BIOINFORMATICS

Vol. 17 no. 1 2001
Pages 73–82

A Web interface generator for molecular biology programs in Unix

Catherine Letondal[1, 2]

[1]Institut Pasteur, Service d'Informatique Scientifique, 28 rue du Docteur Roux, 75724 Paris Cedex 15, France and [2]Laboratoire de Recherche en Informatique, URA 410 du CNRS, LRI - Bâtiment 490 - Université de Paris-Sud, 91405 Orsay Cedex, France

Received on May 16, 2000; revised on June 27, 2000; accepted on July 7, 2000

ABSTRACT

Motivation: Almost all users encounter problems using sequence analysis programs. Not only are they difficult to learn because of the parameters, syntax and semantic, but many are different. That is why we have developed a Web interface generator for more than 150 molecular biology command-line driven programs, including: phylogeny, gene prediction, alignment, RNA, DNA and protein analysis, motif discovery, structure analysis and database searching programs. The generator uses XML as a high-level description language of the legacy software parameters. Its aim is to provide users with the equivalent of a basic Unix environment, with program combination, customization and basic scripting through macro registration.

Results: The program has been used for three years by about 15 000 users throughout the world; it has recently been installed on other sites and evaluated as a standard user interface for EMBOSS programs.

Availability: The program is freely available over the Internet at the following URL: http://www.pasteur.fr/∼letondal/Pise/.

Contact: pise@pasteur.fr

INTRODUCTION

In Unix, some programs are particularly problematic for inexperienced users, as they assume basic skills about the file system or the Unix documentation, such as renaming, moving files or understanding directory naming. Users often cannot see what they have done and which files are available unless they ask for them. Users are also sometimes required to edit files according to strict syntactic rules or even to write small scripts. Finally, it is often necessary to convert from a specific program format into another program format, or to use part of a result as input to another program.

It would certainly be useful to add a friendly interface to each program, but this is totally unfeasible given the increasing amount of new software coming out every day. That is why we have decided to develop a software, called Pise (Pasteur Institute Software Environment), that, given an abstract definition of a program's parameters, can generate a Web-based interface. The software we have written is not currently limited to Web interfaces: there are other modules for Tcl/Tk (as a Web client), X11 graphic interfaces or simple `curses` menus. However, it quickly became apparent that the most useful interface would be a Web interface, because they are widely available and users are accustomed to them. This system addresses the issues introduced above:

1. it provides a *homogeneous* interface to all the programs, suppresses the *syntactic complexity*, and, by giving explanations and access to documentation, lowers the *semantic complexity*,
2. it helps to deal with *files*, to *combine programs* and to register *scripts*.

In the first part of this paper, we present our model of the interfaces to Unix programs and we briefly describe the Web interface generator and its technical aspects. In the second part we focus on the interactive features which, for the main needs of these scientific applications, provide users with the equivalent of a basic scripting environment.

HOMOGENEOUS INTERFACES TO HETEROGENEOUS PROGRAMS

Unix program interface description

Although Unix provides a standard syntax for commands, it is sometimes unused by scientific programs, which are very often written by scientists themselves who do not necessarily have a Unix background. Some programs were designed in order to run the same way on different platforms and therefore do not use a Unix-like style. These are the reasons for a wide range of different program appearances, from command-line with named or positional parameters to menu-driven or parameter files based programs.

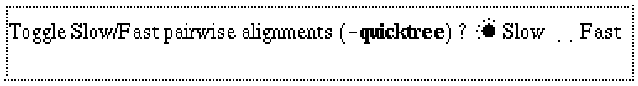
Toggle Slow/Fast pairwise alignments (-**quicktree**) ? ● Slow ○ Fast

Fig. 1. HTML field.

Parameter attributes. Parameters may be typed and described by a common set of attributes. Types include *string*, *integer*, *float*, *list* (0,1 or multiple choices), *data input*, *results* and biology specific types, like sequence data. Parameters may be grouped into paragraphs. Simple dependencies may be described, like a parameter not having to be handled given specific conditions. The attributes of parameters are:

Presentation attributes:

— prompt displayed on the form, size of the field, level of use, displayed default values, comments, paragraph grouping.

Control attributes:

— mandatory, scale for integers or float, default values, pre-conditions. Controls may be used to describe complex dependencies between parameters.

Command building:

— code to convert a value into the command syntax, position on the command line or in a file.

Results:

— pattern of output files names; an attribute may describe types of output files for the purpose of combination of programs: these types are free and application oriented.

The following example shows the definition of the `-quicktree` parameter in the Clustalw program:

```
<parameter ismandatory="1" issimple="1" type="Excl">
<name>quicktree</name>
<attributes>
  <prompt>
    Toggle Slow/Fast pairwise alignments (-quicktree)
  </prompt>
  <vlist>
    <value>slow</value> <label>Slow</label>
    <value>fast</value> <label>Fast</label>
  </vlist>
  <vdef><value>slow</value> </vdef>
  <comment>
    <value>
      slow: by dynamic programming (slow but accurate)
    </value>
    <value>
      fast: method of Wilbur and Lipman
           (extremely fast but approximate)
    </value>
  </comment>
  <group>2</group>
  <format>
    <language>perl</language>
    <code>($value eq "fast")? "-quicktree": " " </code>
  </format>
  <precond>
    <language>perl</language>
    <code>($actions =~ /align/ )</code>
  </precond>
</attributes>
</parameter>
```

Figure 1 shows the corresponding generated HTML field:

A little programming.... Some attributes require a little programming:

1. some code is needed to perform the string conversion from value to the actual command syntax (as the `format` attribute in the example above);
2. boolean expressions may define simple dependencies between parameters (as the `precond` attribute above, specifying that this parameter should not be taken into account unless the `$actions` variable is equal to ‘align’);
3. complex controls may be performed on entered values.

In practice, for the Web interface builder, the language is `perl`, because the CGI is written in that language and small pieces of code may be dynamically evaluated. As the code to be evaluated is under the Web interface maintainer’s responsibility, there is no security issue here.

Formal description. The above description has been formalized as an XML DTD (http://www.pasteur.fr/∼letondal/XML/pise.dtd). Among the advantages of this approach, are the simplicity of the parsers and the availability of many tools, including edition and validation tools. This makes the addition of a new domain specific type of parameter rather easy. As an example, there is a specific type of input for molecular sequences (DNA or proteins), which is dealt with at the CGI level (for file format conversions): this specific type has a special attribute, `seqfmt` that describes the formats that are accepted by the programs for molecular sequence files. The following example shows the definition of such a parameter.

```
<parameter ismandatory="1" issimple="1" type="Sequence">
<name>infile</name>
<attributes>
    <attribute>
        <seqfmt>
            <value>8</value>
```

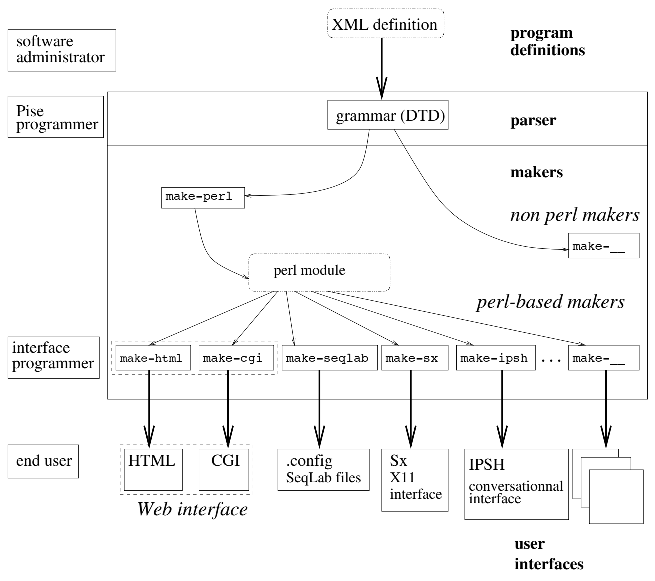

Fig. 2. General architecture.

```
            <value>3</value>
            <value>4</value>
            <value>15</value>
            <value>100</value>
        </seqfmt>
    </attribute>
</attributes>
</parameter>
```

Technical aspects

The preceding description is the only part of the software that is to be understood and mastered by the Web interface maintainer when defining the attribute set for a new program. The following explains how the Web interface builder is implemented.

As shown in Figure 2, interface builders (e.g. `make-html`, `make-cgi`, etc...) rely on a `perl` module generated from the XML description of a program. This module contains all the information needed to generate the HTML form and the CGI script. Other interface builders relying on another language (Java, Tcl, ...) could also be developped.

HTML design. We ran some usability tests with a dozen users grouped in pairs, one person telling the other what s/he intended to do, the other person describing what s/he was seeing. Among a lot of useful information, like where to put the submit button, how to indicate on-line help, these tests showed that it is necessary to design two levels of use:

1. a simple level, without fancy capabilities, giving access only to the mandatory parameters that do not have a predefined default value, as well as the input data;
2. an advanced level, giving access to as many parameters as available in the program, no matter how complex they are. Parameters may be grouped into paragraphs. There is a link from the beginner level form to the advanced one. Documented default values are provided or pre-selected in choices lists.

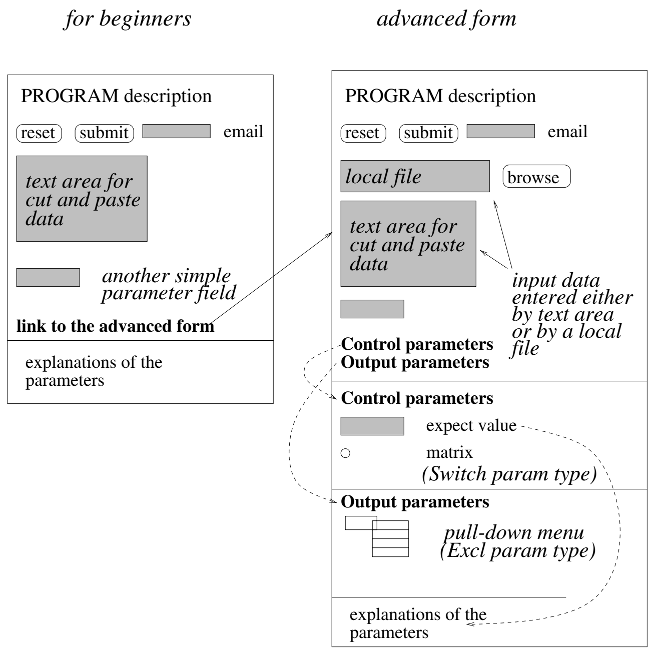

Fig. 3. HTML design: simple and advanced form layout.

The on-line help consists of comments at the end of the HTML document. Parameter headings link to the appropriate part of this section.

As can be seen in Figure 3, except the file upload facility added in the advanced form, our form design relies on basic HTML and we deliberately do not use any frames, applets or even JavaScript on the client side. This facilitates access by everyone.

CGI design.

Immediate vs delayed results. As some analyses can be very long—over several days—and owing to the synchronous aspect of a CGI based Web interface, it was necessary to design a delayed results delivery mechanism, e.g. sending results by email. We have decided to distinguish two cases:

1. short analyses ($\leqslant$60 s), where the results are immediately available;
2. long analyses (>60 s), where the response has to be sent by email.

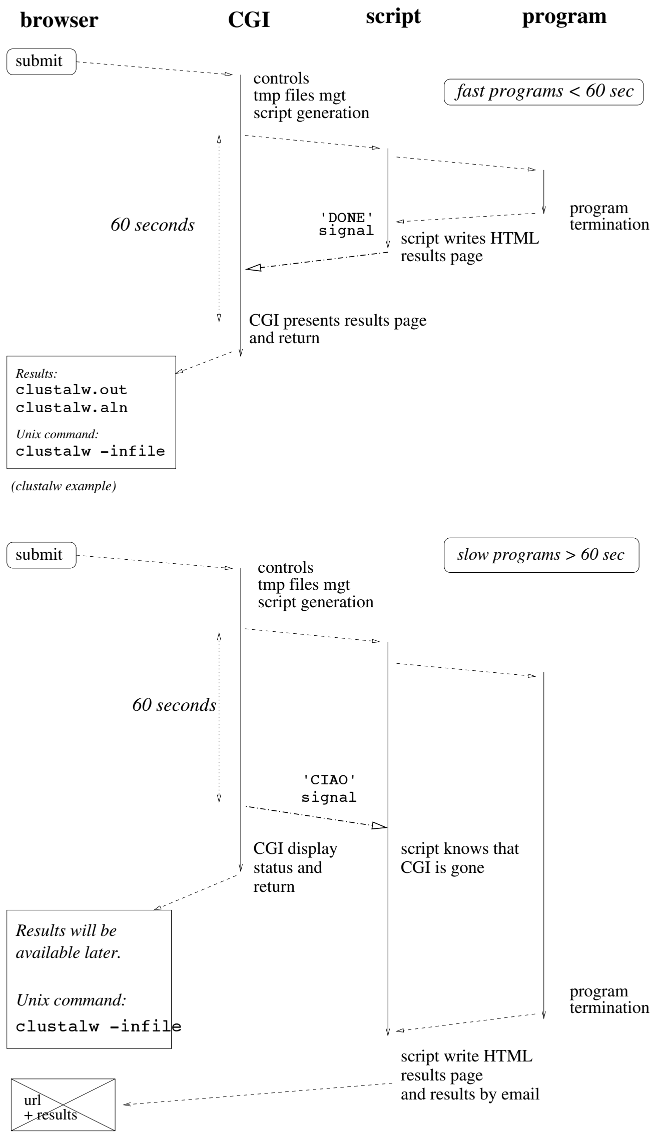

Fig. 4. CGI design: client/server synchronization.

Figure 4 shows how the two different cases are handled by exchanging Unix signals between the CGI and the generated script.

In the case of delayed results, the user can choose:

1. to be notified by an email containing only the URL of the results page (this is the preferred method);
2. to receive all the results by separate emails (default);
3. to receive results as MIME attachments.

The 60 s threshold is somewhat arbitrary and could be more flexible: in Javamatic (Phanouriou and Abrams, 1997), the user may specify a time threshold after which the server returns a URL.

Redundant submissions. As some analyses take a lot of CPU resources, it is very important to prevent similar submissions (easily detected by a checksum on the submitted data). There are two cases to be distinguished:

1. the user has clicked twice on the `Submit` button;
2. the job is not done after a certain time, and the user submits it again (just in case);

The first case is handled during the 'interactive' period within the first 60 s (see above): both submissions are cancelled, and a message is displayed explaining the problem and asking the user to resubmit. In the second case, the first job is not killed. A message is displayed explaining that a similar job has been submitted, and that the user should wait for the results.

User data management issues.

1. Security: the result files are stored in a temporary directory, whose name is built on two concatenated numbers: PID and time. The resulting temporary url thus behaves like a key. The server log files, where these temporary urls are listed, should be read protected.
2. Lifecycle of the results: the distribution includes maintenance scripts for file cleaning. At the Pasteur Institute, the results are stored for 5 days, but this decision belongs to the local software or system administrators.

SCRIPTING BY THE WEB

As we have observed, without the help of a Web system (client and server), biologists have trouble getting the best of all the scientific tools that are available to them. A Web system for scientific applications should therefore aim to fulfil the basic needs of users, which we identify as being:

1. getting the same kind of results as those obtained by stand-alone programs, although not in the same way;
2. combining programs to perform complete analyses;
3. customizing and storing procedures to automate standard or well-proven analyses.

Files

In our system, we have tried to keep the power of a Unix environment and to lower the file manipulation burden for the biologist:

- all the file manipulations required to run the program may be described in the XML specification; for instance, the following shows a `format` attribute (whose aim is to build the command line), with a Unix `mv` command for changing a filename:

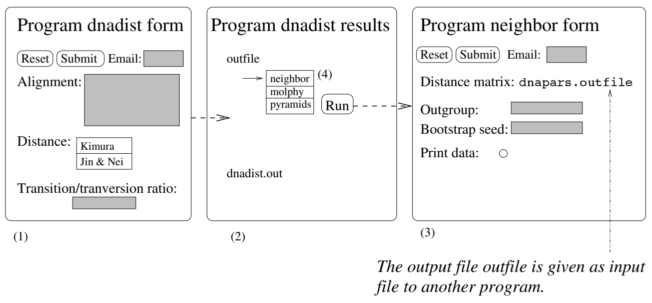

Fig. 5. Piping menus.

```
<attribute>
  <format>
      <language>perl</language>
      <code>
        "seqboot < seqboot.params;
          mv outfile infile;"
      </code>
  </format>
</attribute>
```

- the actual output files described in the program definition as interesting to the user—namely all the files produced by the biology program, are shown as hypertext links on the results HTML page; the following example shows an attribute, `filenames`, which indicates that all the output files ending by `phy`, `msf` or `pir` suffixes are to be presented to the user.

```
<parameter type="Results">
 <name>aligfile</name>
     <attribute>
         <filenames>*.phy *.msf *.pir
           </filenames>
     </attribute>
</parameter>
```

Combining programs

In scientific applications, as well as in many Unix tools, a program's output may be fed into another program's input. This feature, often implemented by the well-known Unix pipeline mechanism, or more simply by intermediate files, has been designed in a way suitable for an hypertext system.

As shown in the Figure 5, an output file presented on the results page (2), `outfile` may be redirected to another program's interface: `neighbor` (3), by means of a menu displayed next to the file name (4), which includes all the programs that could take this file as input (`neighbor`, `molphy` and `pyramids`).

This pipeline feature is rather popular, for it gives an opportunity to explore new programs and eliminates all the file manipulations that would be required to achieve the same thing directly in Unix. The graph of all the possible paths may be seen at this address: http://www.pasteur.fr/~letondal/Pise/gensoft-map.html.

The Tables 4, 5 and 6 in the Appendix gives the actual use that has been logged over 1 year. They show that:

1. some programs would have been used more rarely if not presented in a connection menu `seqgen` or `bionj`);
2. some programs are 'visited' through a connection menu, but not always used (`distquart`, `buildmodel` or `homology`): users may be just curious to see what kind of analysis they would perform, but do not need them at this specific time;
3. the pipeline mechanism is mostly used when there is a data format issue: it is not easy to edit a matrix file, so the program that computes and outputs such a file is very often run first and piped to the one which takes the matrix as input.

This mechanism is implemented by an attribute associated to the parameters at both ends of the pipeline.

1. first end of the pipeline (in program `dnapars` definition): an output file is defined as being from type `dist_matrix`.

```
<parameter type="Results">
<name>outfile</name>
<attributes>
    <attribute>
        <pipe>
            <pipetype>dist_matrix </pipetype>
        </pipe>
    </attribute>
</attributes>
</parameter>
```

2. last end of the pipeline (in program `neighbor` definition): an input file is given the same type.

```
<parameter type="InFile">
<name>infile</name>
<attributes>
    <attribute>
        <pipe>
            <pipetype>dist_matrix
              </pipetype>
        </pipe>
    </attribute>
</attributes>
</parameter>
```

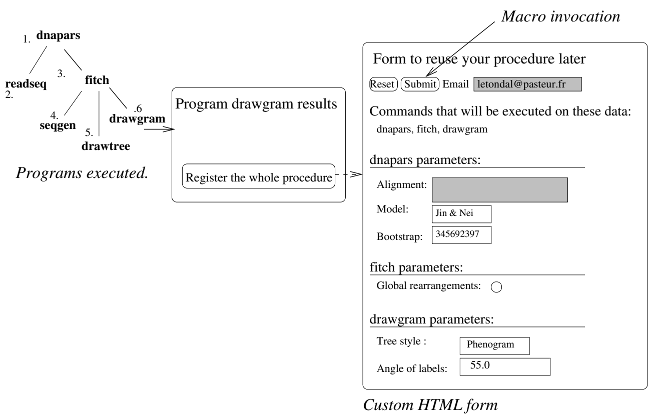

Fig. 6. Web macro registration.

Table 1. `perl` API for Web jobs

`$job=pisejob->new("command");`	*Create a new job for this command*
`$job->args("param", "value");`	*set parameter param to value*
`$job->param_type("param");`	*return the type of this parameter*
`$jobid = $job->submit;`	*submit the job*
`if ($job->terminated($jobid))...`	*test if job has terminated*
`$job->get_results($jobid);`	*get URLs of program's results*
`$job->output($result);`	*return the result as a string*
`$job->save($result, $file);`	*save this result in $file*
`$job->get_pipes ($result);`	*get programs that may take this result as input*
`$job->piped_file_type($result);`	*type of this result for feeding another program's input*

Customizing and registering Web macros

The system offers two ways to save and reuse the user's work. The first one is to save a form pre-filled with the user's values. This is achieved by using the HTML generator to generate a form with the user's values instead of the standard ones.

The second one is to save an entire procedure as a macro. This is useful whenever the user has chained several programs with data pipelines. On the results page, the user is presented with a 'Register the whole procedure' button, that builds a script which is able to redo the actions performed by him/her.

As shown in Figure 6, the set of actions to be registered in this procedure is defined by the entire dataflow chain of piped programs, which starts from the initial form `dnapars`) followed by the intermediate programs having been piped sequentially (`fitch`) until the last one where registration has been requested (`drawgram`). Therefore, only one branch of the tree formed by the user's explorations among the programs is stored, from root to leaf (here: steps 1, 3 and 6 are registered). No specific action is required from the user to start the registration.

The registered procedure is available to the user as a custom CGI perl script, which is stored on the server for a limited amount of time—as decided by the local system administrator. This CGI may be activated from a custom HTML form, that may be either bookmarked or saved. In this form, the user only has to provide mandatory parameters and to fill in initial data.

The registered macro relies on a `perl` API which is able to drive jobs launched on the Web server. This API uses XML parsing tools which makes it possible to get all the data concerning the context of each job (parameters, piped files, ...) from the HTML results pages, which are XHTML complaint. Table 1 shows the main methods of this `perl` module.

This API is also intended for programmers who might be interested in automating their analyses, for it constitutes an easier level of use than the existing Unix program interface: they do not have to worry about the parameter syntax, and may easily combine programs. Several programmers at the Pasteur Institute have already shown interest in this scripting API; the main reason is that it offers the possibility to analyze and filter results. Local users can save the Perl script generated by the macro registration facility and use it as a basis to develop more complex scripts with the above API. This method is easier and faster than writing a script from scratch.

STATISTICS OF USE

Table 2 summarizes the use of this service over the last 19 months (from October 1998 to May 2000). This use has increased a lot during this last year; almost half of the jobs have been submitted during the last 8 months. Some figures must be taken with caution: the number of users, for instance, is computed after email addresses that are required to submit a job. Since there is no semantic check of the email for external users, the only fact we can rely upon is that, whenever the user enters a fake address, s/he will not be able to receive results, which applies only for long jobs. A very rough estimation gives a rate of about 10% errors in email addresses.

The success ratio (submitted jobs/number of POST requests) gives the number of jobs that have passed the controls performed by the CGI: mandatory parameters, allowed values, or more complex controls described in parameters' definitions. This ratio does not mean at all that the program has succeeded, nor that the input data are correct. This is unfortunately a major issue, and one of the main causes of errors, that we are trying to fix by a format converter for molecular sequence data.

Users are defined as 'advanced users' whenever they have run an advanced job, even if only once; related figures are therefore probably over estimated.

Table 2. Statistics of use within the last 19 months

Number of different programs used	Total number of jobs	% advanced jobs	Total number of users	% advanced users	Different programs by user	Jobs by user	Success ratio within last year
253	169 065	56.5	15 044	59.8	2.3	11.2 (7.0)	76.1%

Table 3. Daily and monthly current statistics (today)

Daily number of jobs	Monthly number of users	Number of different programs used monthly
400	1850	175

LESSONS LEARNED

User feedback

In addition to the few usability tests performed at the beginning of the design, the service has evolved as a result of the observations of the log files, reporting of errors and user emails (about 150 messages as of today). We encourage users to signal errors and possible improvments by a help feature that enables the user to send a message pre-filled with all the job's context data and completed by his/her own text. The user receives a copy of this email, which in our opinion is rather important, to give the feeling that something has really happened. We try to answer almost every message. A lot of errors have been corrected thanks to these messages, including errors in the scientific software itself, which are exposed to heavy use through this service.

Developers feedback

One of the aims of this service is to promote bioinformatic research and software development, by providing biologists with access to new algorithms. This explains why feedback from software authors is generally positive.

RELATED WORK

The idea of integrating applications into the Web is not a new one (Lonczewski, 1995; van Doorn and Eliëns, 1995). In the field of molecular biology, there are numerous online services, either to search databases or to perform sequence analyses, such as phylogeny, protein structure prediction, and so on (see the more than 900 links in Pasteur database of resources for biologists).

Web-based interface generators

Application wrappers for using the Web as a program interface, such as Javamatic (Phanouriou and Abrams, 1997), AppLab (Senger, 1999) or W2H (Senger *et al.*, 1998) usually rely on a high-level description of the command-line driven program that is fed to an interface builder. The first two systems referenced above consist of an applet generator that dynamically builds a graphic client issuing requests on a server to run applications. AppLab uses CORBA IDLs as a high-level program description language. Both present the same advantage as our system: no account is needed to perform computations; no knowledge about command-line syntax is necessary. W2H was first designed as a WWW interface to the GCG package (Devereux *et al.*, 1984) and has features for handling user sessions and preferences. The interface is more sophisticated: it includes file management and application menus, as well as database searches and graphical outputs. None of these provides program combination or macro registration features.

Dataflow, program combination

What we have called program combination, or pipelines, is so useful for biologists that there are at least two other systems that implement such an idea: the NCSA Biology WorkBench (Subramaniam, 1998) and (The BioNavigator, provided by eBioinformatics, unpublished data). In the latter, only programs that can process selected input are presented to the user in the 'WorkBench'. Both systems provide a very good integration of database searches and data input: data may be easily extracted from search result lists, and combined with file upload or cut and pasted data. The main drawback of these two approaches is, in our opinion, the cost for users to understand each system model before being able to use it: both systems require the user to log in and to start a session, or a project, and to anticipate the steps of a well defined protocol; that is probably why both systems provide a demo. We have preferred a more 'natural' approach: performing a task involving multiple steps does not need any planning or session

Table 4. Pipes consumers

Piped program (to)	Total uses of connection menu to this program	Number of actual submissions from the piped program	Total submissions of this program	%	How many providers (from)	Biggest provider
neighbor	4742	5283	7714	68.5	3	dnadist (2405)
drawtree	4694	5241	8101	64.7	20	neighbor (1391)
consense	3197	3507	3595	97.6	20	neighbor (936)
drawgram	3192	3831	5022	76.3	18	neighbor (1000)
boxshade	1648	1953	7691	25.4	8	clustalw (1579)
seqgen	1267	708	904	78.3	18	clustalw (320)
fitch	939	1040	1711	60.8	3	protdist (509)
molphy	935	537	2601	20.6	7	protdist (520)
dnadist	932	877	6410	13.7	5	clustalw (889)
loadseq	734	779	1616	48.2	1	loadseq (734)
clustalw	690	541	21 840	2.5	14	loadseq (295)
bionj	659	628	868	72.4	3	protdist (346)
seqsblast	648	345	412	83.7	4	blast2 (356)
distquart	632	248	267	92.9	10	clustalw (372)
protdist	619	571	6728	8.5	7	clustalw (570)
mview_blast	463	328	544	60.3	6	fasta (335)
buildmodel	459	178	394	45.2	12	clustalw (269)
dnapars	436	372	2834	13.1	5	clustalw (408)
kitsch	403	467	787	59.3	3	protdist (243)
pyramids	397	328	622	52.7	3	protdist (266)
drawpyr	377	423	425	99.5	1	pyramids (377)
mview_alig	329	451	635	71.0	8	clustalw (302)
prettyalign	300	281	297	94.6	7	clustalw (241)
maeval	289	117	270	43.3	2	mafold (288)
Puzzle	253	254	2747	9.2	5	clustalw (228)
homology	248	116	397	29.2	6	clustalw (220)
patser	243	138	138	100.0	2	consensus (241)
protpars	212	199	3358	5.9	6	clustalw (187)
fastdnaml	199	174	2562	6.8	4	clustalw (186)
predator	199	104	1045	10.0	4	loadseq (89)
blast2	158	91	6252	1.5	4	readseq (85)
grailclnt	105	52	806	6.5	1	grailclnt (105)
con_filter	104	61	62	98.4	1	consensus (104)
clustalw_convert	104	81	472	17.2	5	clustalw (90)
total others	1344	1245	37 604	3.3		

registration and is simply suggested through pipeline menus. In the same way, our macro registration feature—which has no equivalent in these systems—does not require any action from the user, it is just available when needed. As stated in (Sugiura and Koseki, 1998), users generally tend never to use such features as program combining or macro definition if their cost is greater than that of simply doing the task. Finally, on the practical side, and to our knowledge, these systems are not freely available, and thus not extensible at will to any biology application.

Customization, Web macro registration

There has been a lot of research on customization but our approach is more application-oriented. We already have an interface builder to generate the HTML and CGI parts of the interface: using it to have the users dynamically generate their customized forms was just a step further.

Previous work on macros (Sugiura and Koseki, 1998; Miller and Myers, 1997) focuses more on document authoring or sometimes on navigation adaptation than on action registration. Our macro feature is similar to some database search systems which enable the user to build views and to store complex queries. Like such systems, we do not provide a general macro registration mechanism for the Web, which could be based on generic Web programming languages like WebL (Kistler and Marais, 1998). In our system, a registered procedure thus simply consists in successive calls of CGIs including intermediate HTML result parsings, where XML proved to be a very good tool.

Table 5. Pipes producers

Initial program (from)	Total uses of the connection menu from this program	To how different many clients	Biggest client
clustalw	7531	33	boxshade (1579)
protdist	4321	9	neighbor (2282)
dnadist	3722	9	neighbor (2405)
neighbor	3541	4	drawtree (1391)
protpars	1475	4	drawtree (621)
loadseq	1273	12	loadseq (734)
consense	1242	4	drawtree (466)
dnapars	1216	4	consense (664)
Puzzle	778	13	drawtree (256)
fastdnaml	748	4	drawtree (253)
fitch	740	4	drawtree (330)
bionj	670	4	drawtree (267)
blast2	544	5	seqsblast (356)
pyramids	439	2	drawpyr (377)
seqgen	371	24	distquart (98)
consensus	345	2	patser (241)
fasta	335	1	mview_blast (335)
blast2	293	5	seqsblast (229)
mafold	288	1	rnaeval (288)
kitsch	255	4	drawtree (100)
molphy	218	4	consense (84)
buildmodel	198	5	hmmscore (89)
readseq	152	6	blast2 (85)
clustalw16	133	21	drawtree (17)
clustalw_convert	117	18	clustalw (32)
grailclnt	105	1	grailclnt (105)
total others	1100	70	

CONCLUSION AND FUTURE DIRECTIONS

In this paper, we have presented Pise, a system that, given a description of a program, its parameters and input or output files, generates a Web interface. This Web-based interface stands as a Unix command wrapper, whose role is to build the command line for the given program, formatted with the appropriate parameters, and to manage the results. The main originality of our system is the ability to connect related programs in order to perform more complex scripts. We have shown that, according to the statistics of use, this system actually helps biologists to perform their tasks.

We still have to evaluate and adapt advanced features such as macro registration: for now, it consists of a procedure that will redo the same processing as that already performed, with another initial input. There could be more flexibility, such as leaving more parameters open or putting some well defined branching conditions between each step.

The software described here is available under GPL at the following address: http://www.pasteur.fr/~letondal/Pise. Pise has already been successfully installed at other sites. A collaboration with the EMBOSS project has proven that Pise could interface EMBOSS programs, which are now run from the Pasteur Web server on a daily basis.

Table 6. Summary

Total number of pipes	32 150
Total number of command that give input to another	76
Total number of command that take input from another	93
Total number of users	3535
Mean of pipes by user	9.09 (MaX: 494)

ACKNOWLEDGEMENTS

We are grateful to the persons who helped to make this software easier to install and to use by giving suggestions, and particularly to Stephane Bortzmeyer, Nicolas Joly and Alan Bleasby.

APPENDIX

Statistics on program combinations

Table 4 lists the programs that were selected from the pipe menus, with information about source programs and actual submissions. The number of actually submitted jobs coming from a pipe menu may be greater than the pipe menu selection, since a job may be resubmitted several times from the same form (from October 14, 1998 to May 15, 2000).

Table 5 shows which programs give access to other ones by a pipe menu. This table describes the ***use*** of the pipe menus only, not the actual run of the piped job.

REFERENCES

Devereux,J., Haeberli,P. and Smithies,O. (1984) A comprehensive set of sequence analysis programs for the VAX. *Nucleic Acids Res.*, **12**, 387–395.

van Doorn,M. and Eliëns,A. (1995) Integrating applications and the World Wide Web. *Proceedings of WWW3* Darmstadt, Germany.

Kistler,T. and Marais,H. (1998) WebL—a programming language for the Web. *Proceedings of WWW7* Brisbane, Australia.

Lonczewski,F. (1995) using a WWW browser as an alternative user interface for interactive applications. *Proceedings of WWW3* Darmstadt, Germany.

Miller,R.C. and Myers,B.A. (1997) Creating dynamic World Wide Web pages by demonstration. *Carnegie Mellon University School of Computer Science Technical Report CMU-CS-97-131 and Human Computer Interaction Institute Technical Report CMU-HCII-97-101*.

Phanouriou,C. and Abrams,M. (1997) Transforming command-line driven systems to Web applications. *Proceedings of WWW6* Santa Clara, California, USA.

Senger,M. (1999) AppLab: CORBA-Java based application wrapper. *CCP11 Newsletter, issue 8*.

Senger,M., Flores,T., Glatting,K.-H., Hotz-Wagenblatt,A. and Suhai,H. (1998) W2H: WWW interface to the GCG sequence analysis package. *Bioinformatics*, **14**, 452–457.

Subramaniam,S. (1998) The biology WorkBench: a seamless database and analysis environment for the biologist. *Bioinformatics, Proteins*, **32**, 1–2.

Sugiura,A. and Koseki,Y. (1998) Internet scrapbook: automating Web browsing tasks by programming-by-demonstration. *Proceedings of WWW7* Brisbane, Australia.