Chapter 16

BLUEJAY: A BROWSER FOR LINEAR UNITS IN JAVA

Paul Gordon and Christoph W. Sensen
Institute for Marine Biosciences
National Research Council of Canada
1411 Oxford St.
Halifax, NS B3H 3Z1
Canada
Paul.Gordon@nrc.ca, sensoncw@niji.imb.nrc.ca

Abstract Abstract: The World Wide Web illustrates the power of using a common, simple markup language (HTML) to describe distributed documents. To expand this power to other types of data, a system for the markup and visualization of linear data is introduced. The Browser for Linear Units in Java (Bluejay) is composed of three main components: (i) a set of Java foundation classes for data parsing, visualization, and user event handling, (ii) an HTTP proxy server for data conversion, and (iii) Java classes implementing the user interface for a particular application of the browser. Bluejay users can efficiently browse specialized data on the Web, concentrating on information of particular interest to them. This is achieved through the use of the W3C's Extensible Markup Language (XML) as the data format, a data conversion server, and a document components tree which can be configured to restrict the type of data shown. An application of Bluejay for browsing genetic sequence data encoded in BIOML (an XML) is described to demonstrate the principles of the system. Creating an integrated view of heterogeneous, distributed genetics and genomics databases, the application illustrates the usefulness of borrowing the HTML browser paradigm to create specialized data visualization using XML. Despite the use of proxy servers and drawing abstraction, data retrieval and display performance is kept high through the use of special caching techniques for the proxy, and simple module replacement in the application.

Keywords: XML, BIOML, data visualization, genetics, genomics, bioinformatics, Java, WWW.

16.1 INTRODUCTION

The explosive growth of the World Wide Web illustrates the power of using the Hyper Text Markup Language (HTML), a common, simple markup language [1], to describe shared documents. Unfortunately, HTML is not well suited for describing non-textual data such as molecular genetic sequences. The World Wide Web Consortium's Extensible Markup Language [2] (XML) is an attempt to standardize the markup mechanism for broader ranges of data types. The Browser for Linear Units in Java, Bluejay, is a system for markup and visualization of linear data encoded in XML. For our purpose, linear data are defined as data that can be described by a magnitude distance from a start point, such as the number of letters away from the start of a text or the number of nucleotides away from the 5'-end of a DNA molecule.

The impetus for developing Bluejay was the need to create a view of heterogeneous molecular genetic and genomic sequence databases and documents, but the problems addressed are common to many application domains of information retrieval and display. The data documents, all of which are accessible via the World Wide Web, can vary greatly in terms of size and type of content, and they are formatted in different ways depending on the data source. The user is not necessarily interested in viewing all of the data contained in a retrieved document. Additionally, the data display requirements may vary depending on the how the information is being used. Sometimes the user will only be interested in a small region of the genome, while at other times the entire genome needs to be displayed at once. While solving these problems, we kept in mind that the data should be viewable from computers of several different architectures with often limited processing power and memory.

16.2 THE BLUEJAY ARCHITECTURE

The Bluejay architecture consists of three models that address the problems described above: (i) a model for the creation of documents in a common format, (ii) a model for the configuration of the document view, and (iii) a model for the client-side browser.

The document creation model uses the World Wide Web Consortium's Extensible Markup Language (XML). Although much of the data to be retrieved are currently in HTML format, due to the extensive use of HTML by biologists, HTML is unfortunately not well suited for the logical description of non-textual data such as molecular genetic sequence (i.e. nucleotide sequences of genes and amino acid sequences of proteins). XML is an emerging standard for the markup definition mechanism of broader ranges of data types.

All data sources in Bluejay are required to provide documents in a markup conforming to the XML standard. Due to the impracticality of getting all data sources to serve a common format, the model includes an HTTP proxy [Figure

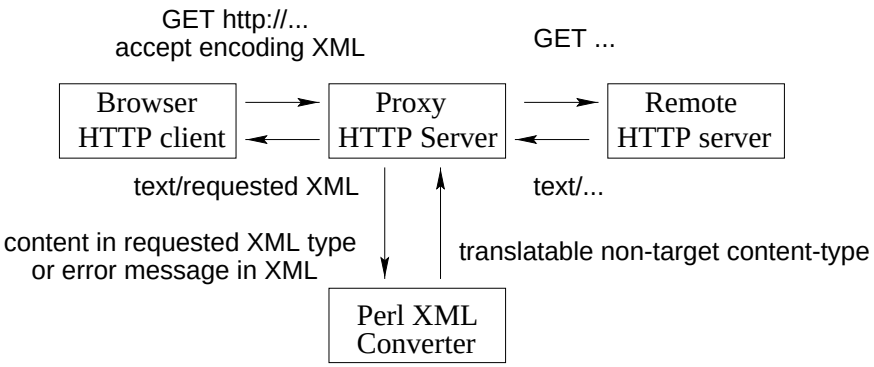

Figure 16.1 Proxy request overview. The Bluejay HTTP client sends the URL request to the proxy and specifies the expected data format, such as BIOML. The proxy then retrieves the document from the remote server. If the document is already encoded in an acceptable format it is returned as is, otherwise it converts the document or returns a message informing the user that the document cannot be viewed. This protocol guarantees the client that any retrieved URL will be readable, regardless of its format at the source.

16.1] which instantaneously converts documents from their native format into the required XML, thus addressing the problem of heterogeneous data sources.

The document view configuration model is strongly influenced by the use of XML as the data format. Since the XML-compliant markup is used to segment the document into its logical components, the hierarchy of those segment instances is the basis for the configuration of the document view. The configuration model is based on the tenet that users can create document views by selecting which type of data should be displayed based on its position in the document segment hierarchy, and by controlling display options for selected data based on the same criterion. View manipulation consists of the expansion and collapsing of nodes in the hierarchy, and the selection of display options that are available at those nodes, thus addressing the need to control viewable document content.

The client-side browser model is also influenced by XML. The browser model defines the interaction between the XML elements, and a drawing toolkit to simplify their drawing. It also provides a framework [Figure 16.2] for standard browsing functionality such as page layout, hyperlinks and user event handling. The model is defined in terms of Java interfaces, since programs written in Java are largely independent of machine architecture; thus addressing issues of variable display requirements and computer platforms.

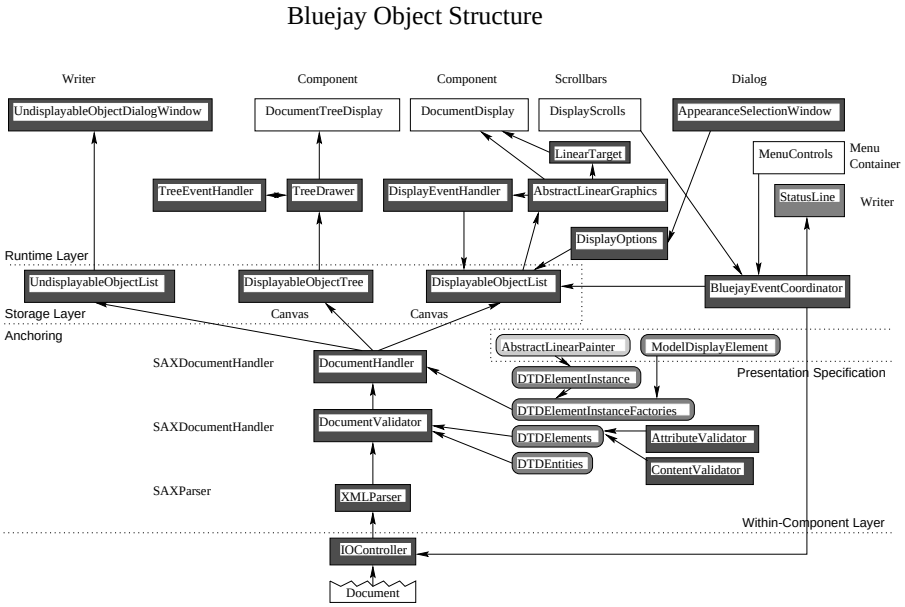

Figure 16.2 The framework for a Bluejay client. Dark gray objects have default implementations, medium gray objects are implemented but require parameters, light gray objects must be specified by the display writer, and white objects are GUI components that must be provided by the application. All objects are accessed through Java interfaces, making object replacement to improve performance easier. Arrows denote the general direction of method calls. The dotted line sectioning roughly divides the system by function according to the Dexter reference model for hypertext systems.

16.3 ARCHITECTURE IMPLEMENTATION & EFFICIENCIES

The Bluejay implementation was built with client-side limitations in mind, both in the proxy and the client browser. Server-side processing and resources are used in Bluejay whenever possible; therefore the system is best suited for multiple clients relying on a high-performance server.

16.3.1 PROXY

The data proxy makes documents available to the application through the provision of a framework for data conversion. Although support for file URLs is part of the Bluejay implementation, the system is mainly designed for hypertext transfer protocol [3] (HTTP) document retrieval. A standard HTTP proxy fetches Web documents on behalf of the client application, but the Bluejay proxy additionally converts the retrieved documents into the requested XML-compliant markup language. To incorporate new types of data, the application writer must supply the code to convert from the actual document format into the desired XML format, but the proxy provides HTTP protocol handling.

The proxy was written in Perl5, using the Apache Web server and mod_perl, and was inspired by Lincoln Stein's AdBlocker [4]. Offloading data formatting from the application to the server reduces the client program size and the required client computing power. The proxy server can become quite taxed in terms of network traffic and computation if many clients are using it to fetch and convert documents at the same time. To reduce this load, the proxy caches converted documents. The caching policy is configurable with regards to the cache size, the types of URLs that can cached (specified using regular expressions) and how long documents should be kept. The latter two factors can be used very effectively if the update schedules for data sources are known.

16.3.2 CLIENT

As shown in Figure 16.2, most of the client architecture classes have default implementations, including those for the view configuration [Figure 16.3]. The classes were designed to be as small as possible, while still proving basic functionality; they can be compressed into a Java archive of about 60 kilobytes. The small code size reduces the client memory requirements, thus enabling the use of the code in Web-browser embedded applets.

Using interfaces for module interaction facilitates the replacement of the default implementations with ones more efficient for the particular application. For example, an application without configurable display settings could replace the initial DisplayOptions and AppearanceSelectionWindow classes with token implementations in order to reduce program size.

The client XML parser conforms to the Simple API for XML (SAX) [5] standard, and can be interchanged with other SAX compliant parsers. The motivation for using SAX is its simplicity: the Bluejay parser is small and does not require the complexity of a parser API such as the Document Object Model (DOM) [6] because the other Bluejay classes are the only modules which are directly interacting with the parser. The parser is validating, i.e. the marked-up data are checked to ensure content and syntax conformity to the markup language definition. The validation of the content offloads work

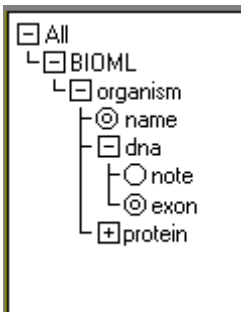

Figure 16.3 Model element tree display. The tree allows the user to specify the type of data to be displayed from the document hierarchy. This allows the user to simplify the display and concentrate on information relevant to their task. Branches with minus signs and leaves with single circles are not displayed, whereas branches with plus signs and leaves with double circles are. Clicking the node with the left mouse button changes its display status, while the right button brings up display options for the node.

from the application writer – there is no need to check for many types of errors in the application code that specifies the data display, because a module located between the SAX Parser and DocumentHandler modules performs the validation. Performance of the parser can be improved by removing the validator, thus avoiding redundant processing if the proxy data converters for the application already ensure data validity.

The key component of the client, which simplifies data display specification, is the drawing toolkit implementing the AbstractLinearGraphics interface. Text layout (i.e. line wrapping and spacing) is handled automatically. Graphical layout is implemented using the LinearTarget objects provided. Linear targets are display components that are inserted into the document's text flow in a manner similar to images in a HTML document. A linear target represents a scale against which other data can be positioned. The drawing primitives (e.g. polygon, line and arc drawing), provided by the abstract linear graphics, use parameters that reference positions on the scale.

For example, a data element representing a feature on a DNA strand might specify that a red-filled rectangle should be drawn from position 20 to position 50 on the linear target representing the DNA, since the feature spans that number of base pairs on the DNA molecule. Such abstraction saves the application developer much effort because the linear target addresses how the rectangle is actually drawn on the display page. In Figure 16.4, the drawing code for the feature element has not changed, but feature display has changed based on page layout and the display-shape of the DNA.

All XML documents contain a single root element containing all the other data elements, therefore the recursive drawing of all elements can be achieved

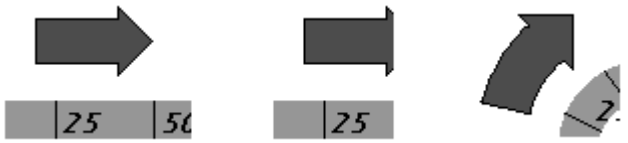

Figure 16.4 Three views of a feature on a linear target. The feature starts at base 20 of the target and ends at base 50. The shape and position of drawn feature changes according to the characteristics of the linear target: full straight, split straight (drawing continued on next line), and curved. The AbstractLinearGraphics class is used to specify how features should be painted relative to the linear target on which they are found. The code specifying how to paint the feature hasn't changed, but its representation has changed with the linear target.

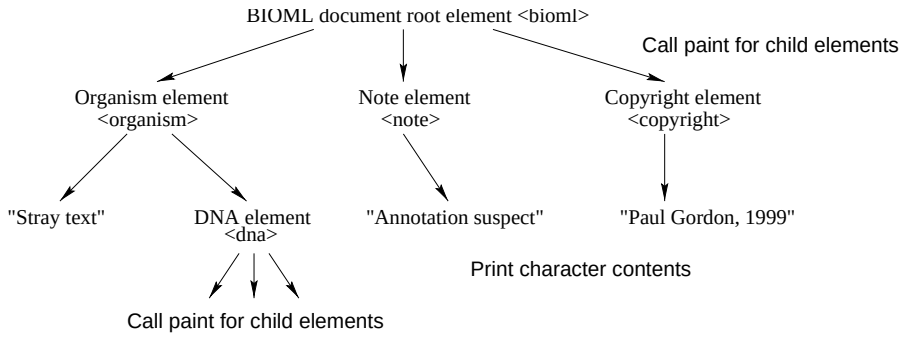

Figure 16.5 An XML document structure with recursive painting. Each element draws itself, and is responsible for calling the paint methods of its children in order. Painting the root element paints the whole document via the in order recursive tree traversal.

by calling the paint method of the root element [Figure 16.5]. The root element paints its character content and calls the paint methods for all its children elements, which paint their children in kind, etc., thereby drawing the entire tree. Each element of the document paints itself, minimizing changes to the element display code and keeping the display specification modular.

16.4 BIOML BROWSER IMPLEMENTATION & OPTIMIZATIONS

Since XML became a World Wide Web Consortium standard just over one year ago, several XML compliant markups for molecular bioinformatics have been implemented. Two markups that have been proposed to markup genetic sequence information are BSML [7] and BIOML [8]. BIOML is still a working draft, but was deemed the most appropriate given the Bluejay conception. Compared to BSML, BIOML has a relatively simple Document Type Definition

Figure 16.6 A tRNA feature displayed at ascending zooms of the DNA target on which it is found. The painter for the tRNA element of BIOML chooses the representation based on the granularity of the DNA display. At high granularity, the feature is denoted only by an empty arrow alongside the DNA line, indicating the feature type. At lower granularity the DNA line is longer and there is enough room to specify the feature's label as well. At the lowest granularity, the actual DNA sequence data is drawn; the relevant tRNA bases hightlighted, and the label integrated into the data.

(DTD), the markup specification. BIOML is also more centered on document content than on document appearance, allowing more degrees of freedom for the data display within the browser.

16.4.1 CLIENT

In terms of the Dexter reference model [9] for hypertext systems, the classes with default implementations are mostly concerned with the within-component, anchoring, storage, and run-time layers. Application writers must supply the presentation specification and a part of the runtime layer implementation, although frameworks and utility classes for both of these are provided. The code specific to the BIOML browser consists of approximately 700 lines of Java code. These set up the user interface (runtime layer) and specify how the elements of BIOML should be displayed in terms of the abstract linear graphics (presentation specification). They also provide the markup specification.

The markup specification is a class that enumerates the valid markup elements, the displayable elements, and element instance factories. It may just contain the language DTD as a string and use the Bluejay document validator to parse it, or, as is the case for the BIOML specification, it may use predefined elements when appropriate to speed up document processing.

Document zooming is the one major function added to the client in the BIOML browser. Although the Bluejay client model does not explicitly support the zooming concept, the scale of the linear targets can be changed during document viewing. The element drawing code can adjust the element display based on the target scale [Figure 16.6]. A feature could potentially not have a representation at a very high or very low zoom level, thus only appearing at scales where its display provides useful information to the user. This drawing-based-on-scale approach provides the ability to view documents of various sizes at many levels.

```
#Translated cache: size in kilobytes, default keep time in seconds
Cache       10000       900

#Data sources with known update times
Scheduled   http://www\.cbr\.nrc\.ca/srs5bin/cgi-bin/wgetz.*   daily 0100
Scheduled   http://www\.cbr\.nrc\.ca/pdb/*                     Sunday 1200

#Locally accessible data file
Local       http://maggie\.cbr\.nrc\.ca/.*      /usr/local/apache/htdocs
Local       http://www.cbr.nrc.ca/localdocs     /cbrdisk/localdocs
```

Figure 16.7 An example Bluejay proxy cache configuration file. The caching policy consists of size and default duration settings. URLs matching the Scheduled lines are potentially kept until the specified time instead of expiring after the default period. Local data files (either on the proxy server or mounted from another machine) are retrieved directly instead of being forwarded. Scheduled and Local configuration data optimize retrieval performance by reducing overhead for update checks and HTTP requests respectively.

16.4.2 PROXY

The BIOML proxy code converts Swissprot-, PIR-, and Genbank-formatted data entries. These are the most common formats for annotated molecular genetic sequence entries, and therefore most of the main data sources are made accessible to the client. Good proxy performance is essential to the system, since client applications rely on the proxy for all HTTP data retrieval. The BIOML proxy was configured [Figure 16.7] for retrieval optimization for two situations: for data sources for which the update schedule is known, and for data files on the proxy server. The scheduling optimization is used for data sources (files or CGI output) stored on the local network. The update schedule for the local databases is known, and the proxy is configured to keep translated documents from those sources up to the known update time if cache space allows, without having to check if documents have been updated. For data files accessible from the machine on which the proxy is located, the serving of requests is additionally accelerated by file retrieval directly from disk instead of loop-back to a non-proxy HTTP request to the local host. This applies to local disk files and files mounted on network file systems (NFS), thus reducing the number of HTTP requests to the host and the processing load.

16.5 DISCUSSION

The goal of the implementation of a BIOML browser was to provide a relatively small application that enables the display genetic sequence data, as well as browse entries of various sizes from large genetic sequence databases linked into applications. The databases accessed require large storage capacity,

and data requests need to be processed on high-performance servers if many clients are connected to the system at the same time. The client browser leverages off the processing power and storage capacity of the server(s) because the proxy performs document translation and data caching.

Other BIOML browsers have been developed, including BioBrow for Windows [10], developed by Proteometrics Inc. BioBrow includes CGI-bin servers for Swissprot and PIR protein database entries. To access data using BioBrow, the user must key-in the script's URL, followed by the database key (the accession number), or the document must be downloaded using an HTML client. The BioBrow distribution is accompanied by a program to create BIOML documents manually.

The Bluejay BIOML system goes beyond the BioBrow concept. The Bluejay proxy implementation allows a user to directly cut and paste URLs of existing documents from their HTML browser, it is focused on the automatic conversion of existing databases. Another advantage of Bluejay is that the client program does not have to be aware of the actual format of the document being requested. This allows a smooth transition between the use of the proxy and direct document retrieval as soon as data sources start supplying documents in a format appropriate for the application.

The Bluejay BIOML browser also differs from the BioBrow in the way in which it presents the document. BioBrow has a document content tree, but its display is limited to the choice of document subtree that is to be displayed. BioBrow is a text-based visualization browser; this paradigm to a certain degree saves the user from being overwhelmed by large documents. Bluejay has more degrees of freedom, as it allows the user to define what type of document elements need to be displayed. The ability to turn the element type display on and off is important in graphical displays, because images can become cluttered with data that are of no interest to the user. The Bluejay browser takes a graphical approach, and thus there is no need to restrict the view in the same manner as BioBrow.

Large genetic sequence structures, including entire chromosomes or complete genomes, can be encoded as BIOML documents, as well as can individual bacterial genes. The ability to display both, high-level graphics representations of large documents, and text for small, highy detailed data is an essential feature of the Bluejay browser. The ability to handle wide ranges of data document sizes, and the ability to zoom in and out from one view, e.g. the entire genome, to another one, e.g. a single gene, is important to scientists who analyze genomic sequence. This is also recognized by other software packages, including the Neomorphic Genome SDK [11].

Although the foundation classes were written with the BIOML browser project in mind, the distinction between universal functionality and BIOML specific functionality was kept clear. All BIOML specific browsing function-

ality is contained in the application code. Nevertheless, some of the BIOML application code could be reused in browsers for other data types with similar requirements. For example, a timeline/schedule browser might reuse the display zoom functionality in order show event elements differently when looking at year, month and day levels.

16.6 FUTURE WORK

The current implementation of the BIOML browser is a proof-of-concept prototype. Elements of the browser will be used for data display in other software being developed at or in collaboration with IMB, such as MAGPIE [12]. The need for specialized data viewing was one of the main reasons for developing Bluejay, and hence this integration is a first priority for further development and testing.

Large data sources are unlikely to keep an XML copy of all documents in addition to the current HTML or plain text versions served through their Web servers. One solution would be to access the XML versions through CGI scripts that do the conversion. A better solution might be to have a proxy configured for the local set of servers on which the data resides. Caching and specialized conversion scripts on the local proxy would improve performance over simple CGI scripts. In order to take advantage of this, a protocol would have to be implemented that allows switching the Bluejay HTTP client from proxy to proxy based on the URLs being accessed. A first draft of such a protocol has been sketched, but is not currently implemented.

Demonstration. A demonstration of the implemented system and the BIOML browser is available on-line at http://maggie.cbr.nrc.ca/bluejay.

References

[1] Raggett, D., Le Hors, A., and Jacobs, I., eds. *HTML 4.0 Specification*, on-line, available at http://www.w3.org/TR/REC-html40.

[2] Bray, T, Paoli, J. and Sperberg-McQueen, C.M., eds. *Extensible Markup Language (XML) 1.0*, on-line, available at http://www.w3.org/TR/REC-xml.

[3] Fielding, R. *et al*. *Hypertext Transfer Protocol - HTTP/1.1*, on-line, available at http://www.ietf.org/rfc/rfc2068.txt.

[4] Stein, L (1998). A Web Proxy Module For mod_perl. *The Perl Journal*, Fall 1998, pp. 23-28.

[5] *SAX 1.0: The Simple API for XML*, on-line, available at http://www.megginson.com/SAX/sax.html.

[6] Wood, L. *et al*, eds. *Document Object Model (DOM) Level 1 Specification*, on-line, available at http://www.w3.org/TR/REC-DOM-Level-1.

[7] Spitzner, J.H., *Bioinformatic Sequence Markup Language*, on-line, available at http://www.visualgenomics.com/bsml.

[8] Beavis, R., Fenyö, D. and Chait, B. *The Biopolymer Markup Language-BIOML Working Draft Proposal*, on-line, available at http://www.proteometrics.com/BIOML/bioml_toc.html.

[9] Halasz, F.G. and Schwartz, M. (1990). The Dexter hypertext reference model. *Proc. NIST Hypertext Standardization Workshop* (Gaithersburg, MD, 16-8 January, 1990), pp. 95-133.

[10] BioBrow. Software. Available at http://www.proteometrics.com/BIOML/Program/biotest.exe.

[11] Neomorphic Genome SDK. Software. Available at http://www.neomorphic.com/GenomeSDK.html.

[12] Gaasterland T., and Sensen, C.W. (1996). Fully automated genome analysis that reflects user needs and preferences - A detailed introduction to the MAGPIE system architecture. *Biochimie* vol. 78, pp. 302-310.