BioPCD - A Language for GUI Development Requiring a Minimal Skill Set

Graham GM Alvare
Dept of Plant Science
University of Manitoba
Winnipeg, MB. Canada. R3T 2N2

Abiel Roche-Lima
Dept of Computer Science
University of Manitoba Winnipeg
MB. Canada. R3T 2N2

Brian Fristensky
Dept of Plant Science
University of Manitoba
Winnipeg, MB. Canada. R3T 2N2

ABSTRACT

BioPCD is a new language whose purpose is to simplify the creation of Graphical User Interfaces (GUIs) by biologists with minimal programming skills. The first step in developing BioPCD was to create a minimal superset of the language referred to as PCD (Pythonesque Command Description). PCD defines the core of terminals and high-level non-terminals required to describe data of almost any type. BioPCD adds to PCD the constructs necessary to describe GUI components and the syntax for executing system commands. BioPCD is implemented using JavaCC to convert the grammar into code. BioPCD is designed to be terse and readable and simple enough to be learned by copying and modifying existing BioPCD files. We demonstrate that BioPCD can easily be used to generate GUIs for existing command line programs. Although BioPCD was designed to make it easier to run bioinformatics programs, it could be used in any domain in which many useful command line programs exist that do not have GUI interfaces.

General Terms

Bioinformatics

Keywords

Graphical User Interfaces; Languages; Formal grammar; Human-Computer Interaction; Computational Biology; Bioinformatics

1. INTRODUCTION

At a time when a popular slogan is “there’s an app for that”, we often take for granted the fact that most human interaction with computers is through Graphical User Interfaces (GUI). It is easy to forget that extensive knowledge of programming and software engineering is needed to create applications with GUIs.

Integrated Development Environments (IDEs) such as NetBeans http://www.netbeans.org and Eclipse http://www.eclipse.org contain GUI builders, in which GUI components are manipulated within a GUI, and the corresponding code is generated by the IDE. While GUI builders make it easier to write GUIs, they still require a substantial knowledge of a programming language and of software engineering practices.

Because of the skill set required to create a GUI, software in fields such as biology are often written as command line programs. At the same time, it is often impossible to find software engineers with enough appreciation of the biological domain who could re-write the program with a GUI. Bioinformaticians often need to assemble data pipelines using many programs written independently by many authors in many languages. Because of rapid advancement in the field, there are always new programs appearing in the literature that replace the current existing “favorite” of researchers working in that field. The work required to modify an existing GUI to handle the new program is also a limiting factor. An additional problem is that many programs are not written with extensibility in mind.

The need for easy ways to create GUIs is particularly important in bioinformatics. Bioinformatics is an interdisciplinary field, in which computer methodologies are applied to the analysis of biological data. The most critical limiting factor in bioinformatics is not computational, but rather human. The complexity of the data, along with the enormous datasets generated, often push the limits of computer resources and algorithmic rigor. However, few biologists have any formal training in computers. Thus, the user group with the greatest need for simple GUI interfaces to complex data pipelines with numerous parameters is not in the position to create them. At the same time, few computer scientists have the necessary background to appreciate the biological subtleties that must be reflected in how the program is presented to the end user.

There have been a few attempts at addressing this problem in the past by creating what could be considered to be a programmable GUI. The EMBOSS package of programs for molecular biology uses a syntax known as ACD to automatically generate menus for running programs through a web browser [1]. JEMBOSS, a Java desktop application, can also read ACD to create menus for running EMBOSS programs [2]. Kaptain, http://kaptain.sourceforge.net is a system for generating graphic interfaces for command line programs using grammar scripts. Similarly, web-based interfaces to over 200 applications have been generated using Pise, which creates HTML interfaces from XML definitions of program parameters [3]. Finally, the Taverna workbench is a Java application in which complex data workflows can be created by linking together icons representing web services available at both local and remote sites [4].

The most flexible programmable GUI in bioinformatics is GDE [5,6], which has a simple syntax for writing menus that does not require formal training in programming. It worked well for what it did, but had limited functionality. While GDE is an open source application which could, in principle, be revised, the parsing logic was written in C in the early 1990s, within an X11 program that depended upon what are now obsolete libraries. No formal grammar exists to describe the menus.

In the spirit of GDE, we have created a formal grammar to describe the means by which GUI menus are created from panels, labels and widgets, which correspond to parameters that are passed to shell commands. This language is referred

to as BioPCD. Because it is described in a formal grammar and compiled into Java by JavaCC, BioPCD is platform independent. We will first introduce a simple example, showing how BioPCD can be used to specify a GUI that runs a program, and then go on to describe BioPCD in a more formal way. We have used BioPCD to create BioLegato, a programmable GUI for molecular biology. BioLegato is written in Java and distributed as part of the BIRCH system for bioinformatics [7]. The details of BioLegato will be described in a separate publication.

2. A SIMPLE EXAMPLE

Suppose we need a GUI to run a program called blrevcomp. blrevcomp reads a single stranded DNA sequence from an input file, and creates the inverse complement using the base-pairing rules of Watson and Crick, which is then written to an output file. In double-stranded DNA molecules, A pairs with T, and G pairs with C , which represent, respectively, the nucleotide bases Adenine, Thymine, Guanine and Cytosine. Thus, if the input strand is 5'ATTCGGGC3', then the complementary strand is written as 3'TAAGCCCG5'. (5' and 3' refer to the locations of the 5' and 3' phosphates, which are used to determine the orientation of nucleotides in a DNA strand). We further add the condition that blrevcomp has a number of command line options that may be included in the command string. Thus, the command we would like the program to run might be

```
blrevcomp -c infile outfile
```

where the -c option tells blrevcomp to create a strand complementary to the input strand.

For the purposes of this paper, we describe BioPCD within the context of a programmable GUI that we have developed called BioLegato. Briefly, BioLegato is Java application in which BioPCD menus are chosen from a pulldown list. Figure 1A shows an example of a GUI that could be used for running blrevcomp. The corresponding BioPCD code is given in Figure 1B. In essence, the task of the menu is to build a command to be executed by the shell, by substituting strings from the var blocks into the "shell" command. The Strand buttons in the GUI are specified in a "var" block, which sends a choice of any of three command line options to the command string. Another "var" block labeled "gdeoutput" gives the user a choice of two output options. The "gdeoutput" block illustrates the fact that even complex statements can be substituted into the shell command line. In this case, the default value of "No" would substitute into the command string code to rename the output file to "out1". In this case, "out1" is defined as a tempfile whose direction is "out", meaning that it is output that can be read directly into the current BioLegato window. In the example, the user has clicked on "Yes", which launches a new BioLegato window. Since the output is DNA, the 'bldna' script is called, which launches BioLegato using DNA-appropriate BioPCD menus.

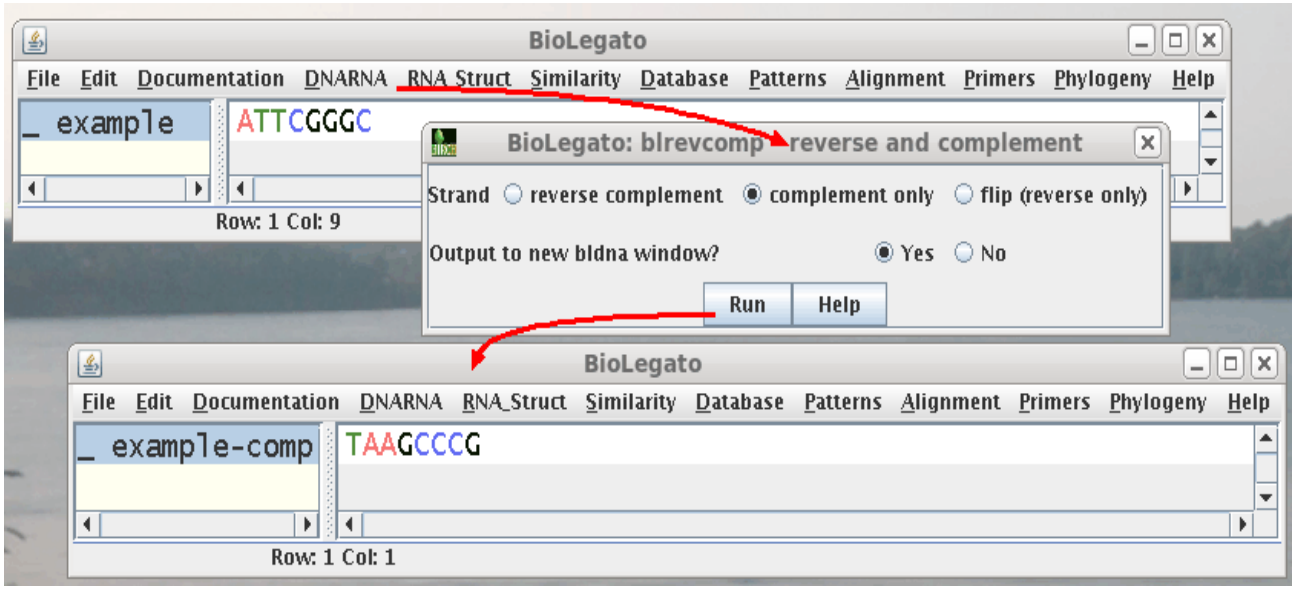

Figure 1. A. Example of PCD menus implemented in the BioLegato application. Top: The BioLegato window at top can run tasks specific for DNA sequences. The tasks are organized into categories (eg. Similarity, Database, Patterns) using pull-down menus. In the example, the blrevcomp program has been chosen from the DNARNA pull down menu, causing the blrevcomp menu to be displayed (center). The contents of menus such as blrevcomp are read at runtime from BioPCD files, an example of which is shown in B. In this example, the Run button tells BioLegato to run blrevcomp using the selected DNA sequence as input, and send output (the complementary DNA sequence) to a new BioLegato window (bottom). In effect, the user is performing data pipelining using a point and click interface, a process we refer to as *ad hoc* data pipelining.

```
name   "blrevcomp - reverse and complement"
var "in1"
    type      tempfile
    direction in
    format    flat
var "strand"
    type    chooser
    label   "Strand"
    default 0
    choices
        "reverse complement" "-r"
        "complement only" "-c"
        "flip (reverse only)" "-f"
var "out1"
    type      tempfile
    direction out
    format    flat
var "gdeoutput"
    type    chooser
    label   "Output to new bldna window?"
    default 1
    choices
        "Yes"   "(echo '' > %out1%; bldna %in1%.blrevcomp; $RM_CMD -f %in1%.blrevcomp)&"
        "No"    "mv %in1%.blrevcomp %out1%"
panel
    var "Run"
        type    button
        label   "Run"
        shell   "blrevcomp %STRAND% %in1% %in1%.blrevcomp; $RM_CMD %in1%; %GDEOUTPUT%"
        close   true
    var "Help"
        type    button
        label   "Help"
        shell   "$BIRCH/script/gde_help_viewer.csh $BIRCH/doc/bioLegato/blrevcomp.html"
        close   false
```

Figure 1. B. PCD code used to implement the blrevcomp menu. Note that while the default value for Strand is "reverse complement" "-r", the window above shows that the user has checked "complement only", which would cause the "-c" option to be substituted into the command line.

This aspect of BioPCD illustrates an important capability inherent in BioPCD, which we refer to as *ad hoc* pipelining. Simply put, ad hoc pipelining is the use of output from one GUI program as input to another GUI program. The pipe '|' is one of the most powerful aspects of the Unix shell, because in principal, any number of commands on the system could be piped to any other command. In typical end user applications, exporting and importing between applications is awkward. BioPCD makes it easy to send data to any other program. In Figure 1, the DNA output generated from one instance of BioLegato is used as input to launch another instance of BioLegato, meaning that all of BioLegato's DNA functions are now available for use on the output. If the output had been text, it could have been sent to a text editor. If the output was HTML, it could be viewed in a web browser.

3. PCD FORMAL SYNTAX

The first step in creating BioPCD was to create a superset of the language referred to as PCD (Pythonesque Command Description). Much as XML is a superset of a large family of markup languages including HTML, PCD is a superset of

BioPCD. PCD defines the core of terminals and high-level non-terminals required to describe a data of almost any type. BioPCD is adds to PCD the constructs necessary to describe GUI components and the syntax for executing system commands. A "skeleton" PCD parser program is found at the URL listed in Appendix 1.

3.1. Terminals (as regular expressions)

All of the below types in PCD are written as regular expressions. The format of these expressions are the same as they would be specified in Perl (see http://perldoc.perl.org/perlre.html).

```
<bool>        ::= true
              ::= false
<text>        ::= "([^"]|"")*"
<id>          ::= [^"]+
<number>      ::= [0-9]+\.?[0-9]*
<comment>     ::= #[^\n]*
```

3.2. Non-terminal productions

PCD code is defined in blocks as follows:

```
<block>       ::= <field> <indent+1>
                  <block> <indent-1>
              ::= <field> <value>
              ::= <block> <indent> <block>
<field>       ::= <data>
<value>       ::= <data>
<data>        ::= <bool>
              ::= <text>
              ::= <id>
              ::= <number>
```

The above tags work similarly to Python, in that indentation defines the scope of parameters. Where this differs from Python is that the indentation is fixed at four spaces per indent level. This prevents the mixing of tabs and spaces for indentation, which is a common problem in Python.

<indent> means to add a new line character and maintain the same indentation level, while <indent+1> indicates that the indentation of the line should be increased by one indentation unit, and <indent-1> indicates that the indentation of the line should be decreased by one indentation unit. Terminal symbols are separated by whitespace, and the only whitespace that is specified in the grammar below are newlines and their indentation effect. Comments may be interspersed at the end of any line.

4. APPLICATION OF BIOPCD – THE BIOLEGATO MENU LANGUAGE

4.1 Working assumptions

BioPCD is designed around two working assumptions to allow for flexibility. The source of the data is unspecified, but is assumed to be data selected elsewhere within an application. This makes BioPCD fit within a range of possible applications. It would even allow BioPCD to be implemented in the context of a web browser. The menu could be implemented as an application with a single standalone window, is assumed to be a file exported by the parent application, given a temporary name (eg. "in1" in Figure 1B.) or as a panel within a larger application, such as BioLegato.

4.2 The Elements of BioPCD

BioLegato is implemented in JavaCC, which allows semantics checking in its parser. In BioPCD every block is distinct in what it can contain. The only identifiers currently supported are predefined keywords, and the only fields that are not keywords are those in the choices subsection, most of the fields must be specified in a specific order (with the exception of `panel`, `var`, and `table`). In the root scope (level zero), the following declarations are allowed (all are optional): `name`, `icon`, `tip`, `exec`, `system`. These tags correspond to the name of the menu item, the icon for the menu item, the tooltip text for the menu item, and the execution command for the menu item, respectively. The exec tag is used only for BioPCD commands which do not require the user to enter any parameters. These commands are launched immediately when the user clicks on the menu item (the user is not prompted for any input). Additionally, `var`, `panel`, `tabset` may be specified in the root scope multiple times in any order after the other tags.

The `tabset` tag is used to indicate that a tabbed pane be created, and a tab created for each child `tab` tag. Each tab tag, may in turn contain multiple `var` tags (widgets). See Figure 2A and 2B for an example.

The `panel` tag is used to create a panel for housing widgets. Panels are used to place more than one widget on the same line.

The `system` tag is accompanied by a list of computing platforms on which the menu should appear. This tag is useful if a program to be called from the shell command is only available on some platforms but not others. By default the menu item should appear on all platforms. If a system tag is provided, then the menu will only appear on the specified platform(s). An example of its usage is:

```
system
    solaris
    linux     x86_64,x86
    osx       x86
```

The set of tags corresponding to valid platform choices are implementation dependent. In the current version of BioLegato, the following operating system tags are allowed: `solaris`, `linux`, `osx`, `windows`; and the following architecture tags can be paired with a system tag: `sparc`, `x86_64`, `amd64`, `intel`, `x86`.

The `var` tag is the most versatile of all of the tags presented thus far. The var tag is used to specify program parameters (aka. Variables, or widgets) for the most part. The only exception to this is program buttons, which use the `var` tag, but are not parameters. Rather, program buttons are used to run commands using the program parameters specified by other `var` tags. The `var` field is specified as follows:

```
var "source"
    type          chooser
```

```
label       "Source"
default     1
choices
    "Commercial" "C"
    "All"        "A"
```

Note that each `var` tag has a name associated with it. The name for a `var` tag is specified on the same line as the `var` tag. In the case above, the name of the var tag is *source*. This means that whenever `%source%` is encountered in an `exec` field or a `shell` field, it is replaced with whatever choice is selected in the variable. In the above example, the variable is a drop-down box with two options ("`Commercial`" and "`All`"). Therefore, if the user were to select "`Commercial`" and click a button to perform a command, each instance of `%source%` in the button's command would be replaced with "C" (the value associated with "Commercial").

Each `var` tag must have a defined set of parameters in BioLegato, depending on what type of variable the `var` tag represents. For instance, using the chooser example above, each chooser variable must have the fields `type` and `choices`; type specifies what type of variable "`Source`" is, and `choices` specifies which `choices` should be available to the user. The fields `default` and `label` are optional fields, which specify the default value to select in the `chooser` (the first value the user will see selected before he or she changes it), and the text to display to the left of the `chooser` (this is so the user can understand what parameter the `chooser` is selecting for the program defined by the BioPCD menu).

The following list contains all of the `var` tag types currently supported in BioLegato, with examples of possible values. The parameters each `var` tag supports are listed below each type. Each parameter below must be specified in the exact order that they are listed within each var type description (e.g. for buttons - type then label, then shell, then close). The tags are in bold, explanations are italicized, and optional parameters are enclosed in brackets []:

Buttons:

type button
[label "the text for the button"]
shell "the command to execute"
[close false]

whether to close the parameter window when the button is clicked

Chooser, Comboboxes, or Lists:

type chooser *or list or combobox*
[label "what to call the choser"]
[default 0] *the default selection*
choices
"abc" "1"

each choice should be specified as "name" "value" – i.e., the name specifies what the user will see, whereas the value specifies the string substituted into the shell command.

Text (textboxes):

type text
[label "what to call the text box"]
[default "the default text"]

Number:

type number
[label "what to call the text box"]
min 0
max 10000

the minimum and maximum numbers selectable by the number widget

[default 0]

External Files/Directories specified by the user:

type file *or dir for directories*
[default "the default filename"]

Temporary files containing data exported from BioLegato's main canvas:

type tempfile
direction out *(or in)*

specifies whether the file will be read as input for the program (in), or for BioLegato (out).

format fasta

the file format used – e.g. csv, tsv, fasta, flat, gde, genbank

[save true]

whether the file should be "saved" after program execution

[overwrite true]

whether the selection in the canvas should be overwritten by the file content – out only.

[content canvas] *or selection.*

whether the data written to the file comes from the canvas's entire contents (canvas) or current selection (selection) – in only.

BioPCD is case-insensitive for two reasons. First, we felt that the flexibility gained by case-sensitivity was outweighed by the frequency with which it results in errors. For example, even experienced Python programmers will often have difficulty recalling that 'True' is a boolean value, whereas 'true' is not. Also, case-sensitivity adds the potential for

generating errors when identical words, which only differ only in their case, are defined differently multiple times.

4.3 JavaCC grammar for BioPCD

A JavaCC grammar contains both the semantic and the syntactic definition for a language. JavaCC compiles the grammar into Java code, which parses the language. In order to change the grammar, one only needs to modify the JavaCC code. The current grammar for BioPCD is available at the URL listed in Appendix 2.

4.4. Examples

The BLASTP [8] program has a large number of parameters, which we have visually organized into categories by splitting the BLASTP menu into several panes using the "`tab`" tag (Figure 2). Only two of the five panes are shown. In the "General search options" pane (Figure 2A), holding the most commonly-used parameters, the first four parameters are implemented as comboboxes, while the "Word size" parameter is implemented as a `chooser`. The "Output" pane (Figure 2B) holds parameters relevant to the output format and the destination of output. This pane illustrates the choices in a combobox, as well as "Output file name", implemented as a text box. A BioPCD code fragment for the BLASTP menu is shown in Figure 2C. Only a fragment of the full BioPCD file is shown, to illustrate how "`tab`" corresponds to the final menu.

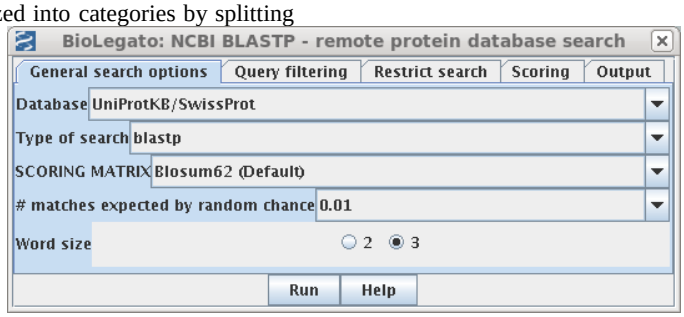

Figure 2A. An example of menus created using BioPCD.

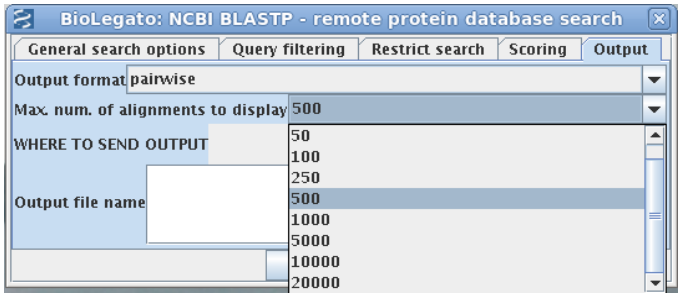

Figure 2B. Another tab selected in the same menu

```
tabset
    tab "General search options"
        var "dbase"
            type    combobox
            label   "Database"
            default 0
            choices
                "UniProtKB/SwissProt"                "swissprot"
                "GenBank NonRedundant (Protein)"     "nr"
                "Reference proteins"                 "refseq_protein"
                "Protein Structure Data Bank (PDB)" "pdb"
                "GenBank Patented"                   "pat"
```

Figure 2C: Excerpt from BioPCD code for the above menu

The BioLegato menu that runs PROML from the Phylip [9] package is shown in Figure 3. The "User tree filename" parameter illustrates use of the "`file`" tag, which adds a file chooser widget to the menu. Of particular interest is the "`number`" tag, that adds a slider to the menu. Sliders solve the problem of allowing the user to set a precise numerical value over a wide numerical range in three different ways. First the slider can be dragged left and right, as a sort of coarse adjustment. Where precision is required, the up and down arrows can be used to increment of decrement the

number, rapidly by holding down the arrow, or slowly by clicking. Finally, a text box accompanies the slider, allowing the user to type or paste in the number if they prefer.

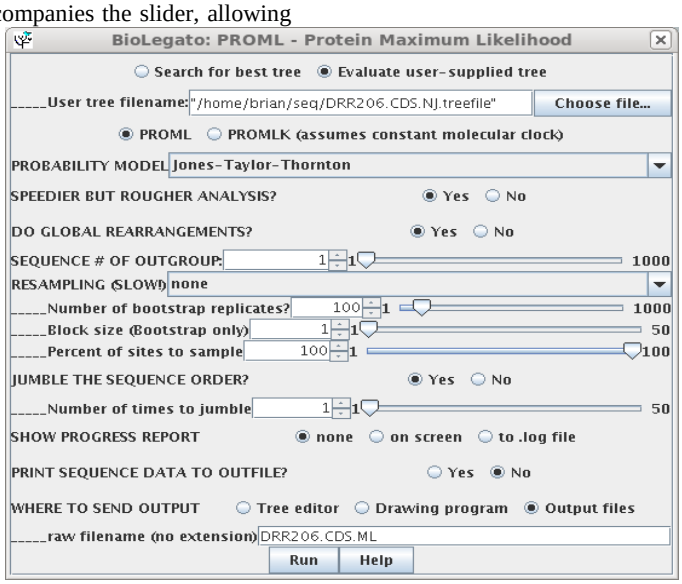

Figure 3. Menu for PROML.

5. DISCUSSION

While PCD has been created primarily to make it easier for non-programmers to create GUIs, it could be useful even to programmers in general. By analogy, we all benefit from having accessibility features in buildings, such as wider hallways, automatic doors, ramps and lifts. By allowing a very rapid develop/test/revise cycle, programmers can bring frequent updates, converging upon an optimal interface suited to the client's needs.

We considered using existing languages such as JSON (http://json.org) or XML (http://www.xml.org), but found that they did not meet our needs. XML is inherently not human readable, especially for non-programmers. In XML, there are no specific rules enforcing visual formatting, and end tags are not always easy to find. When an end tag in XML is incorrectly indented, it can result in confusion about the scope of code blocks. Like Python, PCD scope is determined by indentation, resulting in cleaner code with no need for end tags.

Like XML, JSON does not contain any rules to enforce good formatting practices. Additionally, like XML, JSON also relies on the explicit closing of scope, with the problems already mentioned that accompany end tags. While it could be argued that IDEs such as Eclipse or Netbeans highlight corresponding start and end tags, IDEs also have a high learning curve, which would again undermine the goal of PCD to minimize the learning curve.

BioPCD was specifically designed to facilitate quick addition of menus to a BioLegato instance. In fact, we have found that the hardest part of creating new menus is becoming familiar enough with the program you wish to add to BioLegato. Generally this entails reading the documentation to understand what the program does, what the command line options do, and what the input and output file formats are. At that point, it is easy to make a copy of an existing PCD menu and modify it to run the new program. That is the point of BioPCD. Once those things have been done once, a GUI then exists that saves every end user from having to repeat the learning curve for each new program.

This underscores one way in which menus written in BioPCD make work easier for the end user, compared to previous approaches to programmable GUIs. Pise [3], ACD [1,2] and Kaptain all require the user to prepare input files in the format required for each program to be run, or to be pasted into a text box. BioPCD is designed to be used within the context of an end-user application such as BioLegato. BioLegato, like its predecessor GDE [5,6], hides the implementation of exporting input files and importing output files, giving the end user more of sense of working directly with the data, rather than working with files. As importantly, BioPCD makes it straghtforward to send output to a new BioLegato instance, termed *ad hoc* pipelining, giving the end user numerous choices for what to do next, at each step in the process.

Backward compatibility with other forms of menu specification is easily attained by translators. For BioLegato, we wrote a program called gde2pcd to translate our existing GDE menu files into BioPCD menus for BioLegato. Similar translators could easily be written to create PCD code from formats such as ACD for EMBOSS, or the grammar used in Kaptain.

Both PCD and BioPCD were designed to lend themselves to further development in a number of directions. The ability for PCD to evolve is inherent in the fact that it is an abstract language, where terminals are primitive types eg. Boolean. PCD leaves a lot of room for programmers to modify and create their own dialects of PCD. The formal grammar for PCD can be compiled by any compiler compiler. Thus, the fundamentals of PCD are platform-independent and are not wedded to BioLegato. Rather, different dialects for different applications are possible. Like XML and JSON, PCD could be used as a data interchange language.

The open design of PCD allows a programmer to embed a PCD parser in his or her code (e.g. by using the skeleton PCD parser, Appendix 2). For example, PCD could be used to store DNA sequences and sequence metadata. Additionally, PCD could be used to store non-biological information, such as customer information for a business, database output (as a tree), a specification for a non-bioinformatics GUI, saved data for a game, molecular information for chemistry, or storage of mass spectrometry reads.

BioPCD is intended to be a general purpose and platform-independent language for specification of GUIs. Although it was implemented specifically for bioinformatics, the BioLegato implementation could easily be used without change in any field in which there exists a large body of command line programs that would benefit from being unified within a single GUI.

Currently, we are working to further automate the process of adding 3rd party programs to BioLegato. The first step will be a graphical BioPCD editor in which the user creates new menus by dagging and dropping widgets. Because BioPCD is a formal language, it will be straightforward to directly generate syntactically-validated BioPCD from the editor. The editor would put together definitions of input and output files, parameters, and documentation files associated with the program, and automatically add these to the local copy of BioLegato on the user's computer. The editor would make it even easier for the biologist to seamlessly integrate 3rd party programs of their choice into BioLegato, in a way that is tailored to their specific needs. A logical extension of this process would be to provide a mechanism for submitting these locally-created Add-Ons to a community of BioLegato users.

6. AVAILABILITY

The BioLegato implementation of BioPCD is freely available under the Creative Commons License 2.0. It can be obtained as part of the BIRCH system for bioinformatics at http://home.cc.umanitoba.ca/~psgendb. A wiki for PCD containing all information for developers is located at: http://www.bioinformatics.org/wiki/BioLegato/PCD.

7. ACKNOWLEDGMENTS

This work was funded in part by the following Genome Canada Programs: Competition III, Science and Technology Innovation Centres, and Applied Genomics Research in Bioproducts or Crops. Cofunding was also provided by Manitoba Innovation, Energy and Mines. We would also like to thank Natalie Bjorklund for editorial help with the manuscript and for useful comments.

7. REFERENCES

[1] Rice,P. Longden,I. and Bleasby,A. “EMBOSS The European molecular biology open software suite”. Trends in Genetics. 2000. Vol 16. Issue 6. pp 276-277.

[2] “The design of Jemboss: a graphical user interface to EMBOSS. Bioinformatics”. 2003. Vol 19. Issue 14. pp 1837-1843.

[3] C. Letondal (2001), A Web interface generator for molecular biology programs in Unix, Bioinformatics, Oxford University Press, 17(1), 2001, pp 73-82.

[4] Hull, Duncan; Wolstencroft, Katy; Stevens, Robert; Goble, Carole A.; Pocock, Matthew R.; Li, Peter; Oinn, Tom (2006). "Taverna: A tool for building and running workflows of services". *Nucleic Acids Research* **34** (Web Server issue): W729–W732. DOI:10.1093/nar/gkl320. PMC 1538887. PMID 1684510.8

[5] Smith SW, Overbeek R, Woese CR, Gilbert W, Gillevet PM (1994)The genetic data environment: an expandable GUI for multiple sequence analysis. *Computer Appl. in the Biosciences* 10 671-675

[6] Linton E (2006) MacGDE: Genetic Data Environment for MacOSX. http://www.msu.edu/~lintone/macgde/

[7] Fristensky B (2007) BIRCH: A user-oriented, locally-customizable, bioinformatics system. BMC Bioinformatics , 8:54

[8] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", Nucleic Acids Res. 25:3389-3402.

[9] Felsenstein, J. (1989) PHYLIP Phylogeny Inference Package. Cladistics 5:164-166.

APPENDICES

Appendix 1 - BioLegato 0.7.9 BioPCD JavaCC grammar
http://home.cc.umanitoba.ca/~psgendb/local/ijca_pcd_paper/pcd.jj

Appendix 2 - Skeleton PCD parser
http://home.cc.umanitoba.ca/~psgendb/local/ijca_pcd_paper/PCD.java