database systems to manage their data. They also develop small use-once scripts to automate simple tasks, whenever they had the possibility to attend an introductory programming course or when they know a local developper who can help to deal with potential bugs.

4.3 Programmability for the Scientific User

End-user programming tools are however limited, and biologists very often need to build a piece of software to address a specific problem. But, even when they find a local developer to help them [12], they cannot build new software everytime a simple feature is missing in the software they use. As [11] explains, users should be able to modify or extend tools they already use and find useful. Software mechanisms are required that help cope with unanticipated changes [23]. These should provide:

- explicit and documented areas that users can change,
- tools designed for non-professional programmers that associate user interface elements with system features (for example, a class would be associated with a window and its methods would be available for editing by a menu). This helps to reduce the cognitive distance between the code and the user interface [29].
- a programming environment directly available from the data analysis environment. This eliminates switching from one mode to another: programming becomes just another type of using.

4.4 Biok: a Biological Interactive Object Kit

Biok is a prototype of a programmable environment for biological data analysis. In *biok*, the basic building block for both using and programming is a *graphical object*, that corresponds to a biological entity, such as a 3D molecule, a protein or DNA sequence, an alignment, or to more general objects needed in scientific analyses such as plots. Interaction with graphical objects is available either by direct manipulation or with an associated shell to run objects' methods.

Biok uses a language called XOTcl [30], which is an object-oriented scripting language and uses TK as a graphical toolkit. As a Tcl extension, XOTcl is a full-fledged programming language. XOTcl provides reflective language functionalities which proved to be extremely efficient in the design of *biok* by enabling us to weave *using* and *programming* as two levels of interaction on the user side, directly available from the menus of any graphical object. Our hypothesis is that *it is difficult to program a little*, not only in the sense of programming *occasionally*, but also in the sense of programming *incrementally* [22]. This is made possible in XOtcl, where the code of a class can be dynamically changed at the method level.

One of the central tools of *biok* is a spreadsheet specialized in displaying and editing sequences aligments (Figure 1, back window). As in many other scientific research areas, visualization is critical in biology. Several graphical functions, or *tags*, are available in *biok*. A tag is a mechanism that highlights some parts of an object. For instance, segments of protein sequences showing a specific property, such as a high hydrophobicity, can be highlighted in the spreadsheet.

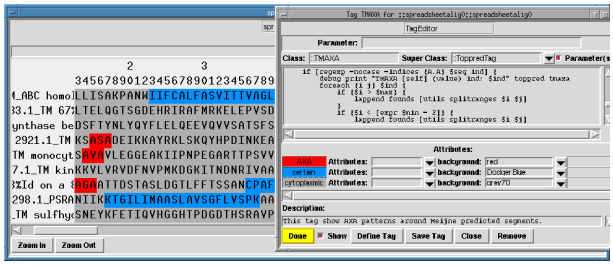

Figure 1. Tag editor (front) and alignment editor (back). In the tag editor, the top part displays a field to enter parameters for this tag, if needed. The part below contains a superclass chooser and an editor for the code to define tag values. The middle part is a tool to associate graphical attributes to tag values. The bottom part displays a short description of the tag. The positions associated with the various tag values are highlighted in the alignment editor. In this example, a tag showing transmembrane segments (in blue) is extended by a sub-tag highlighting small patterns around them (in red).

5 Software Development and Participatory Design at the Institut Pasteur

Before we proceed to the description of our participatory design activies, this section first briefly introduce software development at Institut Pasteur, including our own projects.

5.1 Software Development at the Institut Pasteur

Several biology research projects involve software development, which very often take place outside of the IT laboratory. Typical developments includes databases dedicated to the data related to the laboratory's activities, often made available for access through a Web server. Some laboratories develop systems for bench experiments automation. Most local software is dedicated to data analysis tools though, sometimes including visualization features, or even involving new published methods.

Our own projects, developed in the IT center, include Web interfaces systems [21][19], graphical tools [44], databases, and prototypes for end-user development (see sections 4.4 and 6).

5.2 Interviews and Workshops

We begin each project by visiting the biologists in their laboratories and interviewing them or videotaping them at work. We then hold discussions, brainstorming sessions and design workshops with groups of biologists, bioinformaticians and computer scientists. Some meetings focus on a particular issue, others are general discussions. Participants usually bring examples of their work and we create paper mockups to illustrate and work through design ideas. We also conduct video brainstorming sessions [25], in which participants are videotaped as they act out how they would interact with their prototypes.

Our own research projects have led us to observe, interview and videotape numerous biologists in their laboratories. We have conducted over 65 interviews over the past seven years: Approximately 30 were dedicated to the research on end-user programming [20], 20 have been recently organized for the design of Mobyle [19]. 15 additional interviews focused on an augmented laboratory notebook research project [28]. We asked for volunteers who were willing to be interviewed in their laboratories. We sometimes asked them to arrive at a time when we could videotape them as they performed specific activities relevant to the project, such as performing certain on-line analyses. In some cases, other members of the lab joined in to show us their work, either similar or in contrast to the original person being interviewed. We consider both in situ observation and interviews to be an ongoing activity and observations from one project are often relevant to other projects.

The above projects all use participatory design workshops to create and explore design ideas. We ask participants to bring specific examples of work artifacts, such as DNA sequences or analysis results. We also provide participants with a variety of prototyping materials for creating mockups of proposed software interfaces. The participants then generate ideas and act out how they would like to, for example, visualize a DNA sequence or search for a pattern in their data. Videotaping very short clips [25] forces participants to think concretely in terms of interaction with the software tool and provides a record that other participants can easily understand. We sometimes replay videos from earlier workshops as a source of inspiration, and on some occasions, the videos have been used by programmers as a design specification for software that they then prototype and develop.

Many of the people who volunteered to be interviewed or observed in their labs have also participated in participatory design workshops. The largest project, *biok*, has involved a series of video brainstorming and prototyping workshops over several years. Figure 2 shows the chronology of most of the workshops organized from 1996 to 2004, including the campus-wide survey and other user studies, tied to the development of *biok*. We drew prototyping themes from brainstorming sessions (figure 3) and from use scenarios, which based on interviews and observation. Each workshop involved from 5 to 30 people, with participants from the Institut Pasteur or other biological research laboratories, as well as biology researchers who were students in our programming course.

5.3 Participants: Differing Perspectives

We can identify three categories of participants in our participatory design activities. The largest group are research

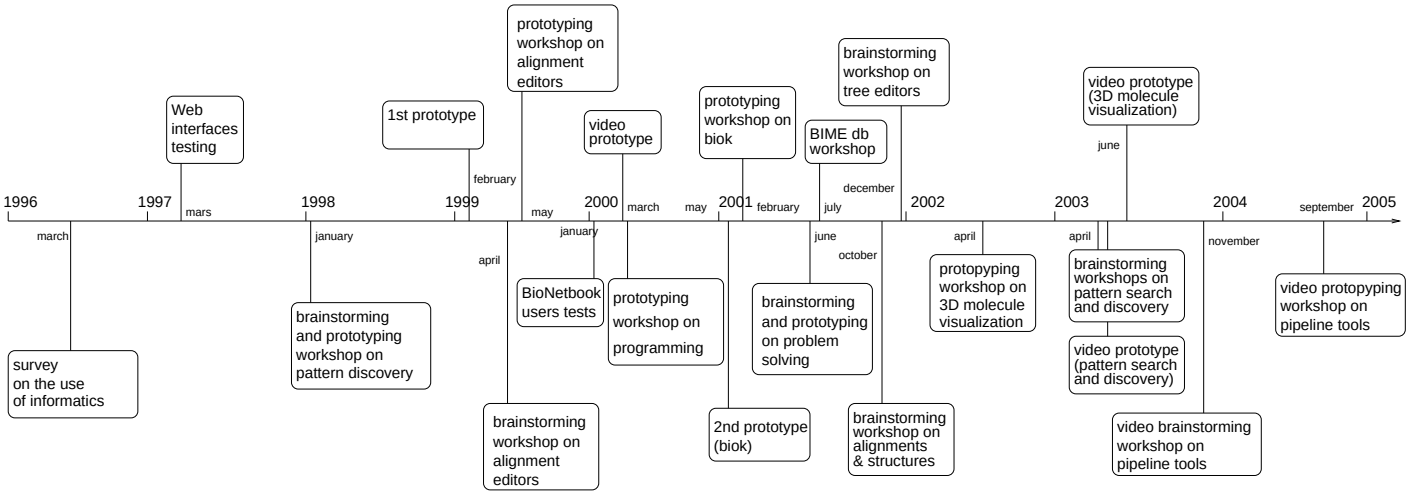

Figure 2. History of workshops.

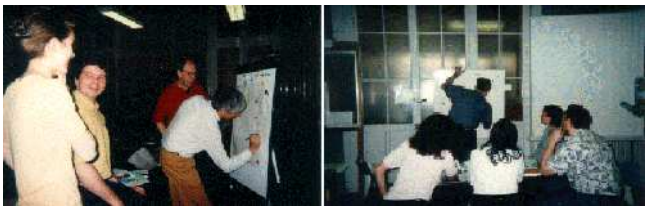

Figure 3. Prototyping a pattern-search and an annotation scenario

biologists, ”end-users” who have little or no training in programming. From their perspective, software is a tool for solving biology research problems, never an end in itself. We assumed that they would value creating flexible tools, if only because they might need to reuse them themselves, but this was rarely the case. In one interview, for example, a biologist expressed both surprise and amusement that his colleague, a bioinformatician, was strongly motivated to make his software usable by others. In another group discussion, several biologists expressed frustration with the IT center’s reluctance to write individual program solutions for them, finding it odd that the latter focus on creating general tools to be used by multiple biologists.

At the other extreme are computer scientists, who have little or no training in biology, but are charged with the problem of creating software tools for them. Unlike biologists, the object of their work is the software they create. Although it’s difficult to say that it is only for creating software... there are also computer scientists who have been “seduced” by biology... W: true, but not necessary to say here. Isn’t it easier to teach oneself programming than biology? One can’t get a job as a biology researcher without the right diplomas, but one can get a job programming without the right diplomas... Their training leads them to value software flexibility and the creation of generic tools that perform a variety of functions for diverse people with different needs.

Bioinformaticians bridge the gap, with training in both biology and computer science. We see two types of bioinformaticians: those who maintain a strong interest in biology and seek to create tools that both they and other biologists in the same research area can use, and those who are ”seduced” by the computer and turn their attention almost exclusively to creating biological software tools. Bioinformaticians have an advantage over computer scientists in that they deeply understand the software problem from the user’s perspective. On the other hand, in some instances, they define the problem very narrowly, from their personal perspective, which makes it more difficult to provide more generally useful tools.

6 Design of EUP and Visualization Tools

6.1 Interviews

Among a total of 65 interviews that were conducted in the context of various projects over the past seven years, about 30 were dedicated to end-user programming. They were mainly intended to collect use scenarios, or to observe biologists using scientific software. Interviews were generally informal and open: we often just asked the biologists to act in front of us a scenario of a recent bioinformatic analysis. Some of the interviews have been videotaped or recorded, and annotated.

Several of these interviews enabled us to observe biologists programming, either by using standard programming environments and languages such as C or C++, or, very often, scripting languages such as awk to parse large text files, perl to write simple Web applications, or Python to build scripts for analysing structural properties of proteins. We also observed uses of visual programming environments such as HyperCard or even visual programming languages. Khoros [34] for image analysis or Labview [18], for instance, are used in some laboratories, mostly due to their good libraries for driving hardware devices, and image or signal processing routines. We also observed various people using spreadsheets for performing simple sequence analyses.

During these interviews, we made several observations:

- *Re-use of knowledge*. Most of the time, biologists prefer using a technique or a language that they already know, rather than a language that is more appropriate for the task at hand, which is referred to as the assimilation bias by [5]. A researcher had learnt HyperCard to make games for his children, and used it in the laboratory for data analysis, even though nobody else knew it and thus was able to provide help. But the result was efficient and he almost never had to ask to the IT Center for help to find or install simple tools. Another researcher wrote small scripts in the only scripting language she knew: awk, although perl that is now often used and taught in biology would have been much more efficient. In summary, as long as the result is obtained, it does not matter how you get it. Similarly, a researcher tends to use a spreadsheet instead of learning to write simple scripts that would be be more suitable to the task.

- *Opportunistic behavior*. Generally and as described in [27], biologists, even if they can program, will not do so, unless they feel that the result will be obtained really much faster by programming. If this is not the case, they prefer to switch to a non-programming methods, such as doing a repetitive task within a word processor or performing an experiment at the bench. There is no requirement nor any scientific reward for writing programs. They are only used as a means to an end, building hypotheses.

- *Simple programming problems*. During his or her everyday work, a biologist may encounter various situations where some programming is needed, such as simple formatting or scripting (for extracting gene names from the result of an analysis program and use them for a subsequent database search) and parsing, or simple operations, not provided in the user interface, such a searching for characters other than A, C, G or T in a DNA sequence.

- *Need for modifying tools rather than building from scratch*. A frequent need for programming that we observed is to make a variant or add a function to an existing tool. Designing variants for standard published bench protocols is often needed in a biology laboratory. For instance, when constructing a primer for hybridization [1], it is often needed to adapt the number of washings according to the required length and composition of the primer, or to the product that is used. With software tools, this is however unfortunately seldom feasible, but it would be highly valuable since there are already many existing tools that perform helpful tasks, and biologists rarely want to build a tool from scratch.

- *Exploratory use of tools*. There is a plethora of tools, including new tools, for the everyday task of biologists, and these tools are often specialized for a specific type of data. This leads to a very interactive and exploratory use of computing tools [31]. For instance, an observed scenario started by the search of a protein pattern published in a recent paper. The user was looking for other proteins than those referred to in this paper and that also contained this motif. After an unsuccessful attempt - the results were too numerous for an interactive analysis - the researcher decided to use another program. This attempt failed again because his pattern was too short for the setting of this specific program. He then decided to extend it by adding another one, also belonging to the set of proteins mentioned in the paper. In the end, this enabled a final iterative analysis of each result. This is a brief summary that stands for many scenarios we have observed, often resulting in many failures due to a problem with the program, or with the format of the data.

  This typical behavior might both be a barrier to and a reason for programming. It can be a barrier by preventing a user to think of a more efficient way to get a result (leading to an "active" user behavior as described by [5]). However, at the same time, it can be a ground for programming since programming could help to rationalize, *a posteriori*, such an exploratory behavior. This, however, involves some kind of anticipation: for instance, it might be a good place for programming instruments such as history and macro recording.

[1] A primer is a short DNA sequence used to generate the complementary DNA of a given sequence

6.2 Workshops

biok (section 4.4) has involved a series of video brainstorming and prototyping workshops over several years from 1996 to 2004. We drew prototyping themes from brainstorming sessions (Figure 3) and from use scenarios, which based on interviews and observation. Each workshop involved from 5 to 30 people, with participants from the Institut Pasteur or other biological research laboratories, as well as biology researchers who were students in our programming course.

Finding potential dimensions for evolution From the very beginning of the design process, it is important to consider the potential dimensions along which features may evolve. Interviews with users help inform concrete use scenarios, whereas brainstorming and future workshops create a design space within which design options can be explore. As Trigg [43], Kjaer [17], [41] or [16] suggest, participatory design helps identify which areas in a system are stable and which are suitable for variation. Stable parts require functionality to be available directly, without any programming, whereas variable parts must be subject to tailoring.

For example, the visual alignment tool in *biok* vertically displays corresponding letters in multiple related sequences (Figure 1, back window). Initial observations of biologists using this type of tool [21] revealed that they were rarely flexible enough: biologists preferred spreadsheets or text editors to manually adjust alignments, add styles and highlight specific parts. It became clear that this functionality was an area requiring explicit tailoring support.

Design of meta-techniques Scenarios and workshops are important to effectively design meta-level features. Scenarios sometimes reveal programming areas as side issues. The goal is not to describe the programming activity per se, but rather to create an analogy between the task, how to perform it, and the relevant programming techniques. We identified several types of end-user programming scenarios:

- *Programming with examples*: One workshop participant suggested that the system learn new tags from examples (tags are visualization functions). Another proposed a system that infers regular expressions from a set of DNA sequences. These led to a design similar to SWYN [3].
- *Scripting*: one participant explained that text annotations, associated with data, can act as a "to do" list, which can be managed with small scripts associated with the data.
- *Command history*: a brainstorming session focusing on data versioning suggested the complementary idea of command history.

The *biok* tag editor design (Figure 1, front window) had to consider the following issues: Must programming be available in a special editor? Must it require a simplified programming interface? Should the user interface be interactive? Should it be accessible via graphical user interface menus?

We found prototyping workshops invaluable for addressing such issues: they help explore which interaction techniques best trigger programming actions and determine the level of complexity of a programming tool. For example, one group in an alignment sequence workshop built a pattern-search mockup including syntax for constraints and errors (Figure 3).

One ¡of the participatory design workshops was organized in the Winter of 2001 with five biologists to work on the *biok* prototype. Among the participants, four had followed a programming course, and programmed from time to time, but not in Tcl, except one who had programmed in Visual Tcl. Before the workshop, interviews had been conducted with discussions about the prototype, and participants were sent a small Tcl tutorial by email. The aim of the workshop was to experiment the prototype and get familiar with it through a scenario (instructions were provided on a Web page). They had to play with graphical objects, and define a simple tag. The issues that would arise during this part would then be discussed and re-prototyped during a second part. The scenario had also spontaneously been "tested" by one of the participants who brought some feedback about it. Although the workshop was not directly aimed at properly testing the prototype, the participants behaved as if it was, and this actually brought some insights on the design of the prototype - briefly and among the most important ones:

- The participants were somewhat disturbed by a too large number of programming areas: formula box, shell, method editor, ...
- They had trouble to understand, at a first sight, the various elements of the user interface and how they interact.
- They had the feeling that understanding the underlying model would help.

One of the the tasks of the scenario was to define a tag. The only tool that the participants had for this was an enhanced text editor, only providing templates and interactive choosers for the graphical attributes. This tool proved completely unusable and participants got lost. The tool was indeed too programmer-centered, and difficult to use, and there was no unified view of the tag definition. This led to another workshop shortly after this one, and after a long brainstorming session, one participant built a paper-and-transparencies prototype. We created an A3-size storyboard mockup and walked through the tag creation scenario with the biologists. The tag editor currently implemented in *biok* is a direct result of these workshops.

Participatory approaches are also helpful when designing language syntax [33, 9] or deciding on the *granularity* of code modification. As observed during the previously described workshop, the object-oriented architecture and the method definition task apparently did not disturb users that much. In a previous workshop that we started by displaying a video prototype showing the main features of *biok*, participants tended to adopt the words "object" and "method" that were used in the video. Interestingly, one of them used the term: "frame" all along the workshop in place of object, probably because objects in *biok* (and in the video prototype) are most often represented by graphical windows. In object-oriented programming terms, we found however that biologists are more likely to need new methods than new classes. Since defining new classes is a skilled modeling activity, we designed *biok* so that user modifications at the user level do not require sub-classing. User-edited methods are performed within the current method's class and are saved in directories that are loaded after the system. However, visualizing tags required the user to create new classes, which lead us to provide this as a mechanism in the user interface.

Setting a design context for tailoring situations Our observations of biologists showed that most programming situations correspond with breakdowns: particular events cause users to reflect on their activities and trigger a switch to programming [26]. Programming is not the focus of interest, but rather a means of fixing a problem. It is a distant, reflexive and detached "mode", as described in [46], [36] or [13]. While end-user programming tools may seek to blur the border between use and programming [10], it is important to take this disruptive aspect into account, by enabling programming *in context*. Developing and playing through scenarios are particularly useful for identifying breakdowns and visualizing how users would like to work around them.

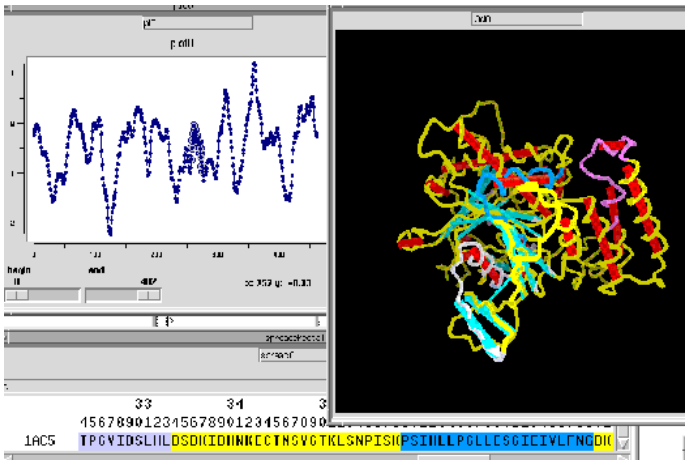

Figure 4. Analysing biological data: a plot displays the hydrophobicity curve of a protein sequence, that is simultaneously displayed in a 3D viewer. Structural properties of the protein, namely transmembrane segments, are simultaneously highlighted in the two protein representations. The user, by selecting parts of the curve, can check to which hydrophobicity peaks these segments might correspond.

7 Design of Web Tools

7.1 Distributed Computing in Bioinformatics

The development of computing tools for biology and genomics has increased at a fast pace to deal with huge genomic data and the need of algorithms to discover their meaning. A major part of these tools are available on the Web, either to provide access to a database, or as a convenient user interface of a program. This situation leads to several problems for the biologist, in particular to combine the use of several tools: each requires a different data format leading to many copy and formatting operations.

7.1.1 GRID Computing and Web services

Electronic workbenches, such as the NCSA Biology Workbench [42] or W2H [35] aim to provide an environment to help the biologist maintain his or her data and to find and combine tools adapted to each type of data. These tools however do not provide minimal workflow or pipeline construction.

Important and dynamic projects illustrate the efforts currently made to provide tools for distributed computing and integration: biopipe [15] for building flexible pipelines, bioperl [37] or biopython [6] for developing components, including remote execution and parsing ones. [38] describes technological attempts to integrate various data sources and services on a common architecture. Likewise, the BioMoby [45] and myGrid [40] projects aim to develop Web services and ontologies.

7.1.2 Mobyle project

Our project, called Mobyle, still in the design phase, relies both on previous software developments, such as Pise, a Web interface generator for programs on Unix [21] and on available components, such as bioperl or biopython. In particular, we intend to integrate the BioMoby project and to start with G-Pipe, a tool to build workflows, developed by external contributors on top of Pise.

One of the main objectives of this project is however, with a participary design approach [14], to address the following issues:

- Do these technologies solve the problems that actually arise in the combination of tools and in the setting of software protocols for the biologists?
- Are actual use problems addressed or even identified at all in these technological approaches?

In the following, we describe the studies, interviews and workshop, that we have conducted at the Pasteur Institute in order to get a deeper view on these issues.

7.2 Interviews

About 20 interviews have been organized during the last two monthes. We first contacted biologists by telephone to explain our project. We choosed biologists having used the Pasteur Institute Web server recently, as well as biologists having a significant activity in bioinformatics. Interviews were informal: we just asked the biologists to play before us a scenario of a recent bioinformatic analysis. All interviews have been videotaped. During these interviews, we made several observations:

- *Need for a stable and predictible set of known tools.*

  Most of the time, biologists prefer to use a technique or a language that they already know, rather than a language that is more appropriate for the task in hand. Since work at the bench can sometimes result in unpredictable outcomes, biologists generally tend to prefer tools that they control. For instance, we observed biologists who:

  – use outdated DOS programs, whereas they have Windows installed, or use outdated bioinformatics package on Unix instead of their improved version on the Web, not because they don't like the Web but rather because they know the Unix version,

  – stay within a given Web server providing an apparently exhaustive set of tools, eventhough better tools exist elsewhere.

Moreover, because Web tools tend to evolve unexpectedly, some biologists we have met prefer to install software on their local computer. This way, they better control the changes.

- *Dealing with equivalent objects*.

  It seems that biologists quite often maintain or have to use several versions of "equivalent" objects, which might be difficult to deal with. For instance:

  - same data files in different formats: they have to be kept, because tools for data (e.g sequence) format conversion *change* these data (see 7.3),
  - software versions: we discussed with a biologist who keeps several versions of a phylogeny inference program, just in case one of the features of an older version has disappeared in the more recent one,
  - data: one of the biologists we met had to deal with two versions of an annotated genome (mosquito); the issue for him was not to lost too much time in re-analysing the same data,
  - file and printout: the same object, either biological data, analysis result or software documentation, is often kept in two (or more) forms, e.g on the disk and on the paper.

- *Interactive nature of some tasks*.

  Analysis tasks supported by computer tools are not always automatic. Indeed, the biologist has either to check the accuracy or significance of a result, such as a score in a database homology search, and to decide to carry on an analysis according to complex criteria that are not possible to automate, or to extract subsets of data before proceeding. Moreover, the biologist has often to edit intermediate results, that are produced in a format that is not recognized by other tools, and automating such edition would require a little programming.

- *Anticipating*.

  Constructing a pipeline is similar to a programming activity, and programming is by nature anticipating. Pipeline tools, of course, aim to help users. In spite of this fact, users are often sceptical regarding sophisticated but difficult to learn systems: they have to be able to anticipate the benefits that they will draw from them. Users also need to anticipate their own needs, in order to perform an action that will help them in the future.

  With respect to anticipation, we observed several type of behaviours about:

  - anticipating the utility of a software for use or reuse,
  - bookmarking: biologist we met often decide not to bookmark an online tool - they seemed to be confident about the retrieval of the site,
  - saving: biologists often save temporary results, or alternative data formats,
  - customization: some bioinformatics tools, such as SeWer [2], enable the users to easily customize Web forms; however, very few biologists actually use them.

- *Constructing and organizing results: annotation, classification, naming, assemblies*.

  The vast majority of biologists we observed maintain a quite organized record of their analyses. Their files are carefully named after their content, the directories are often organized in accordance with species, genes and experiments names. The data, parameters and results that matter for the research activity are often assembled not only in the notebook, but also in Word files. Biologists also annotate printouts and keep them in classified folders.

- *Data flows*.

  General flows of data belong to diverse categories: input to output of a program, piping of an output to the input of another program, reformatting, transforming, filtering, extracting [39]. The most often used flow of data we observed is however by copy-and-paste, and this is not really supported by any smart tool [32].

- *Search and Retrieval*.

  Usually biologist do bibliographic work on a regular basis, but this is not the case with bioinformatics tools. One would expect that biologists, belonging to a fast evolving field, would try to discover the latest new tools in order to improve their work. Only one or two of the people we discussed with did a technological survey, about 3 or 4 times a year.

  However, this does not mean that search and discovery tools, such as Google or Web service directories are not needed: they are. But they tend to be rather used to find an object (data or program) that has already been used.

7.3 Video Brainstorming Workshop

There were four groups of six biologists, and four designers. Among biologists, half were trained bioinformaticians, having a significant knowledge in computing. The workshop was co-organized with the leader of the project to build an augmented laboratory notebook [28], which has several issues in common with our project: organisation of work, building of protocols, just to name a few.

Participants first freely put ideas on large paper sheets. Then, 6 rough paper prototypes of selected ideas were videotaped. The main topics developed by biologists were:

- *reusing executed commands as a script*: all groups played with the idea of reusing executed commands either as an history, a macro or a script, although in a different way. The history or macro was either a simple text file, or a colored list, where the user can remove, edit or re-order commands, and attribute colors to them, according to their importance. One group envisioned a

kind of temporal strip where icons representing data and programs appeared as the actions actually occurred.

- *using and defining a pipeline*: ready-to-use analysis pipelines are a popular idea among biologists, similar to their bench protocols. Two groups, rather composed of bioinformaticians, were interested in the definition of pipelines by the end-user. One of them built a prototype of a visual bench where programs and input data types were connected together in a graphical editor. After execution, clicking on a program's outgoing link enabled the user to access intermediate results. Like several existing tools, such as the Biology Workbench [42], Pise [21], or BioMoby [45] this pipeline editor is data type oriented: the user does only see compatible programs or data types.

  So, unlike the biologists interested in editable histories of commands, this group prefer to *anticipate* the definition of a sequence of command by using a sophisticated editor.

- *dealing with unwanted data transformation*:

  [39] explains that bioinformatics tools can be classified either as: filters, transformers, collections or forks. Unfortunately, transformers or filters often produce results that are not convenient for the biologists. For instance, they truncate data names - and names are very important for organisation and scientific matters - or they remove part of the transformed data without any notice. Two prototypes addressed this problem. One of them showed how to deal with name truncation by means of menus to select among alternative names produced during the various steps. In this prototype, tags on a phylogenetic tree replacing the original names of species could be changed at will by the user. Another prototype dealt with the unwanted removing of a part of a database entry.

- *dealing with desirable data transformation*:

  As exemplified by a prototype showing the extraction of a subpart of a sequence alignment, and as we observed during interviews, interactive manipulations, on the other hand, are also necessary and cannot be automated.

- *need of a global synthetic view on analyses*: the option to visualize several related analyses' results in a coherent way has been proposed. This meets our observation of assemblies of analysis data during the interviews. In a similar category, one group proposed the idea of visualizing the state of parallel analyses in a global window, in order to compare results: for instance the user would clone a subwindow to start a fork to explore an equivalent analysis.

7.4 Video Prototyping Workshop

The next workshop has been organized in September 2004 (see figure 2). This workshop involved 35 participants, including biologists, bioinformaticians, computer scientists, and an HCI researcher. Some prototyping work had been prepared with a bioinformatics student. She built several paper and video prototypes, as well as some actual Web tools to evaluate specific interactive features. We presented these prototypes at the beginning of the workshop, and after a brief brainstorming session to elicit ideas. Main topics were: searching for services, data extraction and filtering, history of commands, presentation and visualization of analyses results, pipelines building and customization. Each group built from 1 to 2 video prototypes.

8 Conclusion

Participatory design workshops proved popular: despite their busy schedules, many people voluntarily attended multiple workshops. They viewed these as a valuable forum for scientific exchange, ranging from simple tips and techniques for using particular software tools to deep exchanges of scientific ideas. Participatory workshops and prototyping activities [4] also proved invaluable for building not only usable, but *useful* tools.

Participatory programming integrates participatory design and end-user programming. Participatory design is used for the design of an end-user programmable tool, yet biologists programming artifacts also participate in the making of tools. These artifacts are either produced by local developers, observed during interviews, or they can be produced by end-users using *biok*. As illustrated by the Web design example, end-user programming issues can unexpectedly show up, at least in our domain, in the context of tools that are not directly related to end-user programming.

References

[1] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schaeffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[2] Malay Kumar Basu. Sewer: a customizable and integrated dynamic html interface to bioinformatics services. *Bioinformatics*, 17(6):577–578, June 2001.

[3] Alan Blackwell. Swyn: A visual representation for regular expressions. In *Your Wish is My Command: Giving Users the Power to Instruct their Software*. Morgan Kaufmann, 2000.

[4] Frederick P. Brooks. No silver bullet,essence and accidents of software engineering. *Computer Magazine*, April 1987.

[5] J. M. Carroll and M. B. Rosson. *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction. J.M. Carroll ed.*, chapter 5, The paradox of the active user, pages 80–111. Cambridge, Mass: MIT Press, 1987.

[6] Brad Chapman and Jeff Chang. Biopython: Python tools for computation biology. *ACM-SIGBIO Newsletter*, August 2000.

[7] M.F. Costabile, D. Fogli, C. Letondal, P. Mussio, and A. Piccinno. Domain-expert users and their needs of software development. In *In Proceedings of the HCI 2003 End User Development Session*, 2003.

[8] Allen Cypher. *Watch What I Do. Programming by Demonstration*. MIT Press, 1993. 652 pages.

[9] C. K. V. da Cunha and C. S. de Souza. Toward a culture of end-user programming understanding communication about extending applications. In *Proceedings of the CHI'03 Workshop on End-User Development*, April 2003.

[10] Y. Dittrich, L. Lundberg, and O. Lindeberg. End user development by tailoring. blurring the border between use and development. In *Proceedings of the CHI'03 Workshop on End-User Development*, April 2003.

[11] Michael Eisenberg. Programmable applications: Interpreter meets interface. *ACM SIGCHI Bulletin*, 27(2):68–93, April 1995.

[12] Michelle Gantt and Bonnie A. Nardi. Gardeners and gurus: patterns of cooperation among cad users. In *ACM conference on Human Factors in Computing Systems (Proceedings) (CHI '92)*, pages 107–117. ACM Press, 1992.

[13] J. Greenbaum. PD, a personal statement. *CACM*, 36(6):47, June 1993.

[14] Joan Greenbaum and Morten Kyng. *Design at Work: Cooperative Design of Computer Systems*. Hillsdale, New Jersey Lawrence Erlbaum Associates, 1991.

[15] S. Hoon, KK. Ratnapu, JM Chia, B. Kumarasamy, X. Juguang, M. Clamp, A. Stabenau, S. Potter, L. Clarke, and E. Stupka. Biopipe: a flexible framework for protocol-based bioinformatics analysis. *Genome Research*, 1363103, 2003.

[16] Helge Kahler. Developping groupware with evolution and participation. a case study. In *Proceedings of the Participatory Design Conference 1996 (PDC'96), Cambridge, MA*, pages 173–182, 1996.

[17] A. Kjaer and K.H. Madsen. Participatory analysis of flexibility. *CACM*, 38(5):53–60, May 1995.

[18] Labview: a demonstration. unpublished, 1987.

[19] C. Letondal and Olivier Amanatian. Participatory design of pipeline tools and web services in bioinformatics. In *Proceedings of the "Requirements Capture for Collaboration in eScience Workshop, NESC, Edinburgh, UK*, January 2004.

[20] Catherine Letondal. Software review: alignment edition, visualization and presentation. Technical report, Pasteur Institute, Paris, France, may 2001. http://bioweb.pasteur.fr/cgi-bin/seqanal/review-edital.pl.

[21] Catherine Letondal. A web interface generator for molecular biology programs in Unix. *Bioinformatics*, 17(1):73–82, 2001.

[22] Catherine Letondal. *End User Development: Empowering people to flexibly employ advanced information and communication technology*, chapter Participatory Programming: Developing programmable software tools for end-users. Kluwer Academic Publishers, 2005.

[23] Catherine Letondal and Uwe Zdun. Anticipating scientific software evolution as a combined technological and design approach. In *USE2003: Second International Workshop on Unanticipated Software Evolution*, 2003.

[24] Henry Lieberman, editor. *Your Wish is My Command: Giving Users the Power to Instruct their Software*. Morgan Kaufmann, 2000.

[25] W.E. Mackay, A. Ratzer, and P. Janecek. Video artifacts for design: Bridging the gap between abstraction and detail. In *Proceedings of ACM DIS 2000, Conference on Designing Interactive Systems. Brooklyn, New York.* ACM Press, 2000.

[26] Wendy E. Mackay. Triggers and barriers to customizing software. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, pages 153–160. ACM Press, 1991.

[27] Wendy E. Mackay. *Users and Customizable Software: A Co-Adaptive Phenomenon*. PhD thesis, Massachusetts Instititute of Technology, 1991.

[28] Wendy E. Mackay. Using video to support interaction design. In *Proceedings of the CHI'02 Conference on Human Factors in Computing Systems, Minneapolis, MN, DVD Tutorial*. ACM Press, April 2002.

[29] Anders Morch. Three levels of end-user tailoring: Customization, integration, and extension. In M. Kyng and L. Mathiassen, editors, *Computers and Design in Context.*, pages 51–76. The MIT Press, Cambridge, MA, 1997.

[30] Gustaf Neumann and Uwe Zdun. Xotcl, an object-oriented scripting language. In *Proceedings of 7th Usenix Tcl/Tk Conference (Tcl2k), Austin, Texas, Feb 14-18*, 2000.

[31] V. L. O'Day, A. Adler, A. Kuchinsky, and A. Bouch. When worlds collide: Molecular biology as interdisciplinary col laboration. In *In Proceedings of ECSCW'01*, pages 399–418, 2001.

[32] Milind S. Pandit and Sameer Kalbag. The selection recognition agent: instant access to relevant information and operations. In *Proceedings of the 2nd international conference on Intelligent user interfaces, Orlando, US*, pages 47 – 52, 1997.

[33] J.F. Pane, C.A. Ratanamahatana, and B. Myers. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, 54(2):237–264, February 2001.

[34] J. Rasure, D. Argiro, T. Sauer, and C. S. Williams. A visual language and software development environment for image processing. *International Journal of Imaging Systems and Technology*, 2:183–199, 1990.

[35] M Senger, T Flores, K Glatting, P Ernst, A Hotz-Wagenblatt, and S Suhai. W2h: Www interface to the gcg sequence analysis package. *Bioinformatics*, 14:452–457, 1998.

[36] Randall B. Smith, David Ungar, and Bay-Wei Chang. The use-mention perspective on programming for the interface. In *Languages for Developping User Interfaces*. Jones and Bartlett, 1992.

[37] Jason E. Stajich, David Block, Kris Boulez, Steven E. Brenner, Stephen A. Chervitz, Chris Dagdigian, Georg Fuellen, James G.R. Gilbert, Ian Korf, Hilmar Lapp, Heikki Lehvslaiho, Chad Matsalla, Chris J. Mungall, Brian I. Osborne, Matthew R. Pocock, Peter Schattner, Martin Senger, Lincoln D. Stein, Elia Stupka, Mark D. Wilkinson, and Ewan Birney. The bioperl toolkit: Perl modules for the life sciences. *Genome Research*, 12(10):1611–1618, 2002.

[38] L. D. Stein. Integrating biological databases. *Nature Reviews, Genetics*, 4(5):337–345, May 2003.

[39] Robert Stevens, Carole Goble, Patricia Baker, , and Andy Brass. A classification of tasks in bioinformatics. *Bioinformatics*, 17(2):180–188, February 2001.

[40] Robert D. Stevens, Alan J. Robinson, and Carole A. Goble. myGrid: personalised bioinformatics on the information grid. *Bioinformatics*, 19(Suppl. 1):i302–i304, July 2003.

[41] O. Stiemerling, H. Kahler, and V. Wulf. How to make software softer - designing tailorable applications. In *In Proc. DIS'97 (Amsterdam)*, pages 365–376, 1997.

[42] S Subramaniam. The biology workbench: A seamless database and analysis environment for the biologist. *"Bioinformatics", Proteins, 32:, 1998.*, 32(1):1–2, July 1998.

[43] Randall H. Trigg. Participatory design meets the mop: Informing the design of tailorable computer systems. In *Proceedings of the 15th IRIS (Information systems Research seminar In Scandinavia) Gro Bjerknes, Tone Bratteteig, Karlheinz Kautz (eds.), August 1992, Larkollen, Norway.*, 1992.

[44] P. Tuffery, B. Neron, M. Quang, and C. Letondal. biok: Molecular visualization. Technical report, Pasteur Institute, Paris, France, 2003. http://www.pasteur.fr/ letondal/biok/i3DMol.html.

[45] MD Wilkinson and M. Links. Biomoby: an open-source biological web services proposal. *Briefings in Bioinformatics*, 3(4):331–341, December 2002.

[46] Terry Winograd. From programming environments to environments for designing. *CACM*, 38(6):65 – 74, June 1995.