

Processing Sequence Annotation Data Using the Lua Programming Language

Yutaka Ueno^{1,2}

uenoyt@ni.aist.go.jp

Toshitaka Kumagai¹

kuma@cbrc.jp

Masanori Arita^{1,3}

arita@k.u-tokyo.ac.jp

Kiyoshi Asai^{1,3}

asai@k.u-tokyo.ac.jp

- ¹ Computational Biology Research Center, National Institute of Advanced Industrial Science and Technology, 2-43-17 Aomi, Koto-ku, Tokyo 135-0064, Japan
- ² Neuroscience Research Institute, National Institute of Advanced Industrial Science and Technology, 1-1 Umezono, AIST Central 2, Tsukuba, Ibaragi 305-8568, Japan
- ³ Department of Computational Biology, Graduate School of Frontier Sciences, The University of Tokyo, 5-1-5 Kashiwanoha, Kashiwa-shi, Chiba 277-8561, Japan

Abstract

The data processing language in a graphical software tool that manages sequence annotation data from genome databases should provide flexible functions for the tasks in molecular biology research. Among currently available languages we adopted the Lua programming language. It fulfills our requirements to perform computational tasks for sequence map layouts, i.e. the handling of data containers, symbolic reference to data, and a simple programming syntax. Upon importing a foreign file, the original data are first decomposed in the Lua language while maintaining the original data schema. The converted data are parsed by the Lua interpreter and the contents are stored in our data warehouse. Then, portions of annotations are selected and arranged into our catalog format to be depicted on the sequence map. Our sequence visualization program was successfully implemented, embedding the Lua language for processing of annotation data and layout script. The program is available at <http://staff.aist.go.jp/yutaka.ueno/guppy/>.

Keywords: sequence annotation, genome database, scripting language

1 Introduction

Large-scale sequencing projects of the human genome and the genomes of other species have revealed rich sets of information coded in nucleic acid sequences. Several database systems have provided sequence annotations using World Wide Web technology, allowing biologists to query and retrieve data of interest through comprehensive graphical interfaces. Since biological studies involve collecting data for different species and experimental conditions, the retrieved data are usually saved intact in desktop computers, then the different formats are converted for comparisons and further analyses.

Although the inconsistencies among data formats are cumbersome, molecular biology databases tend to retain their own schemas and distribution formats in order to maintain the best quality of service with up-to-date information available from their steady servers [15]. While there are standard formats, GenBank and EMBL, in formal data repository systems, a much simpler format is preferable for informal data exchange and practical computational tasks. The General Feature Format (GFF) has been proposed as a tab-separated record format of 9 fields in a line [28], but this simple method does not provide enough fields to describe complex and detailed data. The extensive mark-up language (XML) was suggested as an appropriate language for sequence annotation and other molecular biology data exchanges [10]. Although software tools for XML are available, substantial effort is required to fix the data schema for sequence annotations in practical applications [2]. Therefore, there is a need for optimal solutions for individual computational tasks in bioinformatics.

Perl [25] is a popular and useful scripting language for the inevitable file conversion tasks. Since Perl converts each line of a formatted file, a hierarchical data description in multiple lines forces programmers to parse and arrange the data structure. Therefore, the use of ASN.1, a general communication protocol for such hierarchical data description [12], was proposed. This method provides a rich object model to manipulate sequence annotation supported by a software development toolkit [24]. Although the integrated database services in Entrez were developed with ASN.1, it is not popular among individual biological research projects because its use requires substantial expertise.

There is a demand for visualization tools for increasing sequence annotations that allow biologists to compare their experimental results with data in official annotation repositories. GFF2PS [1], a program that generates postscript graphics from GFF annotation data, is used in many publications. It is written in the Awk programming language and this probably inhibits natural improvements of the code. Because Awk is designed for a text-processing pipeline with pattern-matching and action, it is inadequate for interactive tasks in sequence map management with hierarchical data.

Here we introduce the use of the Lua programming language [4, 7, 23] to describe and process sequence annotation data. Not only is Lua as good as ASN.1 with respect to hierarchical data description, but like Perl, it also contains a necessary programming facility. Even though its programming syntax is very ordinary, it provides well-studied data description facilities, a feature that is especially useful for processing structured data. The implementation of this language is compact and lightweight allowing it to serve as an embedded language for extending application programs.

With the current rich code resources for lexical analyzers and compilers of languages, we could have designed a new language dedicated to the processing of genetic sequence annotations. However, we chose a preexisting language because of advantages regarding various software engineering issues and technical know-how. In constructing a data processing language in a software system, data description and scripting of computational tasks are common among different application areas. In addition, the language needs to integrate with the legacy C language code in which these methods have been extensively studied in preexisting languages. We found that, as Lua provides substantial amounts of these engineering codes for constructing our target software system, we did not need to repeat the same kind of coding effort. In the following section we describe the requirements for an ideal language that is mostly supported by Lua.

We developed a graphical annotation viewer program for the results of gene-finding searches in genomic sequences [3]. It manages the layout of graphical primitives for annotation data that consist of a set of sequence positions, user-defined tags, and references to other data on the sequence map. To support a programmable layout, we also implemented the scripting framework using the Lua language interpreter so that annotations are easily edited or arranged in the sequence map. The result was satisfactory and demonstrated that Lua is a practical and appropriate choice for processing sequence annotation data. We also compared our method with presentations using different languages including XML, and discuss the advanced use of the scripting language in bioinformatics.

2 Methods

2.1 Required Functions for the Language

Based on our experience in processing sequence annotation data, we identified the following tasks as essential functions of a data processing tool language.

- **Data Rearrangement:** A biological study involves extracting information of interest from a huge sequence repository in which various kinds of annotations are differently formatted. The task consists of picking up, grouping, sorting, and comparing information. The information also includes local data obtained in biological experiments.
- **Coordinate Translation:** Annotation data on a fragmented sequence locus are mapped on a different sequence map with a different coordinate. A comparison with different species will introduce

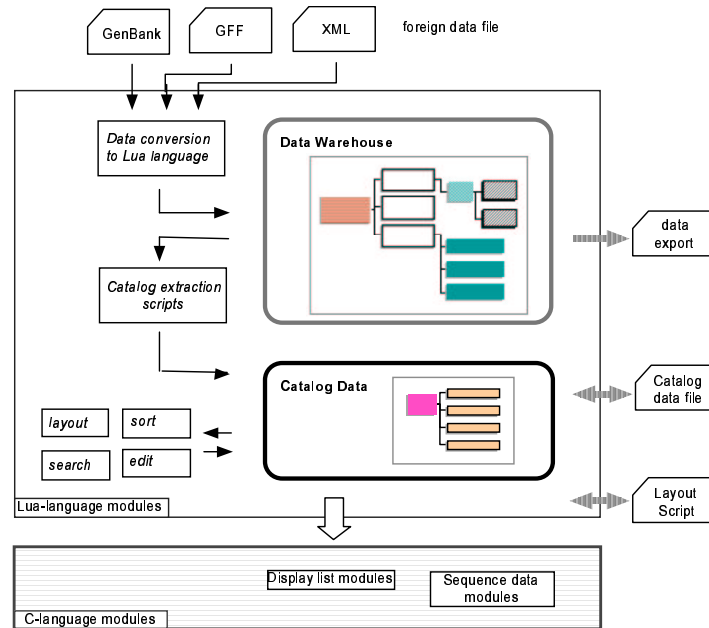


Figure 1: An overview of processing sequence annotation data in our sequence map layout program. The program consists of Lua language area and C-language modules. The data warehouse and catalog data are constructed in the Lua data area, which consist of hierarchical data tree structure, illustrated as imaginary diagrams in rounded rectangles. Solid rectangles are functional modules for processing those data. C-language modules for display list and sequence data are not so large and responsible for quick operations for given tasks.

a new mapping coordinate. This multiple addressing is supported by numerical calculations to the position attribute values of the annotations.

- On-Demand Editing: Examination of sequence annotation data usually appends hypothetical annotations. Semantic inconsistencies in official annotation repositories should also be modified. They are locally edited before submitting corrections to the public data server.

These tasks are greatly facilitated by a data processing meta-language. First, the hierarchical structure of the sequence annotation data is described in the language format. Then, the above tasks are described in programs of procedural operations using symbolic references to individual data elements. Thus, the functions we require of the meta-language are as follows:

- Dynamic creation of a data container, a symbol table for attribute data of strings and numbers, through automatic memory allocations at run time.
- Support of symbolic references to annotation primitives as well as numerical operations.
- Simple language syntax to be used by biologists and individuals with moderate programming knowledge.

2.2 Choosing a Language

Based on the above requirements, we surveyed available programming language implementations. Although a general purpose database management system may be a valid choice, such systems are also enhanced by a programming language. Because C/C++ and Java provide statically structured data only, they are not appropriate for our tasks. Lisp is not adapted to biologists in terms of programming syntax [10]. Hence, the candidates include scripting languages that support associative arrays for dynamically structured data. Perl uses special characters such as \$, @, ->, and [] for data

structures; this makes data description unclear and error-prone. Although Python [26] and Ruby [27] do satisfy our requirements, we chose Lua [7] because it provides the following advantages:

- data description facilities without the introduction of object-oriented programming exercises;
- a simple and stable application programming interface (API) to be embedded in a host program, and
- compact language implementation with a small memory footprint.

We think that the programming syntax of Lua is clearer than that of other languages with respect to a practical code for sequence annotation data (illustrated in Fig. 2).

2.3 The Lua Programming Language

Lua was originally designed for extending application programs, but it is often used as a general purpose scripting language. Its table data type, which implements associative arrays, offers a structured data container. Lua has a simple procedural programming syntax, where statements of assignment or function calls are executed. Conventional flow control structures are supported, e.g., if-then-else-end, while-do-end, repeat-until. Lua also features an automatic memory management and garbage collection. Optionally, Lua programs in text files can also be compiled to the byte code for a virtual machine to accelerate processing.

2.4 Catalog Format

We introduce a catalog format for sequence annotation, the primary data for our sequence map layout program, to display graphical marks on the map. An annotation element is described by a table data type in Lua; it lists attributes in the form KEY = VALUE. The catalog format is defined as an array of tables in Lua language as in the following example:

```
one={
  {symbol="orfD", pos1=18, pos2= 48, category=1},
  {symbol="orfE", pos1=58, pos2= 78, category=3},
  {symbol="gene1",pos1=88, pos2=188;
   {symbol="5'utr",pos1=88,pos2=94},
  }
}
```

This example describes three regions of a sequence, from `pos1` to `pos2`. Each region contains two attribute values, `symbol` and `category`. The third region embeds a subsidiary annotation. Note that a user can use arbitrary key and value pairs, e.g., `db_xref="gdb:N1234"`. By conforming to the Lua syntax for the catalog format, the data can be loaded by a standard Lua interpreter. Foreign data files are converted into this catalog format.

2.5 Data Warehouse

Fig. 1 provides an overview of sequence annotation data processing in our visualization program. In addition to the catalog data area, the data warehouse, an "on-memory" database that uses the tables in Lua is used to facilitate importing foreign data in various file formats. Annotation data are converted in two steps:

1. Foreign data are parsed and decomposed into tables in Lua, maintaining the original structures and database schemas. The converted data file become data for Lua programs.
2. Catalog data are then extracted from the converted tables according to the original data schema. The task involves simple programming to arrange the tables in Lua language.

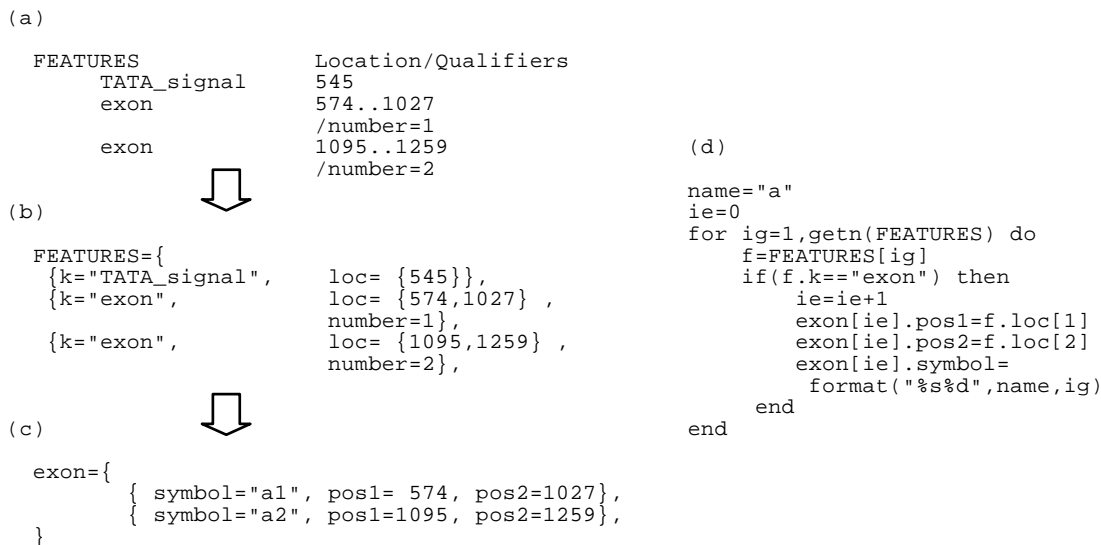


Figure 2: A sample conversion of GenBank data into Lua language. (a) Three FEATURE records from a GenBank entry in HUMHA2WC are shown. (b) A description in Lua language that preserves the original information. The column position does not have any meaning in Lua language, but is appropriately adjusted to the original. (c) Then, catalog data are extracted by a trivial Lua program and depicted on the sequence map. (d) A sample Lua code to extract data into the catalog data.

Fig. 2 illustrates how GenBank data are converted into the catalog format. The first-step parser is a preprocessor that converts foreign data into Lua language. It preserves the original information including nested location specifiers in FEATURES records, e.g. `join()` and `one-of()` defined in GenBank. The converted file is parsed by the Lua interpreter and the contents are stored in the data warehouse. Then, the second script extracts specific annotation data into catalog data, e.g. "exon" in the FEATURE key. In this way, all GenBank data are converted into tables without any inherent difficulty. Since the first-step preprocessor only performs pattern-matching and substitution tasks, unexpected semantic inconsistency in data entries becomes clear mostly in the second step. Such errors are corrected by editing the data warehouse files written in Lua language.

Usually the data warehouse file preserves as much of the original information of the foreign data as possible, thus preserving the data schema. Therefore, it can be a local subset of the formal database and is reusable for other processing tasks. This characteristic is similar to the modern database architecture proposed by Inmon [8]. Current Lua language implementation also allows the use of user-defined algorithms for associative arrays by a mechanism called *tagmethod*. Upon accessing the table marked with the *tagmethod*, it can invoke a user-defined function, for example, to access an external database system automatically. This database interface method through a table reference is especially useful for large molecular biology databases.

3 Results

3.1 Implementation of GUPPY

We have implemented a sequence visualization program, the Genetic Understanding Perspective Preview sYstem (GUPPY). This highly interactive program allows smooth scrolling and zooming from the genomic landscape to discrete nucleic acid sequences in the sequence map window. There are also a catalog list window and a sequence window; selecting annotation primitives by clicking the mouse creates echoes in other windows. The graphical user interface is maintained by our programming library [16] which was ported to Windows (Win32 API, Microsoft Corp.) and MacOS (Classic Toolbox,

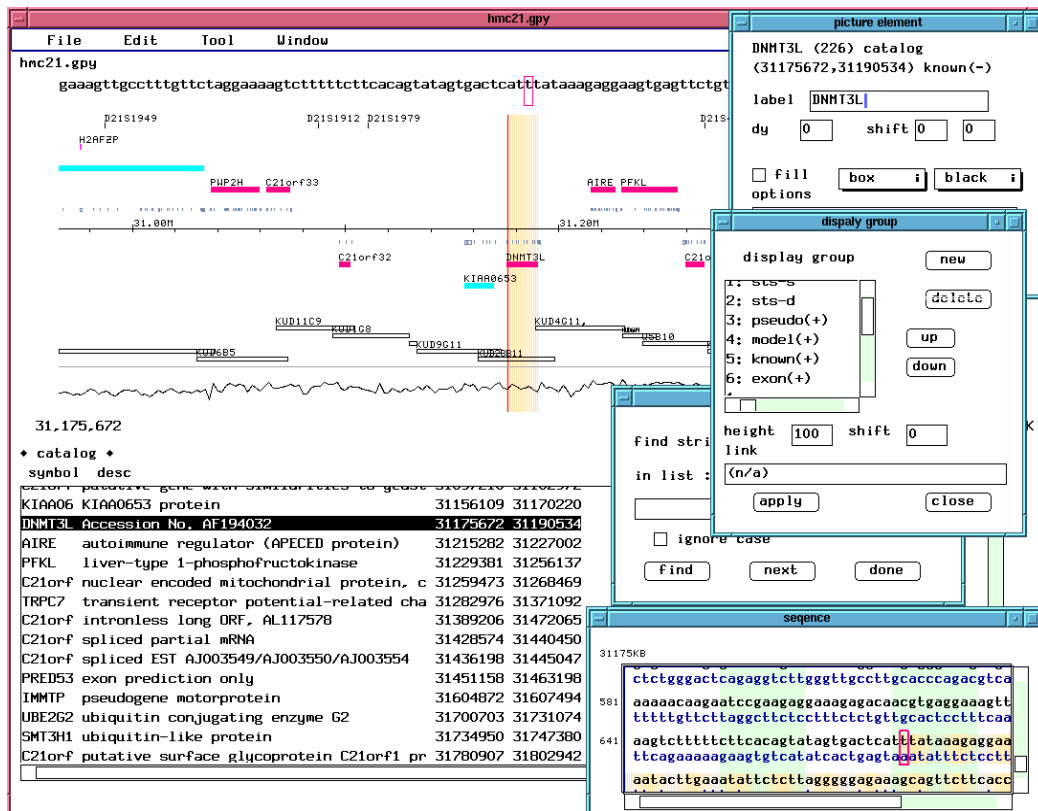


Figure 3: A screen shot of the GUPPY program for the sequence map of human chromosome 21. The main display consists of the sequence map window and the catalog list window below. The sequence window (lower right corner) shows the nucleic acid sequence data. The selection of the gene name is highlighted in the catalog list, and corresponding regions are also selected in other windows.

Apple Computer Inc.) as well as the UNIX operating systems (Linux, IRIX, SunOS, etc.), using the X-Window system. Since the Lua implementation already works in those systems, GUPPY is also a portable program.

The scripting function was implemented by the Lua language library [23] (ver.4.01) written in ANSI C. The library was very well designed so that our program also remained compact: the source code is less than 10,000 lines in Lua and C language. It was very important to separate appropriately the functions of Lua language modules from C language modules. The annotation data are loaded into the Lua data area via a catalog format file; the display list is constructed in a C language module to maximize performance. Large sequence data in FASTA format are also managed in a C language module (Fig. 1).

Fig. 3 is a sample layout for a sequence map of human chromosome 21; it is similar to the layout in the original publication [6]. Segmented GenBank repository data [19] were converted into catalog data for genes, clones, and genetic markers. Some of the semantic inconsistencies in repositories were corrected in our memory database. In addition to private annotation tracks, numeric array data such as the GC contents, can also be appended to the created sequence map.

The program was also designed to use a command line tool from a console. For GFF or GenBank format files, GUPPY accepts the filename as a command line argument to open and display. There are also applications to display sequence alignment by BLAST and Clustal-W. For file conversion tasks, GUPPY also works without a graphical user interface when no window is created by the script. Details of implementation are described in our homepage (<http://staff.aist.go.jp/yutaka.ueno/>)

guppy/).

3.2 Layout Script

Sequence map layouts are described in script files. After creating a main window with the command `guppy:new()`, graphical annotations in the sequence map are created with the command `addlane()` specifying a catalog and its graphical primitive, as in the following example:

```
untitled:addlane("CDS",{ primitive=GPYbox, source=one}, ... options... )
options ... selection={category=1},
pen=RED, height=20, ...
```

This command appends an annotation group named “CDS” to the sequence map designated by the symbol “untitled”. It generates an annotation lane that runs horizontally on the sequence map and contains annotation primitives with a specified height, as shown in Fig. 3. The `primitive` attribute defines the shape of the group, but these attributes are customized in individual annotations. The `selection` attribute provides options to select a specific annotation from the catalog, and the `pen` attribute changes its color. Additional commands to manipulate layout are provided by either scripts or GUI operations. The script written in Lua is easily reused in another layout for different sequences.

3.3 Performance Test

The performance of the on-memory database of Lua was tested by comparisons with Perl, Python, and Ruby. A set of 4463 annotation data [18] with 15 fields of data (426KBytes) for the *E. coli* K-12 (W3110) genome sequence [11] was translated into individual catalog formats, i.e., “array of associative array” in each language. Measured results in computational times to load and simple-dump and memory usage are shown in Table. 1. While Perl and Lua are faster than the others, memory consumption was lowest in Lua. The required amount of memory for loading data was roughly estimated as 50 bytes for each data element; this increases for large strings of data. For example, one million annotation data with 4 fields require 200 MB of memory.

Table 1: Benchmark test of associative array of Lua. Processing time and memory were measured by “time” and “ps” command of Linux (2.2.18-14-SMP) using a ATX-PC (Tiger 100, Tyan Inc.) with dual Pentium3, 800Mhz, 1GB memory.

	Lua	Perl	Ruby	Python
(version)	(4.0.1)	(5.005_03)	(1.6.8)	(2.1.3)
Load time	0.2	0.7	1.2	3.6
Dump time	0.7	0.3	0.3	1.3
total	0.9	1.0	1.5	4.9
(second)				
Memory	4,424	11,904	16,610	52,984
(Kbyte)				

4 Discussion

4.1 Comparisons with Related Works

Various visualization programs, e.g. Apollo [22], Artemis [13], and VectorNTI (InforMax Inc.), provide graphical sequence maps. Language interpreters are embedded to support processing annotation data in ACeDB [20] and BioTk [14]. For sequence data processing, BioPerl [21] provides reusable software

components in Perl, and BioWidget [5] allows users to manage computational tasks with visual software components. GUPPY is unique with respect to its data warehousing method: it allows a new script to import virtually any data format.

Usually, application programs have an underlying data structure that reflect their display-, input-, and output functions. For the importation of foreign data files, additional programs are required to parse the format to pick up necessary data. Consequently, some of the data may be lost due to incompatibilities between the data model and the foreign files. In our method, on the other hand, foreign data are first decomposed systematically in Lua tables without loss of the original information. The necessary features of the data are later arranged into a catalog format for each display using a script program. Once the collected catalog is visualized, rearrangement and layout tasks become straightforward: as all are managed in the Lua interpreter, not only the file format conversion task but also the overall software development effort of the target application program, are minimized.

4.2 Data Sarehouse

Karp, who discussed strategies for database interoperation for a heterogeneous molecular biology database [9], warned that the data warehouse approach will fail unless high-level translation facilities for data schemas are provided. We propose that Lua has solved this problem. Our data warehouse is a simple "on-memory" database system with ad hoc arrangement of tables. Attempts to build a sequence map viewer program on a general purpose relational database management system (DBMS) usually result in burdening of the application program by the programming interface. In fact, the sequence map application does not really require the query features and multiple update transactions provided by a DBMS.

The "on-memory" data warehouse can be substituted with external data written in Lua language. Lua serves as an intermediate language that facilitates reuse in other analyses and comparisons with other data. In addition, it provides a compiled byte code for data and programs that is platform-independent and useful in a distributed network environment. Using the compiled byte code, the loading time of the data warehouse can be accelerated threefold. Unlike the case with a DBMS, Lua language implementation is compact and easily embedded in application programs. Lua greatly facilitates scripting of data processing in an application program, which is a key to our data warehouse strategy.

4.3 XML Versus Lua

While XML is expected to be a standard format for data exchange in bioinformatics, it does not fully support our language function requirements. Although tools such as XSLT (W3C Recommendation) [29] support rearrangement of data on a tagged data tree, numerical operations, and programming facilities, the concept is similar to that of Awk language. Therefore, a more popular alternative is an interface to another programming language, for example, Java. To avoid having to program in 2 languages simultaneously, the "document object model" method allows a program to access XML data as a structured variable on-memory. This is the same as our approach with the data warehouse.

In addition, while XML usually describes a well-formatted static data schema with rigid tags, an active research project sometimes requires modifying and adding new tags on demand. Although XML can provide consensus annotation data file formats, the use of such tags would yield conflicting differentiations from the original XML schema. Therefore, our data warehouse is designed to accept imported data in different data schema. A non-standard data structure is resolved by a custom script that extracts necessary information beyond its formal data specification. For those data rearrangement tasks, the use of scripting language is mandatory for quick programming and testing.

4.4 Embedding a Scripting Language

Lua is used not only as a scripting language for data processing, but also as the language interpreter to parse annotation data. In reusing the parser, a security concern arises because a harmful code in the foreign data may be executed in the course of parsing. Therefore, the parsing environment must be designed for secure execution. For example, execution of external programs and writing to existing files should be disabled in parsing data. Lua provides full control of operating system-related functions; we allow a restricted set of operations to be executed. This is a matter of language design as an embedded language rather than implementation techniques.

In terms of language design, Lua is similar to Lisp. The list in Lisp is a container of data, functions, and other lists. The authors of Lua stated that "... Tables are for Lua what lists are for Lisp: powerful data-structuring mechanisms" [4]. Like Lisp, Lua provides some reflective facilities in which a program may be treated as data. In fact, Lisp fulfills our language requirement if we can tolerate its programming syntax. Thus, the characteristics of Lua support flexible descriptions and manipulations of the data container. Our results demonstrate that Lua is a lightweight programming language for the tasks for which Lisp has been considered appropriate.

4.5 Future Plans

Lua was appropriate for the processing of various kinds of annotation data. We have also applied the Lua language to a molecular structure viewer program [16] to provide a scripting facility. Another application of Lua is currently being tested in an image processing program for single particle analysis of protein images using an electron microscope [17]. In addition to these projects, we are also considering applications for highly complicated data such as graph data for metabolic pathway maps. Using the tables of Lua makes the graph data management a straightforward programming issue. In simulations of biochemical reactions in cells, the programming language also plays an important role. For example, if a program code can be written inside data, the code can be modified by a program during the simulation runtime. As this reflective facility is natural in biological systems, we expect that use of Lua and other scripting languages will facilitate development of a novel self-organizing algorithm for reaction networks in cells.

5 Conclusion

Computational tasks to visualize annotation data for genetic sequences involve data rearrangement, coordinate translation, and local editing. Those tasks are greatly aided by a programming language that provides the necessary functions: handling of data containers, symbolic references, and a simple programming syntax. The Lua programming language fulfilled our requirements. Tables in Lua provide not only the catalog format for the sequence map layout, but also a data warehouse for importing foreign data. Our sequence visualization program was implemented using the Lua language which provides the scripting functions necessary for arrangement of annotation data and a flexible layout. Our results illustrate that Lua greatly facilitates the processing of sequence annotation data with a comprehensive language syntax.

Acknowledgments

We thank Dr. Yutaka Akiyama, Dr. Katsutoshi Takahashi, Dr. Makiko Suwa, Dr. Hirotada Mori, Dr. Takeshi Ara, Dr. Taku Oshima, Mr. Ikuo Okouchi, and Mr. Taishin Kim for useful discussions and encouragement.

References

- [1] Abril, J.F. and Guigo, R., GFF2PS: visualizing genomic annotations, *Bioinformatics*, 16:743–744, 2000.
- [2] Achard, F., Vaysseix, G., and Barillot, E., XML, Bioinformatics and data integration, *Bioinformatics*, 17:115–125, 2001.
- [3] Asai, K., Itou, K., Ueno, Y., and Yada, T., Recognition of human genes by stochastic parsing, *Pac. Symp. Biocomput. '99*, World Scientific, 4:228–239, 1999.
- [4] De Figueiredo, L.H., Ierusalimschy, R., and Celes, W., Lua: an extensible embedded language, *Dr. Dobbs's J.*, 21(12):26–33, 1996.
- [5] Fischer, S., Crabtree, J., Brunk, B., Gibson, M., and Overton, G.C., BioWidgets: data interaction components for genomics, *Bioinformatics*, 15:837–846, 1999.
- [6] Hattori, M. *et al.*, The chromosome 21 mapping and sequencing consortium, The DNA sequence of human chromosome 21, *Nature*, 405:311–319, 2000.
- [7] Ierusalimschy, R., de Figueiredo, L.H., and Celes, W., Lua: an extensible extension language, *Softw. Pract. & Exp.*, 26(6):635–652, 1996.
- [8] Inmon, W.H., *Building the Data Warehouse 2nd ed.*, John Wiley & Sons, New York, 1996.
- [9] Karp, P., A strategy for database interoperation, *J. Comput. Biol.*, 2(4):573–586, 1996.
- [10] McEntire, R., Karp, P., Abernethy, N., Benton, D., Helt, G., DeJongh, M., Kent, R., Kosky, A., Lewis, S., Hodnett, D., Neumann, E., Olken, F., Pathak, D., Tarczy-Hornoch, P., Toldo, L., and Topaloglou, T., An evaluation of ontology exchange languages for Bioinformatics, *Proc. Int. Conf. Intell. Syst. Mol. Biol.*, AAAI Press, 239–250, 1999.
- [11] Mori, H., Isono, K., Horiuchi, T., and Miki, T., Functional genomics of *Escherichia coli* in Japan, *Res. Microbiol.*, 151(2):121–128, 2000.
- [12] Ostell, J., Wheelan, J.S., and Kans, J.A., The NCBI data model, *In Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins (Baxevanis, A.D. and Ouellette, B.F. eds.)*, John Wiley & Sons, 19–43, 2001.
- [13] Rutherford, K., Parkhill, J., Crook, J., Horsnell, T., Rice, P., Rajandream, M.A., and Barrell, B., Artemis: sequence visualization and annotation, *Bioinformatics*, 16:944–945, 2000.
- [14] Seals, D.B., BioTk: componentry for genome informatics graphical user interfaces, *Gene*, 163:1–16, 1995.
- [15] Topaloglou, T., Kosky, A., and Markowitz, V., Seamless integration of biological applications within a database framework, *Proc. Int. Conf. Intell. Syst. Mol. Biol.*, AAAI Press, 272–280, 1999.
- [16] Ueno, Y. and Asai, K., High-throughput graphics library designed for a portable molecular structure viewer, *Pac. Symp. Biocomput. '98*, World Scientific, 3:201–212, 1998.
- [17] Ueno, Y., Takahashi, K., Asai, K., and Sato, C., BESPAs: software tools for three-dimensional structure reconstruction from single particle images of proteins, *Genome Informatics*, 10:241–242, 1999.
- [18] <http://ecoli.aist-nara.ac.jp/>, Genobase database.
- [19] <http://studio.nig.ac.jp/>, DDBJ/CIB Human Genomics Studio.
- [20] <http://www.acedb.org/>, *A. C. elegans* database.
- [21] <http://www.bioperl.org/>, The Bioperl project.
- [22] <http://www.fruitfly.org/annot/apollo/>, Apollo genome annotation tool.
- [23] <http://www.lua.org/>, The programming language Lua.
- [24] <http://www.ncbi.nlm.nih.gov/IEB/ToolBox/index.cgi>, NCBI ToolBox.
- [25] <http://www.perl.com/>, The source for Perl.
- [26] <http://www.python.org/>, The Python programming language.
- [27] <http://www.ruby-lang.org/>, Ruby: programmers' best friend.
- [28] <http://www.sanger.ac.uk/Software/formats/GFF/>, GFF: an exchange format for feature description.
- [29] <http://www.w3.org/TR/xslt.html>, XSL transformations (XSLT).