Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2004-036
AITR-2004-003

May 28, 2004

BioJADE: A Design and Simulation Tool for Synthetic Biological Systems

Jonathan A. Goler

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

BioJADE: A Design and Simulation Tool for Synthetic Biological Systems

by

Jonathan Ari Goler

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of

Master of Engineering

and

Bachelor of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004

© Massachusetts Institute of Technology 2004. All rights reserved.

Certified by: Thomas F. Knight Jr
Senior Research Scientist
Thesis Supervisor

Accepted by: Arthur C. Smith
Chairman, Department Committee on Graduate Students

BioJADE: A Design and Simulation Tool for Synthetic Biological Systems

by

Jonathan Ari Goler

Submitted to the Department of Electrical Engineering and Computer Science on May 18, 2004,
in partial fulfillment of the requirements for the degrees of Master of Engineering
and
Bachelor of Science in Electrical Engineering and Computer Science

Abstract

The next generations of both biological engineering and computer engineering demand that control be exerted at the molecular level. Creating, characterizing and controlling synthetic biological systems may provide us with the ability to build cells that are capable of a plethora of activities, from computation to synthesizing nanostructures. To develop these systems, we must have a set of tools not only for synthesizing systems, but also designing and simulating them. The BioJADE project provides a comprehensive, extensible design and simulation platform for synthetic biology. BioJADE is a graphical design tool built in Java, utilizing a database back end, and supports a range of simulations using an XML communication protocol. BioJADE currently supports a library of over 100 parts with which it can compile designs into actual DNA, and then generate synthesis instructions to build the physical parts. The BioJADE project contributes several tools to Synthetic Biology. BioJADE in itself is a powerful tool for synthetic biology designers. Additionally, we developed and now make use of a centralized BioBricks repository, which enables the sharing of BioBrick components between researchers, and vastly reduces the barriers to entry for aspiring Synthetic Biologists.

Thesis Supervisor: Thomas F. Knight Jr
Title: Senior Research Scientist

Acknowledgments

I would first like to thank my thesis advisor, Professor Thomas F. Knight Jr. for getting me into Synthetic Biology, and showing me the ropes, and supporting my various endeavors in the year and a half that I have worked in his lab.

Randy Rettberg helped me tremendously with learning BioBrick design, and with the development of the BioBricks Registry. Austin Che assisted me in bootstrapping the registry and for writing the code that the basic registry was based on. Reshma Shetty and Drew Endy provided a great deal of insight and help debugging and enhancing the repository. Felice Frankel helped with the design and user interface elements of BioJADE and helped me make it into an elegant and usable tool. Additionally, I thank Gerald Sussman for 6.978 where I first met TK, and worked out some preliminary interesting applications of SynthBio. In addition, I thank Professor Patrick H. Winston for his insights on writing and communicating effectively. His teaching is perhaps the most effective and compelling at MIT.

Naturally, many thanks go out to my parents, and grandparents for their support for my work here. My friends Dana Spiegel, Jose Gonzalez, Ariela Ressler, Stefan Bewley, Tomer Posner and Zofia Gajdos for always being there. MaryEllen McLaughlin for believing in me to give me a chance, and molding me into something a little less flimsy than when she began, and Professor Josef Kurtz for always challenging me to do more and better. And of course, the whole MIT Swimming and Diving team for all of your support through the last five years.

Contents

1 Introduction **8**
- 1.1 Differences in Biology and Silicon . . . 8
- 1.2 System Overview . . . 9
  - 1.2.1 Parts Repository . . . 9
  - 1.2.2 BioJADE . . . 9

2 Background **10**
- 2.1 Basic DNA Functionality . . . 10
- 2.2 Initial System Inspiration . . . 11
- 2.3 The BioBricks Standard . . . 11
  - 2.3.1 BioBrick Definition . . . 11
  - 2.3.2 Assembly . . . 12
  - 2.3.3 Assembly++ . . . 14
  - 2.3.4 Characterizing Parts . . . 14
- 2.4 Biological Limitations . . . 15
- 2.5 Potential Applications . . . 15

3 System Design **17**
- 3.1 Basic Architecture . . . 17
- 3.2 Aspects . . . 17
- 3.3 Part Repositories . . . 17
- 3.4 The Database . . . 18
  - 3.4.1 Implementation . . . 18
  - 3.4.2 Access Methods . . . 18

4 Aspects **20**
- 4.1 The Aspects . . . 20
- 4.2 The Aspects in the Basic BioJADE package . . . 20
- 4.3 Schematic . . . 21
- 4.4 Functional Network Aspect . . . 24
  - 4.4.1 Adding Species and Reactions . . . 25
  - 4.4.2 Refining Auto Part Selections . . . 26
- 4.5 Simulation Aspect . . . 27
- 4.6 Icon Aspect . . . 28
  - 4.6.1 Dynamic Terminals . . . 29

4.7 DNA Aspect 29

5 Simulations **31**
5.1 D-FLUX 31
5.1.1 The Architecture 31
5.2 Standard Simulators 33
5.2.1 Stochastirator 33
5.2.2 Tabasco 33
5.2.3 Electrical Models 33
5.2.4 Logic and State Machines 34

6 Modeling Biology and Devices **37**
6.1 Basic Models 37
6.1.1 General Treatment of Cellular Species 37
6.1.2 Ribosome Binding Sites 37
6.1.3 Coding Regions 38
6.1.4 Termination 38
6.1.5 Promoter Binding and Repression/Expression Control 38
6.2 Device Models 38
6.2.1 Quad Part Inverters 39

7 Compiler Issues and Optimizations **40**
7.1 Logic Reduction 40
7.2 Managing Biological Uncertainty 40
7.3 Possible Optimizations 41

8 Future Work **42**
8.1 Potential Dangers 42
8.1.1 Friendly Warnings Help Avoid Accidents 42
8.1.2 Open and Secret 42
8.1.3 Education 43
8.1.4 Monitoring 43

9 BioJADE's Contributions **44**

A BioBricks SQL Data Model **45**

List of Figures

2.1 RNA Polymerases arrive and produce the mRNA (messenger RNA) transcript in the 5'-3' direction along the template DNA. 11
2.2 Standard BioBrick Restriction Sites. These sites must not be included in any part, otherwise the assembly protocol will fail. 12
2.3 A base part with the BioBrick prefix and suffix. The XbaI and PstI sites are used to cut the part when it is used in an assembly. There are a few extra bases inserted to separate the cut sites. [26] 12
2.4 The base plasmid provides matching XbaI and SpeI sites for inserting the part into the plasmid. 13
2.5 The XbaI and PstI sites on the insert and plasmid are both cut, and the overlapping ends are ligated, producing the part within the vector. 13
2.6 An overview of the Standard Assembly Process [14, 26]. This diagram shows the idempotent assembly procedure where the output from one cycle is an input into the next. 13

3.1 The Parts Database is a star-type schema for part data. In addition, support for users, organizations, part assembly and inventory management is provided. 19

4.1 Schematic Design Mode. This figure shows the initial state of the schematic aspect. We have selected the not part prototype from the palette on the left. 21
4.2 Schematic Design Mode. This figure shows three blank not gates being stamped out on the design. Note the context-sensitive help in the lower right corner. 22
4.3 Schematic Design Mode. This figure shows the three gates being wired together. The terminals change color from red to green when they are connected to other components. 23
4.4 Schematic Design Mode. This figure shows the completed schematic of a repressilator. 24
4.5 The Functional Network Representation shows the same repressilator, but in the form of stretches of DNA. System features such as ribosome binding sites can be selected from the functional network aspect. 25
4.6 The Functional Network mode also enables the designer to specify other species and reactions that affect system performance. This figure shows a degradation reaction involving one of the signaling molecules. 26
4.7 Editing promoter details such as the cooperativity in the Functional Network mode. 27
4.8 The same repressilator simulated with Stochastirator. The figure shows the oscillation of all three signaling molecules. 28

4.9 Drawing a dynamic terminal onto the icon representation for a wired-NOR gate. The red terminal is a normal terminal, and the green arc on the left is the dynamic terminal area. 29
4.10 Examining the repressilator in the DNA Aspect. The DNA Aspect displays sequence data along with a graphical representation highlighting the annotation of the DNA. . 30

5.1 DFLUX gives designers shared access to powerful computational resources to execute simulations. 35
5.2 A Tabasco simulation visualization that shows(in quicktime format) the generation of mRNAs and their associated proteins. The levels of each species is shown at the upper left corner. The DNA is shown as a line, with green blocks representing transcription starts, and yellow blocks representing termination. The blue or green ovals on top of the DNA are the polymerases, which move along the DNA. Finally, below the DNA are lines and a number representing mRNA transcripts produced by the DNA blocks above them. 36

Chapter 1

Introduction

The development of synthetic biological systems will be driven in large part by our ability to design and analyze complicated systems built out of many components. In the development of microprocessors, the introduction of VLSI design and simulation tools greatly improved the capabilities of chip designers. While synthetic biology is at the point where such tools would be highly beneficial, none have yet been developed. BioJADE is based on JADE (JAva Design Environment), which was written by Christopher Terman, James Knight and Milan Minsky of MIT LCS [25]. BioJADE differs from JADE in several respects, and incorporates a number of different data structures and data access methods that are necessary due to the use of biology rather than physical substrates.

1.1 Differences in Biology and Silicon

There are a variety of differences between Silicon and Biology that make existing representations of electric circuits insufficient for describing genetic networks or circuits. The most important of these differences is that the notion of a wire as a localized object in physical space is completely different in biology. The basic units of control, such as the components comprising inverters, are not necessarily in the same place. For example, a basic Lac inverter could be made, using input from another component to produce a protein that represses a promoter on another part of the genome. The wires in our design are constraints that define what components must be in sequential order on the DNA. In addition, the RNA, proteins and small molecule signals constrain the choices of components because those species interact with each other as well as promoter regions of DNA. One significant problem with thinking about wires in this manner is that for components to be distinct, we need distinct proteins/RNAs. Without distinct molecules, our signals will be garbled.

The second primary difference is the speed of operation. Silicon allows us to perform complex computations very quickly due to the nature of electrical signals. Electrical signals propagate near the speed of light (and new photonic systems do operate at the speed of light). Biological signals, particularly prokaryotic ones are molecules, some small like sugars and some large such as proteins or DNA. Most of these signals propagate as they diffuse through the organism. In addition, by using transcriptional regulation to build logic functions, speed is limited by the speed of transcription and translation. For quicker functions, Muir [20] showed how to combine a split protein, which could be non-functional when apart, and functional when combined. The folding process is far faster than transcriptional regulation.

1.2 System Overview

1.2.1 Parts Repository

There are two primary components of the BioJADE system. The first component is the BioBricks repository system, the first instance of which is the MIT Registry for Standard Biological Parts. The repository system is a database driven website that contains all of the BioBricks that are currently either designed or available for use. The Repository contains not only the part definitions, including annotated sequence data, but also interfaces for editing and viewing auxiliary data. The database allows designers to edit and view associated papers, figures, and behavioral parameters.

The Repository system is implemented as a Data Model and Perl interface for the web access. Perl was chosen because of legacy applications, including a prototype Parts Repository written by Austin Che [2]. Che's prototype included the basic data, but was backed by flat files. The Repository system now uses a relational detabase (RDBMS), in this case, Oracle 9i.

1.2.2 BioJADE

BioJADE is the design and simulation tool. With BioJADE, a user can connect to either a partial flat file parts repository in XML format, or a full RDBMS. The designer then has the ability to design new parts or analyze the behavior of parts in his repositories. In the design mode, the designer can build a system by stamping out parts(either prototypes or specific implementations), and connecting them using 'wires'. The designer then 'compiles' the design into more specified designs, such as DNA or functional networks, that expose the underlying biology.

In addition to a system level design, the BioJADE system also generates annotated DNA sequences in plasmid format from the design. Designers can arrange the parts along the plasmid, and modify their interactions. In addition, the designer can view and modify more complicated functional behavior of component interactions such as anti-sense reactions, small molecule transduction and RNA/Protein degredation.

Once a system is designed and its components are specified, the next step is to simulate their combined behavior. Just as in JADE or SPICE, there are a variety of simulations that can be used. These simulators range from a system level simulator that tests the logic without regard to time, to simulators that work down to the level of proteins binding and moving polymerases along the DNA. The simulation interface yields extreme compatibility and flexibility. The interface permits the use of tools written in different languages, on different platforms. The interface wraps existing tools and makes them available to BioJADE designers. In addition, the interface allows distributed simulation, so that designers can harness much more powerful computers to simulate their design.

Finally, the designer submits the design to a BioBricks parts repository for other users to use in their designs. Once on the site, the author executes an assembler and generate assembly instructions for building the parts, making use of the entire library of existing parts.

Chapter 2

Background

This section deals with the fundamentals of synthetic biology as it applies to the problem of building synthetic biological computational units. This section explains how and why synthetic biological systems work, as well as their benefits and limitations. Finally, this section explores a few examples of how synthetic biological systems might be harnessed to improve our lives.

2.1 Basic DNA Functionality

DNA is the basis of how our cells work, and how we develop as human beings. DNA, along with RNA, proteins and small molecules, is the basis for every living thing, from a prion to a giant squid. The sequence of DNA, the A,T,C, and G's, can determine what proteins we produce, and whether or not they are produced correctly. The sequence itself does not entirely determine the resultant proteins due to alternate splicing and these sequences do not completely determine what happens to us, because DNA is only a part of a more complicated system.

When a protein known as a DNA polymerase attaches itself to part of the DNA, it then moves along the DNA, building an RNA transcript. This process is called transcription. Different polymerases exist in different cells, and they each work with differing speeds and accuracies. Transcription is halted when a polymerase encounters a terminator. Terminators are usually stem loop secondary structures in the DNA that literally cause the polymerase to fall off the DNA. Terminators are not perfect, and some BioBrick terminators are actually two in series, so that if one fails, the other is likely to succeed.

The resulting mRNA, which then is free to interact with ribosomes, might then be bound by a ribosome. The part of the mRNA to which a ribosome binds to is known as the ribosome Binding Site (RBS). There are a variety of RBSes, and they have differing binding affinities. Thus, we can control the activity of the ribosome binding through selection of the RBS. Once attached, the ribosome builds a protein one amino acid at a time by matching tRNAs to each codon (3 base sequence), until it reaches a stop codon. Proteins then fold and perform their function, which includes binding to regulatory regions of other DNA parts, or florescing, etc. Proteins are degraded regularly by proteases, and this degredation can be quickened by adding LVA/LAA tails to the proteins that help mark them for 'garbage collection'.

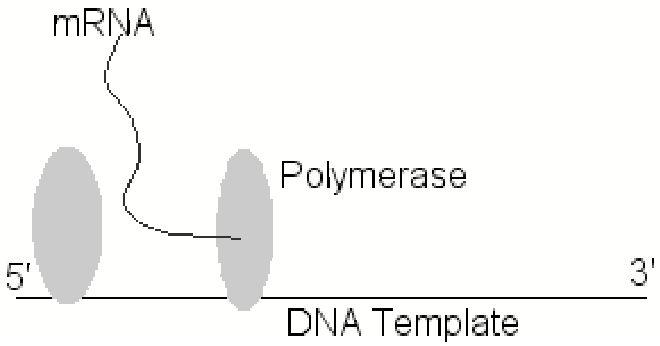

Fig. 2.1: RNA Polymerases arrive and produce the mRNA (messenger RNA) transcript in the 5'-3' direction along the template DNA.

2.2 Initial System Inspiration

A few sample systems have been instrumental in our understanding of gene regulation and cell-cell signaling. One such system, the Lux system, has been studied extensively by Greenberg et al. [6, 7, 10, 15, 3]. These systems were then used by Knight and Weiss to implement communication and logic gates [29, 5].

Other systems, such as the periplasmic binding proteins investigated and exploited by Hellinga, can be modified computationally to respond to non-native ligands [17, 30] , giving us a greater variety of inputs for synthetic systems.

By integrating existing systems with new parts, we can expand our ability to build interesting biological systems. The existing templates we have of control systems in nature give us hints on what is possible, and by altering the existing systems, we learn how to break them. Vilar and Leibler [11, 27, 28] have used synthetic biological systems to study noise resistance in oscillatory networks, and incorporate more robust measurements into models. These synthetic systems are not only useful for engineering systems, but also understanding the mechanisms for natural system performance.

2.3 The BioBricks Standard

The BioBricks Standard, developed by Thomas F. Knight Jr. [14], provides a modular way of builiding and composing parts made out of DNA. The Standard consists of specific prefixes and suffixes that enable the composition of parts in a standard way. There are currently a few ways of composing parts, the Idempotent Standard Assembly method and the Triple Antibiotic Assembly [23] known as the 3A method.

2.3.1 BioBrick Definition

The BioBricks parts are comprised of their contents, and standard biobricks ends. The contents are arbitrary, with the caveat that they may not contain any of the BioBricks restriction sites (EcoRI

XbaI, SpeI, NotI and PstI). These sites can be mutated out through manual edits of the sequence. In most cases, changes can be made that do not affect the system due to the redundancy in the codon specificity.

The prefix for a part is a cctt +XbaI + g site, and the suffix is a t + SpeI + a + NotI + PstI + cctt. The restriction sites enable the idempotent construction, while the extra bases help to separate restriction sites and allow the enzymes some overhang at the ends.

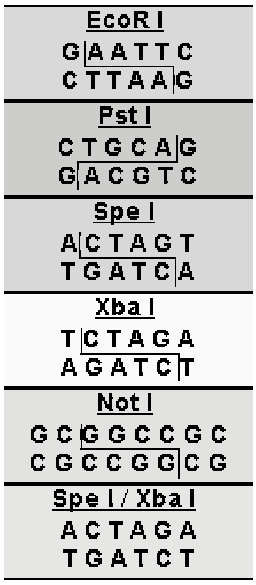

Fig. 2.2: Standard BioBrick Restriction Sites. These sites must not be included in any part, otherwise the assembly protocol will fail.

Parts are grown and stored in plasmid vectors. These vectors are circular pieces of DNA that bacteria exchange with each other. There is a set of standard BioBrick plasmids that are used to build and grow parts. They contain sites that enable the introduction of the BioBrick parts.

Fig. 2.3: A base part with the BioBrick prefix and suffix. The XbaI and PstI sites are used to cut the part when it is used in an assembly. There are a few extra bases inserted to separate the cut sites. [26]

2.3.2 Assembly

To put parts together, we cut the parts with the appropriate restriction enzymes and then ligate the cut products together. In this example, we place one part on the beginning of another part.

Fig. 2.4: The base plasmid provides matching XbaI and SpeI sites for inserting the part into the plasmid.

Fig. 2.5: The XbaI and PstI sites on the insert and plasmid are both cut, and the overlapping ends are ligated, producing the part within the vector.

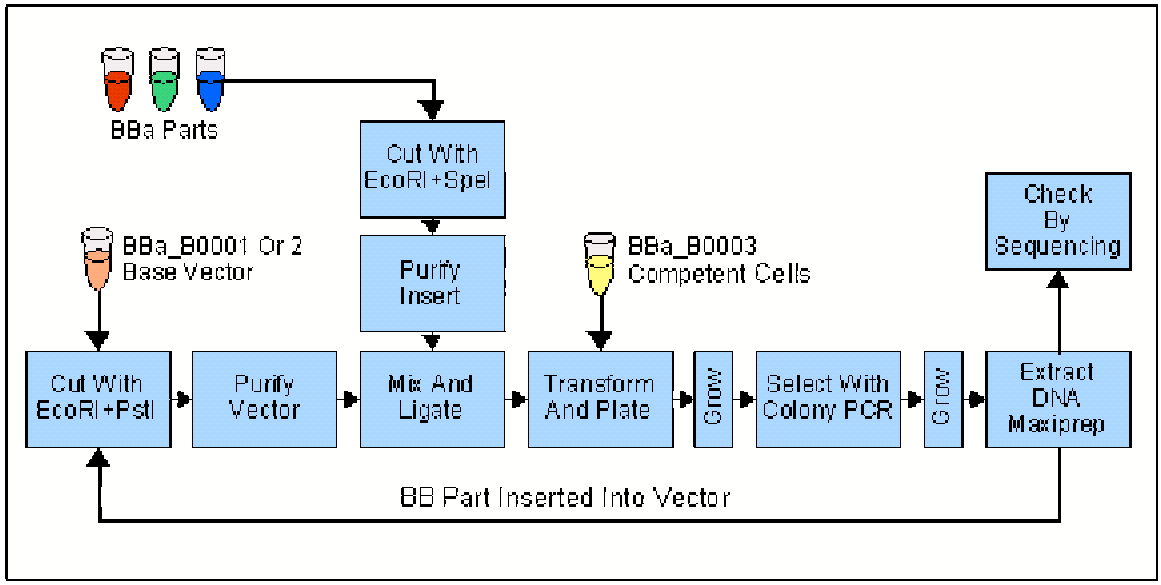

Fig. 2.6: An overview of the Standard Assembly Process [14, 26]. This diagram shows the idempotent assembly procedure where the output from one cycle is an input into the next.

The key to the idempotent assembly process is the convenient nature of the XbaI and SpeI sites. The overhang is identical, but the ends of each cut site are different, so the combined site can be ligated, but not cut. Yet, the ends remain the same, and thus, we can compose parts again and again.

For the component that we are prepending, we take the DNA and cut it with EcoRI, and XbaI. The component we are inserting is cut with EcoRI and SpeI. These components are then ligated together and the result is the combined part, with the same ends, but with an uncuttable mixed SpeI/XbaI site where they were ligated.

The Triple Antibiotic (3A) assembly technique is an extremely useful way to put parts together. We utilize biology's selection mechanisms to do the hard work of ensuring that the resulting part is correct.

The procecure is not discussed here, but see the reference on 3A [23] in the bibiography for more detailed information about the 3A assembly process.

2.3.3 Assembly++

The BioBricks Alpha Specification[16], is one of many possible assembly techniques. There are a variety of possible assembly techniques that would require modifying the ends of the parts, but the basic essence of the parts should remain the same. In addition, improved assembly techniques, such as Austin Che's BioBricks++ [1], offer other assembly options.

Even as new assembly techniques and part ends are developed, the notion of modularity and its advantages continue to apply to the development of systems. Therefore, BioJADE is not vitally concerned with assembly technique until the actual design is compiled into the assembly instructions for a particular assembly process.

2.3.4 Characterizing Parts

Part of having a standard includes having a way of measuring system behavior in a repeatable, and standard way. We want to know several parameters, such as minimum and maximum signal levels, transfer functions, transcription load on the organism, crosstalk and other incompatibilities. With a simple example, like an inverter, which is composed of a coding region, the associated mRNA and proteins, and the regulatory region it binds to already provide a complicated system.

Must we know how fast each of these components is produced, degraded or diffused? Or would it suffice to know how fast and how well an input signal gets mapped to the output signal? The "signal in, signal out" form would be desirable. We have defined two terms: TIPS: transcription initiations per second, and PAR: polymerase arrival rate, to indicate signal level on both input and output. We can measure this characteristic directly using the PARMASAN method of Austin Che[2]. This method allows us to directly observe the polymerase arrival rate on parts.

Nevertheless, the diffusion and degradation properties of these signals are important in determining timing characteristics of the system, including the rise time, fall time, and propagation delays that are vital in system design. Thus, we must measure all of these values.

Finally, these values are not constant across conditions and organisms. Thus, we will have to develop tables of values for the parts, including the organism used and the conditions in which the measurements are conducted. Once a significant number of these values have been determined, more generalized behavior can be better estimated.

2.4 Biological Limitations

Biology, as useful as it is, has significant limitations. For one thing, organisms die, for myriad reasons. In addition, we are hampered by the inherently slow and sometimes very noisy processes of diffusion and transcription/translation. As a result, logic functions(such as super fast arithmetic) might not be ideal applications for computation using synthetic biology, except in cases where massive parallelism is usable, because it is very easy to grow a big vat of biological processors.

Biology also offers us the benefits and curses of evolution. We might have a part that we can use mutation to improve, or we might have a part whose behavior is undesirably modified or removed. In several instances of part synthesis of some of the first biological blinker systems [1], there are systems that are detrimental to the host in their intermediate form, causing too much load on the chassis, and are removed or modified by evolution.

So we must find a way to understand how our parts interact with the existing biology of the cells. The efforts of Knight et al, to create a minimal *Mesoplasma florum* will be useful in removing many interactions and load from the cell. A simple cell, such as the minimal *Me florum*, gives synthetic biologists more freedom in their designs.

2.5 Potential Applications

Synthetic biology has the potential to greatly affect our ability to control and sense the world around us. There are a variety of applications of synthetic biology that have the potential for great benefit, but also for great peril.

At this point, the organisms we create are so weak and stressed that breathing on them the wrong way kills them. However, this feature is partially by design. Nothing we create has much potential to cause problems in the greater world.

However, in the near future, we expect to be able to develop systems that are capable of detecting minute quantities of chemical and biological signals, such as anthrax, poison gas, etc. These sensors would have to be able to react and survive in the external environment, at least for a limited amount of time.

There are numerous applications, the general groups of which are listed below.

- Energy Production and Storage
- New Devices and Assembly (at the nanoscale)
- Molecular Medical Devices
- Bioreactors
- Programmable Devices and Control Logic
- Smart Materials
- Sensors
- Complex Assembly (large functional organisms like self-assembling bio-houses)

[1] In January 2003, a group of MIT students learned the basics of synthetic biology, and designed 'blinkers' that oscillated emitting light. The systems were still being synthesized as of Fall 2003 due to difficulties encountered during synthesis

- Terraforming

In most cases, these applications require control logic within the cells, and our standard parts give us a modular, and standard way way to achieve this control logic. In addition, components such as sensing proteins are also useful and introduced through synthetic biology.

Chapter 3

System Design

This chapter provides a detailed description of the BioJADE system architecture, and discusses the design choices and considerations in building the system. For a user who is merely looking to use the system, this chapter will not be central. However, this chapter should provide some insights into designing a complicated system with the need to merge conventions between Computer Science, Electrical Engineering, and Biology.

3.1 Basic Architecture

BioJADE is built on top of the BioBricks part representation model. This model includes the design and specification of part features, behaviors and graphical representations. The system requires one or more repositories for storage and publication of part data.

The system keeps a cache of parts from the data stores and caches changes until they are committed to the database itself. This cache enables fast response time for the user interface. The basic architecture is derived from Jade[25]. However, we use significantly different data structures to handle the differences between silicon and biological substrates.

3.2 Aspects

Each part has several Aspects, which encapsulate a way of viewing and interacting with a part. Our model of Aspects was derived from Chris Terman's in Jade. When BioJADE is started, it reads a list of available aspects from the configuration file. These Aspects are displayed for the user, and while a part is selected, a user may switch between aspects. To facilitate easy navigation, the Aspect abstract class supports saving data in memory for each part so that changes can impact the current workspace without being committed to the data store.

3.3 Part Repositories

I built in two types of storage options for BioJADE: JDBC/Oracle and XML. To accommodate the two different types of access, I built the interface "BBDataStore" which both BioBricksDatabase-Client(BBDBC) and BioBrickXMLClient(BBXML) implement. BBDBC supports connections to

an Oracle or MySQL database that contains the BioBrick Data Model. BBXML supports connections to an XML Database on a file system or across HTTP. Since primary development of the system relied on the concurrent development of the MIT Registry of Standard Biological Parts, the vast majority of testing was done on BBDBC.

Should new forms of storage become necessary, a user might simply create a new implementation of BBDataStore and install it into the BioJADE packages.

3.4 The Database

3.4.1 Implementation

We implemented the Parts Database on an Oracle 9 RDBMS. The database supports not only the full description of parts, but also tracks stocks of parts that are held at various labs, and associated files such as papers, data charts, or diagrams. The main details for the parts are held in the parts table, while ancillary information and information that would be represented as an array or vector is held in a variety of auxiliary tables. Figure 3.1 describes the organization of the Parts Schema. A complete listing of the table definitions is provided as Appendix A. In addition, the data model has been ported to MySQL for those organizations that do not have a license for Oracle.

This star-type schema permits the simple addition of additional fields should they be necessary. Oracle was chosen as the database for several reasons, including its robustness, powerful Large Object (LOB) support, and the author's familiarity with its operation. It has not escaped our notice that Oracle is both extremely expensive and unwieldy to install for many users. MySQL is a free, viable option for an alternative for the database back-end. We have provided a version of the BioBricks schema that works with MySQL.

3.4.2 Access Methods

JDBC

A JDBC client can access the Parts database remotely over the network. To connect, the client needs the Oracle JDBC driver (provided with the source/binary distributions), along with the host name, database instance name, and authentication information. This data is also included with the distribution. The client relies on the thin implementation of Oracle JDBC, and thus does not require the installation of the Oracle Client Interface(OCI).

The MySQL client requires the MySQL JDBC connection classes, which is freely available from the MySQL website.

Perl/DBI

We use Perl DBI to access the database for the purpose of implementing the web interface to the Parts Database. This method requires installing Perl, DBI and DBM::Oracle modules, as well as the Oracle Client. In the interest of conserving database resources, all web interfaces should utilize cached connection(DB Handles) and Prepared Statements. In addition, prepared statements not only improve performance, but they enhance security through parameter binding, so that there is no chance of introducing hostile code into executed SQL.

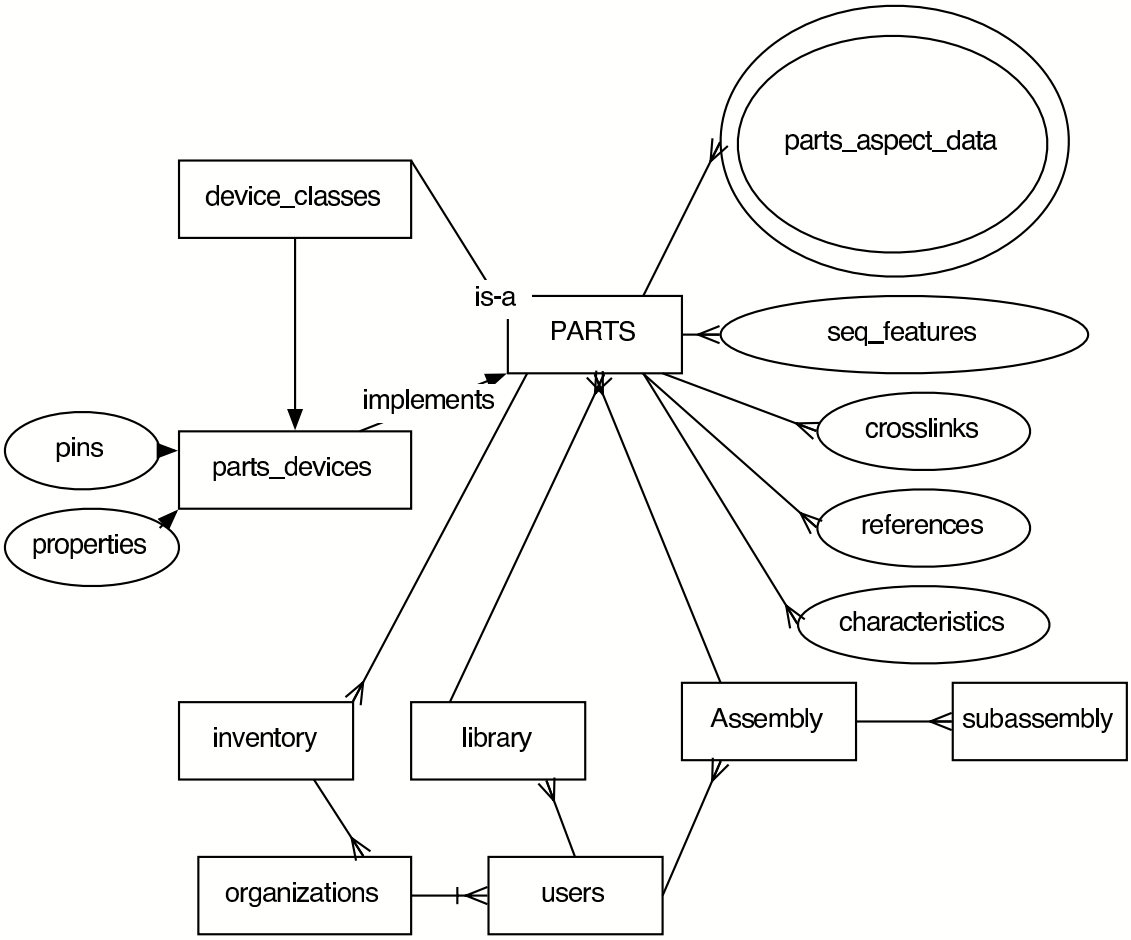

Fig. 3.1: The Parts Database is a star-type schema for part data. In addition, support for users, organizations, part assembly and inventory management is provided.

Chapter 4

Aspects

The Aspects define several different ways of viewing and editing parts. They cover the gamut of specifiying and twiddling individual base pairs of DNA, to importing a pretty picture to represent the parts as they get incorporated into new and larger parts. It is useful to go through the list of aspects as one would go about designing parts. Therefore, the majority of the discussion focuses on the design stages of a part, as a user might go through them.

4.1 The Aspects

Each part has several Aspects, which encapsulate a way of viewing and interacting with a part. Our model of Aspects was derived from Chris Terman's model in JADE. When BioJADE is started, it reads a list of available aspects from the configuration file. The list of available aspects is displayed in the toolbar, so that a designer may switch between them at any time.

The Aspect abstract class essentially handles the loading and saving of aspect data to its data store. Because each aspect has its data represented differently, the individual editors handle actually storing the state to the aspect, which in turn renders it into the data store's XML format.

In addition, the Aspects keep track of the current state of all opened BioBricks, so that if you make a change in one part, and then switch to another part, the changes are not lost. This temporary storage is merely a hash table keyed on the BioBrick. This representation is very useful for making changes to parts while not committing them to the database.

4.2 The Aspects in the Basic BioJADE package

Schematic Supports drag and drop circuit design of more complicated BioBricks. The Schematic Aspect permits the user to lay out generic devices and then, with a mouse click, 'compile' the design into its actual component parts, along with the associated assembly instructions.

Icon Permits the user to upload a PNG/GIF/JPEG file to display for the part, and provides a tool for marking the terminals of the device so that it can be connected to others in the Schematic Mode.

DNA Displays the annotated DNA Sequence.

Functional Network Shows a more detailed Biological-Type view of the system, which permits the user to see more rate constants and tune the design.

Simulation Interfaces to all of the available simulation tools, as well as the status of in-progress simulation. (see Simulation interface)

4.3 Schematic

The Schematic Design mode enables users to lay out circuits made out of genetic components just as they would a circuit made of silicon. The Design mode present a toolbar that contains available libraries of parts to stamp on the design. In addition, a toolbar containing basic functionality such as cut, copy, paste and buttons for drawing wires and stamping parts is docked to the top of the workspace. The rest of the workspace is dedicated to the design itself. Users can scroll and zoom the workspace to focus on different aspects of the design.

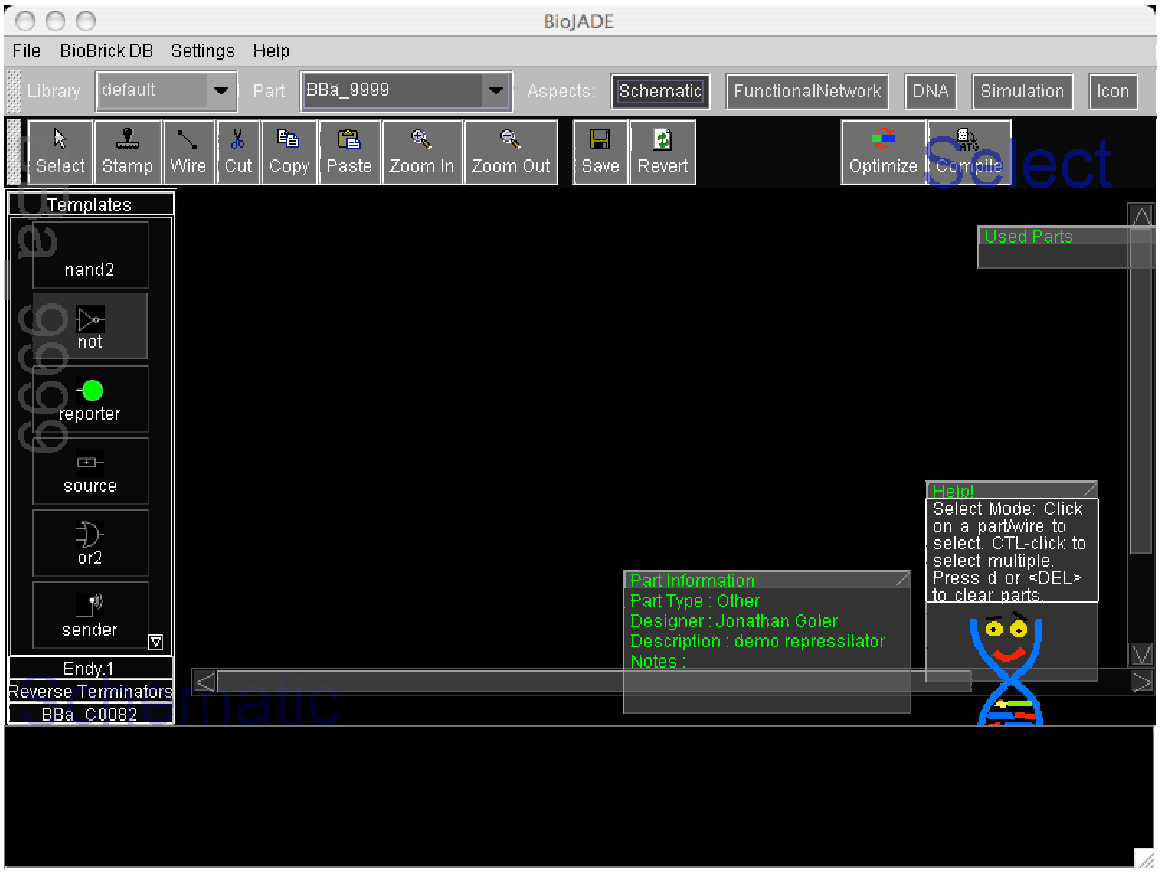

Figure 4.1: Schematic Design Mode. This figure shows the initial state of the schematic aspect. We have selected the not part prototype from the palette on the left.

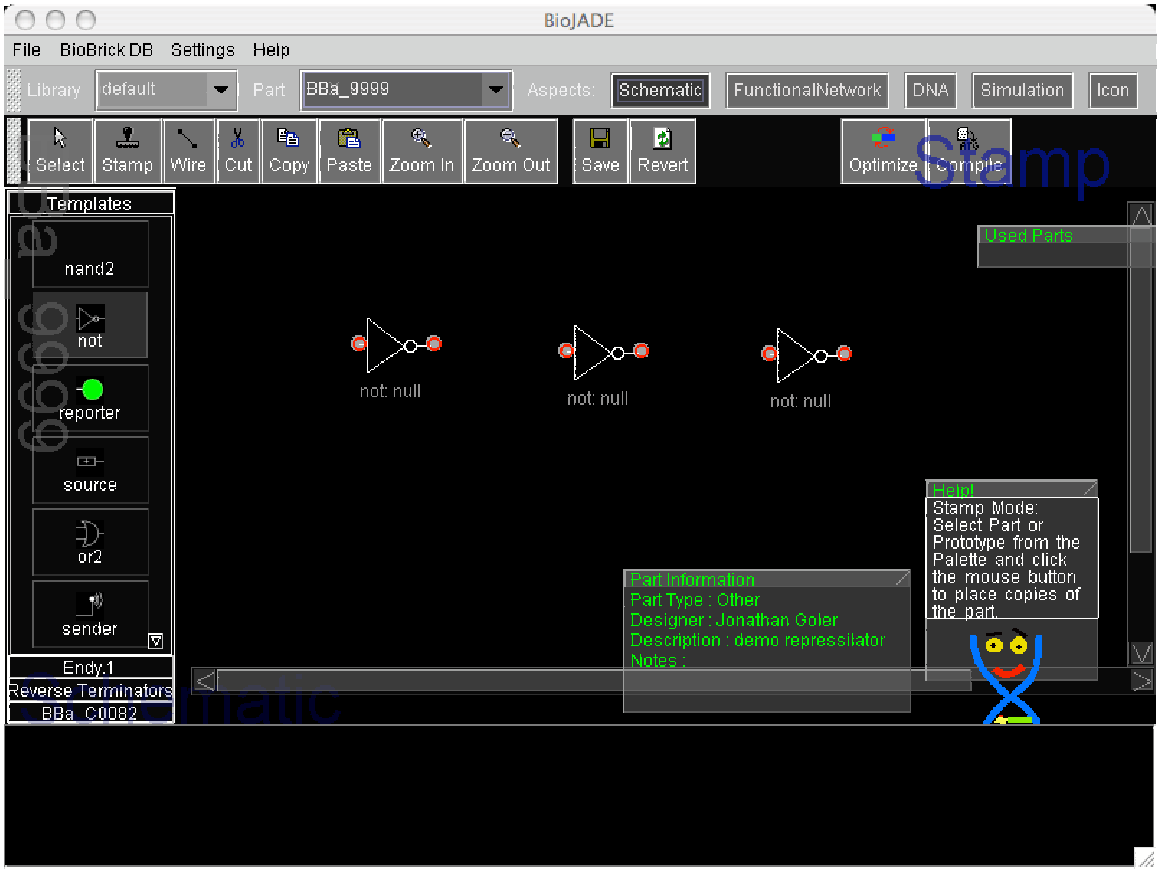

Figure 4.2: Schematic Design Mode. This figure shows three blank not gates being stamped out on the design. Note the context-sensitive help in the lower right corner.

Designers build circuits by selecting a part or prototype from the part palette on the left, and selecting the stamp button on the toolbar. The designer then stamps down the part onto the design. The user can then click on the Select button to revert to being able to merely select objects already on the design and move them around to wherever they wish.

The designer then selects the Wire tool and draws wires between the components. The first method for drawing wires is to select the starting terminal and dragging the wire to the ending terminal. The wires conveniently snap to the terminals of devices and the terminals of other wires that fall within 10 pixels of their own terminals. In addition, wire drawing is very similar to the way it is done in classic circuit drawing programs. When a wire is initiated(Alt-click), the user is presented with a draggable wire L, and can set the waypoints as she draws. The user is able to flip the wire from going horizontal-vertical to vertical-horizontal and vice-versa by typing the f key. Additionally, the waypoints are set using the s key. The wire is completed by clicking again with the mouse. Clicking with the ALT key held down will set another waypoint. In this manner, the user can use the right hand to draw and the left hand to further modify the line. Finally, the user

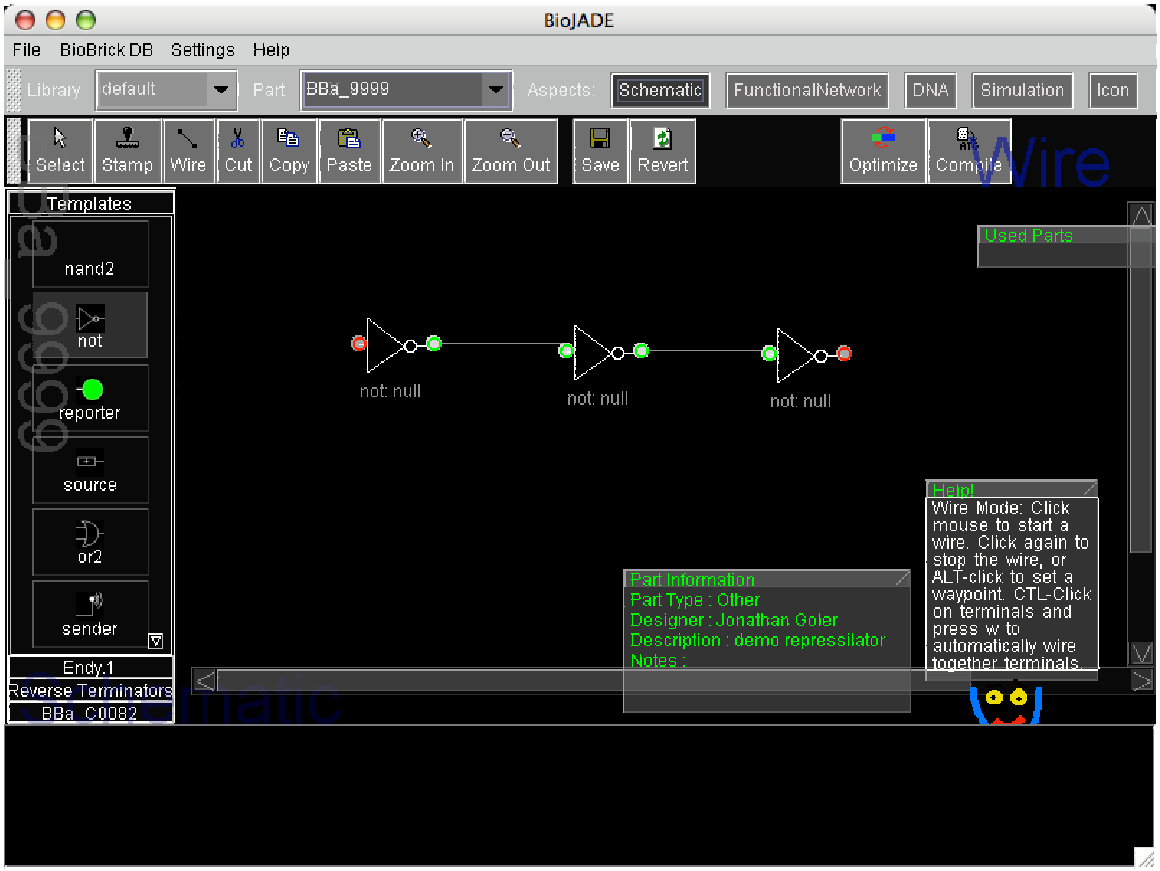

Figure 4.3: Schematic Design Mode. This figure shows the three gates being wired together. The terminals change color from red to green when they are connected to other components.

can press the "s" key to toggle a straight line segment. In addition, the wire drawing mode allows users to CTL-click on a number of terminals, and hit the 'w' key to connect them all of the selected terminals with wires.

Once components and wires are laid out, the user can make further adjustments to the design by dragging components to different locations. If a component moves, all connected wires move to accommodate the new position. The blank part templates that are laid out can be assigned specific BioBricks. The designer simply ALT-clicks on the part and selects the BioBrick from the list of matching BioBricks. Once assigned by the designer, those parts are exempted from the compilation optimization process. The user can then save the design to the database by clicking the Save button on the toolbar. The user can also set basic properties of the design, such as a description, design notes or the target host organism.

After laying out the basic design, the designer clicks the Compile button, selects the appropriate optimizations, and compiles the design into its BioBrick components. Should the compiler run out of assignable parts, it will report the problem to the user. The user could then develop or find

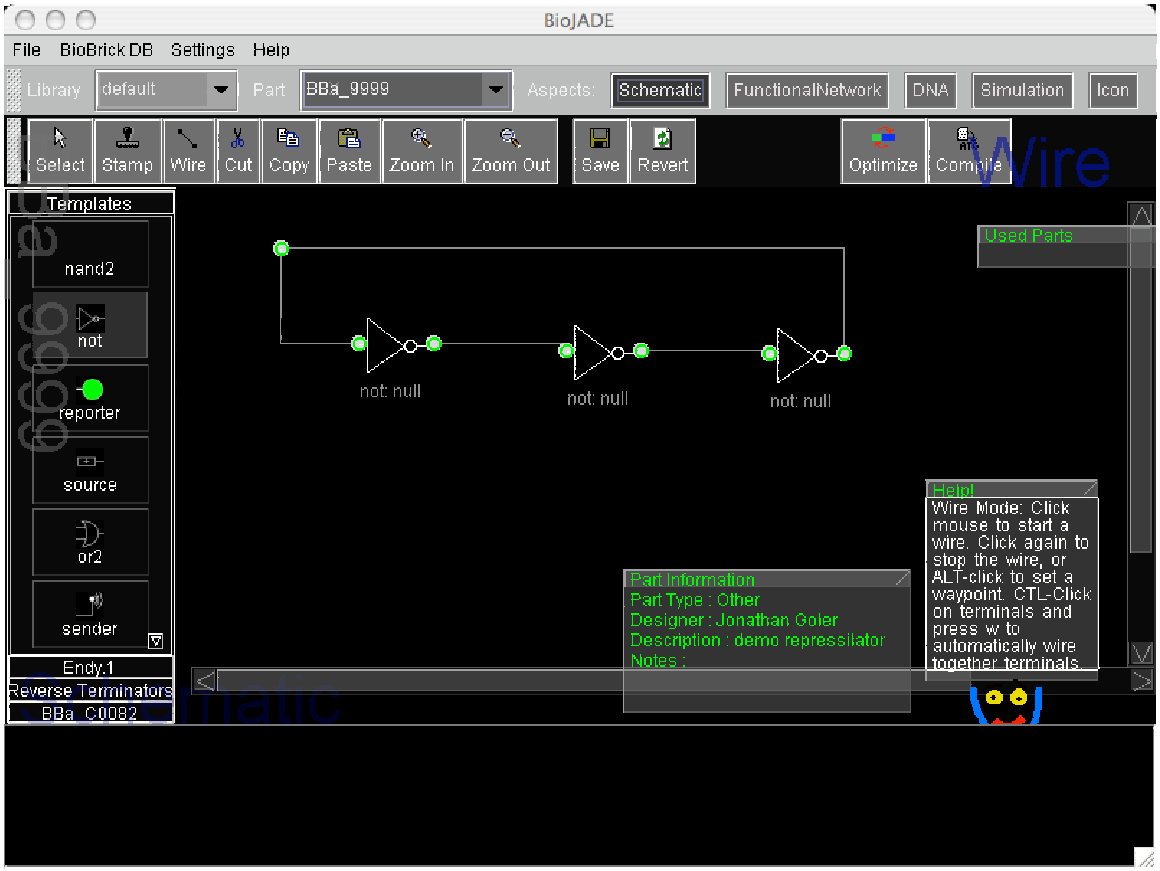

Figure 4.4: Schematic Design Mode. This figure shows the completed schematic of a repressilator.

libraries of additional parts to complete the design.

The completed compiled design is then reported to the user and can be saved to the database. The user can then go to the Functional Network mode to fine-tune the DNA sequences, and specify additional functional interactions.

4.4 Functional Network Aspect

The Functional Network aspect and the DNA aspect share several similarities in representation, but where the DNA aspect merely permits the user to view various aspects of the DNA sequence itself, the Functional Network mode enables the user to edit parts of the sequence, rearrange parts and change subparts such as ribosome binding sites and terminators.

The DNA is laid out vertically left to right as one goes through the circuit. The designer can click on a segment of DNA and edit its various components to optimize binding strengths, or change terminator efficiency. Designers can also physically change the locations of certain genes on the plasmid for whatever reasons might be desirable, such as balancing low-GC regions or exploiting

terminator inefficiency to cheat the modularity abstraction.

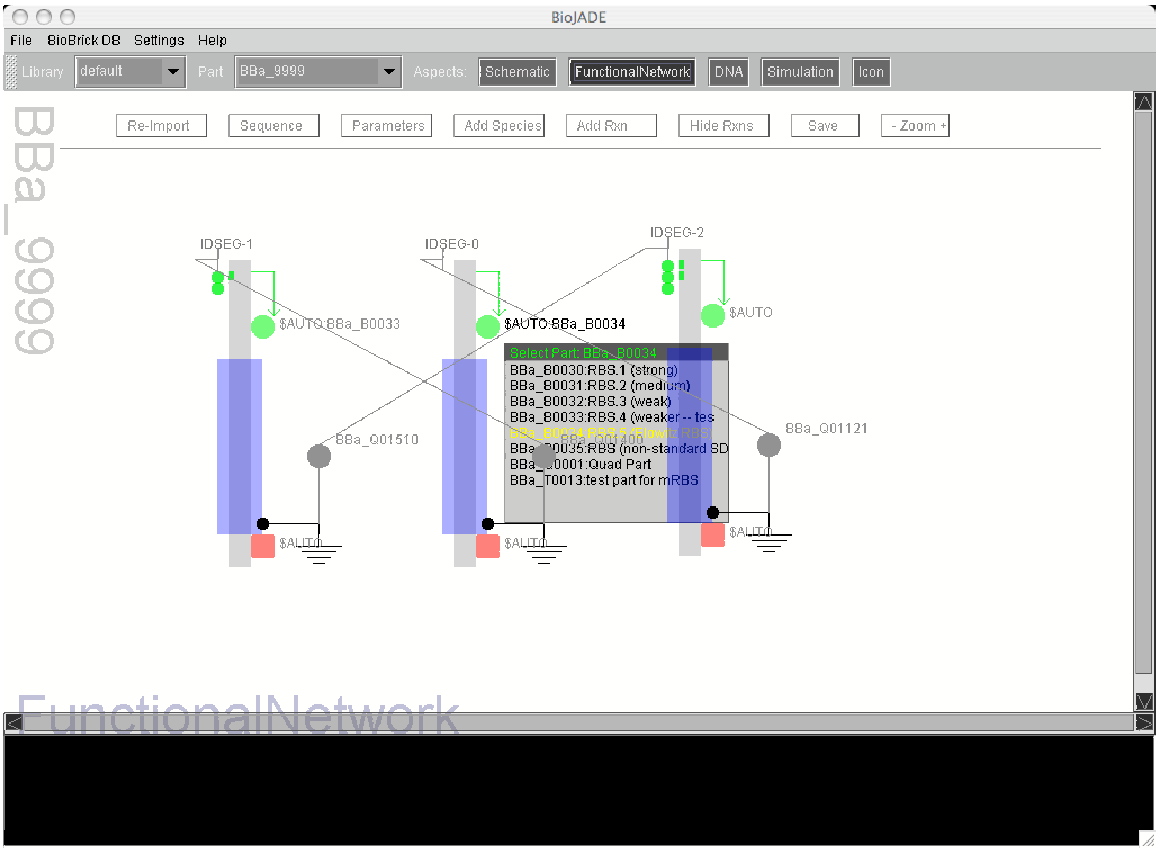

Fig. 4.5: The Functional Network Representation shows the same repressilator, but in the form of stretches of DNA. System features such as ribosome binding sites can be selected from the functional network aspect.

4.4.1 Adding Species and Reactions

In addition to the DNA itself, the Functional Network mode enables the designer to express other important reactions and processes such as protein and RNA degradation rates, small molecule interactions, and anti-sense behaviors. These important interactions are not represented well a schematic, but are vital in expressing how a complicated system works.

To add a molecule, click on the Add Species button on the top of the window, a dialog will pop up and ask for the name of the species, and then the initial concentration and finally, the degradation rate.

Reactions between species (both manually inserted and the protein/RNA from the system) can be assembled by clicking on the Add Rxn button. The reaction will be represented as in FIG, with

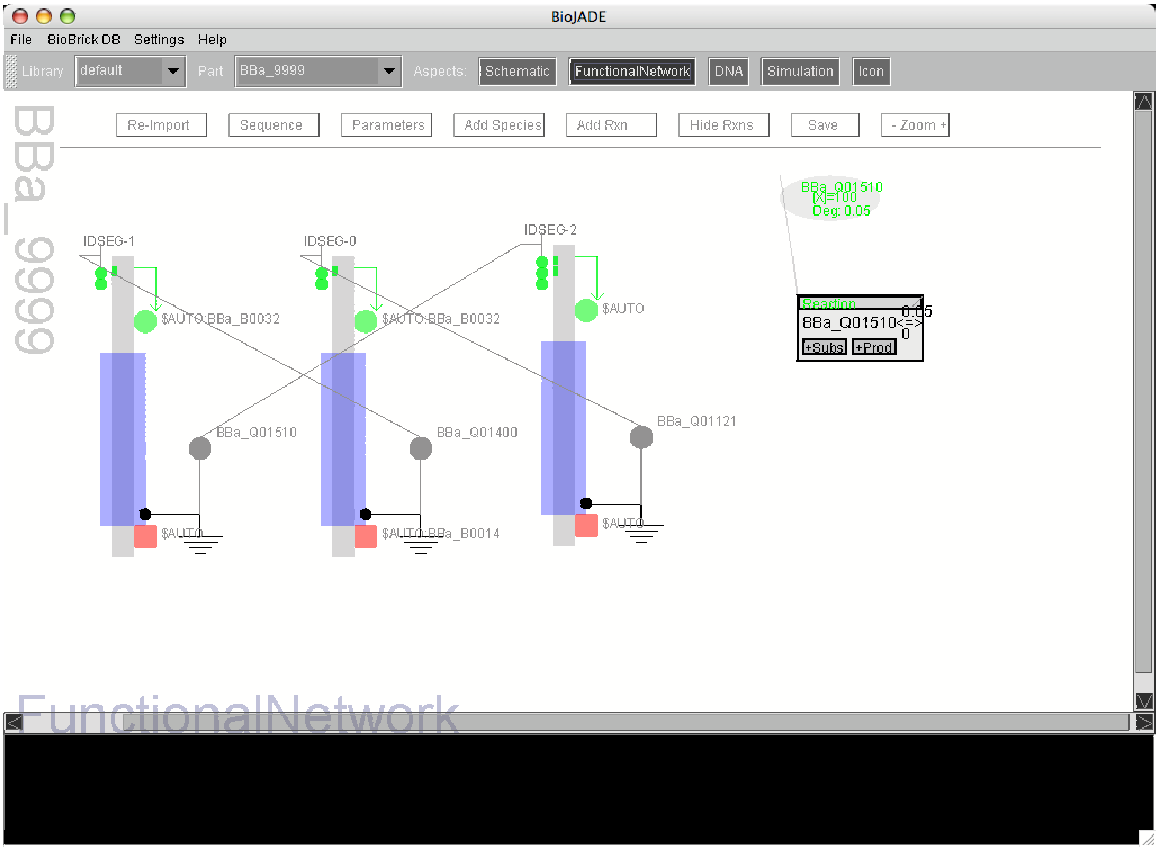

Fig. 4.6: The Functional Network mode also enables the designer to specify other species and reactions that affect system performance. This figure shows a degradation reaction involving one of the signaling molecules.

the two sides of the reaction, as well as forward and reverse rate constants. Clicking on each of these regions allows you to set the rate constants, or add other substrates or products. To add a substrate or product, click on the right, or left side of the <=>, respectively. Then click on the substrate or product of interest on the screen. The species is added to the reaction, and the system draws a line between the reaction and the species connected to it. These reactions will then be modeled and simulated in whatever simulator is later selected.

4.4.2 Refining Auto Part Selections

The compilation process of making a concrete part out of the abstract design gives designers some flexibility in deciding which parts to use for certain common features, such as ribosome binding sites and terminators. For biological and design reasons, it may be desirable to manually make choices based on efficiencies. In addition, selecting different parts will reduce homology and thus

reduce the likelihood of undesired recombination events.

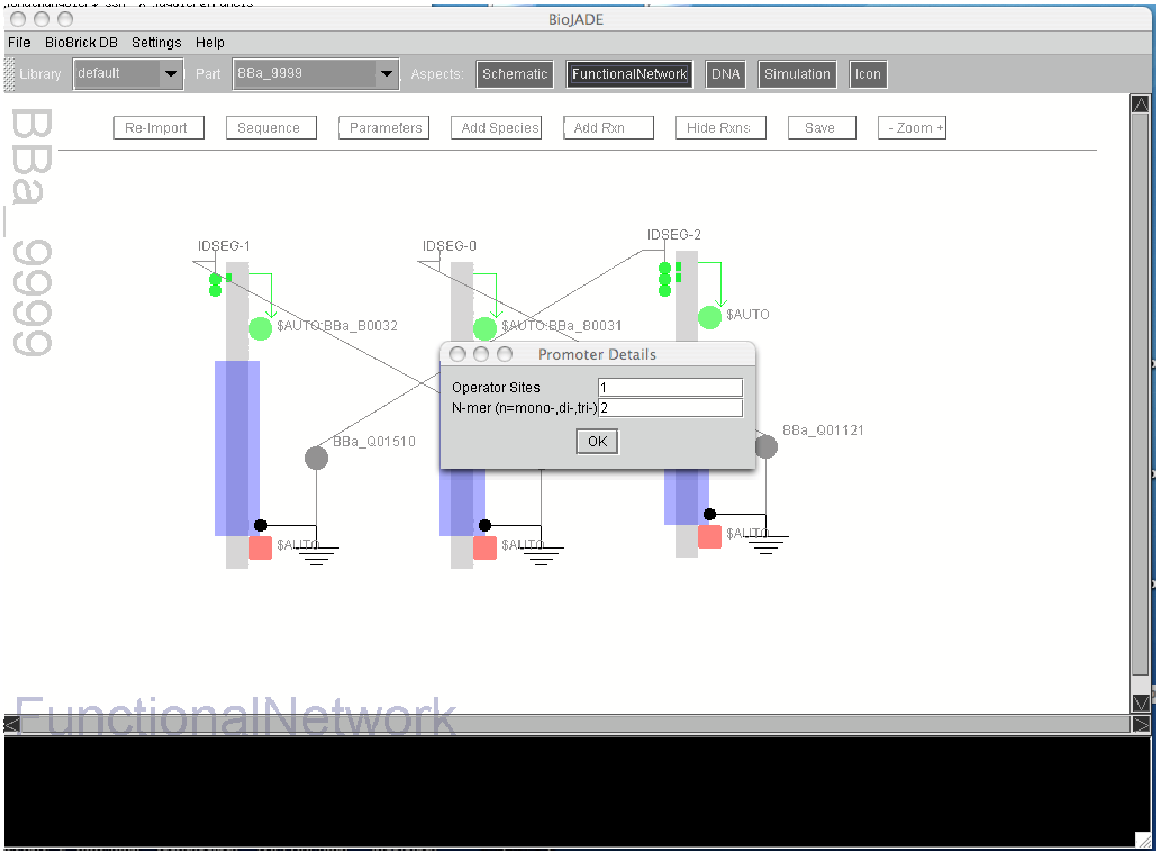

Fig. 4.7: Editing promoter details such as the cooperativity in the Functional Network mode.

To set these variable sequence features such as RBS's and terminators, the designer clicks on that section of the DNA segment and a window pops up with a list of available parts to assign to that module. Using the arrow keys, the user can scroll through the list of options and select the part that fits best in the system.

Once satisfied with the functional representation of the network, the next step in design is to simulate its function. For this feature, we utilize the Simulation Aspect to access a variety of simulators.

4.5 Simulation Aspect

The simulation aspect provides an interface to a variety of simulation tools using the D-FLUX protocol for performing intense computation on distributed computers. The designer can select one or more simulations to execute on the design.

The designer may then execute the simulations and either tinker further with the system or just wait for the results. The simulation results are then displayed in whatever viewer is appropriate for the simulation. Simulation results that are computed on remote systems may be retrieved and viewed at the designer's convenience. This asynchronicity affords the designer with more time to tinker with the system and see the impact small changes have on system characteristics. This feature should help to increase the productivity of BioJADE designers.

Chapter 5 is a more comprehensive discussion of the Simulation Aspect and the available simulations.

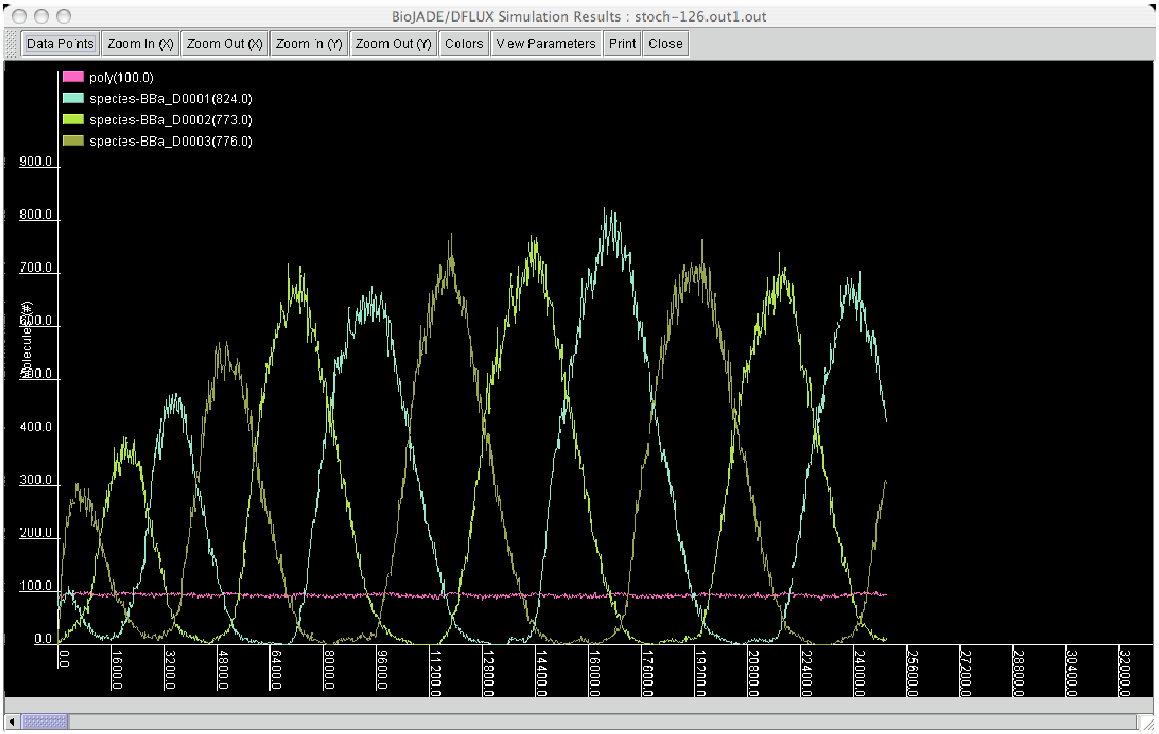

Fig. 4.8: The same repressilator simulated with Stochastirator. The figure shows the oscillation of all three signaling molecules.

4.6 Icon Aspect

No part is complete without its own graphical representation. Because we wanted to give the users flexibility in their choice of iconography, the user can draw a 50x50 pixel representation of their part in any paint program that exports to the PNG/GIF/JPEG formats. The icon aspect allows the user to select the icon file, which is then displayed. Once the graphical representation is selected, the designer selects Mark Terminals to mark the terminals of the device. When the part is used in a schematic view, the terminals provide places to attach wires.

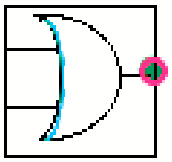

Fig. 4.9: Drawing a dynamic terminal onto the icon representation for a wired-NOR gate. The red terminal is a normal terminal, and the green arc on the left is the dynamic terminal area.

Upon marking a terminal, BioJADE asks the designer what kind of terminal they are putting down: input, output or undefined. ALT-clicking on the terminal can change the choice at any time. In addition, Control-clicking allows the user to select an associated subpart of the device to which the terminal is connected.

4.6.1 Dynamic Terminals

Part of designing general devices includes adding the possibility of having variable numbers of terminals on the device. To account for this possibility, the icon aspect allows the designer to specify dynamic terminals, which define a strip of the device that can grow to an arbitrary number of terminals. To specify a dynamic terminal, click on the Add Dynamic Terminal button, and then click once on the start point for the dynamic terminal, as you drag your mouse across the icon, a line showing the dynamic terminal is shown. Holding down the shift key while dragging the mouse draws an arc shaped dynamic terminal. Clicking again sets the terminal's ending position. The actual number of terminals that implement the dynamic terminal is set by in the schematic mode after the components are stamped onto a design.

Parts that are simply an implementation of a part type (inverters, NANDs) need no special graphical representation of their own. The user can provide an icon, but the default representation for the part type is used instead if there is no icon selected.

4.7 DNA Aspect

DNA mode allows BioJADE users to view the basic DNA features of a part in a format similar to other tools, such as the Biobricks Repository or Vector NTI. In addition, the DNA aspect gives

the user a view and summary of the local DNA features such as GC content, the forward and reverse sequences and the cut sites of restriction enzymes. The DNA aspect shows two panes, one containing the DNA sequence and the other showing an annotation of the sequence. This feature is intended primarily to convey information rather than edit it.

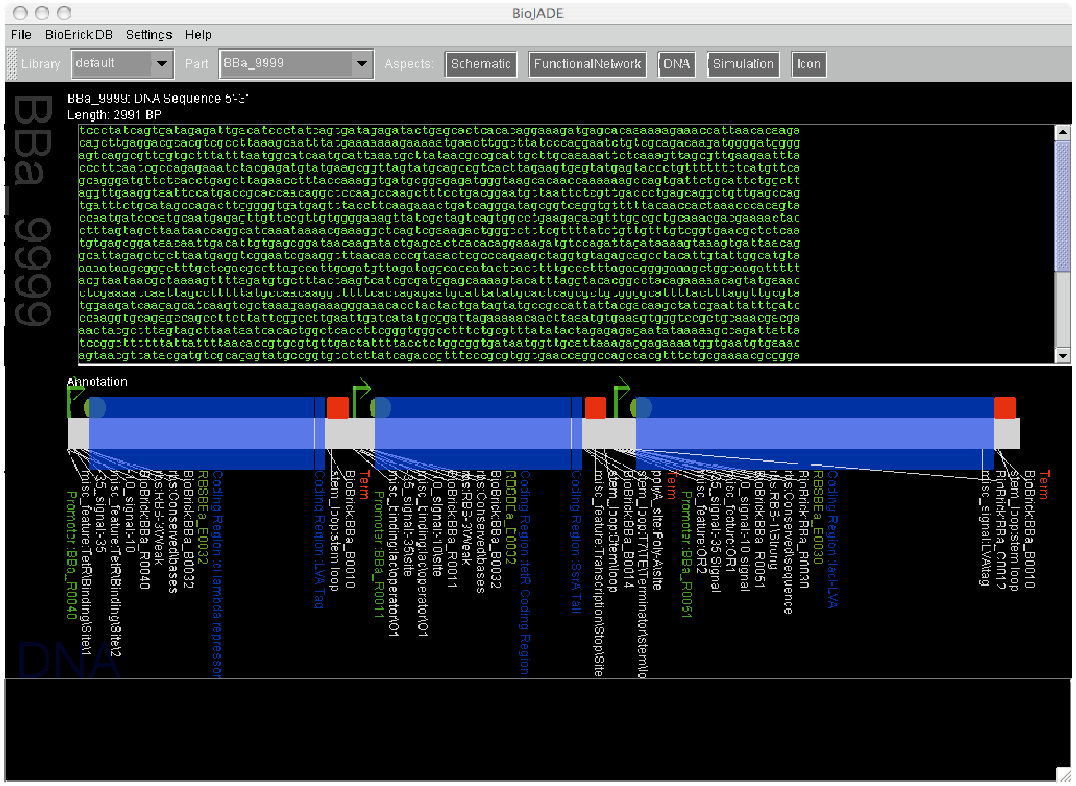

Fig. 4.10: Examining the repressilator in the DNA Aspect. The DNA Aspect displays sequence data along with a graphical representation highlighting the annotation of the DNA.

Chapter 5

Simulations

An abstract design tool is far more useful when combined with powerful and accurate simulation tools. BioJADE includes a number of powerful simulation systems that enable desingers to understand and predict the behavior of their designs. BioJADE is very flexible in its simulation capabilities, offering a plug-in type architecture for wrapping almost any third-party simulator available.

5.1 D-FLUX

We utilize the Distributed, FLexible and User eXtensible protocol for simulations. D-FLUX lets us implement a series of wrappers that encapsulate essentially any simulator we desire. Simulators such as Tabasco [4] are written in Java, which allows us to directly call the necessary functions in the application. However, it is also possible to execute any script, or executable program on the host computer and thus we have access to any simulator that is commercially or freely available.

DFLUX allows designers to create their systems on their local machine and submit the simulation to any available simulation machine. This architecture gives the user access to much more powerful machines and, thus, more complicated simulations than they would be able to execute on their own machine. In addition, batches of simulations can be run on clusters of remote machines, permitting the designer to create a range of parameters to test.

5.1.1 The Architecture

Communication Protocol

The communication protocol is based on work done with developing a secure voting system. In this model, there is a client and a server. The server is the system that is waiting for requests and the client is the system that sends requests. The server listens for socket connections on a specified port. The client initiates a request, the server acknowledges the connection and spawns a process to handle it. At this point, the client merely sends an XML formatted command to the server, and the server generates the appropriate reply and writes it back to the client.

The communication protocol is fairly simple. Clients send messages to the server of the form `<COMMAND><commandName/></COMMAND>` .

The Server replies <RESULTS><result/></RESULTS>. Each command ends with a line with the string "!!!END!!!".

Available Commands are:

- GET-AVAILABLE-SERVICES - Returns the services that are available on the server queried.
- EXECUTE (service, user, password, <params>,<data>) - Returns a process ID on the server to access the status/results of the command.
- GET-STATUS (<processes>,<user>) - Returns the status of the list(s) of processes or users.
- GET-RESULTS (process) - Returns the results from the simulation data in the simulator's format.

Parameters noted in angle brackets are enclosed within the command's opening and closing tags as their own entities while unenclosed tags are passed as parameters within the command's opening tag. Note that by convention, all tags are capitalized, with all parameters lower case. All tags that do not have a closing tag must have a self-closure (/>). As usual with XML, white space is not important in non-quoted strings.

Exceptions are passed back as an <EXCEPTION> and they contain (optionally) the service, and then within the body of the tag, a text description of the error.

Simulation Server

The simulation server itself is highly simplistic. The specification merely requires that it provide appropriate responses for the basic commands. To fulfill this task, we implement the server as a thread spawning while loop. The server listens on its assigned port and whenever a request is received, it invokes a thread to handle the request. How to manage concurrency and the status of local processes are not addressed in the protocol, but the server must keep track of the state of each simulation request. Connections are not held open, so clients must submit separate requests to check on the status and get results from their simulations.

For BioJADE, the simulation server transmits the results to the parts repository. That way, the results are available asynchronously to multiple investigators.

Simulator Address Server

The address server merely maintains a list of the available servers for simulations. Each entry contains the IP address or name of the server, the port number it listens on, and whether or not it requires authentication. The address server is structurally identical to the Simulator Server but it accepts a different set of commands.

- REGISTER-SERVICE (service-name, address port, authentication) - registers the specified service with the address server.
- GET-SERVICES - returns the services registered with the address server in the form <RESULTS><SERVICE name='service-name' address='francis' port='999' authentication='public'></RESULTS>

The SAS provides a central place to store all of the services available, and enables an organization to maintain one or two SAS servers and not have to update every client when a new service is brought online.

5.2 Standard Simulators

5.2.1 Stochastirator

The Stochastirator is a Gibson-Modified-Gillespie [9, 18, 8, 19] simulator written by Eric Lyons in C++. Stochastirator is wrapped by writing out the necessary model definitions and parameters, and then executing the simulator.

Stochastirator is a stochastic simulator useful for dealing with systems at the single molecule level, rather than the concentration levels which are often used. Defining a system with BioJADE allows it to be simulated directly with Stochastirator.

The translation program takes in the design, and builds interactions between the species in the system, and the repressors they bind to. The translator also connects the promoter regions to the genes they promote (wired connections) by simulating the transcription and translation reactions that produce proteins. This process takes into account the cooperative binding of species, an effect that accounts for the non-linearity.

5.2.2 Tabasco

Tabasco, written by Sriram Kosuri and Jason Kelly [4], was originally designed to simulate the behavior of phage DNA entering a cell. The DFLUX Wrapper translates the designer's DNA system into the tabasco format, and alters rate constants to simulate normal transcription and control. Tabasco simulates, with single-base resolution, the behavior of polymerases, small molecules, and proteins within the host cell.

Wrapping Tabasco in D-FLUX was the first practical application of D-FLUX. Writing the wrapper was simple because Tabasco is written in Java. Thus we are able to hook directly into the native data structures and simply execute the simulator.

Since D-FLUX permits us to execute complicated simulations on (multiple) remote computers, a designer can build a matrix of different values that she wants to test the system with, submit that set of simulations to one or more compute clusters, and proceed to design another system while the remote cluster executes the simulations. She can then check the servers to see if the results are ready. If so, she downloads the data sets, and analyzes the data. In the case of Tabasco, this data can be viewed either as plain text values, or, more usefully, as a movie depicting the transcription of each gene. The DFLUX packaged version of Tabasco automatically generates a quicktime movie out of the simulation data and stores it in the parts repository.

For example, suppose that we want to view the behavior of a simple ring oscillator. We can design the oscillator using the Schematic mode, specify the relevant values for constants, transfer curves, etc in the Functional Network mode, and then submit the simulation to Tabasco.

5.2.3 Electrical Models

The electrical type systems use electrical models for simulating the behavior of genetic regulatory systems. One such example is the Represstor [24] simulator written by Gerald Jay Sussman in Scheme.

5.2.4 Logic and State Machines

A higher level simulation that is not yet implemented is a logic level simulation of the behavior of the systems. Each type of device has a logic function associated with it, and this logic function, along with the layout of the system would allow the designer to test the system out with a variety of initialization parameters and observe the behavior.

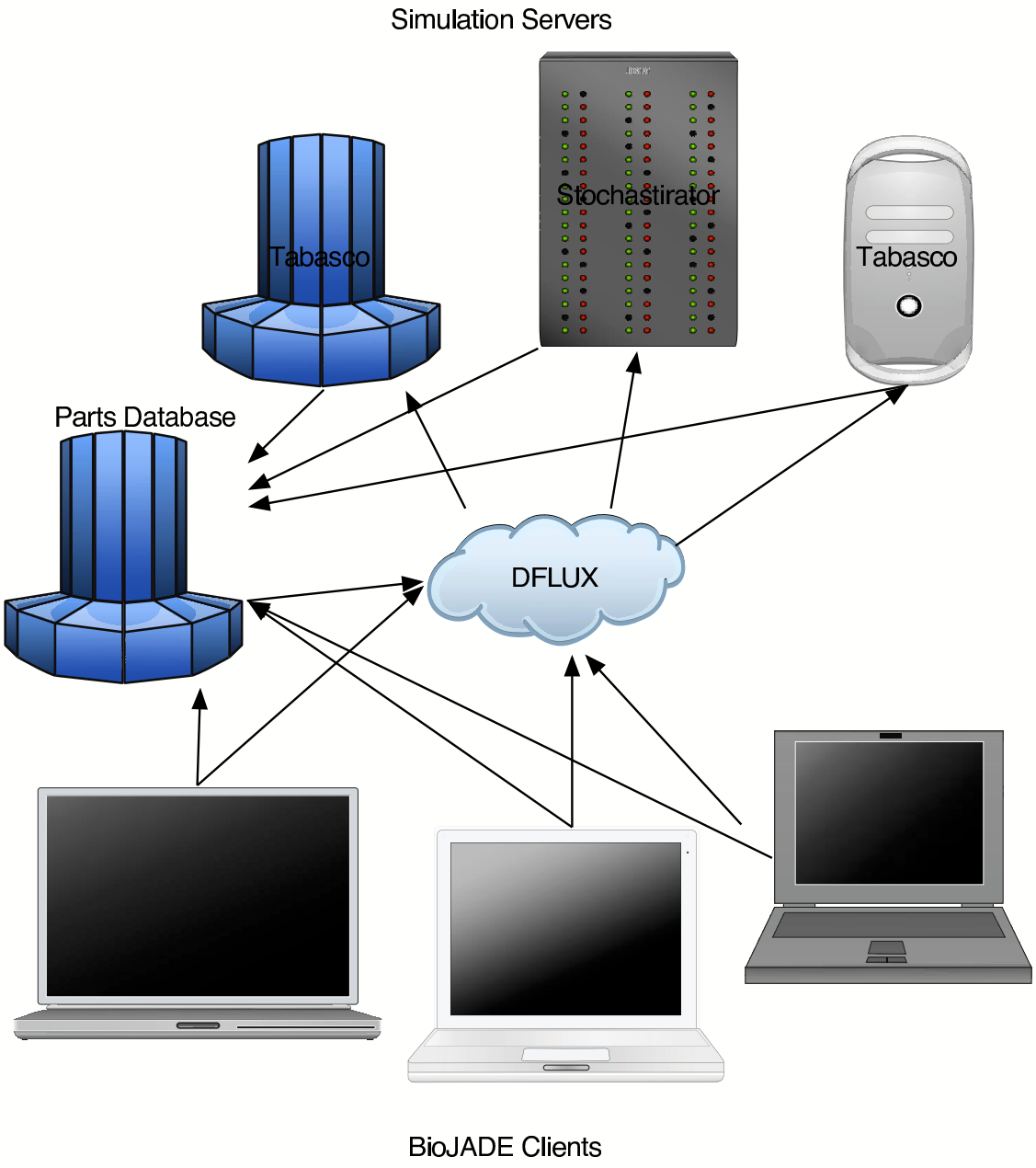

Fig. 5.1: DFLUX gives designers shared access to powerful computational resources to execute simulations.

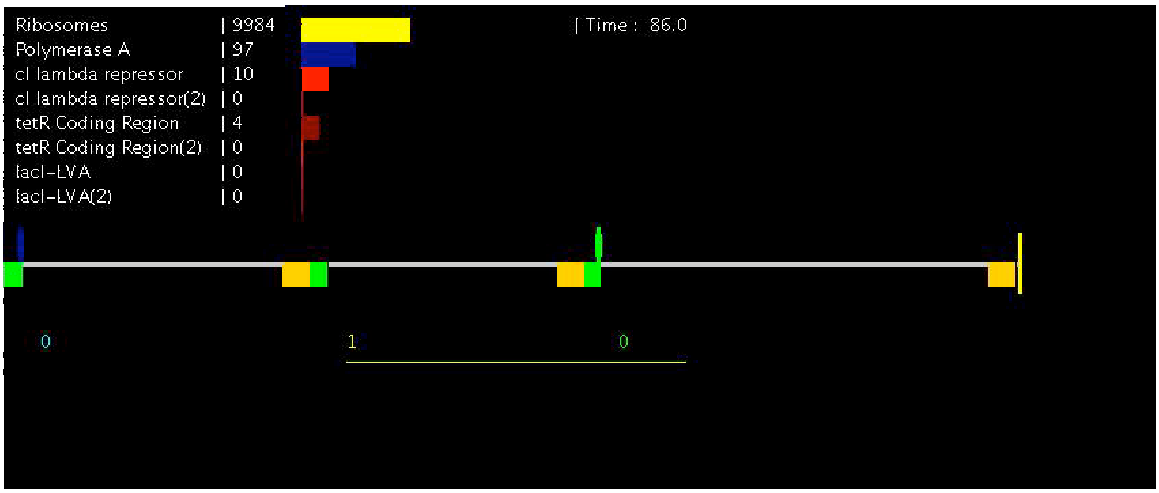

Fig. 5.2: A Tabasco simulation visualization that shows(in quicktime format) the generation of mRNAs and their associated proteins. The levels of each species is shown at the upper left corner. The DNA is shown as a line, with green blocks representing transcription starts, and yellow blocks representing termination. The blue or green ovals on top of the DNA are the polymerases, which move along the DNA. Finally, below the DNA are lines and a number representing mRNA transcripts produced by the DNA blocks above them.

Chapter 6

Modeling Biology and Devices

Having models of the world is useful and convenient, but we must have actual measured data in order to make reasonable predictions about the behavior of a system. This chapter describes the models, parameter measurement and designer interactions that allow BioJADE to execute simulations that provide useful insights into their behavior. BioBricks, while they encapsulate the basic part and interaction, do not include all of the information necessary to simulate systems. Device models encapsulate the general design of devices instead of just functional DNA segments. In addition, device models provide a more comprehensive means of providing data to simulate their behavior.

6.1 Basic Models

6.1.1 General Treatment of Cellular Species

There is a lot of "stuff" floating around in cells, and we really have no good way of knowing exactly how all of the species in the cell interact with each other, as well as our system. Thus, for our purposes we the species that do not directly interact as part of our systems.

Ribosomes and polymerases are the two most important host-specific and host-dependent species we need to model. The polymerase interacts with the promoter regions of DNA to begin producing RNA. The ribosomes bind to the RNA and produce proteins from the RNA. We assume that the cell has enough tRNAs to successfully translate an RNA transcript into protein.

6.1.2 Ribosome Binding Sites

Ribosome binding initiates translation of the mRNA into the polypeptide sequence it encodes. Translation is a complicated process, but for most practical purposes it is valid for us to assume that the cell has enough tRNA anti-codons to put together the proteins we are coding for. With that assumption, we can use the ribosome binding reaction to give us the number of proteins produced by each mRNA.

The substrates in the Ribosome Binding reaction are the ribosomes (r), mRNAs (n), and bound nRNA-ribosome complex($r:n$).

The reactions are defined as follows:

$r + n \underset{k_a}{\longrightarrow} r:n$

and the reverse
$r : n \xrightarrow[k_d]{} r + n$
where k_a and k_d are the association and dissociation constants.

The values of k_a and k_d are stored in the database. Each different RBS part has different values for its association and dissociation constants. Each part's constants may have different values depending on the host organism. The designer specifies the RBS part and the host organism. Changing RBS strengths is one clear way to tune the behavior of systems.

6.1.3 Coding Regions

Coding regions code for mRNA transcripts and eventually the protein product that is translated by ribosomes. The coding region's properties arise from the length of the coding region. The longer the coding region, the more opportunity the polymerases have to fall off without completing a transcript.

6.1.4 Termination

Terminators are characterized by their ability to successfully terminate transcription of DNA. If the terminator does not work, then it is possible for additional coding regions to be transcribed. Leaving off terminators may be useful to create more copies of a gene or to create multiple proteins. Placing multiple coding regions together without termination is known as polycistronic coding and is observed in such systems as the Lux autoinducing system in *V. fischeri* [6].

For simulation, terminators are characterized by a termination efficiency in forward and backward directions.

6.1.5 Promoter Binding and Repression/Expression Control

Promoters are the most complicated part of the simulation behavior. There are both positive and negative regulation sites. In addition, many binding sites are cooperative, with a Hill coefficient greater than 1. This cooperativity is necessary to get the noise margins and signal regeneration properties of digital logic. Cooperativity is achieved when the binding of a single molecule onto a promoter makes it easier for a second or third molecule to bind.

Suppose there is a promoter that is regulated by a dimer of the regulating protein. We can then have species involved: p, the protein, $p : r$, one protein bound to the promoter's regulatory region, and $p : p : r$, a dimerized protein bound to the promoter's regulatory region. When p is single, it falls off the promoter very quickly, but if there is already a p on the promoter, then the dimer does not dissociate quickly. Only the doubly bound state prevents a polymerase from binding to the promoter and initiating transcription.

6.2 Device Models

Once we have obtained fairly reliable values for the basic reactions that comprise the basic parts, we can use those in simulations to determine the lag times, device sensitivities and transfer functions that help us make less detailed but more general models.

To create the device model, the behavior of the system related to the outside world must be defined. We make use of standard electrical engineering nomenclature to define our parts.

Defining parts by their input/ output characteristics requires a boundary to measure inputs and output. One obvious signal to measure is the concentration of proteins. Unfortunately, protein levels vary greatly in their production, degradation and binding affinities. We define TIPS, or Translation Initiations Per Second as the standard I/O parameter. One TIPS is one translation initiation being initiated per second (one polymerase passing the part boundary on the DNA).

T_{ol} Output TIPS for a low signal. T_{ol} must be below T_{il}, which is the upper limit for a valid low signal.

T_{oh} Output TIPS for a high signal. T_{oh} must be above T_{ih}, which is the lower limit for a valid high signal.

T_{il} Upper limit for a low signal input.

T_{ih} Lower limit for a high signal input.

rise time The time it takes for the device to switch from a low state to a valid high state. Rise time is limited by the rate of transcription and translation, and to a lesser degree the size and growth of a cell.

fall time The time it takes for a device to switch from a high state to a valid low state.

fall delay The time it takes for a device's output to become invalid after a change from high to low.

rise delay The time it takes for a device's output to become invalid after a change from low to high.

transfer curve A mapping of TIPS in vs. TIPS out. The transfer curve usually has a nonlinearity, indicating cooperativity.

6.2.1 Quad Part Inverters

Quad-Part inverters(QPIs) are one of the most elementary devices we can construct as part of a genetic circuit. They are composed of 4 parts: RBS, coding region, terminator and promoter. In a basic design that puts those four parts together, Stochastirator or Tabasco can be utilized to simulate the behavior of the gate. It would also be fairly simple (and necessary) to validate the simulation and the models by actually measuring the transcription rates.

Chapter 7

Compiler Issues and Optimizations

7.1 Logic Reduction

Despite years of study, we still do not completely understand how to develop additional logic functionality such as AND/NAND gates as a single gate. For now, we can use DeMorgan's law below to reduce any logical expression to a series of inverters and wired-or gates.

$$\overline{(A \cap B)} = \bar{A} \cup \bar{B} \tag{7.1}$$

$$\overline{(A \cup B)} = \bar{A} \cap \bar{B} \tag{7.2}$$

This reduction operation is simple enough for our compiler to handle, but while the complexity of the functions is reduced, the complexity of the resulting DNA logic introduces a great deal of messiness in the design and implementation. First, we are required to use more parts to accommodate the reduced logic. Second we introduce additional species of proteins floating around the cell, which could lead to interference and crosstalk. In addition, by using a significant amount of additional DNA and coding regions, we are increasing the load on the chassis cell, reducing the likelihood of a successful, thriving cell. Finally, the additional parts required increase the amount of DNA synthesized and the number of assembly steps, which increases the probability of error, even with improvements in synthesis and assembly.

Thus, it would be desirable (Future Work) to develop single modules capable of more complex functions. In any event, the compiler can deal with many of these problems by making optimized choices in the way it assigns parts.

7.2 Managing Biological Uncertainty

Circuit designers must deal with clock skew when dealing with large circuits because even the speed of light is finite, and the operation of transistors is not instantaneous. In our genetic circuits, our signals are diffused through a stew of cellular machinery that occupies the cytoplasm. This diffusion process introduces significant propagation delays and unpredictability in our designs. When we have parts which are fully characterized, we will know their (approximate) propagation delays, and the compiler can attempt to synchronize their behavior more accurately, so that the inputs for a wired

OR would appear at the same time. This behavior would be observable in some of the simulations, and modifications can then be made based on that simulation data.

7.3 Possible Optimizations

Databases such as SWISSPROT[12] are capable of computing sequence and structure homology for proteins. This capability enables us to build a table that would alert us to proteins which have high structure homology and might interfere with each other. In addition, experimental data between pairs of proteins provides our database with information on crosstalk. Using this data, the compiler can find the set of least-interacting proteins to use in coding regions and promoters. Although this computation is very time consuming for any significantly large set, it is still worthwhile because computation is cheaper and faster than building the DNA. The initial (simplistic) compiler runs in the same thread as the designer, while the optimizing compiler runs as a separate process, enabling the designer to continue doing other operations while the design is processed either on the same machine or on a remote machine.

The remote compilation uses the same basic protocol as the remote simulation protocol, which is addressed in Chapter 5.

Chapter 8

Future Work

I plan to extend BioJADE to include not only logical elements, but also metabolic pathway support. This additional support for more general devices (metabolic machines) will enable designers to exert control over cellular processes in addition to simply placing additional circuitry in the cell.

Several other extensions to BioJADE and related tools are made possible through the flexible Aspect oriented architecture. For example, it is possible to add a new Schematic layer that would allow a designer to specify a logic function, completely abstractly, and then generate the necessary schematic and functional description of a system from that logical specification.

The real future work is the actual use of BioJADE to develop new synthetic systems, and in doing so, gain a better understanding of what we are able to abstract out and what we must pay attention to in terms of biological details.

8.1 Potential Dangers

Synthetic Biology has great potential. But it also has great peril. This section outlines a few of the concerns of the improper use of synthetic biology. In addition, I provide a few safeguards and recommendations for avoiding accidental introduction of dangerous parts or organisms.

8.1.1 Friendly Warnings Help Avoid Accidents

In a manner similar to "Are you sure you want to put that botulin toxin gene there?", we can help our designers avoid making obvious mistakes. These friendly warnings only go so far, however, and might end up suffering from the anti-Clippy syndrome that afflicted the annoying paper clip in Microsoft Office that tried to tell the user what to do. Thus, the use of warnings must be careful and appropriate. Parts that are dangerous should all be marked as such.

8.1.2 Open and Secret

While BioJADE for the most part will be an open and free system, the parts of the system designed to guard against bad behavior will be kept closed. While some open source advocates might think this is a bad idea, it may help keep the casual terrorist out of the Synthetic Biological terror business a bit longer.

In a manner similar to cryptography, it might be helpful to restrict programs like BioJADE to friendly countries and not permit export to states that sponsor terrorism, etc. This will not eliminate the danger, but it might mitigate the risk a bit.

8.1.3 Education

It goes without saying that anyone doing synthetic biology should first have some idea of what they are doing. As in every laboratory science, there are protocols to learn to protect both scientists and the rest of the world. Anyone using BioJADE must first learn about the dangers, risks, responsibilities that are inherent in engineering biology.

8.1.4 Monitoring

I shall not digress too much on the subject of monitoring. Needless to say, when the program detects the user trying to put together a VX producing bacteria, it should let the central repository operators know about it. This monitoring should be done without informing the user, lest they figure out some other method of causing mischief.

As mentioned above, the monitoring system should be located in closed code, so that the user cannot easily remove the code that does the reporting. The system will also have to be networked for this monitoring scheme to work. This suggests some sort of central license server connection may be necessary to perform certain operations.

Since all legitimate systems would likely have little problem with this arrangement, it is unlikely that such controls would be a hinderance on acceptance. However, for companies that are highly concerned about the system giving away their trade secrets, it would be fairly simple to deploy a specifically designed license server to the user's company. For example, a large pharmaceutical company might want to use BioJADE, and if they are a law abiding company, they can be granted their own license server, and still protect the confidentiality of their information.

Chapter 9

BioJADE's Contributions

This thesis makes a number of important contributions to the nascent field of synthetic biology. BioJADE itself is only a part of a larger set of tools that enable synthetic biologists to effectively create, share and extend their work. The components of BioJADE and the BioBricks database will enable broader acceptance, understanding and interoperability of BioBrick components.

The main contributions of the BioJADE project are:

The BioJADE Design Environment which enables synthetic biologists to more effectively create and analyze synthetic systems.

BioBricks Database which provides a repository for BioBricks. Designers have access to parts and their characteristics. Researchers have a centralized place to store parts that they discover, build or characterize. The Database and Repository system enables the field of synthetic biology by facilitating the easy exchange of parts.

Synthetic Biology XML Format enables standardized data exchange.

Safeguards help prevent creation of components that pose a danger to existing ecosystems and beings.

D-FLUX protocol for remote, distributed simulations.

Appendix A

BioBricks SQL Data Model

1.0

```
-- Parts Repository Data Model
-- MIT Synthetic Biology Group
-- 1/21/03 Jonathan Goler (jagoler@mit.edu)
-- Reference Revision 2/4/04 Jonathan Goler & Randy Rettberg
-- (C) 2003, 2004 MIT Synthetic Biology Working Group



-- Website/Database Users and groups tables

use bbdb;

create table organizations (
org_id integer auto_increment primary key,
org_name varchar(200),
org_contact varchar(200),
org_email varchar(200),
org_zip varchar(10),
org_country varchar(20),
org_phone varchar(20)
);

create table parts_users (
user_id integer auto_increment primary key,
user_name varchar(20),
password varchar(20),
email varchar(200),
status  char(1) default 'n',
  user_type    varchar(20) default 'Normal',
full_name     varchar(200),
```

```
comments     text,
preferences longtext default '',
affiliation integer references organizations(org_id)
);

create table sessions (
session_id integer auto_increment primary key,
user_id integer references parts_users(user_id),
expiry date
);


-- basic part information

create table parts (
part_id integer auto_increment primary key,
part_name varchar(255),
short_desc varchar(100),
sequence longtext,
description longtext,
source text,
container_id varchar(200),
creation_date date,
lastmod_date date,
version integer,
author varchar(200),
part_type varchar(20),
biology longtext,
status varchar(20) default 'New',
visibility varchar(20) default 'private',
lastmod_user integer references parts_users,
component_list longtext,
first_component integer default 0,
second_component integer default 0,
base_component_count integer default 0,
sequence2 longtext,
barcode varchar(50),
is_current char(1) default 't',
derived_from integer references parts,
license longtext,
notes longtext
);



-- parts auxiliary tables
```

```
create table parts_seq_features (
feature_id integer auto_increment primary key,
feature_type varchar(200),
start_pos integer,
end_pos integer,
label varchar(200),
part_id integer references parts(part_id) on delete cascade
);
```

```
-- crosslink/compatibility information
create table parts_crosslinks (
source integer references parts(part_id) on delete cascade,
target integer references parts(part_id) on delete cascade,
linked char(1) default 'n',
compatible char(1) default 'u',
notes longtext
);

-- journal articles/URLs
create table parts_references(
ref_id integer auto_increment primary key,
part_id integer references parts(part_id),
reference text
);
```

```
-- file storage support
-- with additional (optional) support for simulations
create table parts_files (
part_id integer references parts(part_id),
filename varchar(200),
mimetype varchar(200),
simulator varchar(200),
sim_id varchar(200),
description text,
filedata blob
);
```

```
-- measured properties of parts
create table parts_characteristics (
        char_id         integer not null,
        part_id         integer,
param_name      varchar(200),
param_desc      text,
typical         double,
minimum         double,
maximum         double,
units           varchar(20),
char_type       varchar(200),
chassis         integer
);



-- parts change log
create table parts_change_log (
log_entry_id integer auto_increment primary key,
part_id  integer references parts(part_id) on delete cascade,
change_entry text,
user_id integer
);



-- support for libraries

-- Support for Aspect data

create table parts_aspect_data (
part_name varchar(200),
aspect_name varchar(200),
aspect_data longtext
);



-- support for part devices
-- NEW STUFF HERE

create table parts_devices (
       device_id    integer auto_increment primary key,
       device_name    varchar(200),
       description    longtext,
       biology    longtext,
```

```
       subpart_list    longtext,
       is_expandable    char(1) default 'n',
       base_pinset    longtext,
       icon    blob
);
create table device_pins (
       device_id  integer references parts_devices(device_id),
       pin_type  varchar(20),
       pin_subpart  varchar(20),
       add_pin_code  longtext,
);
create table device_properties (
       device_id   integer references parts_devices(device_id),
       sim_level  varchar(20),
       property_data  longtext
);




-- support for Libraries

create table parts_libraries (
library_id  integer auto_increment primary key,
name varchar(30),
description text,
visibility varchar(10) default 'private'
);

create table parts_library_map (
library_id integer references parts_libraries,
part_id integer references parts on delete cascade
);


create table parts_library_user_map (
library_id integer,
user_id integer,
owner_p char(1) default 'n',
public_p char(1) default 'y'
);
```

```
-- Parts Assembly Program tables

create table parts_assembly (
assembly_id integer auto_increment primary key,
assembly_name varchar(255),
assembly_type varchar(20),
author varchar(200),
restrictions longtext,
current_stage integer,
owner_id integer references parts_users(user_id),
planning_date date,
description longtext,
result_list longtext,
misc longtext,
status varchar(20) default 'draft'
);

create table parts_subassembly (
subassembly_id integer auto_increment primary key,
assembly_id integer references parts_assembly(assembly_id),
stage integer,
result_part_id integer references parts(part_id),
left_part_id integer references parts(part_id),
right_part_id integer references parts(part_id),
flag_available integer default 1,
status varchar(100)
);
create table assembly_history(
history_id      integer auto_increment primary key,
link_id         integer references assembly_history(history_id)
                on delete cascade,
creation_date date default now(),
user_id         integer references parts_users(user_id)
                on delete set null,
assembly_id     integer references parts_assembly(assembly_id)
                on delete set null,
subassembly_id  integer references parts_subassembly
                on delete set null,
part_id         integer references parts(part_id)
                on delete set null,
description     longtext,
text            longtext,
sequence        longtext,
status          varchar (20) default 'New'
);
```

```
create table subassembly_history (
id   integer auto_increment primary key,
subassembly_id integer
            references parts_subassembly(subassembly_id),

flags varchar(200) default '',
comments longtext ,
creation_date date,
status varchar(100) default '',
stage integer default 0,
attempt integer default 0,
result integer default 0,
plasmid varchar (100) default '',
plasmid2 varchar (100) default '',
misc text default ''
);
```

1.0

Bibliography

[1] Austin Che. Biobricks++. *Synthetic Biology Working Group*, August 2003.

[2] Austin Che. PARMESAN. Master's thesis, MIT, 2003.

[3] Kolibachuk d and EP Greenberg. The *Vibrio fischeri* luminescence gene activator LuxR is a membrane-associated protein. *Journal of Bacteriology*, 175, 1993.

[4] J. Kelly D. Endy, S. Kosuri. Tabasco simulation system. 2001.

[5] DNA6. *Engineered Communications for Microbial Robotics*, 2000.

[6] WC Fuqua, SC Winans, and EP Greenberg. Quorum sensing in bacteria: the LuxR-LuxI family of cell density-responsive transcriptional regulators. *Journal of Bacteriology*, 1994.

[7] WC Fuqua, SC Winans, and EP Greenberg. Census and consensus in bacterial ecosystems: the LuxR-LuxI family of quorum-sensing transcriptional regulators. *Annual Review of Microbiology*, 50, 1996.

[8] Cantor Gardner and Collins. Construction of a genetic toggle switch in *E. coli*. *Nature*, January 2000.

[9] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 1976.

[10] EP Greenberg. Quorum sensing in gram negative bacteria. *ASM News*, 63, 1997.

[11] CC Guet, MB Elowitz, WH Hsing, and S Leibler. Combinatorial synthesis of genetic networks. *Science*, pages 1466–1470, May 2002.

[12] http://us.expasy.org/sprot/. Swiss-prot. 1999.

[13] Thomas F. Knight Jr. Biocomp plasmid distribution 1.00 of standard biobrick components. Technical report, MIT AI Lab, 2002.

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

[15] HB Kaplan and EP Greenberg. Overproduction and purification of the LuxR gene product: the transcriptional activator of the *Vibrio fischeri* luminescence system. *PNAS*, 84, 1987.

[16] T. Knight. Biobricks standard distribution. 1999.

[17] LL Looger, MW Dwyer, JJ Smith, and HW Hellinga. Computational design of receptor and sensor proteins with novel functions. *Nature*, 423:185–190, 2003.

[18] J. Bruck M.A. Gibson. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry*, March 2000.

[19] Stanislas Leibler Michael Elowitz. A synthetic oscillatory network of transcriptional regulators. *Nature*, 2000.

[20] HD Mootz, ES Blum, AB Tyszkiewicz, and TW Muir. Conditional protein splicing: A new tool to control protein structure and function in vitro and in vivo. *Journal of the American Chemical Society*, pages 10561–10569, September 2003.

[21] Princeton, editor. *Toward in-vivo digital circuits.* Dimacs Workshop on Evolutation as Computation, 1999.

[22] S. Basu R. Weiss. The device physics of cellular logic gates. Feburary 2002.

[23] Randy Rettberg. Triple-antibiotic (3a) assembly process. Technical report, MIT Synthetic Biology Group, 2003.

[24] G. Sussman. Electrical models for genetic regulatory elements. Technical report, Synthetic Biology Working Group, January 2003.

[25] C. Terman. Jade. 1999.

[26] Randy Rettberg Thomas F. Knight Jr. Biobricks standard assembly. Technical report, http://biobricks.mit.edu, 2002.

[27] JMG Vilar, CC Guet, and S Leibler. Modeling network dynamics: the lac operon, a case study. *Journal of Cell Biology*, pages 471–476, May 2003.

[28] JMG Vilar, HY Kueh, N Barkai, and S Leibler. Mechanisms of noise-resistance in genetic oscillators. *PNAS*, pages 5988–5992, April 2002.

[29] R. Weiss. *Cellular Computation and Communication using Engineered Genetic Regulatory Networks.* PhD thesis, MIT, 2001.

[30] W Yang, LM Jones, L Isley, Y Ye, H-W Lee, A Wilkins, ZR Liu, HW Hellinga, R Malchow, M Ghazi, and JJ Yang. Rational design of a calcium-binding protein. *J. Am. Chem. Soc.*, 125:6165–6171, 2003.