Composable Modular Models for Synthetic Biology

GOKSEL MISIRLI, JENNIFER HALLINAN, and ANIL WIPAT, Newcastle University

Modelling and computational simulation are crucial for the large-scale engineering of biological circuits since they allow the system under design to be simulated prior to implementation *in vivo*. To support automated, model-driven design it is desirable that *in silico* models are modular, composable and use standard formats. The synthetic biology design process typically involves the composition of genetic circuits from individual parts. At the most basic level, these parts are representations of genetic features such as promoters, ribosome binding sites (RBSs), and coding sequences (CDSs). However, it is also desirable to model the biological molecules and behaviour that arise when these parts are combined *in vivo*. Modular models of parts can be composed and their associated systems simulated, facilitating the process of model-centred design. The availability of databases of modular models is essential to support software tools used in the model-driven design process. In this article, we present an approach to support the development of composable, modular models for synthetic biology, termed Standard Virtual Parts. We then describe a programmatically accessible and publicly available database of these models to allow their use by computational design tools.

Categories and Subject Descriptors: J.3 [**Life and Medical Sciences**]: *Biology and genetics*; J.6 [**Computer-Aided Engineering**]: *Computer-aided design (CAD)*

General Terms: Design, Standardization

Additional Key Words and Phrases: Synthetic biology, model-driven design, composable models, model annotation, database of models, standard virtual parts

ACM Reference Format:
Goksel Misirli, Jennifer Hallinan, and Anil Wipat. 2014. Composable modular models for synthetic biology. ACM J. Emerg. Technol. Comput. Syst. 11, 3, Article 22 (December 2014), 19 pages.
DOI: http://dx.doi.org/10.1145/2631921

1. INTRODUCTION

One ambition of synthetic biology is the large-scale engineering of biological systems [Fritz et al. 2010; Hallinan et al. 2010; Young and Alper 2010]. This goal extends conventional genetic engineering to the design and implementation of entire pathways and even whole genomes [Liang et al. 2011]. It is therefore necessary to facilitate biological engineering at a genomic scale, building on developments in high-throughput biology. As the complexity and size of designs increases, the manual design of genetic circuits becomes more challenging and the use of automated strategies and new computational tools becomes important [Misirli et al. 2011]. Computational approaches are highly desirable for designing complex systems and for improving the predictability of the behaviour of an *in silico* system once implemented *in vivo* [Smolke and Silver 2011].

The mapping between the desired properties of a large complex system and the genetic parts necessary to encode the system *in vivo* is difficult, if not impossible for

This work is supported by the Engineering and Physical Sciences Research Council under grant EP/J02175X/1 and the Engineering and Physical Sciences Research Council/National Science Foundation under grant EP/H019162/1.
Authors' address: School of Computing Science and Centre for Synthetic Biology and Bioexploitation, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK. Correspondence email: anil.wipat@ncl.ac.uk.

1550-4832/2014/12-ART22
DOI: http://dx.doi.org/10.1145/2631921

a human to carry out manually. In particular, dynamic models of a system promise to help optimise the design of the system by predicting the behaviour of the system before implementation *in vivo*. Such models seek to capture *in silico* representations of the biological entities involved in the system, the abundance of those entities, their dynamic behaviour and the relationships between the entities. Models can thus potentially act as blueprints for the design of a system to be implemented, since they already contain information about the entities required in a system – information exploited in so-called model-driven design.

Model-driven design is used extensively in a number of different engineering domains including the software, automotive, and aerospace sectors [Feiler et al. 2005]. Models can be constructed to represent the relationships between elements in a system and consequently used to derive the process by which the system is generated [Balasubramanian et al. 2006]. Using this approach, models are transformed into platform-specific electronic designs. For example, in embedded electrical systems, a model that represents a system with a set of requirements can be used to automatically generate the hardware implementation of the system [Henzinger and Sifakis 2006]. As with genetic circuits, these systems are built with individual components such as transistors and logic gates, and the output of one component must be compatible with the input of the next component in order to allow the flow of data among the components. Components can be represented with transfer functions that produce outputs for given inputs using a set of equations. The integration of a set of equations from these transfer functions allows models to be produced that can be simulated in order to choose among and verify designs. The manual creation of simple electronic circuits was long ago replaced with the use of computer-aided design (CAD) tools and design automation. As a result, electronic design automation has become an industry in which very-large-scale integrated circuits can be constructed from well-defined components and subsystems that are electrically and physically correct [MacMillen et al. 2000]. This approach is clearly applicable to the design of synthetic genetic circuits.

Several computational design tools that allow synthetic genetic circuits to be designed and simulated have been developed [Cai et al. 2007; Chandran et al. 2009; Funahashi et al. 2003; Kaznessis 2007; Li et al. 2010; Marchisio and Stelling 2008; Myers et al. 2009; Pedersen and Phillips 2009; Rodrigo et al. 2007a, 2007b]. These tools often use libraries of mathematical models of biological parts in order to aid a user in building complex and predictable designs [Hill et al. 2008; Pedersen and Phillips 2009]. However, there is a lack of modular and reusable models of biological parts in standard formats [Cooling et al. 2010]. The availability of databases of such models would provide a useful computational resource to support tools for the design of synthetic genetic circuits. These databases must store information about parts and their interactions, such as how they function together and regulate each other. These approaches can be used to constrain the possible solution space for computational designs. Qualitative information about parts and their interactions can be used to optimise the computational design of genetic circuits [Zomorrodi and Maranas 2014]. Moreover, by annotating parts with quantitative parameters, the dynamics of systems can be simulated [Endler et al. 2009; Stelling 2004].

Modular modelling approaches for synthetic biology have been demonstrated in several previous studies [Marchisio and Stelling 2008; Rodrigo et al. 2007a, 2011]. However, there is not always a one-to-one mapping between entities in these models and the biological parts used to build actual systems *in vivo* [Rodrigo et al. 2011]. Although these approaches include a formalism for representing models of parts, the number of models available is small. Whilst libraries of models are available for systems biology [Li et al. 2010], there are very few models available for synthetic biology. There is a clear need for computationally accessible and searchable libraries of models of parts for genetic circuit design.

In modular modelling, models represent mathematical representations of biological reactions; however, these models usually do not contain information about the biological entities they represent. As a result, the composition of models is often application specific. In order to automate the process of model composition and to make the composable models available to existing tools, these models may be annotated with metadata. Such metadata can include information about how to interpret and compose these models using computational approaches.

Modular models of parts and their interactions can also be based on standard formats at appropriate levels of abstraction. The Systems Biology Modelling Language (SBML) [Hucka et al. 2003] and CellML [Cuellar et al. 2003] are two modelling languages that allow the computational exchange of models. These languages are also actively used by the synthetic biology community [Marchisio and Stelling 2009]. SBML is supported by 257 tools at the time of writing.[1] Both languages are XML based, allowing the electronic exchange of models between computer applications and supporting the writing of ordinary differential equation (ODE) representations of biological reactions.

We have previously described a modelling formalism for biological systems design, Standard Virtual Parts (SVPs). SVPs are abstract models that are designed to represent basic, physical biological parts such as promoters, RBSs, and CDSs *in silico* [Cooling et al. 2010]. Intracellular events such as biochemical processes and gene products that influence the behaviour of a genetic circuit can also be represented using SVPs. SVPs have standard inputs and outputs that allow the construction of models of large systems to be built from basic biological parts. The interfaces are based on widely accepted biological signals such as polymerases per second (PoPS) and ribosomes per second (RiPS) [Braff et al. 2005] as described in previous modular modelling approaches in synthetic biology [Marchisio and Stelling 2008; Rodrigo et al. 2007a]. Sharing these common signals makes the models composable [Marchisio and Stelling 2009].

This previous work provides the theoretical foundation for a model-based design approach in synthetic biology, but it has drawbacks. First, the modelling of a single biological part requires joining several SVPs together. SVPs, in this approach, refer to any model fragment that models some biology, including molecular interactions. Therefore, composition of models is not straightforward and requires the tracking of different reaction fluxes for every part. Second, SVPs refer to models of parts and also their interactions. Separating models of parts and their interactions with other parts would facilitate constructing higher-level systems using a modular approach. Third, and most importantly, computational composition of models using these SVPs is not possible, since the metadata needed to facilitate such a process has not been defined. Finally, although a repository was made available as part of this original work, there were only few SVPs that could be used in model composition. In addition, the repository was not available for programmatic access, and models could only be browsed manually. In order to construct large, complex biological systems, large amount of information about biological parts and their interactions from the literature and public databases should be integrated and made available in standard formats. Moreover, the requirement for a model repository is different in systems biology approaches. For example, the BioModels database [Li et al. 2010] provides models that can be simulated or can be used in computational analyses. However, for a model-driven design approach, models may be incomplete and may require additional information for the model composition process. Moreover, in order to choose model fragments computationally, models should be associated with functional terms. For example, searching for models of promoters

[1]http://sbml.org/SBML_Software_Guide.

should be possible based on whether the promoters are inducible or repressible, or using their relative strength.

In this work, we extend the SVP concept using annotations to encapsulate biochemical reactions that are specific to the biological part being modelled, allowing a single SVP to be provided for a single genetic part. Annotations are also used to specify the inputs and outputs of SVPs. We defined additional types of operators, spacer sequences and promoters. We extended the SVP concept to other modelling formalisms, including SBML. A public, programmatically accessible repository called the Virtual Parts Repository[2] was developed to provide access to this library of SVPs. This repository was populated by integrating data from several sources and allows different search options for different types of parts. The Repository also stores models of interactions which can be used to combine models of individual parts in order to create simulatable models.

2. METHODS

2.1. SVP Types

The modular nature of SVPs allows new templates to be defined at any desired level of abstraction. In this work, the set of SVP types was extended beyond those described previously [Cooling et al. 2010] to include operators and shims (spacer sequences). A set of promoters acting as two-input logic gates was also defined. The modelling of these promoters takes their relationships with binding transcription factors (TFs) into account. Different types of logic gates, such as NAND and NOR, are modelled based on physical constraints such as the proximity and strength of operators and promoters. SVPs were also extended to encapsulate biochemical reactions, in order to create a one-to-one mapping between SVPs and corresponding sequence features.

2.2. Mapping SVPs to Genetic Elements

In this work, SVPs are constructed as coarse-grained models with inputs and outputs. Entities representing reactions that are specific to an SVP are encapsulated in the model in order to simplify the process of connecting SVPs. For example, recurring entities such as protein translation and degradation elements are incorporated inside the SVPs and thus do not need to be added explicitly when SVPs are joined together.

CDSs and the behaviour of their encoded products are modelled using `FunctionalPart` SVPs. An SVP of type `FunctionalPart` has a RiPS input that can be connected to an RBS SVP. The output of such a `FunctionalPart` SVP may consist of one default entity, but may also include modified forms of gene products. For example, for a protein that can be phosphorylated, both its unmodified and phosphorylated forms are the output of the SVP. Additional inputs can also be defined. For a protein that is induced by an environmental factor, the concentration of the inducer molecule can be used as an input, assuming that the interaction between the inducer and the protein can be abstracted using a Hill-like equation [Alon 2006], so that it can be encapsulated inside the SVPs.

SVPs encapsulate the entities for gene products and their production. The interactions between gene products, environmental factors and biochemical reactions such as degradation and deactivation are modelled as internal interactions. Figure 1 depicts an SVP for a CDS and its encoded protein product. The RiPS signal is converted to a protein flux via a translation reaction. The concentration of the protein is represented by a model entity, such as a species in SBML. The protein is induced by a small molecule which acts as an input to the SVP. The reaction which models the activation of the

[2]http://www.virtualparts.org.

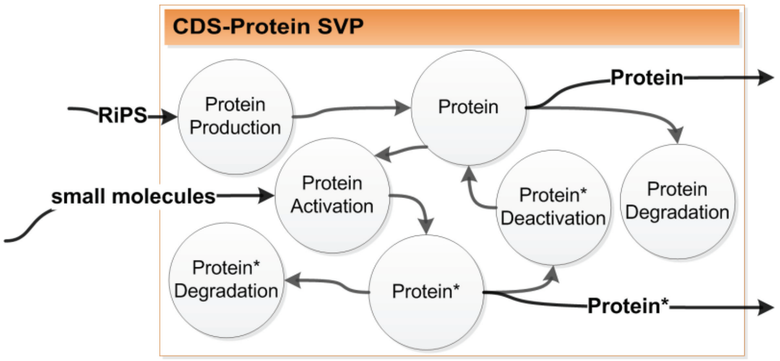


Fig. 1. An example of an SVP for a CDS and its encoded protein. Protein* represents the activated form of the protein in the SVP. The SVP includes the entities for protein production, and biochemical reactions such as activation, degradation, and deactivation.

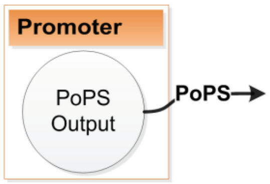


Fig. 2. Promoter SVPs are PoPS generators.

protein links the unmodified and activated forms. The flux from this reaction is an increase in the number of molecules of the activated form of the protein, and a reduction in molecular levels of the unmodified protein. Both forms of the protein are also linked through the deactivation interaction. In the example in Figure 1, the degradation of both forms of the protein is encapsulated as part of the SVP. The SVP has outputs that represent the unmodified and the phosphorylated form of the protein. These outputs can participate in models of interactions that join different SVPs together.

Modular models can also be defined for interactions that are not encapsulated within SVPs. Such interactions are between biological molecules that are represented in different SVPs. Examples include the phosphorylation of a response regulator by a kinase, and the binding of a TF to a promoter in order to activate transcription. In this work, these interactions are represented by distinct models, which are referred to as models of interactions, and are used to join SVPs in order to create simulatable models. For example, in Figure 1, the SVP has outputs that represent the unmodified and the phosphorylated form of the protein. These outputs can be used to connect the SVP to another by participating in an interaction model that represents a phosphorylation interaction.

A promoter SVP is simply a PoPS generator (Figure 2), and mRNA SVPs must be connected to the PoPS output of a relevant SVP. This approach decouples the use of promoters and other parts such as operators that affect the final magnitude of PoPS signals. Relationships between promoters and TFs are represented as models of interactions. These models have PoPS inputs and outputs that are calculated using the PoPS inputs and the promoter occupancies of TFs. This approach also provides more flexibility in the design of genetic circuits, since different TFs can be used to control the activity of one promoter. A similar approach is also used for operator SVPs, in which models of interactions for operator and TF SVPs represent the operator occupancies of TFs.

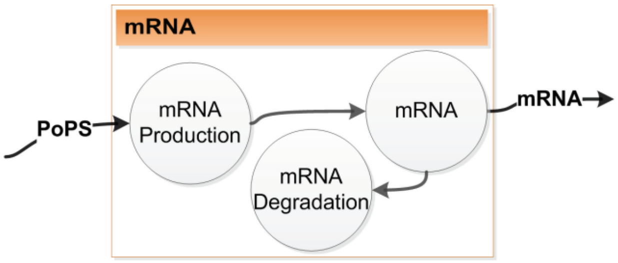


Fig. 3. An mRNA SVP contains modelling entities for mRNA production, degradation and mRNA.

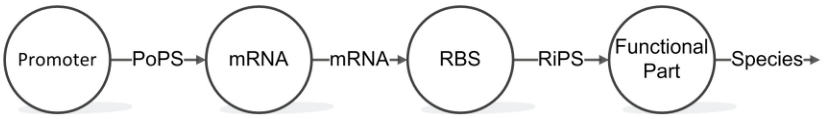


Fig. 4. Interfacing SVPs with common signal carriers.

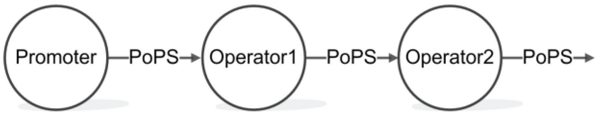


Fig. 5. An example of an interface between a promoter and two operators. Operators independently modulate the PoPS signal.

An mRNA SVP includes entities for mRNA production, degradation and the mRNA itself (Figure 3). Although this SVP does not have an equivalent physical part at the DNA level, it is required to connect the modelling of transcription and translation by transforming PoPS into RiPS.

The definition of inputs and outputs, as demonstrated here, enables the composition of models that encapsulate detailed biochemical reactions.

2.3. Composing Virtual Systems from SVPs

SVPs are specified with inputs and outputs that refer to widely accepted biological signals in synthetic biology, such as PoPS (transcriptional) and RiPS (translational), and the level of a particular biological species. Therefore, as biological parts that send and receive these signals can be combined based on the exchange of these signals, SVPs can be combined using these inputs and outputs. Promoter SVPs have PoPS outputs that are converted into mRNAs (Figure 4). RBS SVPs have mRNAs as the input and produce RiPS as the output, which can then be used as the input for proteins. Each molecular form of a protein is represented as output. These signals specify how SVPs can be combined without handling fine-grained internal modelling entities and their relationships.

Operator and promoter SVPs can be concatenated to amplify or reduce PoPS signals. In our model, these parts do not overlap in genetic context, and their respective activators or repressors act independently [Bintu et al. 2005]. This assumption allows the use of operator parts regardless of the nature of upstream and downstream sequences [Alon 2006] (Figure 5). The PoPS signal from the final design component can then be added to an mRNA SVP in order to convert transcriptional level information to translational-level information. In cases where operators overlap or TFs bind cooperatively, operators are modelled as part of promoters in order to reflect the combined transfer functions.

Table I. RDF Properties with which Entities can be Annotated

Property	Possible values	Note
mts:InterfaceType	Input; Output	
mts:SignalType	PoPS; mRNA; RiPS; Species; EnvironmentConstant; Volume	Enables the composition of models based on biological signal carriers
mts:MolecularForm	Default; Phosphorylated; Dimer; Tetramer	This list can be extended
mts:Species		Species that represent physical molecules can be linked to these molecules with this property

2.4. Computational Composition of SVPs

SVPs can be joined manually to create models of systems. However, this process is time consuming and error-prone. Ideally, SVPs should be joined computationally, for example using CAD tools or biospecific composition languages, to facilitate large-scale, model-driven design. Although it is obvious to an expert that a CDS should be placed after an RBS, to enable computational design the flow of signals between biological parts must be defined in a computationally comprehensible format.

In order to facilitate the computational composition of SVPs, SVPs are annotated with machine-readable metadata about inputs and outputs. These annotations are embedded in the XML structure of SVPs, and can be exchanged between tools with no loss of information, using a formalism such as RDF [Endler et al. 2009]. Each SVP is annotated with a list of inputs and outputs that can be mapped to the relevant entities. These annotations consist of resource-property-value RDF triples. In each triple the resource is the SVP being annotated, the property is whether the interface is an input or output, and the value is the modelling entity referred to by the annotation. Properties belong to a special namespace[3] and are prefixed with `mts`. The uniform resource identifier (URI) for the namespace does not exist physically; however, it is used to construct unique URIs. Input and output properties are specified with `mts:Input` and `mts:Output` URIs.

Modelling entities are annotated with additional information in the form of RDF/XML, and are stored in the XML structure of the corresponding modelling entities. This additional information is in the form of RDF properties (Table I).

2.5. Using Integrated Datasets to Construct SVPs

One approach to expanding the number of genetic parts available for synthetic biology is to mine the definitions (sequence, function, etc.) for these parts from existing biological information which is already publicly available [De Las Heras et al. 2010; Misirli 2013]. Furthermore, the creation of *in silico* models that define the behaviour of these mined parts allows designs to be tested *in silico* via simulation. However, biological data are often heterogeneous and spread in multiple databases, and therefore needs to be integrated and presented to computational tools in suitable formats.

One such integrated dataset, constructed specifically for the model Gram-positive organism *Bacillus subtilis,* is BacillOndex [Misirli et al. 2013]. This database includes information about sequence-based genetic features that can be used as biological parts, together with biological interactions between these parts and annotations. SVPs were constructed for promoters, operators and CDSs encoding TFs from this dataset. Information about relationships such as the TF regulation of promoters and the TF binding of operators is incorporated. Interaction models can be used to join these SVPs

[3]http://purl.org/modeltosequence/1.0#.

together. Sequences for shims, RBSs, and terminators were also extracted as basic biological parts.

Some of these TFs are response regulators that are part of two-component systems. Such TFs must be activated by kinase proteins via phosphorylation. Therefore, SVPs were also constructed to represent enzymes such as kinases. Phosphorylation reactions between these proteins are represented as modular models. Different molecular forms of proteins were incorporated into the SVPs based on the types of their interactions. Metadata about SVPs also include Gene Ontology (GO) terms [The Gene Ontology Consortium 2001] and data from Clusters of Orthologous Groups (COGs) database, which are used to assign orthology group memberships to gene products and thereby indicate protein function [Tatusov et al. 2001].

SVPs are derived from templates which are instantiated using different quantitative parameters. The templates specify the rules regarding particular types of molecular interactions, and SVPs include models of these interactions for different biological entities. Although biochemical parameters are important for simulations [Silva-Rocha and de Lorenzo 2010], obtaining these parameters can be difficult [Westerhoff and Palsson 2004], and in most cases these parameter values are unknown. Here, quantitative parameters were selected from a range of values reported in the literature. In addition to deriving new SVPs, the BacillOndex knowledgebase was used to retrieve an estimate of the relative strengths of promoters. BacillOndex includes normalised values for maximum and minimum gene expression values from 79 microarray experiments [Misirli 2013]. These normalised values were used to estimate transcription rates of promoter SVPs. It has been known that prokaryotic transcription rates vary between 0.0001 and 1 mRNA per second, and 80 bp can be transcribed per second [Alon 2006]. Transcription rates in the literature have also been reported within this range [Braff et al. 2005; Elowitz and Leibler 2000; Ozbudak et al. 2002; Voigt et al. 2005]. In order to construct promoter SVPs with initial values, maximum normalised gene expression values were mapped into a range between 0.0001 and 1. The new rates represent relative transcription rates and can be used in simulations. Based on the literature, the half-lives of mRNA and protein molecules are assumed to be 2 and 10 minutes, respectively [Elowitz and Leibler 2000]. Using the formula `ln(2)/half-life` [Alon 2006], the corresponding rates of degradation of mRNAs and proteins per second are calculated as 0.0058 and 0.0012, respectively. 1000 nM for a TF in a bacterial cell is a high concentration, and Km disassociation constants can change between 1 nM and 10000 nM [Buchler et al. 2003]. Therefore, these constants were initially assigned as 500 nM. Phosphorylation and dephosphorylation rates were selected as 0.0001 per nM per second and 0.1 per second, respectively [Bray et al. 1993]. The RBSCalculator application [Salis et al. 2009] was used to estimate translation rates from RBS parts.

2.6. Construction of the Virtual Parts Repository and Programmatic Access

The Virtual Parts Repository was developed as a Web application using Java. Maven[4] is used to handle subprojects and dependencies, and to build new releases. Metadata about biological parts and molecular interactions are stored in a relational database. In the data model, interactions have parameters and interacting parts which may have properties in the form of name-value pairs. These properties are used to store additional information such as part type, PubMed ID, and host organism, and are used to filter SVPs when querying. The persistence of the objects is carried out by the DataNucleus API.[5] Once the Repository is constructed, SVPs are built using the metadata and are stored as objects in the database.

[4]http://maven.apache.org.
[5]http://www.datanucleus.org.

The current repository was populated with data mined from an ontology representation of the BacillOndex dataset [Misirli 2013]. This ontology was used to classify sequence-based features, based on their composition and the molecular interactions. Examples include the identification of inducible promoters, classification of promoters based on sigma factors and the identification of CDSs using the biological functions of their encoded proteins. The ontology was queried about the sequence features and molecular interactions using the Jena API[6] and the results were stored in the Repository's database. The classification results are assigned to SVPs using the subtype properties.

Computational data exchange is facilitated by using existing data standards such as SBML for models and the Synthetic Biology Open Language (SBOL) [Galdzicki et al. 2014] for DNA sequence information. The Repository has a lightweight, REST-based Web service interface [Curbera et al. 2002] that allows computational tools to retrieve metadata about biological parts and their SVPs. An API that would allow programmatic access to the Repository was also developed using Java. This API uses the Web service interface to retrieve data in the form of Java objects and available from the Repository's website.[7] Models are returned using the `SBMLDocument` objects of the SBML's JSBML library [Dräger et al. 2011] which provides Java objects for SBML entities. SBOL-related data is interpreted using the libSBOLj[8] API. Composite biological circuit designs are visualised using PigeonCAD [Bhatia and Densmore 2013] which has a Web application that can receive designs in a custom format and produces downloadable images.

3. RESULTS

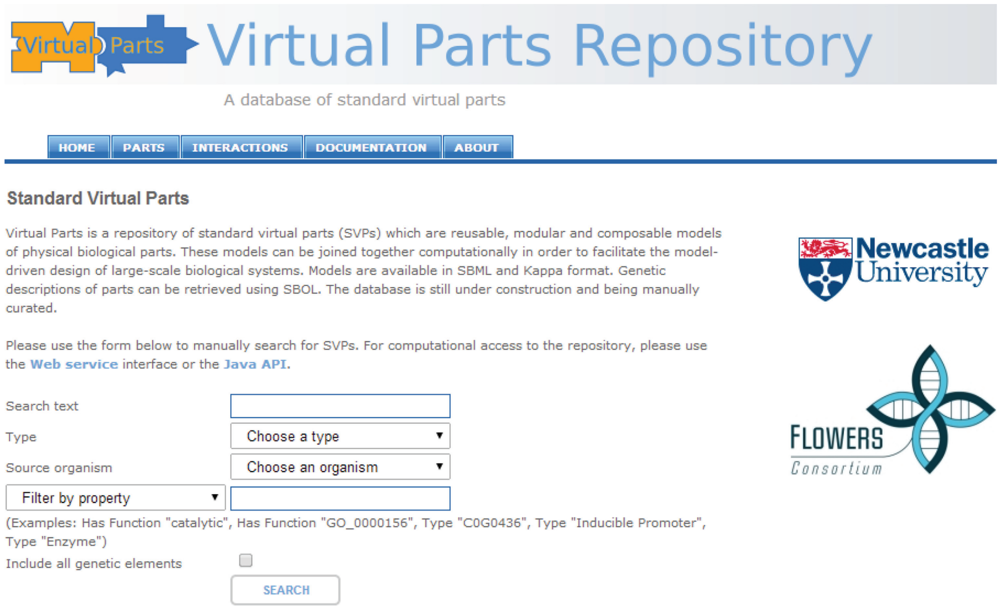
SVPs are publically accessible from the Virtual Parts Repository (Figure 6). The Repository contains, at the time of writing, 3,015 SVPs and 699 interactions. We have currently populated the Repository with SVPs relevant to *B. subtilis*, although the framework is generic enough to support the creation of SVPs for a range of different organisms, and work is currently underway to include parts for other organisms such as *Escherichia coli*.

3.1. Standard Virtual Parts

The core annotations of an SVP include its ID, name, description, type, sequence, source organism and whether the SVP is manually or computationally created. SVPs can also include details of biochemical reactions, such as degradation, that are not dependent on other parts. Such reactions are listed as internal events and are encapsulated as part of the metadata for each SVP. Models of SVPs include these internal interactions and some of the metadata such as type and sequence information about the part (Table II).

SVPs can be assigned additional subtype information to enhance basic type information (Table III). These subtypes are useful for the selection of parts at a finer level of granularity and each SVP may possess more than one subtype. For example, an SVP of basic type `Promoter` may also be both a `SigAPromoter` and a `InduciblePromoter`, annotations which indicate that the part is an inducible SigA promoter. These types refer to terms from ontologies such as the Sequence Ontology (SO) [Eilbeck et al. 2005] and an ontology that has been developed in house in order to define semantically types of biological parts [Misirli 2013]. Similarly, operators can be filtered according to the type of regulation of their TFs. The CDSs and their encoded products are modelled as functional parts which are functionally defined using COG numbers.

[6]http://jena.apache.org.
[7]http://www.virtualparts.org.
[8]https://github.com/SynBioDex/libSBOLj.



Standard Virtual Parts

Virtual Parts is a repository of standard virtual parts (SVPs) which are reusable, modular and composable models of physical biological parts. These models can be joined together computationally in order to facilitate the model-driven design of large-scale biological systems. Models are available in SBML and Kappa format. Genetic descriptions of parts can be retrieved using SBOL. The database is still under construction and being manually curated.

Please use the form below to manually search for SVPs. For computational access to the repository, please use the **Web service** interface or the **Java API**.

Search text	
Type	Choose a type
Source organism	Choose an organism
Filter by property	

(Examples: Has Function "catalytic", Has Function "GO_0000156", Type "C0G0436", Type "Inducible Promoter", Type "Enzyme")

Include all genetic elements

SEARCH

Fig. 6. The Virtual Parts Repository website[7] (version 1.0.2). Computational access is provided by a REST-based Web service interface and also by an API.

Table II. Types of SVPs and the Number of Models for Each Type in the Current Virtual Parts Repository (accessed 01/15/2014)

SVP type	Number
Promoter	455
Operator	554
RBS	467
Shim	291
Terminator	1,123
FunctionalPart	125

Table III. Examples of Subtypes of SVPs and Number of Corresponding Models in the Virtual Parts Repository

SVP subtype	Type	Number
Inducible Promoter	Promoter	51
Repressible Promoter	Promoter	86
Constitutive SigA promoter	Promoter	310
Negatively Regulated Operator	Operator	333
Positively Regulated Operator	Operator	221
Transcription Factor	FunctionalPart	94

SVPs may also possess other properties that are assigned in the form of name-value pairs. These properties include database accession numbers, PubMed IDs, and information about the data sources from which the SVPs are retrieved. For `FunctionalPart` SVPs, properties also exist for their cellular locations and molecular functions in the form of GO terms [The Gene Ontology Consortium 2001]. Parts may be retrieved either manually via the website, or computationally, using Web services.

Table IV. Examples of Types of Interactions in the Current Virtual Parts Repository (accessed 01/15/2014)

Interaction type	Number
Transcriptional activation	44
Transcriptional repression	85
Phosphorylation	27
Transcriptional activation by an operator	196
Transcriptional repression by an operator	333

3.2. Models of Interactions

Models of part interactions are stored in the Virtual Parts Repository and can be retrieved computationally. These models are used to represent interactions between parts, each of which is represented with an individual SVP. Such interactions constrain how biological parts can be used to create functional designs. Models of these interactions may be needed to join SVPs together in order to create simulatable models, particularly when representing higher-order trans-based interactions, such as protein-protein interactions.

The properties of an interaction include information about the interaction's unique ID, its description, a mathematical representation of the reaction and its type. The participating parts, reaction stoichiometries, molecular forms of the parts involved and kinetic parameters are also part of the metadata for each interaction.

Interaction types currently include transcriptional activation, transcriptional repression, phosphorylation, dephosphorylation, transcriptional activation by an operator, and transcriptional repression by an operator (Table IV).

3.3. Computational Access to the Repository

A REST-based Web service has been implemented to retrieve models (SVPs and models of interactions) and metadata about parts and their interactions. Types, subtypes, and properties of SVPs can also be used to retrieve a subset of SVPs using this Web service. An API that enables programmatic access to the Virtual Parts Repository via a Web service is also available.

The API, called JParts, returns Java objects in response to Web service calls. For example, in order to retrieve the first 50 SVPs that represent SigA promoters, the '`GetParts(1, "Promoter", "type", "SigAPromoter")`' method call can be used. Here, Promoter is the type of an SVP and `SigAPromoter` is the type of a promoter SVP. SVPs can also be searched for according to their functional roles. For example, protein SVPs with associated GO classes can be searched for based on their molecular functions and cellular locations. Parts can also be retrieved by their IDs. A list of interactions for a part can be retrieved using a different method call. Additionally, SBML models of parts and interactions can be retrieved. JParts also provides utility methods to create a container model into which SVPs and their interactions can be placed, and to join inputs and outputs of SVPs and models of interactions. The `PartsHandler` class is used to access models of parts and interactions. The `ModelBuilder` class is used to add these SVPs into an SBML model and to join them using the defined inputs and outputs of SVPs. The resulting file can be simulated using tools such as COPASI [Hoops et al. 2006].

3.4. A Use Case: Computational Composition of the Subtilin Receiver System from SVPs

In order to demonstrate SVPs and their interface, the construction of a subtilin receiver device is provided. In this system, the lantibiotic (a class of small peptide antibiotic) subtilin is sensed by a two-component system (TCS) [Bongers et al. 2005; Klein et al.

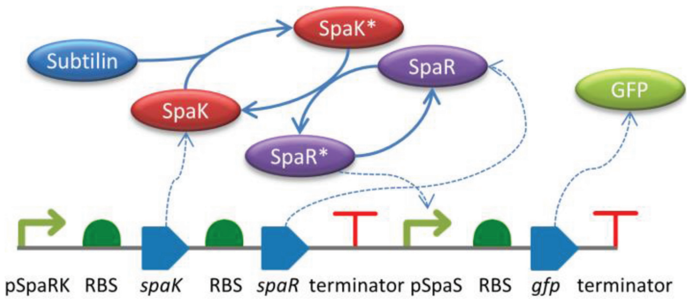


Fig. 7. The subtilin receiver device. Adapted from Cooling et al. [2010]. SpaK* and SpaR* represent the phosphorylated proteins. Dotted and continuous arrows respectively represent gene-protein and protein-protein interactions.

1993], which comprises the kinase protein SpaK and the regulatory protein SpaR. CDSs encoding these proteins are located in the same operon and are transcribed by a constitutive promoter. Each CDS has its own RBS.

In this synthetic system, this operon comprises the phosphate-mediated signalling system required to form an activated SpaR protein (Figure 7). Upon sensing subtilin, SpaK activates SpaR via phosphorylation. The activated SpaR then activates the downstream *pSpaS* promoter to express the green fluorescent protein (GFP) [Spiller et al. 2010] as a reporter protein. Both SpaK and SpaR are autodephosphorylated at a constant rate.

The computational model of the subtilin receiver device includes modelling entities for the activation and deactivation of the SpaK and SpaR proteins, the transcription of mRNAs from the two promoters, and the translation of proteins. Modelling entities representing the degradation of all of the proteins and mRNAs in the device are also added to the model. In the model, the PoPS signal from the *pSpaRK* promoter is converted into mRNAs. It is assumed that the mRNA transcription is uniform for both CDSs. Therefore, both RBSs for *spaK* and *spaR* use the same mRNA output from the *pSpaRK* promoter to produce RiPS signals which are then converted into the SpaK and SpaR proteins. The activation of SpaR is modelled via an interaction between the SpaK and SpaR SVPs. Similarly, the activation of the *pSpaS* promoter by the activated SpaR is modelled via an interaction between the corresponding SVPs. This interaction has a PoPS output which is converted into mRNA, the input for the third RBS in order to produce GFP.

The subtilin receiver model can be constructed from SVPs using the API according to the following rules.

(1) Create an empty model and add the SVPs.

SVPs required to construct the device are the “pSpaRK” and “pSpaS” promoter SVPs; three SVPs representing the RBSs for *spaK*, *spaR,* and *gfp* CDSs; and three `FunctionalPart` SVPs for SpaR, SpaK and GFP proteins and their encoding CDSs (Figure 8).

(2) Add mRNA SVPs.

An mRNA SVP is used for SVPs representing the RBSs for *spaK* and *spaR* CDSs, and another one is used for the RBS preceding the *gfp* CDS.

(3) Join SVPs by connecting their inputs and outputs with the same types using the following rules:

—Connect the PoPS output of a promoter SVP to the PoPS input of an mRNA SVP.
—Connect the mRNA output of an mRNA SVP to the mRNA input of an RBS SVP.
—Connect the RiPS output of an RBS to the RiPS input of a `FunctionalPart` SVP.

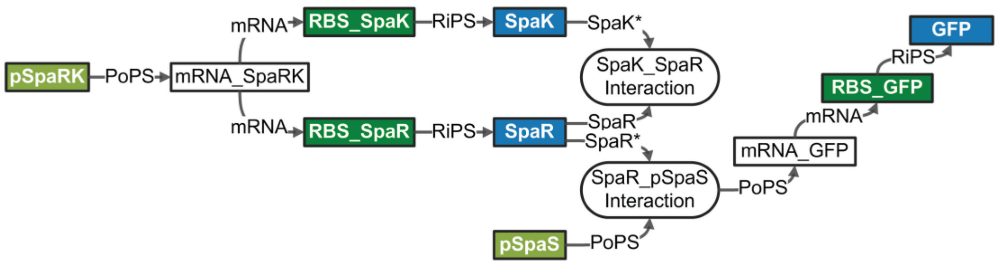


Fig. 8. Relationships between the SVPs of the subtilin receiver model. Ovals represent interactions and other boxes represent SVPs. Each SVP has internal modelling entities that are encapsulated. SVPs are connected with signal carriers such as PoPS and RiPS.

The JParts API's `ModelBuilder` class provides a method to add a connection, which is a simple assignment between the output of one SVP and the input of another SVP. In the case of SBML, these assignments are implemented using `AssignmentRules`. SVPs are already annotated with machine-level information to facilitate model composition and these annotations are used by the API when connecting SVPs to create systems models.

(4) Join SVPs by adding models of interactions.

The interaction between the SpaK and SpaR proteins can be added directly to the container model without any explicit connection, since the connections for reactions in SBML are implicit. For the interaction between the "pSpaS" promoter and "SpaR" SVPs, the PoPS output of the "pSpaS" SVP must be connected to the PoPS input of the model of the interaction between these two parts. Moreover, the PoPS output from this interaction model should be connected to the corresponding mRNA SVP.

4. DISCUSSION

Design automation for synthetic biological systems is gaining popularity as a means of constructing large-scale genetic circuits that are not possible to achieve manually [Densmore and Hassoun 2012]. Computational design, simulation, synthesis and testing strategies are key technologies for these systems. However, computational design tools often lack access to simulatable models of biological parts [Clancy and Voigt 2010], a situation which limits the design of complex genetic constructs. SVPs and their associated repository, as described in this work, therefore provide a useful resource for CAD and automation tools by offering a predefined library of modular models that encapsulate the behaviour and function of genetic parts. The Web-based interface facilitates manual selection and browsing by a user, whilst the Web service interface facilitates programmatic access to allow seamless access by remote software applications.

Another issue with design automation for synthetic biology is that biological systems have very large design spaces. In order to design biologically plausible systems in a cost- and time-effective manner, these designs should ideally be constrained to potentially valuable regions of the search space, using existing knowledge and computational simulation. The use of abstractions that hide the physical details of molecular interactions has already facilitated the construction of biological devices from simple biological parts. In order to automate this process, SVPs can be used to derive models from which biological systems can be generated automatically. Such a systems design process [Henzinger and Sifakis 2006] facilitates a model-driven approach to automatically designing and verifying genetic circuits *in silico* before implementing them physically. SVPs are suitable for automating the searching of large design spaces for both bottom-up and top-down approaches.

SVPs can be combined computationally to construct a range of large, simulatable models with the same behaviour, from which optimal solutions can be chosen in a bottom-up approach. The output of one SVP can be connected to the input of another SVP. However, not all parts interact with each other, and hence not all designs are biologically plausible [Densmore et al. 2010]. Without considering such constraints, the possible solution space for biological systems would grow exponentially [Rodrigo et al. 2011]. Information about interactions between SVPs can be used to identify suitable SVPs, restricting the number of solutions to biologically plausible designs. Moreover, SVPs are categorised according to their types, GO terms and COG numbers, allowing SVPs to be selected using these criteria, prior to the assembly of larger models.

The automation of genetic circuits have been demonstrated via the domain specific languages (DSLs) and the tools implementing them [Beal et al. 2011]. A design specified with these languages is then mapped to individual parts. However, solutions should be verified via simulation. SVPs already provide a mapping between biological function and DNA sequence. Each SVP may contain a set of equations that takes inputs and converts them into outputs. Therefore, SVPs can be combined to integrate all of the transfer functions needed for the verification of a selected solution. SVPs can therefore be plugged into tools that implement DSLs.

There are also tools, such as SBROME [Huynh et al. 2013] and MatchMaker [Yaman et al. 2012], that use graph-based approaches to map abstract designs to individual parts. These tools already work with libraries of parts and biological constraints in order to realize genetic circuit designs and could benefit from the readily available SVPs. Optimization-based algorithms have also been proposed to automate genetic circuit designs. Tools such as OptCircuit [Dasika and Maranas 2008] can similarly work with libraries of parts and their interactions to construct genetic circuits and can also suggest kinetic parameters to implement a desired behaviour. With small libraries of parts, it may be valuable to use directed evolution. Such tools can also take advantage of SVPs to search for genetic parts that provide desired quantitative parameters, reducing the effort to carry out these experiments. SVPs would then be ideal for the application of a dual evolutionary strategy to evolve genetic circuits *in silico* first and then to evolve the biological system *in vivo* or *in vitro* [Hallinan et al. 2012].

Recently, it has been shown that computational intelligence approaches, such as genetic algorithms, can be used to guide the composition of SVPs for the design of specified biological system [Hallinan et al. 2014]. SVPs can also be used by CAD tools when manually designing biological systems. Virtual parts can be selected and joined together using a CAD environment. We are currently in the process of integrating iBioSim [Myers et al. 2009], a CAD tool that is developed for model-based design of genetic circuits, with the Virtual Parts Repository.

Although model annotation has been used extensively in systems biology, the needs of model annotation for synthetic biology are unique. In systems biology, models are used to understand existing biological systems. Therefore, entities representing biological reactions and molecules such as the proteins that participate in those reactions are annotated to aid machine-level understanding of these entities [Lister et al. 2010]. However, in synthetic biology the emphasis is on the design of new biological systems using existing or novel biological parts. Here, annotations are used to facilitate the model composition, and to identify the types and nucleotide sequences of biological parts. These annotations are essential to computationally design biological systems and to derive DNA sequences necessary to encode these systems [Misirli et al. 2011].

The construction of models of biological systems, particularly novel biological systems, is not a trivial task. This process can be facilitated by using the extensive information available from biological databases and experimental results [Endler et al.

2009]. The Virtual Parts Repository was populated using a data mining approach, which demonstrates how existing biological data can be integrated and used as a basis upon which to construct dynamic models for synthetic biology.

The use of SVPs exemplifies the advantages of using standardised models for synthetic genetic circuits [Marchisio and Stelling 2008; Rodrigo et al. 2007a]. SVPs are stored in a publicly accessible catalogue and can be used by any tools that adopt standards such as SBML and the widely accepted biological signals [Braff et al. 2005] that are used as the inputs and outputs of SVPs. The genetic descriptions of parts are available in the SBOL format. The use of data standards can facilitate the construction of computational workflows that may include several different tools for biological system design. This approach has recently been used to specify an abstract genetic circuit and to automate the finding of solutions [Galdzicki et al. 2014]. The Virtual Parts Repository is used to store the resulting genetic circuit designs alongside their computational models. Having a single repository of models allows standardisation of models and facilitates a central curation approach. Currently, work is in progress to provide facilities for curating and submitting user-defined models.

One of the current limitations of SVPs is that parts are considered to be independent of their genetic context. However, it is known that other factors such as the length of an operon can affect rates of transcription and translation. Although translation rates are largely affected by the strength of RBSs, upstream and downstream sequences can affect three-dimensional structures to prevent the binding of ribosomes and hence reduce translation rates. There are already tools such as RBSCalculator [Salis et al. 2009] and RBSDesigner [Na and Lee 2010] that can be used to predict such behaviour. These context dependent design issues are currently difficult to represent, since these issues only arise when parts are joined together. Therefore, design time refinements would be useful. In future work, we will update the JParts API and provide methods that will take genetic context information into account by incorporating existing tools.

There are a number of design patterns designed to remove context dependency, such as the use of insulating parts to reduce the effect of genomic context for transcription [Davis et al. 2010] and translation [Na et al. 2010]; the use of negative feedback to increase the robustness of genetic circuits to parameter variations [Silva-Rocha and de Lorenzo 2010]; and reducing retroactivity [Del Vecchio et al. 2008; Kim and Sauro 2011], a quantity that is used to measure how the behaviour of a system changes when it is connected to downstream components. SVPs can be used to implement these design patterns. For example, SVPs can be used to implement genetic circuits with positive or negative feedbacks to create stochastic [Süel et al. 2006] or fast-response behaviours [Alon 2007] respectively. Feed-forward loops (FFLs) can also be ideal to implement a desired behavior; for example, to create a pulse of a signal or to filter noisy signals (using incoherent type 1 FFL and coherent type 1 FFL, respectively). Negative feedback can further be coupled with large input amplification (by using a strong promoter or incorporating fast protein-protein interactions such as phosphorylation) to reduce retroactivity.

The modular modelling approach and the catalogue of models of biological parts presented here facilitates the large-scale engineering of complex biological systems using computational methods. The models are specified with defined levels of abstraction in the form of mathematical formulae, an approach which is important for standardising the modelling of biological parts for synthetic biology. These modular models can be accessed computationally from the Virtual Parts Repository. The models also include annotations that facilitate their computational composition, which is particularly important when constructing models of large systems that are not feasible to build manually.

In this work, we extended previous work on SVPs described by Cooling and colleagues and provided a database of SVPs. We encapsulate model entities specific for a biological part in a single model, an SVP. This encapsulation provides a one-to-one mapping between SVPs and biological parts and reduces the complexity when building complex models from smaller models. We also provide a framework to computationally join SVPs, by identifying inputs and outputs for different types of SVPs. These inputs and outputs are annotated in SVPs in order to interpret the information necessary for computational model composition. We integrated existing biological data sources and mined the resulting dataset as a base to provide a model repository with computational access to information about large numbers of biological parts and their models. SVPs can be used to explore the space of possible solutions via computational simulation. The availability of these standard models and associated metadata are useful for computational tools, for example those that implement DSLs or stochastic heuristics, and therefore for the automation of the design of predictable large-scale biological systems.

We are currently working to add new parts, curate models, and include support for other modelling languages. In the future, the Repository will also be updated to extend the range of organisms and to add chassis-specific data. The API will be enhanced to provide additional features and to reduce the complexity of building virtual systems.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Michael Cooling, Owen Gilfellon, Dr. Matthew Pocock, the UK Flowers consortium, and the SBOL community for helpful discussions.

REFERENCES

U. Alon. 2006. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman & Hall/CRC.

U. Alon. 2007. Network motifs: theory and experimental approaches. *Nat. Rev. Genet.* 8, 450–461.

K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema. 2006. Developing applications using model-driven design environments. *Computer* 39, 33–40.

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

S. Bhatia and D. Densmore. 2013. Pigeon: A design visualizer for synthetic biology. *ACS Synthet. Biol.* 2, 348–350.

L. Bintu, N. E. Buchler, H. G. Garcia, U. Gerland, T. Hwa, J. Kondev, and R. Phillips. 2005. Transcriptional regulation by the numbers: Models. *Curr. Opin. Genet. Develop.* 15, 116–124.

R. S. Bongers, J.-W. Veening, M. Van Wieringen, O. P. Kuipers, and M. Kleerebezem. 2005. Development and characterization of a subtilin-regulated expression system in *Bacillus subtilis*: Strict control of gene expression by addition of subtilin. *Appl. Environ. Microbiol.* 71, 8818–8824.

J. C. Braff, C. M. Conboy, and D. Endy. 2005. Definitions and measures of performance for standard biological parts. In *Proceedings of the International Conference Systems Biology (ICSB'05)*.

D. Bray, R. B. Bourret, and M. I. Simon. 1993. Computer simulation of the phosphorylation cascade controlling bacterial chemotaxis. *Molec. Biol. Cell* 4, 469–482.

N. E. Buchler, U. Gerland, and T. Hwa. 2003. On schemes of combinatorial transcription logic. *Proc. Nat. Acad. Sci.* 100, 5136–5141.

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

A. A. Cuellar, C. M. Lloyd, P. F. Nielsen, D. P. Bullivant, D. P. Nickerson, and P. J. Hunter. 2003. An overview of CellML 1.1, a biological model description language. *Simulation* 79, 740–747.

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

K. Clancy and C. A. Voigt. 2010. Programming cells: Towards an automated 'Genetic Compiler'. *Curr. Opin. Biotechnol.* 21, 572–581.

M. T. Cooling, V. Rouilly, G. Misirli, J. Lawson, T. Yu, J. Hallinan, and A. Wipat. 2010. Standard virtual biological parts: A repository of modular modeling components for synthetic biology. *Bioinformatics* 26, 925–931.

F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. 2002. Unraveling the Web services web: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Comput.* 6, 86–93.

M. Dasika and C. Maranas. 2008. OptCircuit: An optimization based method for computational design of genetic circuits. *BMC Syst. Biol.* 2, 24.

J. H. Davis, A. J. Rubin, and R. T. Sauer. 2010. Design, construction and characterization of a set of insulated bacterial promoters. *Nucl. Acids Res.* 39, 1131–1141.

A. De Las Heras, C. A. Carreño, E. Martínez-García, and V. De Lorenzo. 2010. Engineering input/output nodes in prokaryotic regulatory circuits. *FEMS Microbiol. Rev.* 34, 842–865.

D. Del Vecchio, A. J. Ninfa, and E. D. Sontag. 2008. Modular cell biology: Retroactivity and insulation. *Molec. Syst. Biol.* 4, 161.

D. Densmore and S. Hassoun. 2012. Design automation for synthetic biological systems. *Des. Test Comput.* IEEE 1–1.

D. Densmore, J. T. Kittleson, L. Bilitchenko, A. Liu, and J. C. Anderson. 2010. Rule based constraints for the construction of genetic devices. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'10)*. 557–560.

A. Dräger, N. Rodriguez, M. Dumousseau, A. Dörr, C. Wrzodek, N. Le Novère, A. Zell, and M. Hucka. 2011. JSBML: A flexible Java library for working with SBML. *Bioinformatics* 27, 2167–2168.

K. Eilbeck, S. Lewis, C. Mungall, M. Yandell, L. Stein, R. Durbin, and M. Ashburner. 2005. The sequence ontology: A tool for the unification of genome annotations. *Genome Biol.* 6, R44.

M. B. Elowitz and S. Leibler. 2000. A synthetic oscillatory network of transcriptional regulators. *Nature* 403, 335–338.

L. Endler, N. Rodriguez, N. Juty, V. Chelliah, C. Laibe, C. Li, and N. Le Novere. 2009. Designing and encoding models for synthetic biology. *J. Roy. Soc. Interf.* 6, S405–S417.

P. H. Feiler, B. Lewis, S. Vestal, and E. Colbert. 2005. An overview of the SAE architecture analysis & design language (AADL) standard: A basis for model-based architecture-driven embedded systems engineering. In *Architecture Description Languages*, Springer, 3–15.

B. R. Fritz, L. E. Timmerman, N. M. Daringer, J. N. Leonard, and M. C. Jewett. 2010. Biology by design: From top to bottom and back. *BioMed Res. Int.* 2010.

A. Funahashi, M. Morohashi, H. Kitano, and N. Tanimura. 2003. CellDesigner: A process diagram editor for gene-regulatory and biochemical networks. *BIOSILICO* 1, 159–162.

M. Galdzicki, K. P. Clancy, E. Oberortner, M. Pocock, J. Y. Quinn, C. A. Rodriguez, N. Roehner, M. L. Wilson, L. Adam, J. C. Anderson, B. A. Bartley, J. Beal, D. Chandran, J. Chen, D. Densmore, D. Endy, R. Grunberg, J. Hallinan, N. J. Hillson, J. D. Johnson, A. Kuchinsky, M. Lux, G. Misirli, J. Peccoud, H. A. Plahar, E. Sirin, G.-B. Stan, A. Villalobos, A. Wipat, J. H. Gennari, C. J. Myers, and H. M. Sauro. 2014. The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nat. Biotech.* 32, 545–550.

The Gene Ontology Consortium. 2001. Creating the gene ontology resource: Design and implementation. *Gen. Res.* 11, 1425–1433. http://genome.cshlp.org/citmgr?gca=genome%3B11%2F8%2F1425.

J. Hallinan, O. Gilfellon, G. Misirli, and A. Wipat. 2014. Tuning receiver characteristics in bacterial quorum communication: An evolutionary approach using standard virtual biological parts. In *Proceedings of the IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*.

J. Hallinan, G. Misirli, and A. Wipat. 2010. Evolutionary computation for the design of a stochastic switch for synthetic genetic circuits. In *Proceedings of the 32nd IEEE Annual International Conference on Engineering in Medicine and Biology Society (EMBC'10)* (Buenos Ares, Argentina).

J. Hallinan, S. Park, and A. Wipat. 2012. Bridging the gap between design and reality - A dual evolutionary strategy for the design of synthetic genetic circuits. In *Proceedings of the (Bioinformatics'12) - International Conference on Bioinformatics Models, Methods and Algorithms*, February 1–4, J. Schier, C. M. B. A. Correia, A. L. N. Fred, and H. Gamboa, Eds., SciTePress, 263–268.

T. A. Henzinger and J. Sifakis. 2006. The embedded systems design challenge. In *Proceedings of the International Symposium on Formal Methods (FM'06)*. Springer, 1–15.

A. D. Hill, J. R. Tomshine, E. M. B. Weeding, V. Sotiropoulos, and Y. N. Kaznessis. 2008. SynBioSS: The synthetic biology modeling suite. *Bioinformatics* 24, 2551–2553.

S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. 2006. COPASI—a COmplex PAthway SImulator. *Bioinformatics* 22, 3067–3074.

M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and W. J. 2003. The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics* 19, 524–531.

L. Huynh, A. Tsoukalas, M. Köppe, and I. Tagkopoulos. 2013. SBROME: A scalable optimization and module matching framework for automated biosystems design. *ACS Synthet. Biol.* 2, 263–273.

Y. Kaznessis. 2007. Models for synthetic biology. *BMC Syst. Biol.* 1, 47.

Kyung H. Kim and Herbert M. Sauro. 2011. Measuring retroactivity from noise in gene regulatory networks. *Biophys. J.* 100, 1167–1177.

C. Klein, C. Kaletta, and K. D. Entian. 1993. Biosynthesis of the lantibiotic subtilin is regulated by a histidine kinase/response regulator system. *Appl. Environ. Microbiol.* 59, 296–303.

C. Li, M. Donizelli, N. Rodriguez, H. Dharuri, L. Endler, V. Chelliah, L. Li, E. He, A. Henry, M. Stefan, J. Snoep, M. Hucka, N. Le Novere, and C. Laibe. 2010. BioModels database: An enhanced, curated and annotated resource for published quantitative kinetic models. *BMC Syst. Biol.* 4, 92.

J. Liang, Y. Luo, and H. Zhao. 2011. Synthetic biology: Putting synthesis into biology. *Wiley Interdisciplinary Reviews: Syst. Bio. Med.* 3, 7–20.

A. Lister, P. Lord, M. Pocock, and A. Wipat. 2010. Annotation of SBML models through rule-based semantic integration. *J. Biomed. Semant.* 1, S3.

D. MacMillen, R. Camposano, D. Hill, and T. W. Williams. 2000. An industrial view of electronic design automation. *IEEE Trans. Comput.-Aided Des. Integ. Circ. Syst.* 19, 1428–1448.

M. A. Marchisio and J. Stelling. 2008. Computational design of synthetic gene circuits with composable parts. *Bioinformatics* 24, 1903–1910.

M. A. Marchisio and J. Stelling. 2009. Computational design tools for synthetic biology. *Curr. Opin. Biotechnol.* 20, 479–485.

G. Misirli. 2013. Data integration strategies for informing computational design in synthetic biology. Ph.D. dissertation. School of Computing Science, Newcastle University, UK.

G. Misirli, J. S. Hallinan, T. Yu, J. R. Lawson, S. M. Wimalaratne, M. T. Cooling, and A. Wipat. 2011. Model annotation for synthetic biology: Automating model to nucleotide sequence conversion. *Bioinformatics* 27, 973–979.

G. Misirli, A. Wipat, J. Mullen, K. James, M. Pocock, W. Smith, N. Allenby, and J. Hallinan. 2013. BacillOndex: An integrated data resource for systems and synthetic biology. *J. Integrat. Bioinf.* 10, 224.

C. J. Myers, N. Barker, K. Jones, H. Kuwahara, C. Madsen, and N.-P. D. Nguyen. 2009. iBioSim: A tool for the analysis and design of genetic circuits. *Bioinformatics* 25, 2848–2849.

D. Na and D. Lee. 2010. RBSDesigner: Software for designing synthetic ribosome binding sites that yields a desired level of protein expression. *Bioinformatics* 26, 2633–2634.

D. Na, S. Lee, and D. Lee. 2010. Mathematical modeling of translation initiation for the estimation of its efficiency to computationally design mRNA sequences with desired expression levels in prokaryotes. *BMC Syst. Biol.* 4, 71.

E. M. Ozbudak, M. Thattai, I. Kurtser, A. D. Grossman, and A. van Oudenaarden. 2002. Regulation of noise in the expression of a single gene. *Nat. Genet.* 31, 69–73.

M. Pedersen and A. Phillips. 2009. Towards programming languages for genetic engineering of living cells. *J. Roy. Soc. Interf.* 6, S437–S450.

G. Rodrigo, J. Carrera, and A. Jaramillo. 2007a. Asmparts: Assembly of biological model parts. *Syst. Synthet. Biol.* 1, 167–170.

G. Rodrigo, J. Carrera, and A. Jaramillo. 2007b. Genetdes: Automatic design of transcriptional networks. *Bioinformatics* 23, 1857–1858.

G. Rodrigo, J. Carrera, and A. Jaramillo. 2011. Computational design of synthetic regulatory networks from a genetic library to characterize the designability of dynamical behaviors. *Nucl. Acids Res.* 39, e138.

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

R. Silva-Rocha and V. de Lorenzo. 2010. Noise and robustness in prokaryotic regulatory networks. *Ann. Rev. Microbiol.* 64, 257–275.

C. D. Smolke and P. A. Silver. 2011. Informing biological design by integration of systems and synthetic biology. *Cell* 144, 855–859.

D. G. Spiller, C. D. Wood, D. A. Rand, and M. R. H. White. 2010. Measurement of single-cell dynamics. *Nat.* 465, 736–745.

J. Stelling. 2004. Mathematical models in microbial systems biology. *Curr. Opin. Microbiol.* 7, 513–518.

G. M. Süel, J. Garcia-Ojalvo, L. M. Liberman, and M. B. Elowitz. 2006. An excitable gene regulatory circuit induces transient cellular differentiation. *Nature* 440, 545–550.

R. L. Tatusov, D. A. Natale, I. V. Garkavtsev, T. A. Tatusova, U. T. Shankavaram, B. S. Rao, B. Kiryutin, M. Y. Galperin, N. D. Fedorova, and E. V. Koonin. 2001. The COG database: New developments in phylogenetic classification of proteins from complete genomes. *Nucl. Acids Res.* 29, 22–28.

C. A. Voigt, D. M. Wolf, and A. P. Arkin. 2005. The bacillus subtilis sin operon: An evolvable network. *Motif. Genet.* 169, 1187–1202.

H. V. Westerhoff and B. O. Palsson. 2004. The evolution of molecular biology into systems biology. *Nat. Biotechnol.* 22, 1249–1252.

F. Yaman, S. Bhatia, A. Adler, D. Densmore, and J. Beal. 2012. Automated selection of synthetic biology parts for genetic regulatory networks. *ACS Synthet. Biol.* 1, 332–344.

E. Young and H. Alper. 2010. Synthetic biology: Tools to design, build, and optimize cellular processes. *J. Biomed. Biotechnol.*

A. R. Zomorrodi and C. D. Maranas. 2014. Coarse-grained optimization-driven design and piecewise linear modeling of synthetic genetic circuits. *Europ. J. Oper. Res.*

Received January 2014; revised April 2014; accepted May 2014