ACS SyntheticBiology

Letter

pubs.acs.org/synthbio

PaR-PaR Laboratory Automation Platform

Gregory Linshiz,[†,‡] Nina Stawski,[†,‡] Sean Poust,[§] Changhao Bi,[‡] Jay D. Keasling,[†,‡,§] and Nathan J. Hillson*[,†,‡,∥]

[†]Fuels Synthesis Division, Joint BioEnergy Institute, Emeryville, California 94608, United States

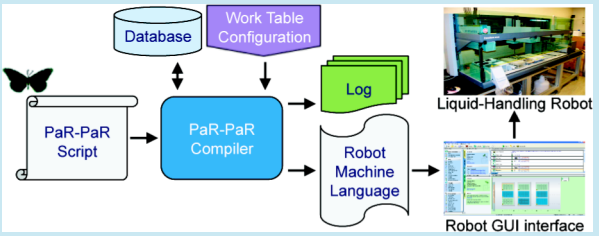
[‡]Physical Bioscience Division, Lawrence Berkeley National Laboratory, 1 Cyclotron Road Mail Stop 978R4121, Berkeley, California 94720, United States

[§]Department of Chemical & Biomolecular Engineering, Department of Bioengineering, University of California, Berkeley, California 94720, United States

[∥]DOE Joint Genome Institute, Walnut Creek, California 94598, United States

Supporting Information

ABSTRACT: Labor-intensive multistep biological tasks, such as the construction and cloning of DNA molecules, are prime candidates for laboratory automation. Flexible and biology-friendly operation of robotic equipment is key to its successful integration in biological laboratories, and the efforts required to operate a robot must be much smaller than the alternative manual lab work. To achieve these goals, a simple high-level biology-friendly robot programming language is needed. We have developed and experimentally validated such a language: Programming a Robot (PaR-PaR). The syntax and compiler for the language are based on computer science principles and a deep understanding of biological workflows. PaR-PaR allows researchers to use liquid-handling robots effectively, enabling experiments that would not have been considered previously. After minimal training, a biologist can independently write complicated protocols for a robot within an hour. Adoption of PaR-PaR as a standard cross-platform language would enable hand-written or software-generated robotic protocols to be shared across laboratories.



KEYWORDS: *synthetic biology, design automation, liquid-handling robotics, laboratory automation, high-level programming language*

Laboratory automation systems were first developed decades ago to improve the accuracies and decrease the costs and turn-around times of clinical laboratory services. Automation technologies have since been adopted by the pharmaceutical industry to screen vast chemical libraries for new drug lead compounds. Laboratory automation achieves increased productivity and lowered costs by reducing experimental error rates (eliminating the human factor) and producing more reliable and reproducible experimental data.

Laboratory automation companies largely target the highly repetitive industrial laboratory operations market and have devoted little effort to developing flexible easy-to-use programming tools for dynamic nonrepetitive research environments. To implement new protocols on robotic platforms, research biologists are dependent either on professional programmers or on vendor-supplied graphical user interfaces that often lack flexibility, force users too keep track of too many details (e.g., pipet tip logistics), and require excessive effort to modify or debug existing protocols.[1] Moreover, protocols developed for one robotic platform are not transferable to other platforms, precluding researchers from sharing protocols between laboratories, or even between different robots in the same laboratory. Since more time and effort is often required to instruct a robot to perform a new task than the robot saves a researcher by performing the task, researchers can only effectively automate a small fraction of their workflows. Nevertheless, well-funded research laboratories have invested in liquid-handling robots aiming to accelerate research, save time, and provide high-throughput solutions. While proudly shown to visitors during laboratory tours, these robots frequently remain under-utilized with very low duty cycles. To successfully integrate robotics into academic biological laboratory workflows, the efforts required to instruct and operate a robot must be much smaller than the alternative manual lab work. To achieve this goal, a simple high-level robot programming language for biologists is required.

Beyond enabling biologists to manually instruct robots in a time-effective manner, a simple high-level robot programming language would also amplify the utility of biological design automation software tools. There are several recent examples of software tools generating protocols for specific liquid-handling robotic platforms, including BglBrick assembly protocols for the Beckman Biomek 3000[2−4] and Tecan Freedom Evo,[5] PCR-setup protocols for the NextGen/eXeTek expression workstation,[6,7] Gibson/Gateway assembly protocols for the Tecan

Special Issue: IWBDA 2012

Received: August 15, 2012
Published: October 4, 2012

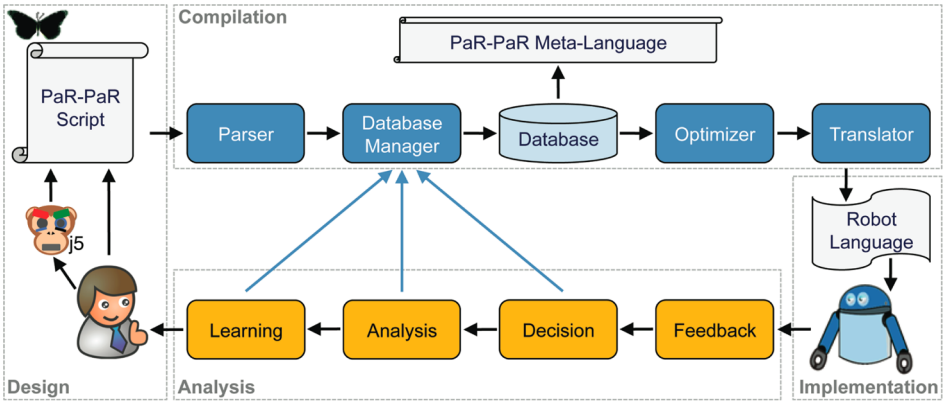


Figure 1. PaR-PaR modules and data flow. A researcher (optionally in conjunction with biological design automation software such as j5) composes a PaR-PaR script. The PaR-PaR script is parsed by the parser, which sends all definitions, declarations, and commands (translated into the PaR-PaR meta-language) to the database manager, which deposits this data in the database. The optimizer optimizes the operational flow of the sequence of commands and adapts it to the configuration of the robotic platform. The translator translates the commands from the PaR-PaR meta-language into the robotic scripting language. Finally, the robot executes the sequence of translated commands. Anticipated feedback, decision, analysis, and learning modules, slated for future development, will provide postexecution analysis to the researcher, which will feed back into the protocol design process.

Freedom Evo,[5] and recursive DNA construction and automated cloning protocols for Tecan robots.[8,9] Ideally, the protocols generated by these software tools would be executable across robotic platforms and not limited to one specific robot. As the biological design/implementation process becomes increasingly software-automated,[10−26] achieving this ideal will become increasingly important so that researchers across laboratories with different robotics platforms can all benefit from design automation software innovations.

Here, we report the development of a biology-friendly high-level robot programming language PaR-PaR (Programming a Robot). High-level languages such as PaR-PaR simplify writing and maintaining programs by making the programming code more intuitive and understandable. High-level languages can also be translated (i.e., compiled) into specific instructions for a variety of robotic platforms. The development of PaR-PaR is guided by computer science principles along with a deep understanding of biological workflows. PaR-PaR is based on an object-oriented approach that represents physical laboratory objects, including reagents, plastic consumables, and laboratory devices, as virtual objects. Each object has associated properties, such as a name and a physical location. Several objects can be grouped together to create a new composite object with its own properties, and in this way hierarchies of objects can be created. PaR-PaR allows actions to be performed on objects, and sequences of actions can be consolidated into protocols, which in turn can be issued as PaR-PaR commands. Collections of protocol definitions can be imported into PaR-PaR via external modules.

A PaR-PaR script consists of two logical sections. In the first section, the user declares and defines the robotic work table, consumables, reagents, recipes, and protocols for the current experiment. PaR-PaR has several declarative commands: *TABLE* (load the robot's work table configuration), *PLATE* (define a plates's name and location), *COMPONENT* (define a reagent's name, location, pipetting method, and other properties), *VOLUME* (define an alias for a given volume), *RECIPE* (define mixtures of liquids to prepare), and *PROTOCOL* (define a set of commands to perform). While the user currently specifies COMPONENT locations manually, PaR-PaR will notify the user, for example, if there is no such well on the plate, or if a plate has not yet been defined. Program control flow features (e.g., loops and conditionals) will be added to PROTOCOL declarations when the anticipated feedback module (Figure 1) is developed. In the second section, the user issues commands that make use of the definitions declared in the first section. PaR-PaR provides three basic liquid handling commands: *SPREAD* (transfer liquid from one location to many locations), *TRANSFER* (transfer many to many), and *PREPARE* (prepare a recipe); and the MESSAGE command, which allows for interactive features (e.g., prompting the user to change a plate on the robotic work table; see the Supporting Information file "colony_pcr.par"). Each location on the robot work table is defined by three coordinates: plate or carrier name, row, and column. We have developed a compact way to define lists of wells on the plate (Table S1, Supporting Information), as well as a few useful predefined methods for liquid transfer (i.e., liquid classes) (Table S2, Supporting Information). All three basic liquid handling commands can optionally specify pipet tip management and mixing after transfer instructions.

In addition to the development of PaR-PaR, we report its integration with biological design automation software as well as its experimental validation. We have further developed the j5 DNA assembly design automation software[6] to output PCR-setup protocols in PaR-PaR script. The 33-line j5-generated PaR-PaR script we tested (Supporting Information file "distribute_pcr.par") was compiled into a 98-line (one command per line) Tecan Freedom EVOware scripting language file (Supporting Information file "distribute_pcr.esc"). As a point of comparison, it would have required about an hour using the EVOware graphical user interface to achieve the equivalent of the j5-generated PaR-PaR script that took less than a minute to compile. For the set of 11 j5-designed PCR reactions we tested, the PCR reactions setup using a j5-output PaR-PaR script compiled for the Tecan Freedom Evo have results very similar to those setup manually (Figure S1, Supporting Information). For both manual and robotic PCR reaction preparations, 9 of the 11 PCR reactions performed very well. The two remaining PCR reactions, with the largest (10,931 bp; PCR_ID_0) and smallest (76 bp; PCR_ID_2) expected product sizes, yielded either no product or only a modest amount of product, respectively.

dx.doi.org/10.1021/sb300075t | *ACS Synth. Biol.* 2013, 2, 216−222

To explore another representative biological use-case scenario, we composed a simple PaR-PaR script to automate the setup of 144 colony PCR reactions. Following DNA assembly and transformation, colonies can be screened for the presence of constructs with correct assembly junctions with colony PCR reactions that span across assembly junctions. For complex DNA assemblies with multiple junctions (i.e., multipart DNA assembly reactions), multiple PCR reactions are required to screen each colony. For challenging DNA assembly tasks, it may be necessary to screen many colonies before identifying a correct construct. In these situations, high-throughput automated colony PCR screening is very desirable. We interrogated 48 colonies for an 8-fragment DNA assembly that should result in a 13.3-kb plasmid (Figure S2, Supporting Information). To screen each colony, three pairs of primers were designed. The first two pairs of primers span distinct pairs of contiguous assembly junctions, and the third pair of primers spans three contiguous junctions (Figure S2, Supporting Information). Three PCR master mixes (each with one of the three pairs of primers) were prepared manually. DNA templates were manually prepared by picking and boiling each of the 48 colonies. A simple 15-line PaR-PaR script (Supporting Information file "colony_pcr.par") was composed and compiled into a 307-line EVOware scripting language file (Supporting Information file "colony_pcr.esc") to direct a Tecan Feedom Evo robot to automate the preparation of the 144 (48 × 3) PCR reactions. As a hypothetical point of comparison, had we wanted to change the PCR master mix volume in each PCR reaction from 25 to 27 μL, it would have taken less than a minute to make the 3 requisite changes in the PaR-PaR script and recompile, while using the EVOware graphical user interface it would have taken over an hour to modify 162 commands. While it is not necessarily valid to directly compare PaR-PaR and Tecan scripting language line counts as relative measures of composition effort (unless writing each line for the two languages is equally demanding), enabling the user to dynamically configure protocols (by adjusting volume and component location variables) without modifying the underlying code, demonstrates a compelling time-savings advantage for PaR-PaR. The resulting PCR products were visualized in an agarose gel (Figure S3, Supporting Information). Colonies with two or more PCR reactions with DNA bands consistent with the expected product sizes were subsequently mini-prepped and further probed by restriction digest as a secondary screen (not shown). Following this secondary screen, one correct clone was obtained. Only one correct clone out of the 48 screened colonies may seem low for routine DNA construction tasks, but for an 8-fragment DNA assembly that results in a relatively large 13.3-kb plasmid, the observed efficiency is not unexpected. This particular low-efficiency DNA assembly is representative of complex DNA construction tasks that will increasingly depend on high-throughput colony PCR screening to identify correct clones.

PaR-PaR can import optional modules to take advantage of the robotic arms and peripheral devices that may be connected to a given robot. Robotic systems consist of the following major components: work table space, liquid handling arms, manipulation arms, and integrated equipment and devices. The work table can be equipped with various carriers and racks for tubes, plates, tips, and reagents. Liquid handling arms (LiHas) can consist of a various number (e.g., 1, 8, or 96) of pipettes, each capable of handling a particular range of liquid volumes that either use disposable or have fixed tips. Robotic manipulation arms (RoMas) can move objects (e.g., tubes and plates) around the work table and can load or unload them from integrated devices (e.g., a PCR machine, vacuum manifold, centrifuge, or plate reader). A PaR-PaR RoMa module (currently under development) with the command *MOVE* (move object from source to destination) could be imported for robotic platforms containing RoMas, and a PaR-PaR Vacuum module (anticipated development) with the command *VACUUM* (toggle vacuum) could be imported for platforms containing a vacuum manifold.

Going forward, we will work toward developing additional modules as well as an application programming interface (API) for PaR-PaR. Along with new translation modules (i.e., compilers for platforms other than the Tecan) that will enable the same PaR-PaR protocol to be executed across multiple robotic platforms, feedback, decision, analysis, and learning modules (Figure 1) are slated for development. The feedback module will receive experimental data from researchers, robots, or peripheral devices during protocol execution in real-time and store these data in the database. The decision module will compare measured with expected values and determine experimental success or failure given a set of deviation thresholds. The analysis module will perform cross-experiment analysis. The learning module will propose models to describe systemic or specific object behavior. The development of a PaR-PaR API will standardize communication between peripheral devices, robotic and design automation software programs, which can automatically produce protocols in PaR-PaR script or in the PaR-PaR meta-language. It is important to emphasize that the modular structure of PaR-PaR allows for these anticipated modules to be developed on top of its already useful functionality.

Finally, it is important to point out that the adoption of PaR-PaR as a standard cross platform high-level robot programming language would enable protocols written in PaR-PaR to be performed on comparable platforms across laboratories. That is to say, a researcher in one laboratory could provide reagents and a PaR-PaR script to another researcher in another laboratory to build the same construct even with different sets of robotic hardware. Such a standardization effort could be pursued as an extension to the Synthetic Biology Open Language (SBOL; http://sbolstandard.org) effort.[27] Protocols could be deposited in a public PaR-PaR repository (or exchanged directly between researchers), and all protocol changes would be captured by a PaR-PaR version control system.

METHODS

PaR-PaR Language Syntax. The complete syntax of the PaR-PaR language, presented in Backus Naur form, is documented in the PaR-PaR source code repository (https://github.com/JBEI/parpar/blob/master/developer/parpar_bnf.txt).

PaR-PaR Compiler. The PaR-PaR compiler consists of four modules: parser, database manager, optimizer, and translator (Figure 1). While the PaR-PaR compiler currently only outputs instructions for the Tecan Evo liquid-handling robotic platform, the compiler's modular structure will facilitate its further development for other robotic platforms as well as its further extension to satisfy additional user needs.

Parser. The parser parses each PaR-PaR input script and identifies the user's declarations and definitions, such as those

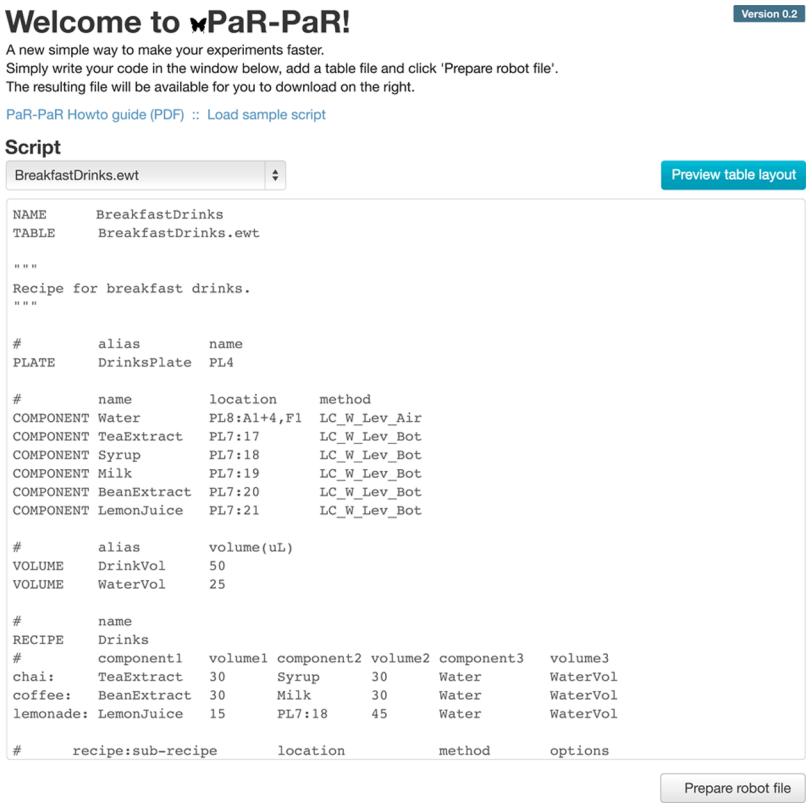


Figure 2. PaR-PaR web interface and script example. The web interface provides a link to the "How-to write a PaR-PaR script" user's guide and provides the user with the opportunity to load a built-in example script (shown here) to quickly get started. The user can either choose from a built-in set of table configurations or select a robot work table configuration file to upload. Pressing the "Preview table layout" button visually renders the robotic work table configuration (Figure 3). The PaR-PaR script can be edited in the web interface itself or copy/pasted from another software tool. Pressing on the "Prepare robot file" button sends the PaR-PaR script and the work table configuration file to the compiler, which triggers the compilation process (Figure 4). The resulting robot scripting language file output can then be downloaded by clicking on the "Download" button (not shown).

relating to the robotic work table, components, volumes, and recipes. The parser sends these definitions and declarations to the database manager (see below). The parser additionally recognizes and decomposes procedures (if any have been defined) into basic PaR-PaR commands, and then translates all commands into the PaR-PaR meta-language. The PaR-PaR meta-language represents a sequence of object transfers in terms of "transfer units", and may reference external devices. Each "transfer unit" consists of a source, destination, quantity (measured in microliters) and transfer method.

Database Manager. All PaR-PaR virtual objects and PaR-PaR scripts (translated into the PaR-PaR meta-language) are stored in the database. All sources and destinations are linked to objects stored in the database. During the experiment, each operation with objects (such as reagents, devices, data and etc) triggers object tracking that updates the object's history record. An object's history contains descriptions of all the events that the object was involved in. At compilation time, the compiler updates the object parameters in the database and produces log files that reflect the changes in object parameters expected after running the protocol.

Optimizer. The optimizer optimizes the operational flow and adapts it to the specific configuration of the given robotic platform. The optimizer receives a list of transactions (in the PaR-PaR meta-language) from the Database Manager. Optimization in PaR-PaR is currently limited to optimizing sequential liquid transfers. For example, if the optimizer recognizes that the specified volume in a "transfer unit" is larger than the maximum possible volume allowed for a given pipet, the optimizer subdivides the transfer unit into multiple transfer units with allowable volumes and manages the order of each transfer. For platforms with different pipet tips, the optimizer identifies which tip size is the most suitable for each transfer. The optimizer also reduces the number of the robot arm movements by reordering transfer units, while maintaining the temporal logic of the overall protocol. The optimizer achieves this in part by clustering together sequential liquid transfers according to the number of tips concurrently available to the robot, which allows for the use of more than one tip at the time. Refer to the PaR-PaR source code repository (see below) for additional details concerning how this optimization process has been implemented.

Translator. The translator currently translates a protocol composed in the PaR-PaR meta-language into the Tecan Evo robotic platform scripting language. The translator could be further developed to translate into one of many low-level robotic scripting languages or human languages, depending on the translation module. For semiautomated protocols, automatable portions could be translated into a robotic scripting language and the off-line/manual portions into a human language.

PaR-PaR Interfaces, Workflow, License, and Availability. PaR-PaR has two primary interfaces: a web interactive mode and a command line mode. The web interactive mode

dx.doi.org/10.1021/sb300075t | *ACS Synth. Biol.* 2013, 2, 216–222

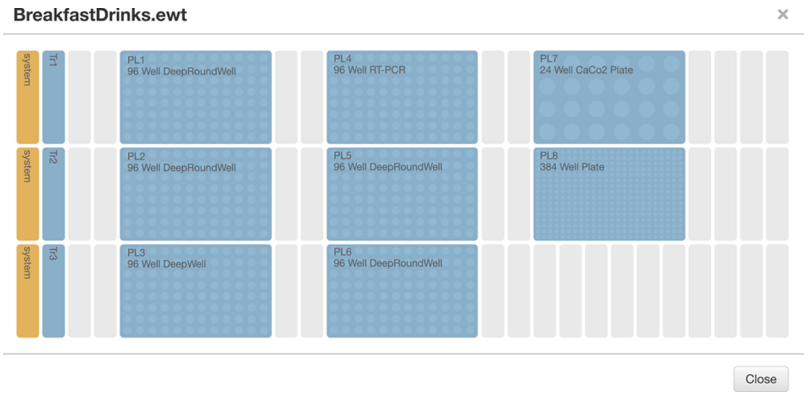


Figure 3. PaR-PaR web interface robotic table preview.

(Figures 2, 3) provides a visual editor that facilitates the composition of PaR-PaR scripts. The PaR-PaR workflow is shown in Figure 4. PaR-PaR is open-source software under the

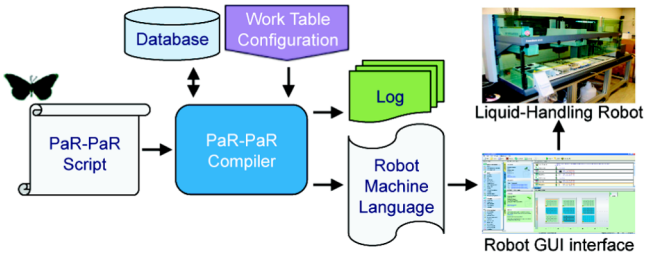


Figure 4. PaR-PaR workflow. The compiler takes a PaR-PaR script and a work table configuration file as input, and following compilation, outputs a robot scripting language file. Following this step, the output file is loaded into the vendor-supplied robot GUI interface software (e.g., Tecan Freedom EVOware). The researcher arranges all reagents and labware on the robotic work table according to the protocol and then runs the experiment.

BSD license, is freely available from GitHub (https://github.com/jbei/parpar), and is also available through its web interface on the public PaR-PaR webserver (http://parpar.jbei.org).

PaR-PaR Software Implementation. PaR-PaR is implemented in the Python 3 (http://www.python.org/) programming language.

PaR-PaR Objects, Memory Allocation, and Variable Scope. At the start of parsing a PaR-PaR script, PaR-PaR creates a new Experiment object with an assigned unique ID. Initially, the Experiment object contains empty dictionaries for each type of PaR-PaR object (plate, well, component, volume, recipe, protocol). Python class constructors automatically allocate memory for the Experiment object and all other PaR-PaR objects. At the end of parsing the PaR-PaR script, the Experiment object is passed to the Database Manager. The database stores the Experiment object's information in PaR-PaR meta-language. Variable scope is at the PaR-PaR script/Experiment object level.

Database. The current database solution is SQLite3 (http://www.sqlite.org/), a lightweight software library available as a built-in Python module, which implements a self-contained, server-less, transactional SQL database engine. SQLite3 stores data in the file system, and therefore requires little effort to set up. Given the modularity of PaR-PaR, SQLite could be readily substituted with a classical relational database such as PostgreSQL (which would, however, require a more demanding initial setup process). Data is stored in the database in the fourth normal form, with table attributes representing objects that are defined within the PaR-PaR configuration script. This simplifies access to data within PaR-PaR.

Command Line Interface. The command line interface utilizes argparse, a Python 3.2 built-in module (http://docs.python.org/dev/library/argparse.html), to handle user input (e.g., files) and allow the user to specify additional arguments when running PaR-PaR (e.g., configuration file name, logging enabled/disabled). Since Python is a cross-platform programming language, the command line interface is operating system independent.

Web Interface. The web interface (Figures 2, 3) utilizes bottle.py (http://bottlepy.org), either independently or in conjunction with Apache/mod_wsgi, and is built with jQuery (http://jquery.com/), HTML5, CSS3 (http://www.w3.org/), and Twitter Bootstrap (http://twitter.github.com/bootstrap), allowing for a cross-platform browser-independent seamless no-page-refresh user-experience.

j5 License and Availability. j5 is available at no cost to noncommercial (e.g., academic, nonprofit, or government) users under a Lawrence Berkeley National Lab end-user license agreement (http://j5.jbei.org/index.php/License). The software is available through the public j5 webserver (http://j5.jbei.org). The j5 software has been exclusively licensed to TeselaGen Biotechnology, Inc. (http://teselagen.com) for commercial use and distribution.

j5 Software Implementation. The core software implementation of j5 has been reported previously.[6] For the purposes of the further development required for j5 to output PaR-PaR scripts, it suffices to mention that j5 is written in the Perl programming language (http://www.perl.org/) and draws upon the Text::CSV_XS library available from Comprehensive Perl Archive Network (CPAN, http://www.cpan.org) repository to output tab-delimited text files. j5 PaR-PaR output is documented in the j5 user's manual (http://j5.jbei.org/j5manual/pages/98.html). The JBEI-ICE biological part registry platform[15] is indirectly integrated with PaR-PaR via VectorEditor → DeviceEditor[7] → j5 → PaR-PaR. Further integration between PaR-PaR and JBEI-ICE will be pursued when sample tracking functionality is integrated into PaR-PaR.

Sequence Availability. DNA sequences (pj5_00047, pSP5, pSP8, pSY49, pSY055, and pSY61), along with their associated information (annotated Genbank-format sequence files, and colony PCR DNA oligo sequences) have been deposited in the public instance of the JBEI Registry (https://

public-registry.jbei.org; corresponding Part IDs JPUB_000476-81).

■ ASSOCIATED CONTENT

Supporting Information

Supporting tables, methods, figures, and files. This material is available free of charge via the Internet at http://pubs.acs.org.

■ AUTHOR INFORMATION

Corresponding Author

*Tel: +1 510 486 6754. Fax: +1 510 486 4252. E-mail: njhillson@lbl.gov.

Author Contributions

G.L. and N.S. designed and developed the PaR-PaR software; N.J.H. designed and developed the j5 software to output PaR-PaR scripts; G.L., N.S., and N.J.H. wrote the "How-to write a PaR-PaR script" user's guide; G.L. and S.P. performed the j5-designed PCR reaction setup experiments; G.L. and C.B. performed the colony PCR setup experiments; and G.L., N.S., S.P., C.B., J.D.K, and N.J.H. wrote the manuscript.

Notes

The authors declare the following competing financial interest(s): N.J.H declares competing financial interests in the form of pending patent applications related to the j5 software, and equity in TeselaGen Biotechnology, Inc., whose value may be affected by the publication of this article. Otherwise, the authors declare no competing financial interests.

■ ACKNOWLEDGMENTS

The authors thank Steve Lane for providing information technology support; Peter Su for assistance in the construction of plasmid pj5_00047; Satoshi Yuzawa for providing plasmid templates pSY49, pSY055, and pSY61; and Joanna Chen, James Carothers, and Vivek Mutalik for constructive comments on the manuscript. This work conducted by the Joint BioEnergy Institute, and the U.S. Department of Energy Joint Genome Institute was supported by the Office of Science, Office of Biological and Environmental Research, of the U.S. Department of Energy (Contract No. DE-AC02−05CH11231); the Department of Energy, ARPA-E Electrofuels Program (Contract No. DE-0000206−1577); the National Science Foundation Graduate Research Fellowship Program (Grant No. DGE 1106400, to S.P.); and the Berkeley Laboratory Directed Research and Development Program (to N.J.H.).

■ REFERENCES

(1) Gu, H., Unger, S., and Deng, Y. (2006) Automated Tecan programming for bioanalytical sample preparation with EZTecan. *Assay Drug Dev. Technol. 4*, 721−733.

(2) Leguia, M., Brophy, J., Densmore, D., and Anderson, J. C. (2011) Automated assembly of standard biological parts. *Methods Enzymol. 498*, 363−397.

(3) Xia, B., Bhatia, S., Bubenheim, B., Dadgar, M., Densmore, D., and Anderson, J. C. (2011) Developer's and user's guide to Clotho v2.0 A software platform for the creation of synthetic biological systems. *Methods Enzymol. 498*, 97−135.

(4) Densmore, D., Hsiau, T. H., Kittleson, J. T., DeLoache, W., Batten, C., and Anderson, J. C. (2010) Algorithms for automated DNA assembly. *Nucleic Acids Res. 38*, 2607−2616.

(5) Beal, J., Weiss, R., Densmore, D., Adler, A., Appleton, E., Babb, J., Bhatia, S., Davidsohn, N., Haddock, T., Loyall, J., Schantz, R., Vasilev, V., and Yaman, F. (2012) An end-to-end workflow for engineering of biological networks from high-level specifications. *ACS Synth. Biol. 1*, 317−331.

(6) Hillson, N. J., Rosengarten, R. D., and Keasling, J. D. (2012) j5 DNA assembly design automation software. *ACS Synth. Biol. 1*, 14−21.

(7) Chen, J., Densmore, D., Ham, T. S., Keasling, J. D., and Hillson, N. J. (2012) DeviceEditor visual biological CAD canvas. *J. Biol. Eng. 6*, 1.

(8) Ben Yehezkel, T., Nagar, S., Mackrants, D., Marx, Z., Linshiz, G., Shabi, U., and Shapiro, E. (2011) Computer-aided high-throughput cloning of bacteria in liquid medium. *BioTechniques 50*, 124−127.

(9) Linshiz, G., Yehezkel, T. B., Kaplan, S., Gronau, I., Ravid, S., Adar, R., and Shapiro, E. (2008) Recursive construction of perfect DNA molecules from imperfect oligonucleotides. *Mol. Syst. Biol. 4*, 191.

(10) Beal, J., Lu, T., and Weiss, R. (2011) Automatic compilation from high-level biologically-oriented programming language to genetic regulatory networks. *PLoS One 6*, e22490.

(11) Bilitchenko, L., Liu, A., Cheung, S., Weeding, E., Xia, B., Leguia, M., Anderson, J. C., and Densmore, D. (2011) Eugene—a domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS One 6*, e18882.

(12) Cai, Y., Wilson, M. L., and Peccoud, J. (2010) GenoCAD for iGEM: a grammatical approach to the design of standard-compliant constructs. *Nucleic Acids Res. 38*, 2637−2644.

(13) Chandran, D., Bergmann, F. T., and Sauro, H. M. (2009) TinkerCell: modular CAD tool for synthetic biology. *J. Biol. Eng. 3*, 19.

(14) Chandran, D., Bergmann, F. T., Sauro, H. M., and Densmore, D. (2011) Computer-aided design for synthetic biology, in *Design and Analysis of Bio-molecular Circuits* (Koeppl, H., Densmore, D., di Bernardo, M., and Setti, G., Eds.) 1st ed., pp 203−224, Springer-Verlag, New York.

(15) Ham, T. S., Dmytriv, Z., Plahar, H., Chen, J., Hillson, N. J., and Keasling, J. D. (2012) Design, implementation and practice of JBEI-ICE: an open source biological part registry platform and tools. *Nucleic Acids Res.*, DOI: 10.1093/nar/gks531.

(16) Lux, M. W., Bramlett, B. W., Ball, D. A., and Peccoud, J. (2012) Genetic design automation: engineering fantasy or scientific renewal? *Trends Biotechnol. 30*, 120−126.

(17) MacDonald, J. T., Barnes, C., Kitney, R. I., Freemont, P. S., and Stan, G. B. (2011) Computational design approaches and tools for synthetic biology. *Integr. Biol. 3*, 97−108.

(18) Olsen, L. R., Hansen, N. B., Bonde, M. T., Genee, H. J., Holm, D. K., Carlsen, S., Hansen, B. G., Patil, K. R., Mortensen, U. H., and Wernersson, R. (2011) PHUSER (Primer Help for USER): a novel tool for USER fusion primer design. *Nucleic Acids Res. 39*, W61−67.

(19) Richardson, S. M., Liu, S., Boeke, J. D., and Bader, J. S. (2012) Design-A-Gene with GeneDesign. *Methods Mol. Biol. 852*, 235−247.

(20) Salis, H. M., Mirsky, E. A., and Voigt, C. A. (2009) Automated design of synthetic ribosome binding sites to control protein expression. *Nat. Biotechnol. 27*, 946−950.

(21) Bates, J. T., Chivian, D., and Arkin, A. P. (2011) GLAMM: genome-linked application for metabolic maps. *Nucleic Acids Res. 39*, W400−405.

(22) Carothers, J. M., Goler, J. A., Juminaga, D., and Keasling, J. D. (2011) Model-driven engineering of RNA devices to quantitatively program gene expression. *Science 334*, 1716−1719.

(23) Copeland, W. B., Bartley, B. A., Chandran, D., Galdzicki, M., Kim, K. H., Sleight, S. C., Maranas, C. D., and Sauro, H. M. (2012) Computational tools for metabolic engineering. *Metab. Eng. 14*, 270−280.

(24) Dymond, J. S., Richardson, S. M., Coombes, C. E., Babatz, T., Muller, H., Annaluru, N., Blake, W. J., Schwerzmann, J. W., Dai, J., Lindstrom, D. L., Boeke, A. C., Gottschling, D. E., Chandrasegaran, S., Bader, J. S., and Boeke, J. D. (2011) Synthetic chromosome arms function in yeast and generate phenotypic diversity by design. *Nature 477*, 471−476.

(25) Hillson, N. J. (2011) DNA assembly method standardization for synthetic biomolecular circuits and systems, in *Design and Analysis of Bio-molecular Circuits* (Koeppl, H., Densmore, D., di Bernardo, M., and Setti, G., Eds.) 1st ed., pp 295−314, Springer-Verlag, New York.

(26) Shabi, U., Kaplan, S., Linshiz, G., Benyehezkel, T., Buaron, H., Mazor, Y., and Shapiro, E. (2010) Processing DNA molecules as text. *Syst. Synth. Biol. 4*, 227−236.
(27) Galdzicki, M., Rodriguez, C., Chandran, D., Sauro, H. M., and Gennari, J. H. (2011) Standard biological parts knowledgebase. *PLoS One 6*, e17005.