

Capturing Multicellular System Designs Using Synthetic Biology Open Language (SBOL)

Bradley Brown,[†] Bryan Bartley,^{||} Jacob Beal,^{||} Jasmine E. Bird,[¶] Ángel Goñi-Moreno,^{‡#} James Alastair McLaughlin,[‡] Göksel Mısırlı,[§] Nicholas Roehner,^{||} David James Skelton,[‡] Chueh Loo Poh,[⊥] Irina Dana Ofiteru,[†] Katherine James[♭] and Anil Wipat^{*,‡}

[†] School of Engineering, Newcastle University, NE1 7RU, United Kingdom

^{||} Raytheon BBN Technologies, Cambridge, MA, 02138, United States

[¶] School of Natural and Environmental Sciences, Newcastle University, NE1 7RU United Kingdom

[‡] School of Computing, Newcastle University, NE4 5TG, United Kingdom

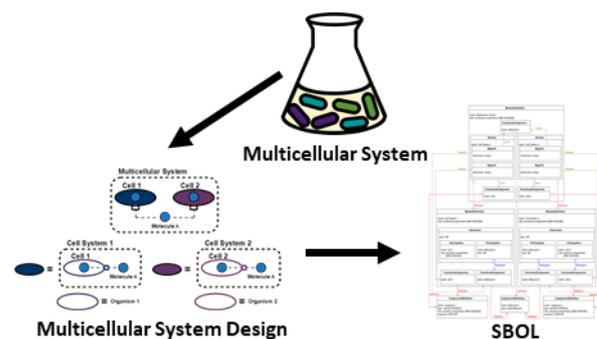
[#] Centro de Biotecnología y Genómica de Plantas (CBGP, UPM-INIA), Universidad Politécnica de Madrid (UPM) - Instituto Nacional de Investigación y Tecnología Agraria y Alimentaria (INIA) Campus de Montegancedo-UPM, 28223 Pozuelo de Alarcon, Madrid, Spain

[§] School of Computing and Mathematics, Keele University, ST5 5BG, United Kingdom

[⊥] Department of Biomedical Engineering and NUS Synthetic Biology for Clinical and Technological Innovation (SynCTI), National University of Singapore, Singapore

[♭] Department of Applied Sciences, Northumbria University, United Kingdom

***Email: anil.wipat@newcastle.ac.uk**



Abstract

Synthetic biology aims to develop novel biological systems and increase their reproducibility using engineering principles such as standardisation and modularisation. It is important that these systems can be represented and shared in a standard way to ensure they can be easily understood, reproduced, and utilised by other researchers. The Synthetic Biology Open Language (SBOL) is a data standard for sharing biological designs and information about their implementation and characterisation. Previously, this standard has only been used to represent designs in systems where the same design is implemented in every cell, however, there is also much interest in multicellular systems, in which designs involve a mixture of different types of cells with differing genotype and phenotype. Here, we show how the SBOL standard can be used to represent multicellular systems,

and hence, how researchers can better share designs with the community and reliably document intended system functionality.

Key words

Multicellular Systems, Microbial Communities, Synthetic Biology Open Language (SBOL), Data Standards

The increasing popularity of synthetic biology has yielded a wealth of biological systems that have been designed, implemented, and characterised to various degrees ^{1 2}. These systems span a wide range of functionalities, from fundamental genetic circuits such as oscillators ³ and toggle switches ⁴, to applied devices such as biosensors ⁵ and microbial factories ⁶. To date, most synthetic biology systems have been homogenous in nature, meaning that every cell in the population is intended to have both the same genotype and the same phenotype. Whilst this approach has yielded promising results, the complexity and optimisation of such systems can become limited for certain applications ⁷. One reason for these limitations is that the larger genetic circuits required by more complex systems can place cells under metabolic strain, resulting in sub-optimal performance. Another reason is that elements in large and complex designs tend to be drawn from many diverse sources, which have different optimal environments ⁸. Therefore, the host cell chosen to express these circuits tends to be a compromise which limits overall performance. Finally, it is also the case that many applications inherently involve cells with multiple distinct phenotypes, such as organoid models ⁹ and microbiome engineering ¹⁰.

To tackle the issues mentioned above, there has been an increased interest in multicellular biological designs that involve more than one distinct population of cells and interactions between populations. In such systems, the overall design is split across the multiple cell populations. These populations can be engineered to communicate with each other, creating a co-culture of cells that, together, can perform a desired function ^{11 12 13 14 15 16}. This approach can reduce the metabolic burden on individual cells, which now only have to perform a fragment of the overall system ¹⁷. In addition, different host cells can be chosen for each element of the design, allowing optimal cells to be chosen for each element of the design. This could be extended further to create designs which involve both engineered and non-engineered cells ¹⁸.

Splitting designs between cell populations can also assist with the concept of modular design, where modules with specific functions can be designed, shared, and easily re-used in other system designs. This modularity can also be achieved by splitting designs across different plasmids which are then implemented in the same host ¹⁹. However, as this can already be easily captured using SBOL (The Synthetic Biology Open Language), it is not focussed on here.

One of the major hallmarks of synthetic biology is standardisation, which aims to increase the reproducibility of engineered biological systems and facilitate re-use by other researchers. To fully realise this aim, it is important that information about the design, implementation, and characterisation of engineered systems can be easily shared with and understood by other members of the synthetic biology community. SBOL ²⁰ has been developed by a community of synthetic biologists to capture information about engineered systems in a standardised format. In a similar fashion to the way that file formats such as the GenBank flat file format (GBF) were developed to capture information about natural biological systems ²¹, SBOL enables the design-build-test cycle to be standardised by storing the information required at each stage. This information can detail designs, build plans, implementation details, and experimental information/results. SBOL aids the sharing of information between researchers and labs and promotes more reliable documentation ^{22 23}. Previously, however, this standard has only been used to represent homogeneous designs and has not explicitly been used to represent strain or other aspects of host context. Here, it is shown how the host context of a design can be represented using SBOL, and how this can be further applied to represent multicellular system designs.

Methods

In this section, the relevant portions of the SBOL data model are reviewed and it is shown how they may be used to represent host context and multicellular systems. These representational practices are based on SBOL version 2.3.0 ²⁴.

Note that in this discussion the word “class” is used to refer to types of entities in the SBOL data model.

Defining parts, devices, and systems with the *ComponentDefinition* and *ModuleDefinition* Classes

The current SBOL data model has two main classes used to capture biological designs: *ComponentDefinition* and *ModuleDefinition*. The *ComponentDefinition* class is usually used to store information about physical structures,

such as DNA and proteins, whereas the *ModuleDefinition* class is used to group together biological entities in a design in order to define the functional interactions between such entities.

The designs captured can range in complexity, from the representation of single parts (such as promoters, coding sequences, proteins), to devices composed of multiple parts (for instance an expression construct), to complex systems comprising many devices (for example a genetic biosensor). In the case of devices and systems, each of the individual parts must be described by a separate *ComponentDefinition* or *ModuleDefinition*. The use of that part is then described within a *ModuleDefinition* class, with the part being referred to using the *FunctionalComponent* class.

The *ModuleDefinition* class may contain interactions between biological entities in the design (for example a coding sequence (CDS) encoding a protein, which in turn represses a promoter, or a small molecule inhibiting a protein), whereas instances of *ComponentDefinition* may not. These relationships are formally captured using two other SBOL classes, *Interaction* and *Participation*: the *Interaction* class specifies the type of interaction (e.g.; genetic production) and contains instances of *Participation* giving the role played by each interacting object (for example, a CDS as genetic template and a protein as product). Additionally, the *ComponentDefinition* class has both type and role properties, but *ModuleDefinition* has only role properties. The type property in SBOL is used to describe the category within which a biological entity falls (for example DNA molecule, small molecule, protein), and the role property describes the intended biological function for an entity or design. For example, a metabolic pathway might have roles of 'metabolic process' and 'small molecule biosynthetic process' from the Gene Ontology (GO) ²⁵, and a biosensor might have a role of 'response to chemical' also from GO.

Ontologies in SBOL

An ontology can be thought of as a set of formal descriptions for specific terms and their relationships. In synthetic biology, ontologies allow a standardised language to be used when describing biological systems. A number of separate ontologies are used in SBOL to better describe entities within a system, and how those entities interact. The SBOL-OWL ontology defines the relationships between classes in the SBOL data model and terms from other ontologies ²⁶. For example, the role property used by instances of the *ComponentDefinition* class is recommended to be defined using terms taken from the Sequence Ontology (SO) ²⁷. Examples of terms used in this property are 'Promoter' (SO:0000167), 'Ribosome Entry Site' (SO:0000139), and 'CDS' (SO:0000316). Another key ontology is the Systems Biology Ontology (SBO) ²⁸, which is used for defining types for *Interaction* instances, such as 'Genetic Production' (SBO:0000589) and roles for *Participation* instances, such as 'Template' (SBO:0000645) and 'Product' (SBO:0000011). Other ontologies commonly used in SBOL include the Gene Ontology (GO) ^{25,29}, and Chemical Entities of Biological Interest (CHEBI) ³⁰.

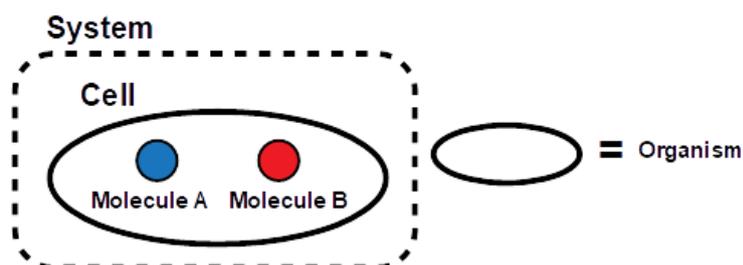


Figure 1. A Cell Encoded depiction using SBOL Visual. A UML diagram of the system shown here is represented in *Supplementary Figure 1*. Using the best practices described here, each cell should be contained within its own system. Entities, such as the two molecules in this diagram, can be represented within the system and the SBOL *Interaction* class can be used to convey that they are contained within the cell. Additional interactions, not shown here for clarity, can also be used to represent behaviour such as active transport of molecules into the cell, or binding of entities to the cell's surface. Taxonomic information about the cell in the system is captured using an instance of the *ComponentDefinition* class, which is referenced from the data structure representing the cell in the cell system. This allows for a distinction between an organism in general, and an actual cell in a system.

Representing Cells in SBOL

When attempting to re-produce a previously designed multicellular system, it is necessary to know some fundamental information about the cells used. The most crucial information comprises the strains that were used, any plasmids transformed into the cells, and the expected functionality. By providing precise taxonomic information, it can be ensured that the correct strains are used when other researchers attempt to re-create a system, hence increasing reproducibility. Finally, it is useful to record and share the exact functionality of a design to ensure that future users select the correct system for their desired application, and to allow for informed modification of the system.

Using the classes and ontologies described in the section above, a recommended approach for representing cells in a biological system using SBOL has been developed. This approach captures: (i) taxonomy, (ii) interactions occurring within the cell, and (iii) components inside the cell (for example DNA and small chemical molecules).

The approach uses an instance of the *ModuleDefinition* class to represent a system that involves a specified type of cell (Figure 1). Usage of the cell type is represented by an instance of the *FunctionalComponent* class inside the *ModuleDefinition*, whose definition is a *ComponentDefinition* instance that is used to capture information about the species and strain of the cell in the design. This *ComponentDefinition* has a type of 'cell' from the Gene Ontology (GO:0005623), and a role of 'physical compartment' (SBO:0000290). Taxonomic information is captured by annotating the class instance with a URI that leads to a description of the strain. As a best practice, and where possible, the organism's species and strain should be defined by providing a link to the relevant entry in the NCBI (National Center for Biotechnology Information) taxonomy database. This standardised approach would allow for easier automated retrieval of information about the organism. Whilst a link to an NCBI entry would be preferable, there are instances where this may not be possible (for example when using a novel strain that is not yet recorded in NCBI). In these cases, it is suggested that a different database, which does contain the organism, is used. If the organism is not in any database, then a description of the organism should be provided.

Other relevant entities, such as inducer molecules or plasmid DNA, are also captured using instances of the *FunctionalComponent* class. Interactions that occur within the system are captured using the *Interaction* and *Participation* classes, and interactions that occur within the cell are specified by including a *Participation* for the cell with a role of 'physical compartment'. An additional *Interaction* class instance can also be used to explicitly define which entities are only present within the cell and, therefore, not available to the rest of the system. This interaction has a type of 'containment' (SBO:0000469) and has at least two participants: the cell, which has a role of 'physical compartment', and one or more contained entities, which have roles of 'contained' (SBO:0000064) (Supplementary Figure 1B).

It should be noted that when a cell is included in a SBOL design, it is actually representing a 'pool' of cells of that type. This is like how, for example, in SBOL genetic production of a protein from a plasmid is interpreted as production of a pool of some number of proteins from a collection of some number of copies of the plasmid. Thus, for example, a containment interaction such as in Supplementary Figure 1B may be interpreted as stating: "in this system, cells of type X contain plasmids of type Y".

Representing Designs with Multiple Cells

Once cells have been individually defined, they can be included in a design for a multicellular system. In systems involving more than one cell, it is important to capture the relative amounts of each cell type, as this can have a large effect on the system's behaviour. Additionally, it is important to define how each cell type interacts with other cells in the system, as these intercellular interactions are usually the basis for a multicellular system's functionality. Intercellular interactions normally occur by the same type of molecule being involved in processes of different cells. For example, two cell types in the system may require the same molecule for metabolic pathways to facilitate cell growth and hence are competing for resources, or one cell may produce a molecule which interacts with genetic circuits in a second cell, which is the basis for intercellular communication.

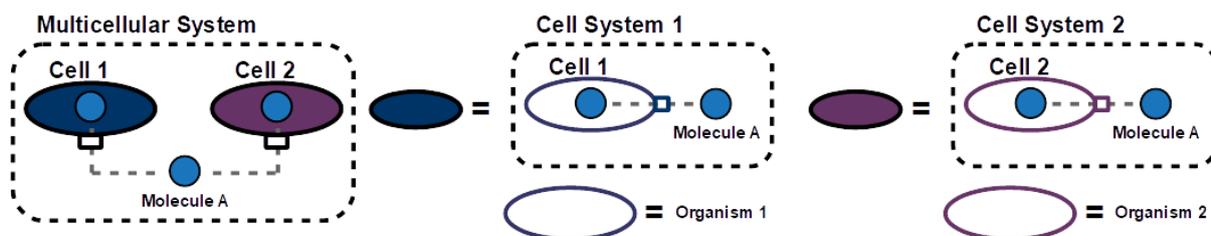


Figure 2. Multicellular System Representation using SBOL Visual. A UML diagram of the system shown here is represented in *Supplementary Figure 2*. In this diagram, a multicellular system composed of two different cells of different organism types is shown. The two cell types are represented by cell system 1 and 2 and are depicted similarly to the cell in *Figure 1*. Here, the two cells contain molecule A which is imported into the cell from the extracellular environment. In SBOL, the multicellular system itself is represented by an instance of the *ModuleDefinition* class. This *ModuleDefinition* contains elements representing the two cells and Molecule A, which are referenced from the original cell systems.

Given the above representation for a single class of cells, the same approach can be used to represent designs that incorporate more than one cell type. At the simplest level, one can simply have one *FunctionalComponent* for each cell type and appropriate *Interaction* and *Participation* instances to specify which aspects of the systems are associated with each cell type.

One can also compose a multicellular system by using the *Module* class to link together *ModuleDefinition* instances that each define a design for a system containing a single type of cell. Figure 2 shows an example of this approach. Here, each *Module* instance has its definition pointing to the *ModuleDefinition* class that is used to represent each single system containing a cell-type. In order to capture links between the same entities present in multiple parts of the same design, the *Module* classes contain instances of the *MapsTo* class. Here, a *MapsTo* class with a refinement value of 'merge' is used to link *FunctionalComponent* classes that represent a cell type in the multicellular system to the *FunctionalComponent* class used to represent the same cell in the lower-level cell system design. Instances of the *MapsTo* class are also used to capture that non-cell entities in the multicellular system are identical to those same entities when used in the cell system design, such as a small molecule produced by one cell population and utilised by another population.

Finally, it is recommended that the proportion of cell types in a multicellular system can be captured using the *Measure* class³¹. The *Measure* class has value, unit, and type properties, which allow specification both of a parameter and of how to interpret it in the context of the biological system. For example, Figure 3 shows how an instance of the *Measure* class can be used to annotate the *Module* instance in the multicellular design which represents a cell system. The *Measure* instance can capture the proportion of cells using any relevant units, including percentage, cell count, mass, or culture volume. It should also be noted that the *Measure* class could be used to help represent structured multicellular systems by describing the location of cells in space.

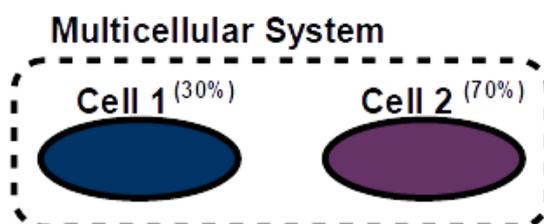


Figure 3. Diagram Depicting how to Capture Cell Ratios using SBOL. A UML diagram of the system shown here is represented in *Supplementary Figure 3*. This diagram shows a multicellular system composed of two undefined cells; Cell 1 and Cell 2. Cell 1 composes 30% of all cells in the system, and Cell 2 composes the other 70%.

Results and Discussion

Having presented a method for using SBOL to represent multicellular designs, in this section it is applied to two recent complex designs, one a sensor system distributed across three types of cells, the other an inducible cell-sorting system with two types of cells.

Example system: A Modular, Multicellular Biosensor

Figure 4 shows an example of a multicellular system, the Modular, Multicellular Biosensor (MMB) described by the Newcastle team for the 2017 International Genetically Engineered Machines (iGEM) competition³³. The MMB consists of three cell types: (i) a detector cell that converts the presence or absence of a specific stimulus into a genetic signal; (ii) a processor cell that modifies the signal from the detector cell in some way (for instance amplifies it); and (iii) a reporter cell that converts the genetic signal into a response, such as colour change or regulation of a metabolic pathway.

The three cell types in the MMB exhibit unidirectional communication, in which the detector cell passes a signal to the processor cell, and the processor cell passes a signal to the reporter cell. This communication is enabled using two orthogonal quorum sensing (QS) mechanisms. The LasIR QS mechanism is used to pass the signal from detector cell to the processor cell. When the stimulus is present, the detector cell produces the acylhomoserine lactone (AHL) C12-HSL (homoserine lactone), which diffuses out of the detector cell and activates gene expression in the processor cell. The RhlIR QS mechanism is used to pass the signal from the processor cell to the reporter cell in a similar way, except that the processor cell produces AHL C4-HSL to activate gene expression in the reporter cell³⁴.

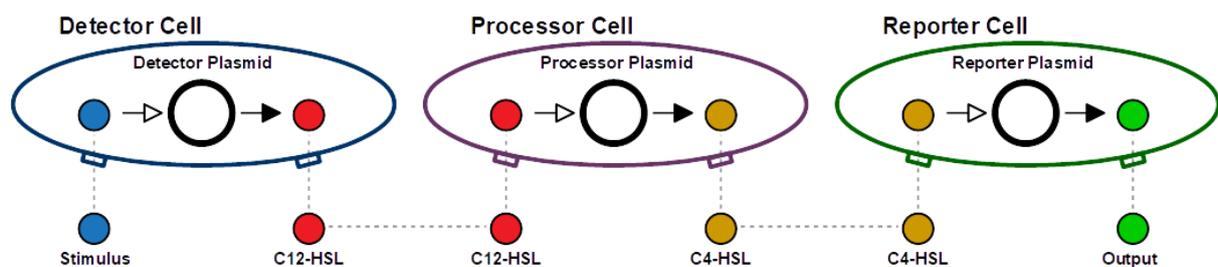


Figure 4. Schematic of a Generic Modular, Multicellular Biosensor. The Modular, Multicellular Biosensor (MMB) Framework described by Newcastle iGEM 2017 consists of three modules: a detector, a signal processor, and a reporter. These three modules are expressed on separate plasmids and transformed into *Escherichia coli* cells. A co-culture of these three cell types is then created to form a functional biosensor. The signal propagates from Detector Cells to Processor Cells to Reporter Cells using AHL (acylhomoserine lactone)-based quorum sensing mechanisms.

Figure 5 illustrates key aspects of the SBOL representation of a variant of the Detector Cell in the MMB, in this case designed to detect IPTG (Isopropyl β -D-1-thiogalactopyranoside). A full XML file with this file can be found in supplementary materials. An instance of the *ModuleDefinition* class is used to represent the system in which the IPTG Detector cell is implemented, and a separate *ComponentDefinition* instance is used to capture taxonomic information about the cell; in this case that it is an *Escherichia coli* DH5 α strain. This *ComponentDefinition* is used to define an instance of the *FunctionalComponent* class, which represents the cell within the system. The important molecules in the design are captured using instances of the *FunctionalComponent* class, and transformation of the cell with a plasmid can be captured in the same way. However, the containment of the plasmid within the cell must also be captured explicitly using a 'containment' interaction. This *Interaction* instance has the host cell and plasmid DNA as participants with roles of 'physical compartment' and 'contained', respectively. Other molecules which are produced by the cell and are not transported out into the extracellular space can also be defined in this way.

It is possible to explicitly capture the movement of molecules in/out of the cell if desired by using the *Interaction* class to define specific transport mechanisms. This approach can provide additional information that may be

important, such as if transportation is passive or relies on additional cellular machinery. In this case, however, we do not add this additional information because it is not anticipated to be of significance for the MMB design.

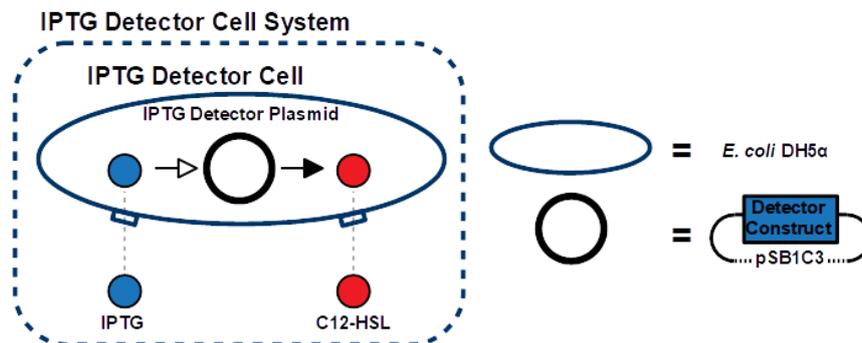


Figure 5. SBOL Visual Depiction of the IPTG Detector Cell. A UML diagram of the system shown here is represented in *Supplementary Figure 4*. This diagram depicts how the IPTG Detector Cell from the MMB in Figure 4 can be represented using SBOL. The cell's species is *E. coli* DH5α and it contains two small molecules (IPTG and C12-HSL), and the IPTG Detector Plasmid. In SBOL, these entities are defined using *ComponentDefinitions* and implemented within the cell as *FunctionalComponents*. The small molecules travel from the extracellular environment into the cell, whereas the plasmid is contained only within the cell. This information is stored in SBOL using the direction property of the *FunctionalComponent* class. An instance of the *Interaction* class could also be used to explicitly state this behaviour, along with any other information.

Figure 6 depicts how a design containing the IPTG Detector Module and Processor Module from the MMB can be captured using SBOL and the best practices described in section 2. An XML file describing this system can also be found in the supplementary materials. In this design, there are two cell populations; each population contains identical bacterial strains (*E. coli* DH5α) but are transformed with different plasmids (either the IPTG Detector Plasmid or the Blank Processor Plasmid). The design contains two *ModuleDefinitions* to capture information about each cell type. These *ModuleDefinitions* include one of the cell types as a *FunctionalComponent*, that is defined by a *ComponentDefinition* that links to the NCBI entry from *E. coli* DH5α, which conveys that both cell populations are composed of identical bacterial strains. The *ModuleDefinitions* representing the cell systems also contain other important entities; in the case of the Detector or Processor plasmid, this is an inducer molecule (IPTG for the Detector system and C12-HSL for the Processor system) and a molecule which is produced by the cell (C12-HSL for the Detector system and C4-HSL for the Processor system). Each of these entities are included as a *FunctionalComponent* which is defined by a *ComponentDefinition*. The small molecule C12-HSL is involved in both cell systems, and therefore the *FunctionalComponents* in both systems are defined by the same *ComponentDefinition*, which directly conveys that this molecule pool is identical.

Each *ModuleDefinition* representing a cell system also contains an *Interaction* which defines the function of that cell system. This design captures that, within the Detector cell, the small molecule IPTG stimulates something on the Detector plasmid to produce C12-HSL, and, within the Processor cell, C12-HSL stimulates the production of C4-HSL. More details could be included at this point (such as exact mechanisms for how IPTG stimulates the production of C12-HSL or the diffusion of the small molecules across the cell walls), but for clarity this functionality is abstracted here. It should also be noted that ordinarily it would be recommended that an additional *Interaction* class be included to capture that the plasmids that are contained within each cell (as depicted in Figure 1), but again this is omitted in Figure 6 for clarity.

Each cell system can now be included in a new *ModuleDefinition* as a *Module* to convey that they are members of a multicellular design. The other important biological entities, such as the small molecules, can also be included in the multicellular design as *FunctionalComponents*, along with the cell populations. *MapsTo* classes are used to explicitly link identical entities between the cell system designs and the multicellular design. In this way, interactions between the cell system become apparent. In this example, the small molecule C12-HSL is produced by the Detector cell population, which then stimulated production of C4-HSL in the Processor cell population, therefore conveying unidirectional communication from the Detector cells to the Processor cells.

Finally, the proportion of each cell population is captured by annotating the *Modules* which represent each cell system in the multicellular design with the *Measure* class. In the example in Figure 6, each cell population is annotated as making up 33.33% of the entire cell population. As this does not add up to 100%, it can be inferred that another cell population may be required, or that there are unknown cell types in the design.

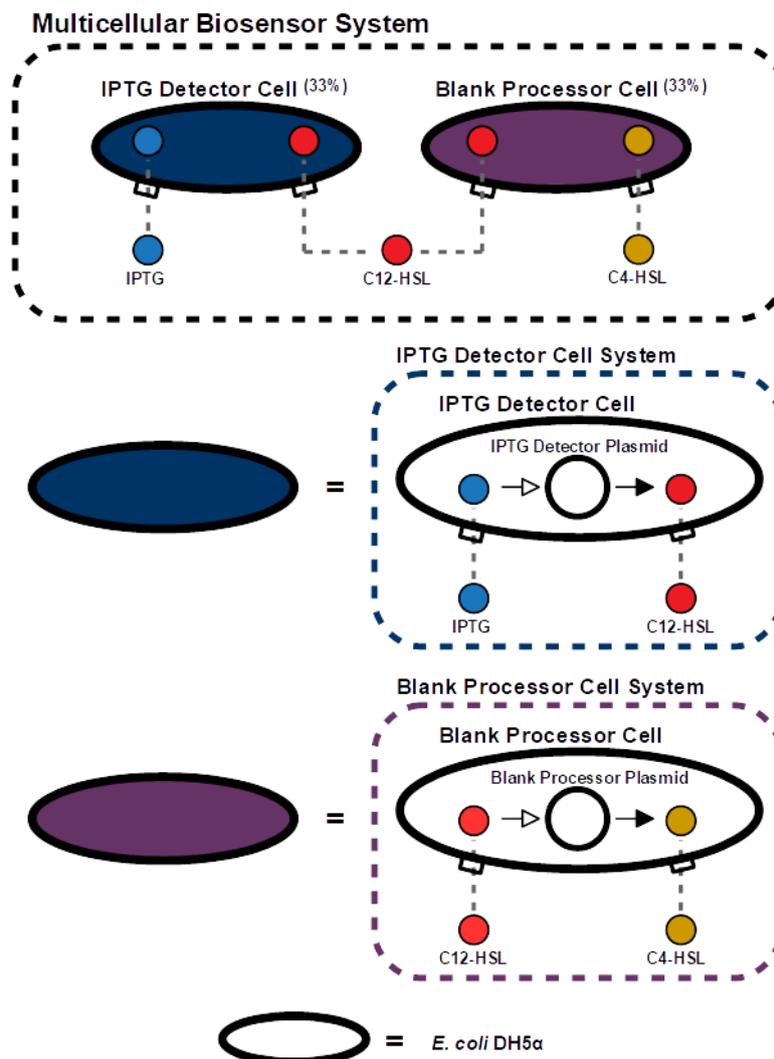


Figure 6. SBOL Visual Diagram Showing Intercellular Interactions using SBOL. A UML diagram of the system shown here is represented in *Supplementary Figure 5*. This diagram depicts how the IPTG Detector Cell and Blank Processor Cell variants of the MMB interact. The two cell types are represented using the same principles described by *Figure 5*. Both cells contain three entities (captured in SBOL as *FunctionalComponent* objects). One of these entities, the small molecule C12-HSL, is present in both cell systems; in the Detector Cell it has a role of ‘product’, and in the Processor Cell it has a role of ‘stimulator’. When both cell types are combined into a multicellular system (represented in SBOL as a *ModuleDefinition*), the sharing of this molecule is captured as a shared *FunctionalComponent*. This feature can be used to capture intercellular interactions in a non-explicit way. In this case, the interaction between the Detector and Processor Cells can be derived as the following: The Detector Cell produces C12-HSL, which stimulates the Processor Cell to produce the small molecule C4-HSL.

An Inducible Cell-Sorting System

Another prototypical example of systems involving multiple types of cells are pattern-formation systems based on cell sorting. Here, we consider a recent work in this area on programmable cell sorting^{35 36}, in which the pattern formed is controlled predictably by mixing cells with high cadherin expression and cells with low cadherin expression. If the cadherins used are all the same, then cell motility will result in high-adhesion cells

gradually sorting into clusters with low-adhesion cells on the outside of each cluster. The shape formed in this manner is controlled by the fraction of high-adhesion cells: above a critical threshold, they form a “sorted ball” comprising a single large cluster with a surface of low-adhesion cells. At lower fractions, the high-adhesion cells instead form “polka dots”, with small clusters embedded in a unified background of low-adhesion cells. Controlling cadherin expression (e.g. by adding a synthetic expression cassette with an inducible promoter) can further allow sorting behaviour to be selected dynamically.

Figure 7 shows an example of how such an inducible cell-sorting system can be represented in SBOL following the recommendations given above. In this case, the two strains of cells are both Chinese Hamster Ovary (CHO) cells, which are natively low in cadherin expression and clump only weakly. One of the two strains, however, has been transformed with the addition of a synthetic Doxycycline-inducible cadherin expression cassette. Within the *ModuleDefinition* describing this system, each cell strain’s representation is based on a *FunctionalComponent*, both using the same definition of a CHO cell *ComponentDefinition*. The inducible CHO (iCHO) cells, however, are enhanced with an Interaction of type containment that sets them as the physical compartment that contains the *FunctionalComponent* instantiations of both the cadherin cassette and the cadherin that is its output. The production of cadherin from this cassette is represented by a second Interaction (additional details of the structure of the cassette and its induction by Doxycycline are omitted for space purposes). The actual cell-to-cell adhesion relationships that implement the sorting behaviour are included in the *ModuleDefinition* as more Interactions, each representing one of the three adhesion relations in the system: CHO cells with CHO cells, CHO cells with iCHO cells, and iCHO cells with iCHO cells. Finally, the parameters and dynamics of this system may be represented by attaching Measures to the *FunctionalComponents* and a Model to the *ModuleDefinition* (not shown).

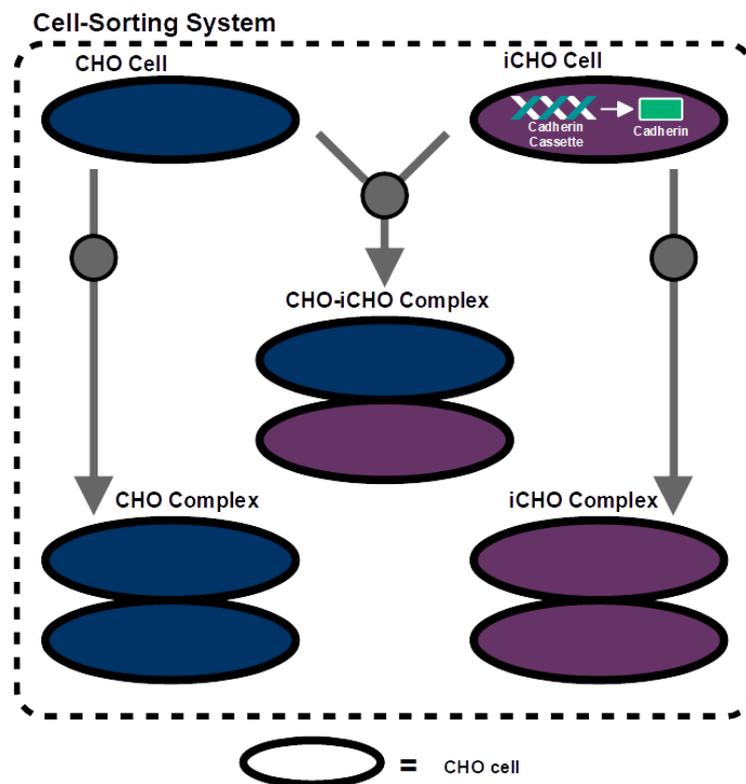


Figure 7. Visual Representation of a Cell Sorting System using SBOL. A UML diagram of the system shown here is represented in *Supplementary Figure 6*. Two cell types are captured in this system, both of which are CHO cells. The natural CHO cells (blue) clump together weakly (shown by the association glyph). In SBOL, this clumping is represented using the *Interaction* class. The iCHO cells (purple) have been transfected with a cadherin cassette encoding cadherin, which enhances cell clumping. This allows the iCHO cells to clump together at a greater rate. The CHO and iCHO cells can also associate to form a CHO-iCHO cell complex.

Discussion

Whilst the SBOL data model has previously been used to capture information about genetic constructs and intracellular interactions, it has not been widely used to describe and share information about multicellular systems. This paper describes a set of best practices for how multicellular system designs can be captured in a standard way using SBOL. Examples have been provided to illustrate specific concepts and demonstrate feasibility, and valid illustrative SBOL documents are available (Supplementary Information). The SBOL documents provided were created using the currently available python SBOL libraries.

The best practices described focus on ensuring that there is sufficient flexibility to describe a wide variety of multicellular designs, and incorporates the concept of modular design, which is an important principle in synthetic biology. Additionally, to ensure that the approach described here is as backward compatible as possible, terminology from ontologies already widely used in the current version of SBOL are used, and no new classes or features are required. Additionally, it is recommended that, where possible, existing resources such as the NCBI database be used to reference in-depth information to avoid replication of information.

The best practices described here have been included with the latest version of SBOL version 2 (2.3.0). It should be noted that SBOL version 3 is now available, however most libraries and tools are not compatible with this latest version and the underlying data model is subject to change. Therefore, the approaches described here for capturing information about multicellular systems using SBOL 2 should be relevant for the coming years.

Funding Sources

This document does not contain technology or technical data controlled under either the U.S. International Traffic in Arms Regulations or the U.S. Export Administration Regulations. This work was supported in part by NSF Expeditions in Computing Program Award #1522074 as part of the Living Computing Project and by the Defense Advanced Research Projects Agency under Contract No. W911NF-17-2-0098. The views, opinions, and/or findings expressed are of the author(s) and should not be interpreted as representing official views or policies of the Department of Defense or the U.S. Government.

A.G.-M. was supported by the SynBio3D project of the UK Engineering and Physical Sciences Research Council (EP/R019002/1) and the European CSA on biological standardization BIOROBOOST (EU grant number 820699)

Acknowledgements

We thank Jesse Tordoff and Ron Weiss of MIT for contributing information about inducible CHO cell lines.

Supplementary Information

Supplementary Diagrams: UML diagrams for the systems described in the paper

SBOL File 1: SBOL file representing the system depicted in *Figure 1*

SBOL File 2: SBOL file representing the system depicted in *Figure 2*

SBOL File 3: SBOL file representing the system depicted in *Figure 5*

SBOL File 4: SBOL file representing the system depicted in *Figure 6*

SBOL File 5: SBOL file representing the system depicted in *Figure 7*

References

- 1 Amos, M.; Goñi-Moreno, A. (2018), Cellular Computing and Synthetic Biology, *In: Stepney S.; Rasmussen S.; Amos M. (eds) Computational Matter. Natural Computing Series. Springer, Cham*, pp 93-110.
- 2 Church, G. M.; Elowitz, M. B.; Smolke, C. D.; Voigt, C. A.; and Weiss, R. (2014), Realizing the potential of synthetic biology. *Nat. Rev. Mol. Cell Biol.* 15, pp 289–294.
- 3 Elowitz, M. B.; and Leibler, S. (2000), A synthetic oscillatory network of transcriptional regulators, *Nature*, 403, pp 335–338.
- 4 Gardner, T. S.; Cantor, C. R.; and Collins, J. J. (2000), Construction of a genetic toggle switch in *Escherichia coli*, *Nature*, 403, pp 339-342.

- 5 Karig, D. K. (2017) Cell-free synthetic biology for environmental sensing and remediation, *Curr. Opin. Biotechnol.*; 45, pp 69–75.
- 6 Reed, J.; Stephenson, M. J.; Miettinen, K.; Brouwer, B.; Leveau, A.; Brett, P.; Goss, R. J.; Goossens, A.; O'Connell, M. A.; and Osbourn, A. (2017), A translational synthetic biology platform for rapid access to gram-scale quantities of novel drug-like molecules, *Metab. Eng.*; 42, pp 185–193.
- 7 Urrios, A.; Gonzalez-Flo, E.; Canadell, D.; de Nadal, E.; Macia, J.; and Posas, F. (2018), Plug-and-Play Multicellular Circuits with Time-Dependent Dynamic Responses, *ACS Synth. Biol.*; 7, pp 1095–1104.
- 8 Johns, N. I.; Blazejewski, T.; Gomes, A. L.; and Wang, H. H. (2016), Principles for designing synthetic microbial communities, *Curr. Opin. Microbiol.*; 31, pp 146–153.
- 9 Yin, X.; Mead, B. E.; Safaee, H.; Langer, R.; Karp, J. M.; and Levy, O. (2016), Engineering stem cell organoids, *Cell stem cell*, 18, no. 1, pp 25–38.
- 10 Mimee, M.; Citorik, R. J.; and Lu, T. K. (2016), Microbiome therapeutics—advances and challenges, *Adv. Drug Delivery Rev.*; 105, pp 44–54.
- 11 Kylilis, N.; Tuza, Z. A.; Stan, G.-B.; and Polizzi, K. M. (2018), Tools for engineering coordinated system behaviour in synthetic microbial consortia, *Nat. Commun.*; 9, pp 2677.
- 12 Goni-Moreno, A.; Redondo-Nieto, M.; Arroyo, F.; and Castellanos, J. (2011), Biocircuit design through engineering bacterial logic gates, *Nat. Comput.*; 10, pp 119–127.
- 13 Regot, S.; Macia, J.; Conde, N.; Furukawa, K.; Kjellén, J.; Peeters, T.; Hohmann, S.; de Nadal, E.; Posas, F.; and Solé, R. (2011), Distributed biological computation with multicellular engineered networks, *Nature*, 469, pp 207–211.
- 14 Macia, J.; Posas, F.; and Sole, R. V. (2012), Distributed computation: The new wave of synthetic biology devices, *Trends Biotechnol.*; 30, pp 342–349.
- 15 Goñi-Moreno, A.; Amos, M.; and de la Cruz, F. (2013), Multicellular Computing Using Conjugation for Wiring, *PLoS ONE*, 8, e65986.
- 16 Grozinger, L.; Amos, M.; Gorochoowski, T.; Carbonell, P.; Oyarzún, D.; Stoof, R.; Fellermann, H.; Zuliani, P.; Tas, H.; Goñi-Moreno, A. (2019) Pathways To Cellular Supremacy In Biocomputing. *Nat. Commun.*; 10.; 5250
- 17 Tsoi, R.; Wu, F.; Zhang, C.; Bewick, S.; Karig, D.; You, L. (2018), Metabolic Division Of Labor In Microbial Systems, *Proc. Natl. Acad. Sci.*; 115, pp 2526–2531.
- 18 Kong, W.; Meldgin, D.; Collins, J.; Lu, T. Designing Microbial Consortia With Defined Social Interactions, *Nat. Chem. Biol.*; 14, pp 821–829
- 19 Voyvodic, P.L.; Pandi, A.; Koch, M.; Conejero, I.; Valjent, E.; Courtet, P.; Renard, E.; Faulon, J.; and Bonnet J. (2019), Plug-and-play metabolic transducers expand the chemical detection space of cell-free biosensors. *Nat. Commun.*; 10
- 20 Roehner, N.; Beal, J.; Clancy, K.; Bartley, B.; Misirli, G.; Grünberg, R.; Oberortner, E.; Pocock, M.; Bissell, M.; Madsen, C.; Nguyen, T.; Zhang, M.; Zhang, Z.; Zundel, Z.; Densmore, D.; Gennari, J.; Wipat, A.; Sauro, H.; Myers, C. (2016), Sharing Structure and Function in Biological Design with SBOL 2.0, *ACS Synth. Biol.*; 5, pp 498–506.
- 21 Benson, D. A.; Karsch-Mizrachi, I.; Lipman, D. J.; Ostell, J.; and Wheeler, D. L. (2005), GenBank, *Nucleic Acids Res.*; 33, pp 34–38.
- 22 Goni-Moreno, A.; Carcajona, M.; Kim, J.; Martinez-Garcia, E.; Amos, M.; and de Lorenzo, V. (2016), An Implementation-Focused Bio/Algorithmic Workflow for Synthetic Biology, *ACS Synth. Biol.*; 5, pp 1127–1135.
- 23 Myers, C. J.; Beal, J.; Gorochoowski, T. E.; Kuwahara, H.; Madsen, C.; McLaughlin, J. A.; Misirli, G.; Nguyen, T.; Oberortner, E.; Samineni, M.; Wipat, A.; Zhang, M.; and Zundel, Z. (2017), A standard-enabled workflow for synthetic biology, *Biochem. Soc. Trans.*; 45, pp 793–803.
- 24 Madsen, C.; Goñi Moreno, A.; P, U.; Palchick, Z.; Roehner, N.; Atallah, C.; Bartley, B.; Choi, K.; Cox, R.; Gorochoowski, T.; Grünberg, R.; Macklin, C.; McLaughlin, J.; Meng, X.; Nguyen, T.; Pocock, M.; Samineni, M.; Scott-Brown, J.; Tarter, Y.; Zhang, M.; Zhang, Z.; Zundel, Z.; Beal, J.; Bissell, M.; Clancy, K.; Gennari, J.; Misirli, G.; Myers, C.; Oberortner, E.; Sauro, H.; Wipat, A. (2019), Synthetic Biology Open Language (SBOL) Version 2.3, *Journal of Integrative Bioinformatics*, 16 (2)
- 25 Ashburner, M.; Ball, C.; Blake, J.; Botstein, D.; Butler, H.; Cherry, J.; Davis, A.; Dolinski, K.; Dwight, S.; Eppig, J.; Harris, M.; Hill, D.; Issel-Tarver, L.; Kasarskis, A.; Lewis, S.; Matese, J.; Richardson, J.; Ringwald,

- M.; Rubin, G.; Sherlock, G. (2000), Gene Ontology: tool for the unification of biology, *Nat. Genet.*; 25, pp 25–29.
- 26 Misirli, G.; Taylor, R.; Goñi-Moreno, A.; McLaughlin, J. A.; Myers, C.; Gennari, J. H.; Lord, P.; and Wipat A. (2019), SBOL-OWL: An Ontological Approach for Formal and Semantic Representation of Synthetic Biology Information, *ACS Synth. Biol.*; 8 (7), pp 1498-1514
- 27 Eilbeck, K.; Lewis, S. E.; Mungall, C. J.; Yandell, M.; Stein, L.; Durbin, R.; and Ashburner, M. (2005), The Sequence Ontology: a tool for the unification of genome annotations, *Genome Biol.*; 6, R44.
- 28 Juty, N.; and Novere, N.; *Encyclopedia of Systems Biology*; Springer New York: New York, NY, 2013; pp 2063–2063.
- 29 The Gene Ontology Consortium (2017), Expansion of the Gene Ontology knowledge base and resources, *Nucleic Acids Res.*; 45, D331–D338.
- 30 Hastings, J.; Owen, G.; Dekker, A.; Ennis, M.; Kale, N.; Muthukrishnan, V.; Turner, S.; Swainston, N.; Mendes, P.; and Steinbeck, C. (2016), ChEBI in 2016: Improved services and an expanding collection of metabolites, *Nucleic Acids Res.*; 44, pp 1214–1219.
- 31 Beal, J.; and Roehner, N. (2018), SEP 028 – Measurements/Parameters and Units, github.com/SynBioDex/SEPs/blob/master/sep_028.md, accessed: 10/01/2020
- 32 Gkoutos, G. V.; Schofield, P. N.; & Hoehndorf, R. (2012), The Units Ontology: a tool for integrating units of measurement in science, *Database: the journal of biological databases and curation*, bas033
- 33 Newcastle iGEM 2017 (2017), Sensynova, 2017.igem.org/Team:Newcastle, accessed: 10/01/2020
- 34 Papenfort, K.; and Bassler, B. L. (2016), Quorum sensing signal-response systems in Gram-negative bacteria, *Nat. Rev. Microbiol.*; 14, pp 576–588.
- 35 Tordoff, J.; Beal, J.; Weiss, R.; Bartley, B.; Gumuskaya, G.; Kiwimagi, K.; Krajnc, M.; Lebo, K.; Shvartsman, S.; Tseng, A.; and Walczak, N. (2018), Toward Programming 3D Shape Formation in Mammalian Cells, *10th International Workshop on Bio-Design Automation (IWBDA)*
- 36 Tordoff, J.; Krajnc, M.; Walczak, N.; Lima, M.; Beal, J.; Shvartsman, S.; and Weiss, R. (2020), Incomplete cell sorting creates engineerable structures with long term stability, *in revision*