

A Software Stack for Specification and Robotic Execution of Protocols for Synthetic Biological Engineering

Viktor Vasilev, Chenkai Liu, Traci Haddock, Swapnil Bhatia¹,
Aaron Adler, Fusun Yaman, Jacob Beal², Jonathan Babb, Ron Weiss³, and
Douglas Densmore¹,

¹Department of Electrical and Computer Engineering, Boston University, Boston, MA

²BBN Technologies, Cambridge, MA

³Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA

{vvasilev, nykai55, thaddock, swapnilb, dougd}@bu.edu,
{aadler, fyaman, jakebeal}@bbn.com, {jbabb, rweiss}@mit.edu

1. INTRODUCTION AND MOTIVATION

Synthetic biology is an emerging field in which biologists modify or design the behavior of organisms to engineer systems that perform computation in diverse biological applications. Synthetic biologists design such a complex system by composing basic functional units—e.g., a promoter or a gene—into a regulatory network that exhibits the desired transcriptional behavior. As the desired behavior becomes more sophisticated, the size of the network grows, the complexity of the design becomes an impending concern [2], and its assembly and verification, an arduous task. The design complexity may be addressed by powerful design tools, large circuits may be assembled using novel assembly protocols, and the result may be verified by executing a comprehensive test suite. Performing design, assembly, or verification manually however, is tedious, error-prone, not easily reproducible, and hence unscalable. We address the problem with a chain of tools [3, 8], each tool providing an optimized solution to the problem at a discrete grade of abstraction. This report focuses on the assembly and verification stage—specifically, the design of a software stack for high level specification of biological protocols used in assembly and verification.

The primitive genetic parts comprising a larger system are constructed to be compatible with one of the standard assembly protocols such as BioBricks [7] or BioBytes [6]. Assembly planning algorithms [5] take a library of such assembly-ready parts and a list of genetic devices to be assembled, and produce an optimized hierarchy of assembly steps. The execution of each step in this sequence requires the execution of one or more basic biological procedures such as a ligation or a restriction digest. A critical gap exists between the specification of assembly plans at this level—a sequence of biological procedures—and the programming interface of liquid handling robots. Liquid handling robots are typically programmed by chaining together a sequence of basic actions such as pipetting from a specific well in one plate to another well, moving a plate or other labware from a specific location on the robot deck to another, or an action involving interfacing with auxiliary instruments such as incubators or mixers. While low-level access to a robot may afford greater flexibility, it has

at least two disadvantages: 1) Low-level languages make the specification of complex protocols and their composition into higher-level human understandable units difficult. The absence of modularity, powerful and well-defined composition operators, and the need to manage labware minutiae makes low-level programming difficult, unscalable, and the resulting code unmaintainable. 2) Specifying protocols with a vendor-provided low-level language ties the resulting code to a particular robot architecture, removing the secondary advantages of programmed automation: portability, open exchange, and accelerated development through reuse. In this work, we bridge the gap between high-level assembly protocols and a low-level robot interface by designing a high-level language for biological protocol specification called Puppeteer, and a robot Hardware Abstraction Layer. The proposed design will allow high level specification and composition of protocols, accelerate protocol design, and make the assembly and verification of large systems tractable. Below, we describe our design, its advantages, the current state of its implementation, and plans for future work.

2. ARCHITECTURE

Our solution comprises a five-layer stack as illustrated in Figure 1. Using the Clotho platform [4], we develop two applications for specifying and executing biological protocols. The Assembly Planner [5] is the end-point of an end-to-end design workflow [3] that produces an assembly plan for synthetic biological devices, with each assembly step annotated with the name of a biological protocol. Each such protocol itself may be fully specified using another Clotho application called PuppetShow, which provides an environment for writing, testing, debugging, and executing biological protocols.

The protocols are written in a new high-level language called Puppeteer. The Language layer comprises the Puppeteer interpreter and linker. A protocol specified in Puppeteer may contain Puppeteer instructions as well as references to previously created Puppeteer programs available in a library. The Language layer expands and translates a Puppeteer protocol to a sequence of low-level commands expressed in a Common Robot Instruction Set (CRIS). CRIS provides a standardized instruction set that high level biological protocol languages like Puppeteer may assume to be supported by any robot. Any high-level language may produce CRIS programs and any robot ven-

dor may support a superset of CRIS: this decouples robot hardware details from biological protocol and specification details and supports our goal of portability and protocol library reuse. The Hardware Layer—the external control and I/O interface of a robot—is wrapped under a Hardware Abstraction Layer (HAL). Vendor-provided software for programming the robot may be proprietary and is used to control the robot. An interface to it is provided by a software bridge, which maps protocols expressed in CRIS to sequences of native robot instructions.

The Resource Management layer maintains resource state information and provides a standardizable high-level interface for initializing, requesting, naming, aggregating, and accessing resources to the Language layer, analogous to a “system call” suite. This interface supports our goal of removing the minutiae of resource management from the protocol specification language.

3. IMPLEMENTATION

We have finished implementation of the Assembly Planner, with protocol name annotation, and will shortly begin work on PuppetShow. Our current implementation of the Puppeteer Language layer comprises an interpreter that recognizes ten primitive instructions and a library of two protocols. In its current embodiment, the interpreter is a command line program that accepts Puppeteer instructions and uses JSON objects for communication with the HAL. The interpreter can be run in one of two modes: *user* mode and *API* mode. In user mode, the user inputs a Puppeteer instruction, and the interpreter, after parsing and executing it, returns information about the result. In API mode, the interpreter uses JSON objects to communicate with the Applications layer allowing any application or programming language to interact with it. We have also begun implementation of a bridge for the Tecan Freedom Evo 150 robot. Our software stack is independent of the Tecan robot and API—we use the Tecan as a prototypical testbed for implementing the proposed stack. Our current implementation is capable of accepting a BioBrick assembly plan, linking it to a Puppeteer protocol library, and executing it on a Tecan robot or simulator.

4. RELATED AND FUTURE WORK

To our knowledge, Biocoder [1] is the only language for high-level specification of biological protocols with the goal of clarity, precision, and eventually, automation and reuse of protocol code. Biocoder is constructed as a C++ library thus allowing protocol specifications to leverage C++ features. When a protocol is compiled, a goal of the BioCoder library is to produce a human executable list of unambiguous English-language instructions equivalent to the original biological protocol. In addition to supporting these goals, Puppeteer will be the first to achieve an end-to-end translation from a high-level biological protocol specification to a sequence of actions on a robot.

Beyond protocol specification, Puppeteer also aims to aid the testing of biological protocols and ease the verification of large systems, through specialized instructions. The Resource manager may also exploit hardware parallelism and schedule actions or protocols on one or more robots efficiently. We plan to pursue these goals in subsequent versions of Puppeteer.

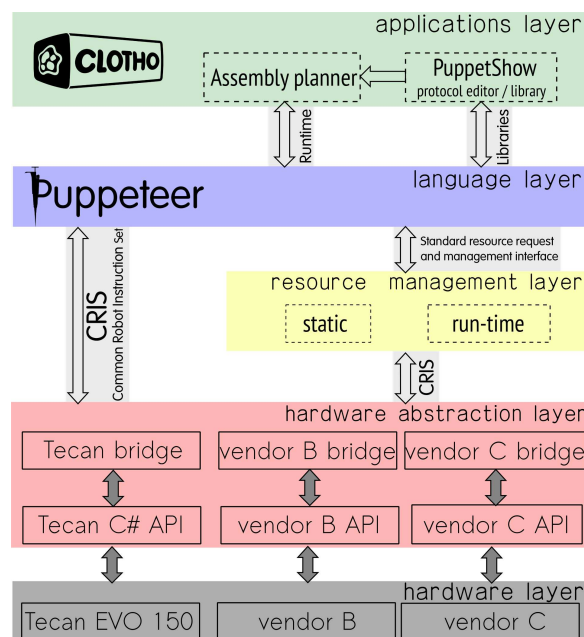


Figure 1: A software stack that abstracts away details of automated robotic assembly, enabling automated synthetic biological engineering on a variety of robotic platforms.

5. REFERENCES

- [1] ANANTHANARAYANAN, V., AND THIES, W. Biocoder: A programming language for standardizing and automating biology protocols. *J. of Biological Engineering* 4 (2010).
- [2] ANDRIANANTOANDRO, E., BASU, S., KARIG, D. K., AND WEISS, R. Synthetic biology: new engineering rules for an emerging discipline. *Mol Syst Biol* 2 (May 2006).
- [3] BEAL, J., WEISS, R., DENSMORE, D., ADLER, A., BABB, J., BHATIA, S., DAVIDSOHN, N., HADDOCK, T., YAMAN, F., SCHANTZ, R., AND LOYALL, J. TASBE: A toolchain to accelerate synthetic biological engineering. In *IWBDA* (June 2011). (submitted).
- [4] DENSMORE, D., DEVENDER, A. V., JOHNSON, M., AND SRITANYARATANA, N. A platform-based design environment for synthetic biological systems. In *TAPIA '09* (2009), ACM, pp. 24–29.
- [5] DENSMORE, D., HSIAU, T. H. C., BATTEN, C., KITTLESON, J. T., AND DELOACHE, W. Algorithms for automated DNA assembly. *Nucleic Acids Research* (2010).
- [6] ELLISON, M., RIDGWAY, D., FEDOR, J., GARSIDE, E., ROBINSON, K., AND LLOYD, D. BioBytes assembly standard. Tech. Rep. BBF RFC 47, The BioBricks Foundation, 2009.
- [7] KNIGHT, T. Idempotent vector design for standard assembly of BioBricks. Tech. rep., MIT Synthetic Biology Working Group Technical Reports, 2003.
- [8] YAMAN, F., BHATIA, S., ADLER, A., DENSMORE, D., BEAL, J., WEISS, R., AND DAVIDSOHN, N. Toward automated selection of parts for genetic regulatory networks. In *IWBDA* (June 2011). (submitted).