A Platform-Based Design Environment for Synthetic Biological Systems

Douglas Densmore[1*], Anne Van Devender[1+], Matthew Johnson[2*], Nade Sritanyaratana[2*]
Department of Electrical Engineering and Computer Sciences[1]
Department of Bioengineering[2]
University of California, Berkeley[*]
Washington and Lee University[+]
vandevendera@wlu.edu {densmore@eecs, matthewjohnson, nadesri}.berkeley.edu

ABSTRACT

Genomics has reached the stage at which the amount of DNA sequence information in existing databases is quite large. Synthetic biology is now using these databases to catalog sequences according to their functionality thus creating a system of standard biological parts. Flexible tools are needed which both permit access and modification to that data and also allow one to perform meaningful, intelligent manipulation. A Platform-Based Design approach views genetic information as having a particular functionality and assembles platforms (collections of DNA elements) to perform this functionality. Specifically this paper presents the *Clotho* toolset which uses these concepts to create a complete design environment for standardized biological parts.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Design, Management

Keywords

Platform-based Design, Synthetic Biology

1. INTRODUCTION

Synthetic biology is a rapidly growing field in which the techniques of chemistry, biology, and engineering merge. Synthetic biologists look to create new microorganisms by manipulating the basic building blocks of life to create living material which interacts with, manipulates, and responds to the environment in which it lives. Synthetic biology is very much a *design science* where a new system is created by researchers in laboratories using a series of design steps along with their understanding of biological processes. Synthetic biology has the potential to greatly impact greater society through the development of new technologies in drug production, biofuels, and drug delivery vessels.

In an attempt to standardize this process, leverage previous design experiences, and begin to create a predictive design environment, registries of standard biological parts are beginning to emerge [14]. Researchers have begun to talk about how to classify these parts, create CAD systems, and establish standards [13], [12], [7], [19], [10]. [18] lays out very nicely an example of how these parts can be used to *program* bacteria and discusses how they can be characterized (e.g. sensors, switch logic, inducers, etc). The fact that these collections of *parts* can be discussed in terms of their functionality along with rules for their composition, raises the interesting question of how the Electronic Design Automation (EDA) community (traditionally in electrical engineering and computer science) possibly can leverage its techniques in the creation of biological systems.

This paper describes the design of a toolset called *Clotho* (named after the Greek fate which spun the thread of life) which uses a methodology called Platform-Based Design (PBD) [16] to approach the problem of designing synthetic biological systems. In particular, we will describe its separation of computation, communication, and coordination, the concept of a "platform" as a common semantic meeting place for designs, and the notion of both "top down" and "bottom up" design styles.

1.1 Requirements

In the world of biology one can roughly separate tool offerings into three broad categories. The first category are those tools which provide computational power to specific biological algorithms. BLAST (Basic Local Alignment and Search Tool) [15] aligns nucleotide and protein sequences to allow for functional prediction and to aid in locating sequences in databases. ORBIT [11], [4] is protein design software which allows the design of an amino acid sequence that folds into a particular 3D structure. Mfold [20] enables the prediction of mRNA secondary structures which aids in predicting mRNA regulation and ribosome binding site strengths. These types of tools require a strong understanding not only of the underlying biology but also that the biology be *predictive*.

The second category of tools are those tools which allow the

Tapia'09, April 1-4, 2009, Portland, Oregon, USA.

user to design biological systems. Many biological systems are not well understood, thus requiring a very empirical approach in which one must observe a number of experiments and create hypotheses based on the collected data. In order to perform this research in an efficient, repeatable manner tools must exist to help in this process. Examples of these tools are APE (A Plasmid Editor) [5], BioJADE [8], Gene Designer [17] by DNA 2.0, and GenoCAD [3].

Finally, the third category are "glue tools" which do not require a great deal of biological knowledge but are used routinely by biologists in a laboratory setting. Functionality needed includes: translation of a gene into an amino acid sequence, producing the reverse complement of a sequence, calculating the melting point for a region of DNA, identifying restriction sites and other DNA motifs, calculating transmembrane regions, calculating the probability that a protein has a secretion signal sequence (signalP), etc. This list continues to grow and each lab has its own set of favorites.

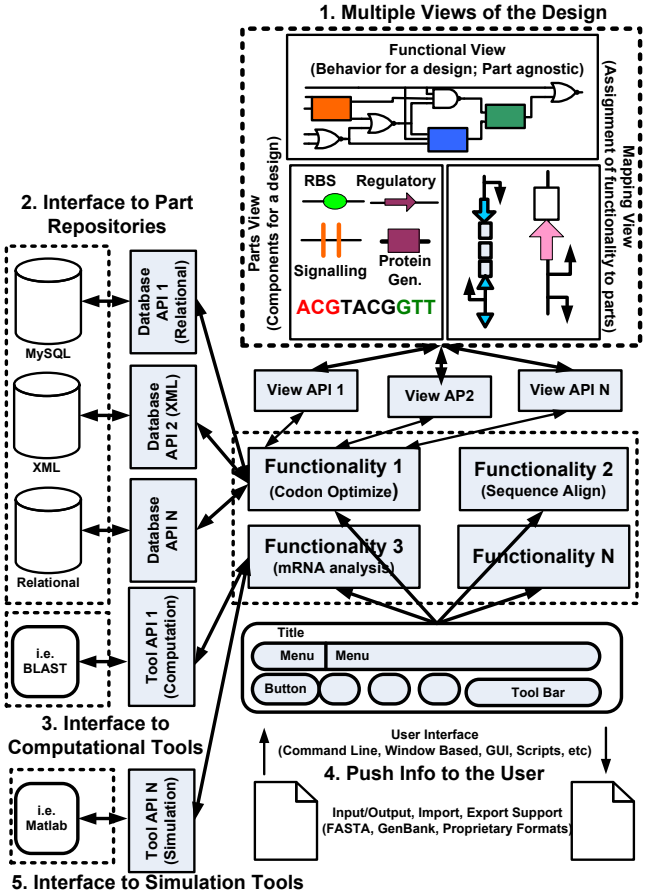

Figure 1: Framework for the Design of Synthetic Biological Systems

This work recognizes the fact that a successful design environment will encompass all three categories. It should do so in such a way that allows for the tool to grow and expand while not being overwhelming complex for an end user. Any new tool offering in the design area needs to minimally support five aspects.

1. **Provide various views of the design**. It is important that the designer be able to view the system at various abstraction levels and from various perspectives. Examples of abstraction levels in DNA manipulation may include viewing the entire genome, individual genes, and various nucleotide sequences. Perspectives include DNA "design" functionality (e.g. sensors, actuators, switches) or DNA "biology" functionality (e.g. terminators, ribosome binding sites, promoters).

2. **Interface with both local and remote part repositories**. The tools should support both importing and exporting parts to well know databases such as MIT's Registry of Standard Biological Parts [14].

3. **Support the interaction with computational tools**. The environment should allow the design under examination to be operated on by a large variety of external computational tools. It should also support the export and import of designs in a variety of standardized formats (such as FASTA, GFF3, and GenBank).

4. **"Push" information to the user**. The tools should not passively allow for the designer to create a system. Constant feedback as to the validity of the design should be provided whenever possible.

5. **Provide simulation support**. Simulation of a completed system or aspects of the system should be available when such simulation engines exist [2]. The internal design representation should be modular as to allow subsections to be extracted for this purpose.

Figure 1 illustrates such a framework with these five aspects. Platform-based design will very nicely lend itself to the first requirement (various views) as will be shown. The other aspects will require a very structured software engineering approach with a polymorphic, inheritance based approach to API design. These issues are touched on in Section 2.

Finally we should state the goals of such a design environment clearly. A tool in this space should take the design of small biological circuits and systems (<20,000 base-pairs) and push it to the whole engineering of a genome (4 megabases+). It should allow both designs for commercial synthesis as well as design in by researchers in an academic setting (i.e. be technology agnostic). It should also be freely available under an open source ideology such as BSD.

1.2 Organization

The rest of this paper is organized as follows: Section 2 discusses Clotho's system architecture. This demonstrates how we adhere to PBD concepts as well as those of disciplined, object oriented software design. Section 3 discusses Clotho's design flow. We illustrate how each of the five aspects of a successful design tool are included. We also show how "top down" and "bottom up" design can be achieved. Finally, Section 4 provides conclusions and future work.

2. CLOTHO SYSTEM ARCHITECTURE

Platform-based design is a very complex methodology which cannot be completely covered in this paper. For brevity's sake we will focus on three aspects of PBD in this paper:

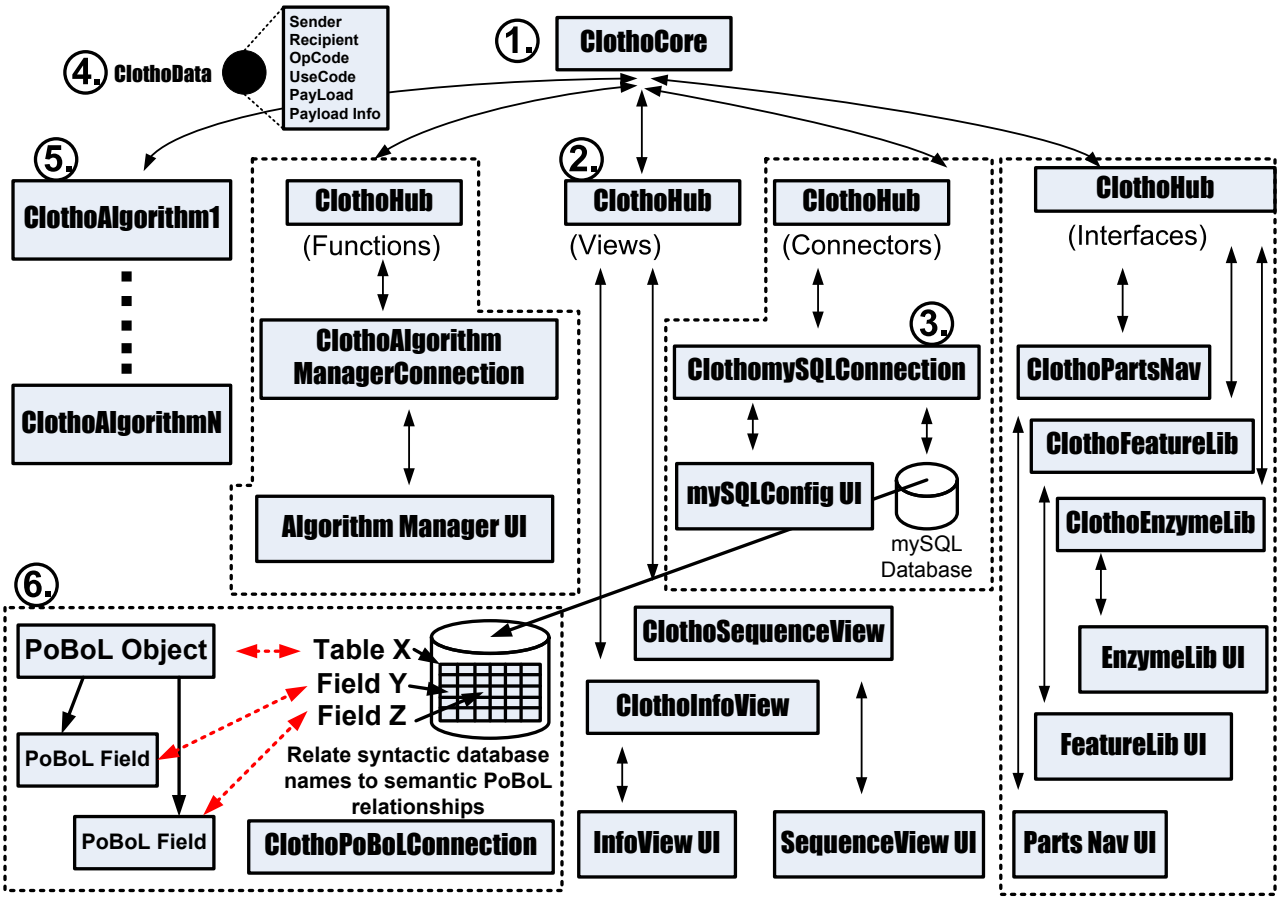

Figure 2: Clotho System Architecture

"orthogonalization of concerns", the concept of a "platform", and a "meet in the middle" approach to design. This section will focus on orthogonalization of concerns and Section 3 will focus on the other two. For a more complete picture of PBD we refer the reader to the references.

"Orthogonalization of concerns" is the separation of communication, coordination, and computation. This aids in reuse, debugging, and system analysis. The added modularity allows for system expansion as well as configurability. Therefore each aspect of the Clotho system is classified as to what type of operations it is involved with and the communication between components is explicitly separated from the computation of each component. The coordination of the system is also removed from each individual component and maintained in a central location. Figure 2 illustrates Clotho's overall system architecture. In particular there are six areas of interest: the *ClothoCore*, *ClothoHubs*, *ClothoConnections*, *ClothoData*, *ClothoAlgorithms*, and *PoBoL Bindings*. Each of these areas will be described in the following subsections beginning with the most basic element, ClothoConnections.

2.1 ClothoConnection

ClothoConnections are the "workhorses" of the system. They represent the *computational* aspect of PBD. Connections are categorized as view type, connector type, function type, and interface type. View connections deal with the display of biological information. This can be both graphical or textual. Views may also present (push) system information to the user. Connector type connections connect Clotho to external tools or data sources. Function connections are processing engines for data. Interface connections are points of interaction for the user to control the operation or settings assigned to Clotho. Interface connections can also manage libraries which Clotho uses. An example of a ClothoConnection is marked by #3 in Figure 2. Notice that connections are explicitly separated from the user interface (UI). This allows complete aesthetic overhauls of Clotho without having to modify the connections.

ClothoConnections are by far the most prevalent objects in the system. They are derived Java classes which inherent methods to process data, communicate with other connections, display information to specific debugging sources, group themselves with other connections, and to make themselves explicitly available to the user via a Java Swing GUI interface. Key to the operation of ClothoConnections, is the ClothoData object which will be described next.

2.2 ClothoData

In Figure 2, #4 marks a ClothoData object. ClothoData objects are the means by which ClothoConnections communicate with one another. ClothoData objects are classes encapsulating the following information:

- Sender - The connection which generated this object. This is the connection which is typically initiating a transaction.
- Recipient - The intended destination for this object. This is the connection which is typically responding to a transaction.
- Op Code and Use Code - These codes determine the type of operation which the data should be used for (e.g. calculate a DNA sequence's open reading frames) as well as how it should be used within the operation itself (e.g. the data is the sequence itself). There are an explicit enumeration of both Op Codes and Use Codes which enforce type safety in the system.
- Payload - This is the bulk of the data. This is a generic data object which allows the system to pass back and forth whatever is required for the transaction.
- Payload Information - An additional mechanism for detailing the payload should the Op and Use codes not be sufficiently granular.

Each individual connection is responsible for both being able to generate their own ClothoData objects and well as process incoming data objects. ClothoData objects are routed throughout the system by a connection addressing scheme. A key aspect of this addressing scheme is a ClothoHub.

2.3 ClothoHub

To make the location and management of connections easier, connections are grouped and linked to ClothoHubs (#2 in Figure 2). Like connections, hubs are categorized as view, interface, connector, and function. There is one type of each of these hubs in the system. Connections belong to one hub each and belong to the hub corresponding to the type of connection they are derived from. Hubs maintain a list of all connections they are responsible for and provide information about these connections to the ClothoCore during initialization. This information includes the hub address of the connection and connection abilities. Hubs allow for not only point to point communication between connections but also can broadcast a single ClothoData object to all the connections of the hub. This is useful if an application wishes to send one piece data to multiple connections simultaneously. All hubs are connected to one ClothoCore.

2.4 ClothoCore

The ClothoCore object (#1 in Figure 2) maintains control over the hubs and (by implication) the connections in the system. The core is responsible for routing ClothoData objects to the correct hub. The core is also responsible for setting up initial connections in the system. The core has both a hub and connection addressing scheme. This system allows for the core to know both how to address a connection in a hub as well as where connections are without having to directly speak to the hub. This bypassing is useful if general system information needs to be queried or to perform common operations faster. For example, once a connection has been contacted by the core, it then can initiate a transaction back to the core in response to the sender of the data. The core can store the information about this link and therefore prevent redundant setup information to be repeatedly passed back in forth in the event that each connection wants to transfer more than one ClothoData object. This can occur as long as needed to finish the transaction. This is similar to direct memory access (DMA) transfers in computer architecture.

Because of the modular way in which Clotho has been designed, a developer wishing to use Clotho need only create a connection derived from one of the 4 basic types of connections. Once the connection has been defined, it need only be instantiated in the Clotho main file and then call the required activate method. The ClothoCore takes care of the rest. Activation of a connection associates it with the core, a specific hub, provides it a global and hub address, and runs any needed start up routines. The core is used to not only activate connections but it can also run regular start up operations, load preference data on start up, and save preference data to various files in the Clotho system.

2.5 ClothoAlgorithm

Finally, in addition to connections, Clotho also supports ClothoAlgorithms. These interact specifically with the *Clotho Algorithm Manager*. This specific connection makes user created algorithms available to the rest of the system. The user can create an extension of the ClothoAlgorithm class and simply instantiate the algorithms in the Clotho software architecture netlist and register them with the ClothoCore. This then makes the algorithm available to use through a flexible GUI. The algorithm manager also allows the algorithms access to any of the database connections available to Clotho. This can be used to look up part information or save and create new parts. ClothoAlgorithms will be touched on more in Section 3.3.

3. CLOTHO DESIGN FLOW

As the previous section illustrated Clotho's ability to cleanly separate computation, communication, and coordination, this section will illustrate how designs can be viewed functionally as well as structurally. In addition, the idea of supporting a "platform" (a unification of functional descriptions and implemented designs) is shown in Clotho's support for BioBricks [12] and the Provisional BioBrick Language (PoBoL) [1].

Clotho currently uses the architecture described to create a general design flow which entails: **1)** Connecting to a database of standardized biological parts. **2)** Associating the syntactic database fields (text) to a standardized semantic definition (PoBoL). **3)** Manipulating the part data, assembling parts, and analyzing a design. **4)** Packaging the result as a new part and distributing/saving the part back to existing databases. The following subsections will elaborate on this flow and provide some examples of the UIs in Clotho.

3.1 Data Retrieval and Semantics

In Figure 2 the *ClothomySQLConnection* is responsible for the first step in the design flow. Clotho currently supports the connection to mySQL databases. This was the second requirement of a tool as outlined in Section 1.1. The second step is to assign the fields found in the database (e.g. name,

DNA sequence, part number) to a standardized semantic definition. This allows Clotho to connect to databases organized in a wide variety of ways, containing a wide variety of information, and still perform standard operations which require a semantic meaning behind the data. For example, a user may want to view all the DNA samples in a particular plate. If the database names the plates "trays", there needs to be a way to indicate that trays == plates.

Clotho uses PoBoL to provide this semantic meaning. This standard provides a basic data model for BioBricks. BioBricks are a DNA sequence held in a circular plasmid. This standard is becoming increasing popular as a format for the distribution of biological parts. BioBricks are an example of a *platform* support by Clotho. PoBoL contains "object" and "fields". Figure 3 shows the user interface to "bind" aspects of the database to PoBoL objects and fields. Once this binding has been defined, Clotho can then perform various look ups to the database and "interpret" the database in a standard way. Clotho maintains a list of binding files allowing the users to have a unique configuration on a per database basis.

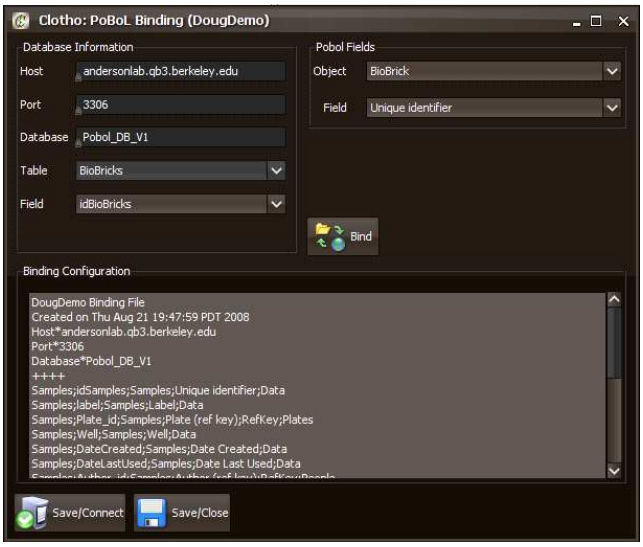

Figure 3: Clotho PoBoL Binding Manager

3.2 Multiple Views

The first requirement in Section 1.1 was support for multiple views. These are used for design and analysis (step 3). The currently available view is the *Sequence View*. Figure 4 illustrates this user interface. Using the Sequence View the user can perform protein translation, find open reading frames, highlight specific user defined features, identify restriction enzymes, view reverse complemented sequences, determine the melting point of subsequences, and the %GC content.

3.3 Computational Tools

Step 3 in the flow is aided by the Clotho Algorithm Manager shown in Figure 5. Illustrated are various operations available. The first is the ability for the user to select an algorithm from a drop down menu populated automatically by the ClothoCore during initialization. Each provides their own customized instructions. One can run the algorithm on

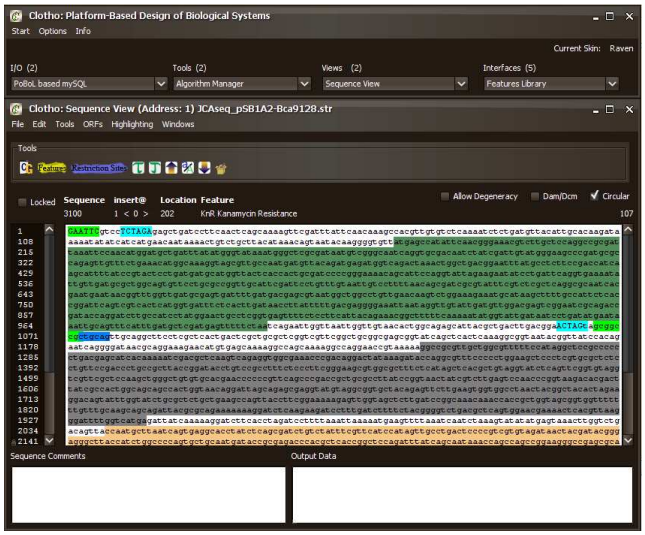

Figure 4: Clotho Sequence View

the data provided either in the text window provided or from an imported file. The user can view the results of the algorithm textually (a report file) or visually (a graph or diagram). Finally, the user can save the result of running the algorithm. Computation was the third requirement in Section 1.1 and currently Clotho includes algorithms for the assembly of BioBrick based systems.

3.4 Clotho Interfaces

Finally the completed design can then be "packaged" as part or saved in a common file format. Part packaging allows the design to be saved as a PoBoL based BioBrick. Once this process is done, it can then be exported back to an existing database for use by the worldwide synthetic biology community. Figure 6 shows an example of a part info viewer which allows the information currently associated with a part to be viewed during this packaging process.

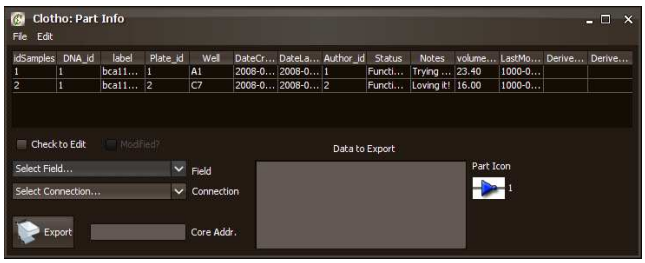

Figure 6: Clotho Parts Info

4. CONCLUSIONS

We have presented *Clotho* which adheres to a Platform-based design methodology. This creates a very structured software architecture and design flow. This design flow lends itself very nicely to the designed of standardized biological parts. This is due to the fact that it allows for flexible data syntax and semantics, the separation of the data being manipulated from the location in which it is stored, and both a top down (function driven) and bottom up (parts driven) design approach.

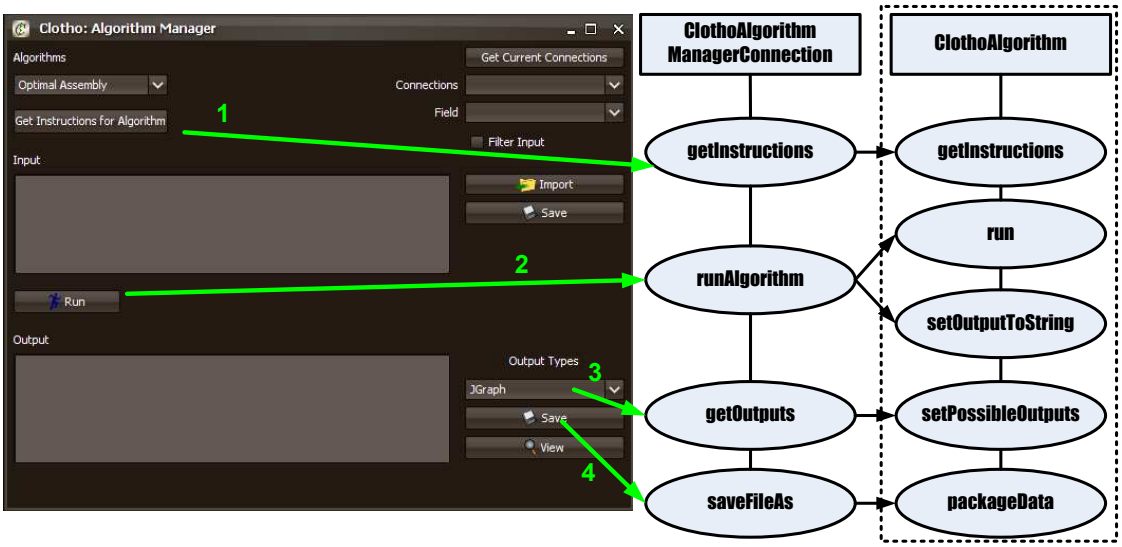

Figure 5: Clotho Algorithm Manager

Clotho is still very much in development. Currently the alpha version of Clotho is available at [6] and a beta version should be available by Fall 2008. Future work includes the inclusion of specific user interfaces based on specific PoBoL objects (e.g. DNA samples, plates, biobricks, etc), the integration of simulation, and the addition of more views. In addition, we expect to fully integrate Clotho into the larger synthetic biological community by working with such efforts such as BrickIt [9].

5. ACKNOWLEDGMENTS

The authors would like to thank Prof. Chris Voigt, Prof. J. Christopher Anderson, Prof. Alberto Sangiovanni-Vincentelli, Dr. Tom Knight, Dr. Ryan Owen, Ethan Mirsky, and Josh Kittleson for their valuable input on the paper and related discussions.

6. REFERENCES

[1] *Provisional BioBrick Language*. World Wide Web, http://www.pobol.org, 2008.

[2] S. A. Becker, A. M. Feist, M. L. Mo, G. Hannum, B. O. Palsson, and M. J. Herrgard. Quantitative prediction of cellular metabolism with constraint-based models: the cobra toolbox. *Nature protocols*, 2(3):727–738.

[3] Y. Cai, B. Hartnett, C. Gustafsson, and J. Peccoud. A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts. *Bioinformatics*, 23(20):2760–2767, October 2007.

[4] B. I. Dahiyat and S. L. Mayo. De novo protein design: fully automated sequence selection. *Science*, 278(5335):82–87, October 1997.

[5] M. W. Davis. *A Plasmid Editor*. World Wide Web, http://www.biology.utah.edu/jorgensen/wayned/ape/, 2008.

[6] D. Densmore. *Platform-Based Design of Synthetic Biological Tools*. World Wide Web, http://biocad-server.eecs.berkeley.edu/wiki, 2008.

[7] D. Endy. Foundations for engineering biology. *Nature*, 438:449–453, Nov 2005.

[8] J. Goler. BioJADE: A Design and Simulation Tool for Synthetic Biological Systems. Master's thesis, MIT, MIT Computer Science and Artificial Intelligence Laboratory, May 2004.

[9] R. Gruenberg. *BrickIt*. World Wide Web, http://brickit.wiki.sourceforge.net/, 2008.

[10] S. Hayat, K. Ostermann, L. Brusch, W. Pompe, and G. Rödel. Towards in vivo computing: quantitative analysis of an artificial gene regulatory network behaving as a rs flip-flop and simulating the system in silico. In *BIONETICS '06: Proceedings of the 1st international conference on Bio inspired models of network, information and computing systems*, page 5, New York, NY, USA, 2006. ACM.

[11] L. Jiang, E. A. Althoff, F. R. Clemente, L. Doyle, D. Rothlisberger, A. Zanghellini, J. L. Gallaher, J. L. Betker, F. Tanaka, C. F. Barbas, D. Hilvert, K. N. Houk, B. L. Stoddard, and D. Baker. De novo computational design of retro-aldol enzymes. *Science*, 319(5868):1387–1391, March 2008.

[12] T. F. K. Jr. Idempotent vector design for standard assembly of biobricks. Technical report, MIT AI Lab, 2002.

[13] T. Knight. *Computer Aided Design and Construction of Living Systems*. Synthetic Biology Conference 3.0, ETH Zurich, Switzerland, 2007.

[14] MIT. *Registry of Standard Biological Parts*. World Wide Web, http://parts.mit.edu, 2008.

[15] A. Pertsemlidis and J. W. Fondon. Having a blast with bioinformatics (and avoiding blastphemy). *Genome Biol*, 2(10), 2001.

[16] A. Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes*, February 2002.

[17] A. Villalobos, J. E. Ness, C. Gustafsson, J. Minshull, and S. Govindarajan. Gene designer: a synthetic biology tool for constructing artificial dna segments. *BMC Bioinformatics*, 7:285, June 2006.

[18] C. A. Voigt. Genetic parts to program bacteria. *Curr Opin Biotechnol*, 17(5):548–57, Oct 2006.

[19] R. Weiss, G. E. Homsy, and T. Knight. Towards in vivo digital circuits. *DIMACS Workshop on Evolution as Computation*, 1:1–18, January 1999.

[20] M. Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic acids research*, 31(13):3406–3415.