Received: 24 January 2023 | Revised: 31 March 2023 | Accepted: 10 April 2023

DOI: 10.1002/psp4.12972

TUTORIAL

QSP Designer: Quantitative systems pharmacology modeling with modular biological process map notation and multiple language code generation

Richard J. Matthews[1] | David Hollinshead[1] | Daniel Morrison[1] | Piet H. van der Graaf[1,2] | Andrzej M. Kierzek[1,3]

[1]Certara UK, Sheffield, UK

[2]Leiden Academic Centre for Drug Research, Universiteit Leiden, Leiden, the Netherlands

[3]School of Biosciences and Medicine, University of Surrey, Guildford, UK

Correspondence
Andrzej M. Kierzek, Certara UK, 1 Concourse Way, Sheffield S1 2BJ, UK.
Email: andrzej.kierzek@certara.com

Abstract

Typical Quantitative Systems Pharmacology (QSP) workflows involve discussion of biology, supported by graphical diagrams, followed by construction of large Ordinary Differential Equation models. QSP Designer facilitates this process by providing enhanced graphical notation, which enables hierarchical presentation with modules and handling of combinatorial complexity with diagram node arrays. Whereas the software includes a simulation engine, a major feature is full model code generation in MATLAB, R, C, and Julia to support multiple modeling communities.

Quantitative Systems Pharmacology (QSP)[1] combines pharmacokinetic (PK) and pharmacodynamic (PD) modeling with Systems Biology[2] models of biology underlying disease to support development of new therapeutics. A typical workflow[3] starts from extensive literature mining and discussion with domain experts to agree the scope and granularity of the biological mechanisms represented in the model. Because biologists commonly use informal mechanistic diagrams in textbooks, publications, and presentations, specification of the biological scope and main modeling assumptions usually involve construction of a graphical biological process map. The map is then converted into a mathematical model, most frequently formulated as a system of Ordinary Differential Equations (ODE). The model is subsequently calibrated with data and validated by simulation of datasets not used for calibration. Lessons learned from discrepancies between model predictions and data inform changes to the model. Frequently, multiple learn and confirm cycles are conducted before the model gains sufficient confidence of the drug development team to inform their decisions. If the biological process map is drawn informally in a diagram editor separate from modeling software, the inconsistencies between original graphical representation and complex mathematical model inevitably arise during the iterative modeling project. This confuses communication, increases effort of passing the model between team members, and may negatively impact confidence of interdisciplinary experts in the model. These problems can be resolved by the modeling software providing a biological process map editor with graphical notation, allowing creation of diagrams easy to understand by biologists, and at the same time formally associating state variables, rate equations, parameters, algebraic assignments, events, and constraints with the map.

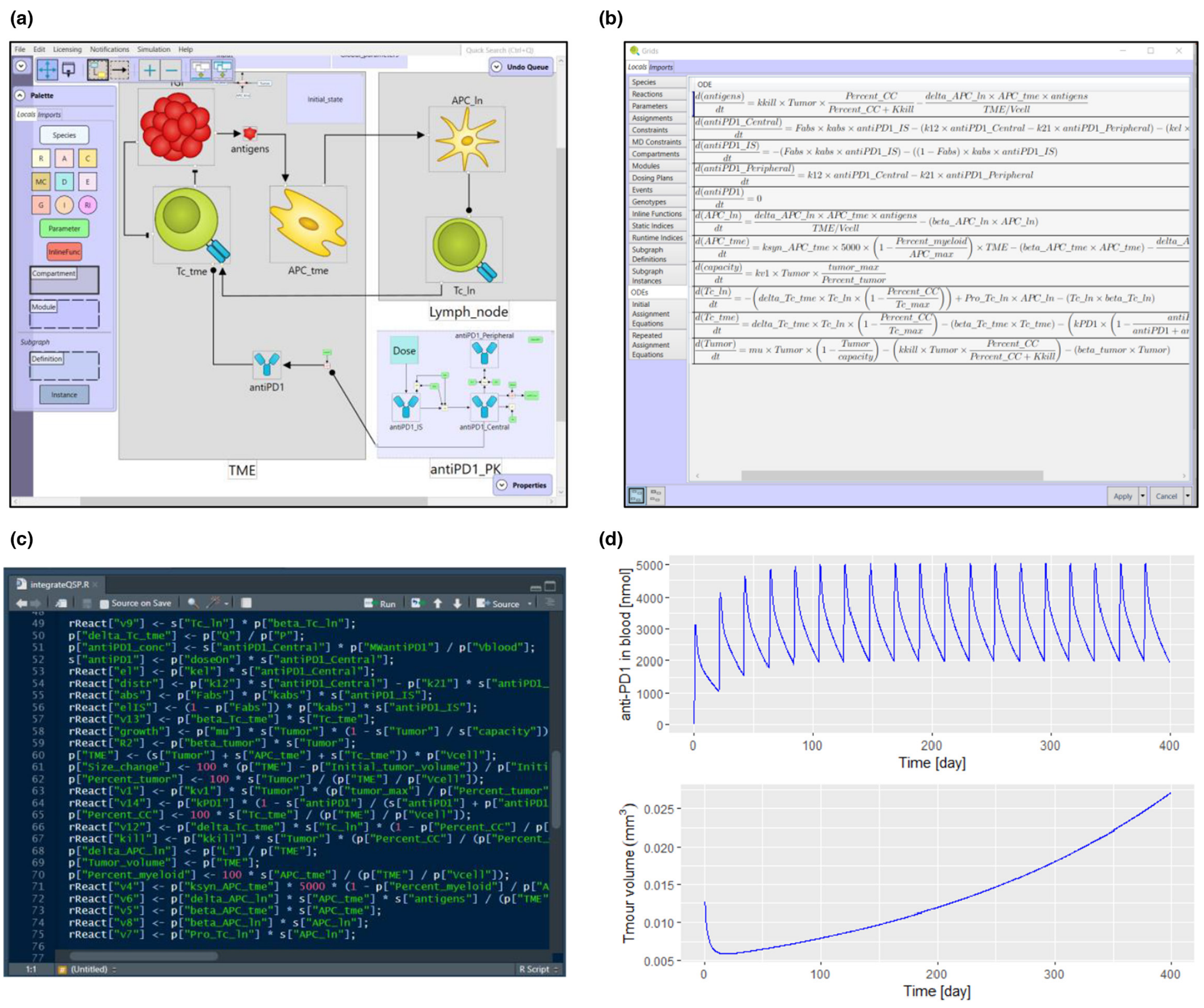
The idea of using formal graphical notation and graphical model building is of course not new nor invented by QSP and Systems Biology communities. Formal technical drawings and blueprints have been used in engineering long before computers were invented. Electronic circuit diagrams formally specify complex technological systems allowing precise design communication between engineers who may not even speak the same language. In software engineering, Uniform Modeling Language[4] is a commonly used graphical notation supporting every stage of design and development. The idea to represent chemical reactions and complex, concurrent, technological systems by formal graphical notation dates back to the 1960s when Carl Petri introduced bipartite graphs representing system variables and transitions.[5] Petri Nets became an active field of research producing tools and algorithms applied in multiple fields of science and technology. Graphical mathematical model editing, coupled to simulation algorithms has been used in numerous software tools, such as the popular SimuLink[6] toolbox of MATLAB.

Systems Biology[2] emerged as a field aiming to bring the methodology of exact science and engineering to molecular biology, which included research on formal graphical notation and software tools for graphical model building. The Systems Biology Graphical Notation (SBGN)[7] implemented first in CellDesigner software[8] is commonly used. Petri Nets were also applied. For example, Snoopy software provides a Petri Net editor and a wide range of ODEs, stochastic, and hybrid simulation algorithms.[9] In the PK/PD field, models are simpler and have standard structures making graphical model editors less beneficial and popular. However, Certara Phoenix software does allow graphical model building. When Systems Biology motivated the expansion of mechanistic modeling in PK/PD, leading to emergence of the QSP field, graphical model editors were also adopted. The MATLAB SimBiology toolbox,[10] a frequently used tool in QSP, provides a graphical model editor with bipartite graph notation, similar to SBGN and Petri Nets, where state variables are represented as species interacting through reactions representing rate laws. The MoBi[11] software, part of the Open QSP[11] environment, also provides a graphical editor using species/reactions notations.

QSP model editors need to address considerable challenges and trade-offs. Graphical notation must be flexible and easy to understand for scientists who do not have modeling experience, but at the same time formal, so they can represent mathematical models as precisely as possible. Ideally, notation should represent the full relationship among variables, parameters, and rate laws, but for a model with tens or hundreds of variables and rate law terms this may lead to overcomplicated diagrams. Frequently, models involve combinatorial explosions of species and reactions (for example, full allosteric binding model) necessitating creation of very large graphs, with repeated copies of the same basic sub-graph. Having encountered these challenges in our QSP practice, we proposed an extended graphical notation and implemented it in a software tool called QSP Designer. We introduced modules to represent complex models hierarchically, with different levels of granularity addressing different audiences. We also introduced arrays of modeling objects and rules for automatic rate law generation to handle the challenge of combinatorial explosions within the graphical language. We recognize that the QSP community is using several different mathematical modeling environments and therefore created a software where the model formulated in the diagram editor can be exported to multiple languages. In the following part of this tutorial, we will introduce graphical modeling with a Modular Biological Process Map and demonstrate its benefits for both the QSP modeling community and the wider community of pharmacology scientists interacting with QSP modelers and possibly using QSP models. We will also demonstrate how the model built with the QSP Designer can be exported to MATLAB, R, or Julia code and therefore used by different modeling communities.

INTRODUCTION TO QSP DESIGNER AND MODULAR BIOLOGICAL PROCESS MAP

QSP Designer is a QSP modeling and simulation software with graphical, Modular Biological Process Map interface (Figure 1). The user builds the model using the graphical editor (Figure 1a). The editor provides a palette with objects, which can be moved to the canvas and connected by edges. Each object has properties, such as rate laws and initial values, that are specified via a property inspector. The diagram is compiled into the ODE system, which can also be examined within the graphical user interface (Figure 1b). The model can then be calibrated and simulated using the parameter estimation and simulation engine available within QSP Designer. Simulation output is saved either in formatted Excel files with plots or in CSV files facilitating custom plotting. Alternatively, the user may choose to use the model within a mathematical modeling environment to develop custom analysis methods or capitalize on a legacy of simulation and analysis code already available within the group and community. The QSP Designer provides model code export to R (Figure 1c), R with model equations in C, Julia, and MATLAB. Full model code is exported, rather than the code using calls to application programming interface (API). Therefore, the user can transfer the model to mathematical modeling environment of their choice and use it independently of QSP Designer. Figure 1d shows an example simulation output, which can be produced with stand-alone engine or exported code.



ODE

$$\frac{d(antigens)}{dt} = kkill \times Tumor \times \frac{Percent\_CC}{Percent\_CC + Kkill} - \frac{delta\_APC\_ln \times APC\_tme \times antigens}{TME/Vcell}$$

$$\frac{d(antiPD1\_Central)}{dt} = Fabs \times kabs \times antiPD1\_IS - (k12 \times antiPD1\_Central - k21 \times antiPD1\_Peripheral) - (kel \times$$

$$\frac{d(antiPD1\_IS)}{dt} = -(Fabs \times kabs \times antiPD1\_IS) - ((1 - Fabs) \times kabs \times antiPD1\_IS)$$

$$\frac{d(antiPD1\_Peripheral)}{dt} = k12 \times antiPD1\_Central - k21 \times antiPD1\_Peripheral$$

$$\frac{d(antiPD1)}{dt} = 0$$

$$\frac{d(APC\_ln)}{dt} = \frac{delta\_APC\_ln \times APC\_tme \times antigens}{TME/Vcell} - (beta\_APC\_ln \times APC\_ln)$$

$$\frac{d(APC\_tme)}{dt} = ksyn\_APC\_tme \times 5000 \times \left(1 - \frac{Percent\_myeloid}{APC\_max}\right) \times TME - (beta\_APC\_tme \times APC\_tme) - \frac{delta\_A}{}$$

$$\frac{d(capacity)}{dt} = kv1 \times Tumor \times \frac{tumor\_max}{Percent\_tumor}$$

$$\frac{d(Tc\_ln)}{dt} = -\left(delta\_Tc\_tme \times Tc\_ln \times \left(1 - \frac{Percent\_CC}{Tc\_max}\right)\right) + Pro\_Tc\_ln \times APC\_ln - (Tc\_ln \times beta\_Tc\_ln)$$

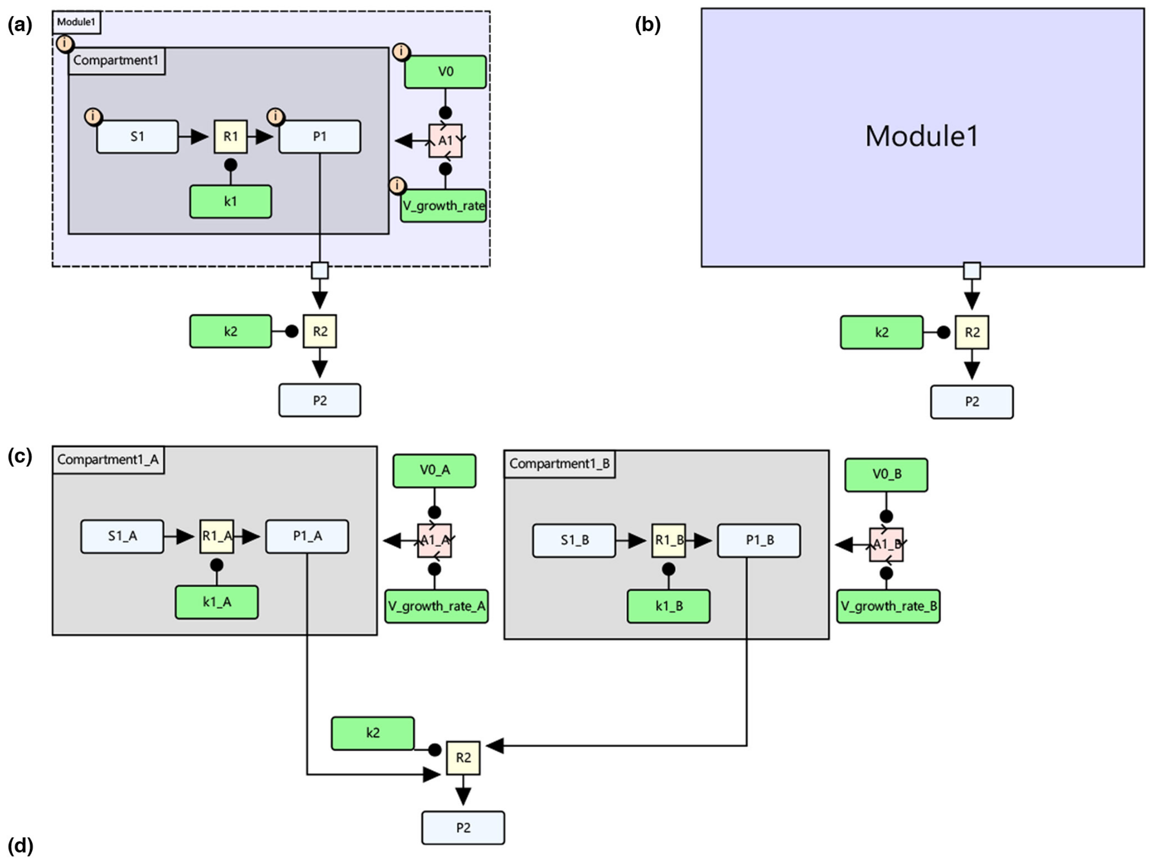
$$\frac{d(Tc\_tme)}{dt} = delta\_Tc\_tme \times Tc\_ln \times \left(1 - \frac{Percent\_CC}{Tc\_max}\right) - (beta\_Tc\_tme \times Tc\_tme) - \left(kPD1 \times \left(1 - \frac{antiI}{antiPD1 + ar}\right.\right.$$

$$\frac{d(Tumor)}{dt} = mu \times Tumor \times \left(1 - \frac{Tumor}{capacity}\right) - \left(kkill \times Tumor \times \frac{Percent\_CC}{Percent\_CC + Kkill}\right) - (beta\_tumor \times Tumor)$$

```
rReact["v9"] <- s["Tc_ln"] * p["beta_Tc_ln"];
p["delta_Tc_tme"] <- p["Q"] / p["P"];
p["antiPD1_conc"] <- s["antiPD1_Central"] * p["MWantiPD1"] / p["Vblood"];
s["antiPD1"] <- p["doseOn"] * s["antiPD1_Central"];
rReact["el"] <- p["kel"] * s["antiPD1_Central"];
rReact["distr"] <- p["k12"] * s["antiPD1_Central"] - p["k21"] * s["antiPD1_
rReact["abs"] <- p["Fabs"] * p["kabs"] * s["antiPD1_IS"];
rReact["elIS"] <- (1 - p["Fabs"]) * p["kabs"] * s["antiPD1_IS"];
rReact["v13"] <- p["beta_Tc_tme"] * s["Tc_tme"];
rReact["growth"] <- p["mu"] * s["Tumor"] * (1 - s["Tumor"] / s["capacity"])
rReact["R2"] <- p["beta_tumor"] * s["Tumor"];
p["TME"] <- (s["Tumor"] + s["APC_tme"] + s["Tc_tme"]) * p["Vcell"];
p["Size_change"] <- 100 * (p["TME"] - p["Initial_tumor_volume"]) / p["Initi
p["Percent_tumor"] <- 100 * s["Tumor"] / (p["TME"] / p["Vcell"]);
rReact["v1"] <- p["kv1"] * s["Tumor"] * (p["tumor_max"] / p["Percent_tumor"
rReact["v14"] <- p["kPD1"] * (1 - s["antiPD1"] / (s["antiPD1"] + p["antiPD1
p["Percent_CC"] <- 100 * s["Tc_tme"] / (p["TME"] / p["Vcell"]);
rReact["v12"] <- p["delta_Tc_tme"] * s["Tc_ln"] * (1 - p["Percent_CC"] / p[
rReact["kill"] <- p["kkill"] * s["Tumor"] * (p["Percent_CC"] / (p["Percent_
p["delta_APC_ln"] <- p["L"] / p["TME"];
p["Tumor_volume"] <- p["TME"];
p["Percent_myeloid"] <- 100 * s["APC_tme"] / (p["TME"] / p["Vcell"]);
rReact["v4"] <- p["ksyn_APC_tme"] * 5000 * (1 - p["Percent_myeloid"] / p["A
rReact["v6"] <- p["delta_APC_ln"] * s["APC_tme"] * s["antigens"] / (p["TME"
rReact["v5"] <- p["beta_APC_tme"] * s["APC_tme"];
rReact["v8"] <- p["beta_APC_ln"] * s["APC_ln"];
rReact["v7"] <- p["Pro_Tc_ln"] * s["APC_ln"];
```

FIGURE 1 Overview of the QSP Designer. (a) The user builds the model using the Modular Biological Process Map Editor. (b) The ODE model is compiled and ODEs can be examined within the QSP Designer interface. (c) The user can choose to generate model code in R, R with model equations in C, Julia, or MATLAB. (d) The user simulates the model using exported code or simulation engine provided within the QSP Designer. ODE, Ordinary Differential Equation; QSP, Quantitative Systems Pharmacology.

QSP Designer uses Modular Biological Process Map graphical notation to facilitate building and communication of complex models, saved in workspaces, within interdisciplinary scientific communities. Figure 2 shows the basic graphical notation, a full set of symbols and formal specification is available in Section 1 of Data S1. As depicted in Figure 2a, the core graphical building blocks are nodes representing species, parameters, and reactions, edges connecting them, and compartments. The chemical species symbol represents a state variable. Reactions represent changes to state variables and contain rate law formulas. Substrate and product edges connecting species to reactions define whether reactions consume (decrease) or produce (increase) the species (state variable value). Parameters of rate laws can be connected to reactions by modifier edges to visualize parameter dependencies. Modifier edges, indicating that quantities connected to a reaction are not changed, can also be used to connect a species to a reaction in the case when the state variable value is used as rate law parameter, but not changed (e.g., enzyme concentration in enzyme catalysis reaction). Upon compilation of the ODE system, species define ODEs, reactions define ODE terms, and edges define the sign (positive or negative) with which each term is included in the ODE. Assignment symbols allow re-setting of species or parameter values either at the start of simulation (initial assignment) or at every iteration of the solver (repeated assignment). The assignment symbol contains an algebraic formula.



(d)

$$\frac{d(P1\_A)}{dt} = k1 \times S1\_A - (k2 \times P1\_A)$$

$$\frac{d(P1\_B)}{dt} = k1 \times S1\_B - (k2 \times P1\_B)$$

$$\frac{d(P2)}{dt} = k2 \times P1\_A + k2 \times P1\_B$$

$$\frac{d(S1\_A)}{dt} = -(k1 \times S1\_A)$$

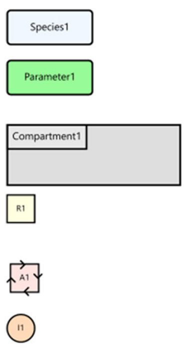
$$\frac{d(S1\_B)}{dt} = -(k1 \times S1\_B)$$

(e)

$$Compartment1\_A = V0\_A + V\_growth\_rate\_A \times t$$

$$Compartment1\_B = V0\_B + V\_growth\_rate\_B \times t$$

(f)



- Represents states in ODE model
- May be converted to arrays via index attachment

- Represents parameters in ODE model
- May be time-varying due to assignments
- May be converted to arrays via index attachment

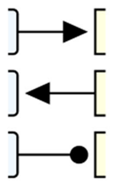
- Represents generalised volumes containing species
- Like parameters in terms of assignment, indices etc.

- Represents reactions that consume and produce species or, more generally, ODE terms
- Implicit array due to referencing indexed variables

- Assigns values of evaluated expressions to quantities
- Implicit conversion to array

- Has property to define range as list of names
- Converts quantities into arrays via attachment

(g)



Indicates that the species is consumed by the reaction and is in-scope to be used in its rate expression

Indicates that the species is produced by the reaction but is not in-scope to be used in its rate expression

Indicates that the node that the edge connects from is in-scope to be used in expressions of the node connected to

FIGURE 2 Basic elements of the Modular Biological Process Map notation. (a) A simple model with a single reaction between substrate and product species in a compartment with a rate constant parameter is created. A repeated assignment is added to make the compartment volume vary with time. An index is attached to species, parameters, and the compartments to convert into an array of parallel models. Part of the model is placed within a module and connected to the rest via a channel. (b) The module is collapsed allowing hierarchical representation. (c) An equivalent model without indices. (d) When the ODE system is compiled, indices are used to automatically generate multiple species and parameter names as well as reactions and assignments, thus combining graphical modeling and scripting functionalities. (e) The algebraic equations for compartment volumes. (f) Definition of basic node symbols used in examples a–c. (g) Definition of basic edge types used. See Data S1 for documentation of all features. ODE, Ordinary Differential Equation.

Parameter dependencies can be visualized by connecting parameter symbols to assignments with modifier edges. Compartments contain information about volumes. Compilation of the graphical notation to ODE system involves unit dimensionality verification and unit conversion, allowing user to input values with suffixes (e.g., milli and micro) of their convenience. The unit system also allows the automatic generation of volume multiplications for reactions which transfer substance between compartments.

Building large QSP models using only the basic graphical notation described above is challenging, as diagrams quickly become overcomplicated and thus unsuitable for communication of the model to an interdisciplinary audience. In QSP Designer, we introduced modules, which encapsulate part of the model and connect it to other modeling objects through interfaces. The module can be collapsed, hiding the underlying complexity, or expanded to allow full examination. Modules are used for visualization purposes only and do not translate to ODEs. In case study 1, we will show how modules can be used to present cell / cell interactions to biology experts whereas also allowing modelers to access all details. Another challenge frequently encountered in graphical model building is combinatorial explosion of species and reactions due to binding reactions (e.g., allosteric binding) or repeated occurrences of the same model structure (different clones in immune system models, physiologically-based pharmacokinetic [PBPK] models for multiple compounds). To enable handling of combinatorial complexity, we introduced indices, which define arrays of species and parameters. Upon compilation of an ODE system, QSP Designer generates ODEs automatically for all array elements. Such an automatic generation of ODEs usually requires coding by expert modeler. QSP Designer provides this functionality through graphic user interface, thus eliminating the need for error prone ad hoc scripting and making it available to scientists who are comfortable with modeling, but not coding in programming languages. In case studies 2 and 3, we show how this functionality is applied in modeling of immune system and allosteric binding.

The enhanced modular biological process map notation contains other advanced features, which will not be covered in the main text of this tutorial but are presented in detail in different sections of extensive Data S1. Modularity is further enhanced by import symbols (Section 1.8.2), which represent and connect models stored in separate workspace files. Index ranges can be defined run-time by parameters (Sections 1.6 and 3.3). We also provide graphical notation for doses (Sections 1.1.2 and 1.2), algebraic constraints (Sections 1.1.2 and 3.5), and functions (Sections 1.3 and 1.4).

CASE STUDY 1: MECHANISTIC MODEL OF T-CELL PROLIFERATION ASSAY

Undesired immune response to therapeutic protein is a major challenge in drug development.[12] The immune system may generate antidrug antibodies (ADAs) binding engineered, non-self fragments of therapeutic proteins, thus affecting the concentration of free drug and efficacy. This response is patient-specific, with ADA positive and negative patient groups observed in many trials. A major, although not the only, source of this variability is variability of human leukocyte antigen (HLA) genes encoding major histocompatibility complex II (MHC II) receptors. Different affinity of MHC II receptors encoded by HLA alleles affects the presentation of T-cell epitopes, therapeutic protein fragments resulting from endosomal digestion. Antigen presentation is required to recruit the helper T-cell response, which is in turn required for initiation of antibody production by B-cells. Several bioinformatics approaches are used to predict T-cell epitopes and their affinity to MHC II receptors. Currently, these methods are used for calculation of static ADA risk indicator for a compound. In 2014, Chen, Hickling, and Vicini published QSP, a model (CHV model)[13,14] for simulation of ADA response dynamics for a patient with specific HLA genotype. This model uses bioinformatics prediction of T-cell epitopes and MHC II as input but allows prediction of ADA synthesis as a function of dosing regime and crucially, impact of ADA on compound PKs. Expansion of this model by an industry consortium has led to development of the IG Simulator.[15,16]

T-cell proliferation assays are used in biologics drug development as a preclinical screen for the propensity of the compound to cause unwanted ADA response. Peripheral blood mononuclear cells obtained from blood donors are treated with the compound in vitro. T-cell proliferation is assessed and compared with a baseline. Frequently, blood samples are genotyped for HLA genes. TCPro is a mechanistic model of T-cell proliferation assay intended by the authors to be used in lieu of actual, expensive, experimental work.[17] TCPro is based on the MHC II pathway and T-cell proliferation processes in the CHV model. The IG Simulator team also created a T-cell proliferation assay model. The assay model is used to infer parameters of T-cell response from T-cell proliferation assay data and subsequently these parameters are used in the IG Simulator model of clinical ADA response.

Here, we use the T-cell proliferation assay model to demonstrate the application of modules and indices, two major features of the enhanced graphical notation in QSP Designer. Figure 3 shows the Modular Biological Process Map of the T-cell proliferation assay model

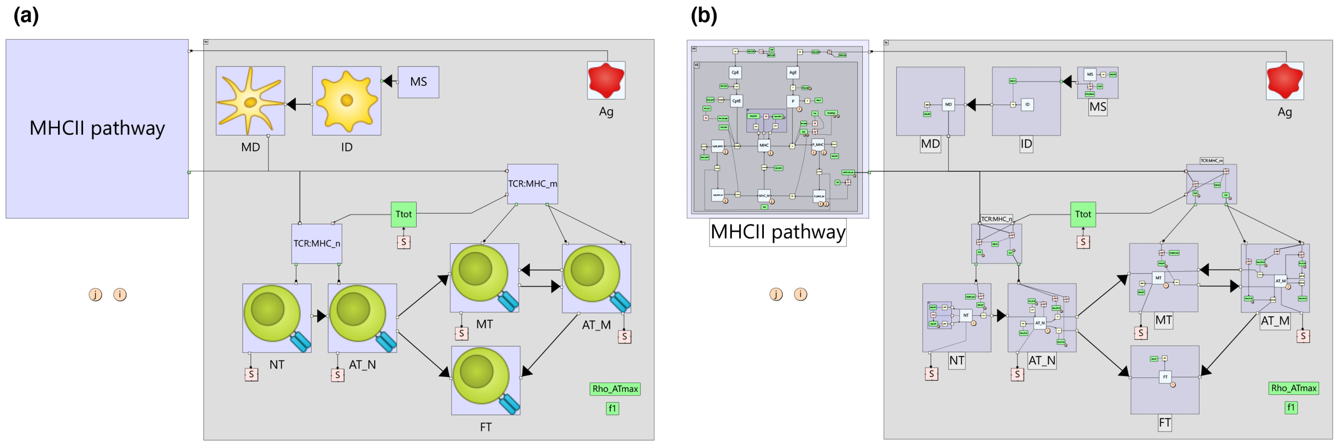
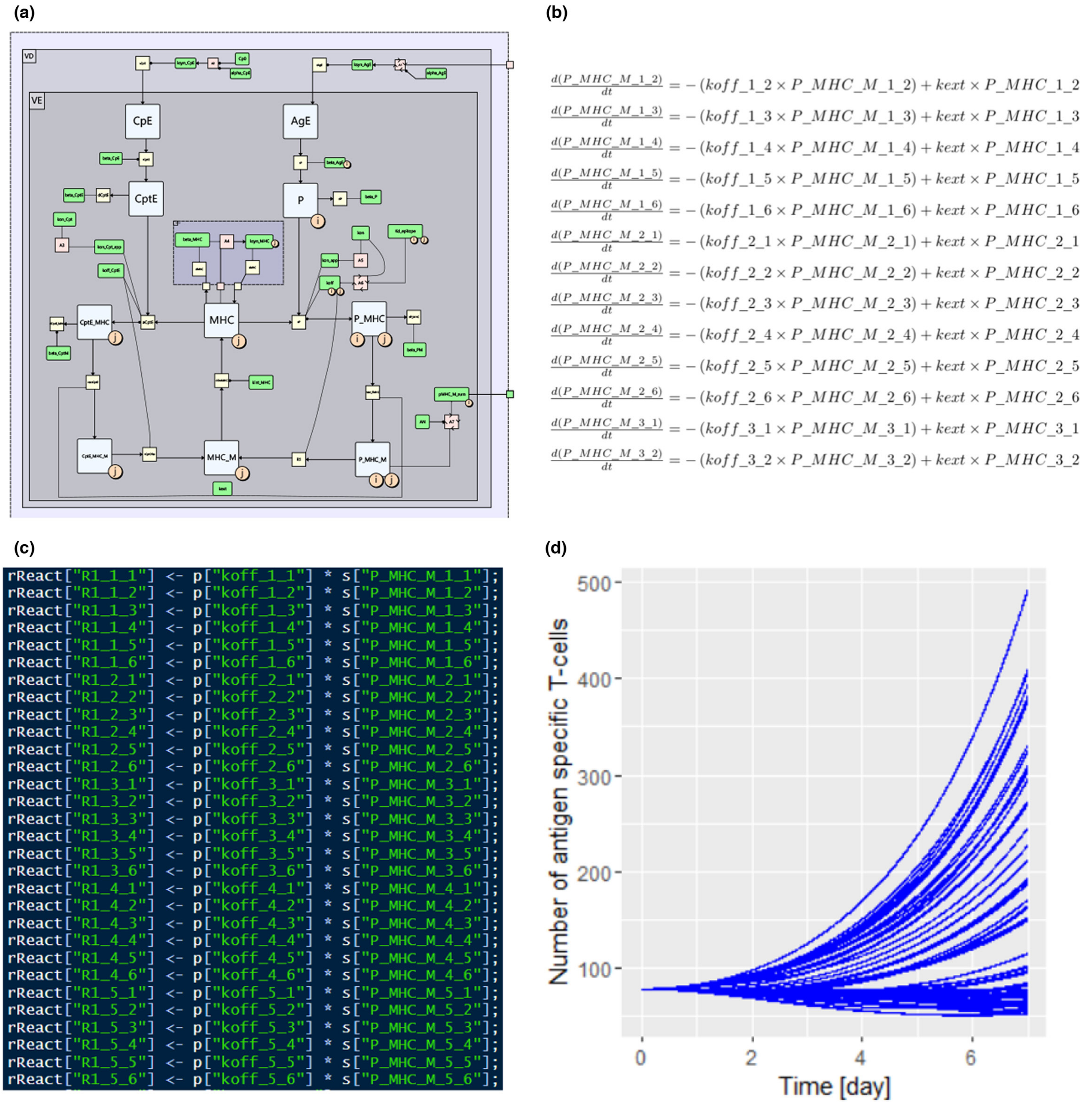


FIGURE 3 Mechanistic model of T-cell proliferation assay. Modular Biological Process Map diagrams created by SVG export from QSP Designer. (a) The map with collapsed modules focusing on biological scope. Compound (Ag) is dosed into the in vitro reaction compartment, taken up by dendritic cells, and subject to digestion and presentation through the MHC II pathway. Maturation signal (MS) induces differentiation of immature dendritic cells (ID) to mature dendritic cells (MD). Naïve helper T-cells (NT) are activated (AT_N) by MD which involves formation of TCR:MHC complex. Activated T-cells may differentiate to memory (MT) or functional T-cell (FT). Memory T-cells may be re-activated (AT_M), which requires MD and TCR:MHCII complex. Total number of T-cells (Ttot) is readout of the assay and also affects activation through cell density feedback. (b) The map with expanded modules showing all variables, reactions, parameters, compartments and assignments (466 visual entities). Indices i and j are used to automatically generate species and reactions for five T-cell epitopes and six MHCII receptors, creating the model with 120 ODEs and 177 parameters. QSP, Quantitative Systems Pharmacology; SVG, Scalable Vector Graphics.

created in QSP Designer and exported to Scalable Vector Graphics (SVG) image file. Figure 3a shows the map with all modules collapsed, which illustrates the major biological processes: dendritic cell activation, uptake of antigen into MHC II pathway, activation of T-cells involving interaction of MHC II/T-cell epitope and T-cell receptor (TCR), proliferation of T-cells, and differentiation of naïve T-cells into memory and functional T-cells. QSP Designer allows embedding of arbitrary images into diagram symbols. Here, T-cells and dendritic cells are distinguished by images embedded into collapsed module symbols. This diagram can be used to discuss the biological scope of the model with an interdisciplinary team. The diagram in Figure 3b shows the map with all modules expanded. Now the user can appreciate the full model structure. The MHC II pathway is a detailed mechanistic model of endosome digestion and MHC II binding including binding of competing peptides. Each of the modules representing cells contains detailed representation of state variables, reactions, parameters, and algebraic assignments defining dynamics of cellular differentiation and proliferation processes. This view can be used to communicate details of the model to modeling team. The user of QSP Designer software can navigate the map fully interactively, expand or collapse individual volumes, and zoom on any part of the map. Parameter values, variable initial states, rate, and assignment equations can be then accessed in property inspector invoked upon selection of the diagram symbol.

Binding of five T-cell epitopes to MHC receptors encoded by six HLA alleles (2 × HLA-DR, 2 × HLA-DQ, 2 × HLA-DP) creates 30 T-cell epitope/MHC II receptors complexes, all following association/dissociation, degradation, and membrane circulation processes. Therefore, there are 30 instances of the ODEs describing these processes. Moreover, the model contains T-cell clone specific to each of five epitopes, which creates five instances of T-cell differentiation, proliferation, and death processes. Graphical model building of these reactions is not practical, both due to the repetitive effort of graphical editing and the clarity of the resulting diagram. Even if such a diagram were created, changing it for a compound specific number of T-cell epitopes would not be practical. In the CHV model,[13,14] the MATLAB code for ODEs was automatically generated, for a specific number of epitopes, by a bespoke MATLAB script. This way of working has a number of disadvantages. First, a model diagram consistent with the equations can no longer be created. Even if automatic layout techniques were used, the resulting diagram would not be as good an illustration of the underlying processes as a map created manually by an expert modeler. Furthermore, scripting as a way of model building requires advanced coding skills in a specific language and thus limits some expert scientists from participating in model building. In addition, ad hoc coding is error prone and the resulting code difficult to communicate even between different members of the same team. Figure 4 shows how these challenges are addressed by



$$\frac{d(P\_MHC\_M\_1\_2)}{dt} = -(koff\_1\_2 \times P\_MHC\_M\_1\_2) + kext \times P\_MHC\_1\_2$$
$$\frac{d(P\_MHC\_M\_1\_3)}{dt} = -(koff\_1\_3 \times P\_MHC\_M\_1\_3) + kext \times P\_MHC\_1\_3$$
$$\frac{d(P\_MHC\_M\_1\_4)}{dt} = -(koff\_1\_4 \times P\_MHC\_M\_1\_4) + kext \times P\_MHC\_1\_4$$
$$\frac{d(P\_MHC\_M\_1\_5)}{dt} = -(koff\_1\_5 \times P\_MHC\_M\_1\_5) + kext \times P\_MHC\_1\_5$$
$$\frac{d(P\_MHC\_M\_1\_6)}{dt} = -(koff\_1\_6 \times P\_MHC\_M\_1\_6) + kext \times P\_MHC\_1\_6$$
$$\frac{d(P\_MHC\_M\_2\_1)}{dt} = -(koff\_2\_1 \times P\_MHC\_M\_2\_1) + kext \times P\_MHC\_2\_1$$
$$\frac{d(P\_MHC\_M\_2\_2)}{dt} = -(koff\_2\_2 \times P\_MHC\_M\_2\_2) + kext \times P\_MHC\_2\_2$$
$$\frac{d(P\_MHC\_M\_2\_3)}{dt} = -(koff\_2\_3 \times P\_MHC\_M\_2\_3) + kext \times P\_MHC\_2\_3$$
$$\frac{d(P\_MHC\_M\_2\_4)}{dt} = -(koff\_2\_4 \times P\_MHC\_M\_2\_4) + kext \times P\_MHC\_2\_4$$
$$\frac{d(P\_MHC\_M\_2\_5)}{dt} = -(koff\_2\_5 \times P\_MHC\_M\_2\_5) + kext \times P\_MHC\_2\_5$$
$$\frac{d(P\_MHC\_M\_2\_6)}{dt} = -(koff\_2\_6 \times P\_MHC\_M\_2\_6) + kext \times P\_MHC\_2\_6$$
$$\frac{d(P\_MHC\_M\_3\_1)}{dt} = -(koff\_3\_1 \times P\_MHC\_M\_3\_1) + kext \times P\_MHC\_3\_1$$
$$\frac{d(P\_MHC\_M\_3\_2)}{dt} = -(koff\_3\_2 \times P\_MHC\_M\_3\_2) + kext \times P\_MHC\_3\_2$$

```
rReact["R1_1_1"] <- p["koff_1_1"] * s["P_MHC_M_1_1"];
rReact["R1_1_2"] <- p["koff_1_2"] * s["P_MHC_M_1_2"];
rReact["R1_1_3"] <- p["koff_1_3"] * s["P_MHC_M_1_3"];
rReact["R1_1_4"] <- p["koff_1_4"] * s["P_MHC_M_1_4"];
rReact["R1_1_5"] <- p["koff_1_5"] * s["P_MHC_M_1_5"];
rReact["R1_1_6"] <- p["koff_1_6"] * s["P_MHC_M_1_6"];
rReact["R1_2_1"] <- p["koff_2_1"] * s["P_MHC_M_2_1"];
rReact["R1_2_2"] <- p["koff_2_2"] * s["P_MHC_M_2_2"];
rReact["R1_2_3"] <- p["koff_2_3"] * s["P_MHC_M_2_3"];
rReact["R1_2_4"] <- p["koff_2_4"] * s["P_MHC_M_2_4"];
rReact["R1_2_5"] <- p["koff_2_5"] * s["P_MHC_M_2_5"];
rReact["R1_2_6"] <- p["koff_2_6"] * s["P_MHC_M_2_6"];
rReact["R1_3_1"] <- p["koff_3_1"] * s["P_MHC_M_3_1"];
rReact["R1_3_2"] <- p["koff_3_2"] * s["P_MHC_M_3_2"];
rReact["R1_3_3"] <- p["koff_3_3"] * s["P_MHC_M_3_3"];
rReact["R1_3_4"] <- p["koff_3_4"] * s["P_MHC_M_3_4"];
rReact["R1_3_5"] <- p["koff_3_5"] * s["P_MHC_M_3_5"];
rReact["R1_3_6"] <- p["koff_3_6"] * s["P_MHC_M_3_6"];
rReact["R1_4_1"] <- p["koff_4_1"] * s["P_MHC_M_4_1"];
rReact["R1_4_2"] <- p["koff_4_2"] * s["P_MHC_M_4_2"];
rReact["R1_4_3"] <- p["koff_4_3"] * s["P_MHC_M_4_3"];
rReact["R1_4_4"] <- p["koff_4_4"] * s["P_MHC_M_4_4"];
rReact["R1_4_5"] <- p["koff_4_5"] * s["P_MHC_M_4_5"];
rReact["R1_4_6"] <- p["koff_4_6"] * s["P_MHC_M_4_6"];
rReact["R1_5_1"] <- p["koff_5_1"] * s["P_MHC_M_5_1"];
rReact["R1_5_2"] <- p["koff_5_2"] * s["P_MHC_M_5_2"];
rReact["R1_5_3"] <- p["koff_5_3"] * s["P_MHC_M_5_3"];
rReact["R1_5_4"] <- p["koff_5_4"] * s["P_MHC_M_5_4"];
rReact["R1_5_5"] <- p["koff_5_5"] * s["P_MHC_M_5_5"];
rReact["R1_5_6"] <- p["koff_5_6"] * s["P_MHC_M_5_6"];
```

FIGURE 4 MHC pathway module of T-cell proliferation assay model. (a) Biological process map. Antigen in endosome (AgE) is digested into T-cell epitope peptides P, which bind MHC receptors. Index i represents five epitopes, index j six loci of MHC II receptor genes. Competitive binding of other peptides in endosome (CptE) is also included. (b) Automatically generated ODEs for a complex of peptide and MHC receptor in membrane (P_MHC). QSP Designer LaTex rendering of 13 out of 30 ODEs is shown. Indexes are used to automatically generate specie and parameter names and equations. (c) Part of exported R code defining all 30 dissociation reactions of P_MHC species. Reactions define first term of ODE shown in b. (d) Simulation of full T-cell proliferation assay model for 100 blood samples from donors with different HLA genotypes. The 30 binding constants for five epitopes and six MHC receptors produced by six HLA alleles are predicted by bioinformatics and input into Kd_epitope parameter indexed with i and j indexes. ODE, Ordinary Differential Equation; QSP, Quantitative Systems Pharmacology.

QSP Designer. The detailed diagram of the MHC II pathway module is shown in Figure 4a. Indices i and j are used to create one- or two-dimensional arrays of variables and parameters. Index i assumes integer values of 1 to 5 and denotes T-cell epitopes, whereas index j is used to create an array of six MHC receptors. Two dimensional arrays of

30 complexes are created by both indices. Indices are an intuitive way of creating arrays, very similar to diagram drawing. Upon compilation of the model, all ODEs are automatically generated, without any coding required from the user. Figure 4b shows 13 out of 30 ODEs created by QSP Designer for the state variables describing the number of membrane-bound complexes of MHC II receptors and peptides. The automatically generated R codes for all 30 ODEs describing this molecular species are shown in Figure 4c. The same index i is also used to create arrays of variables and parameters describing proliferation, differentiation, and death of T-cell clones specific to T-cell epitopes. Figure 4d shows the application of the model to population simulation of subjects with different HLA genotypes. QSP Designer reads files with patient specific parameters generated based on bioinformatics predictions of T-cell epitope binding to MHC II receptors and allele frequencies in North American populations.

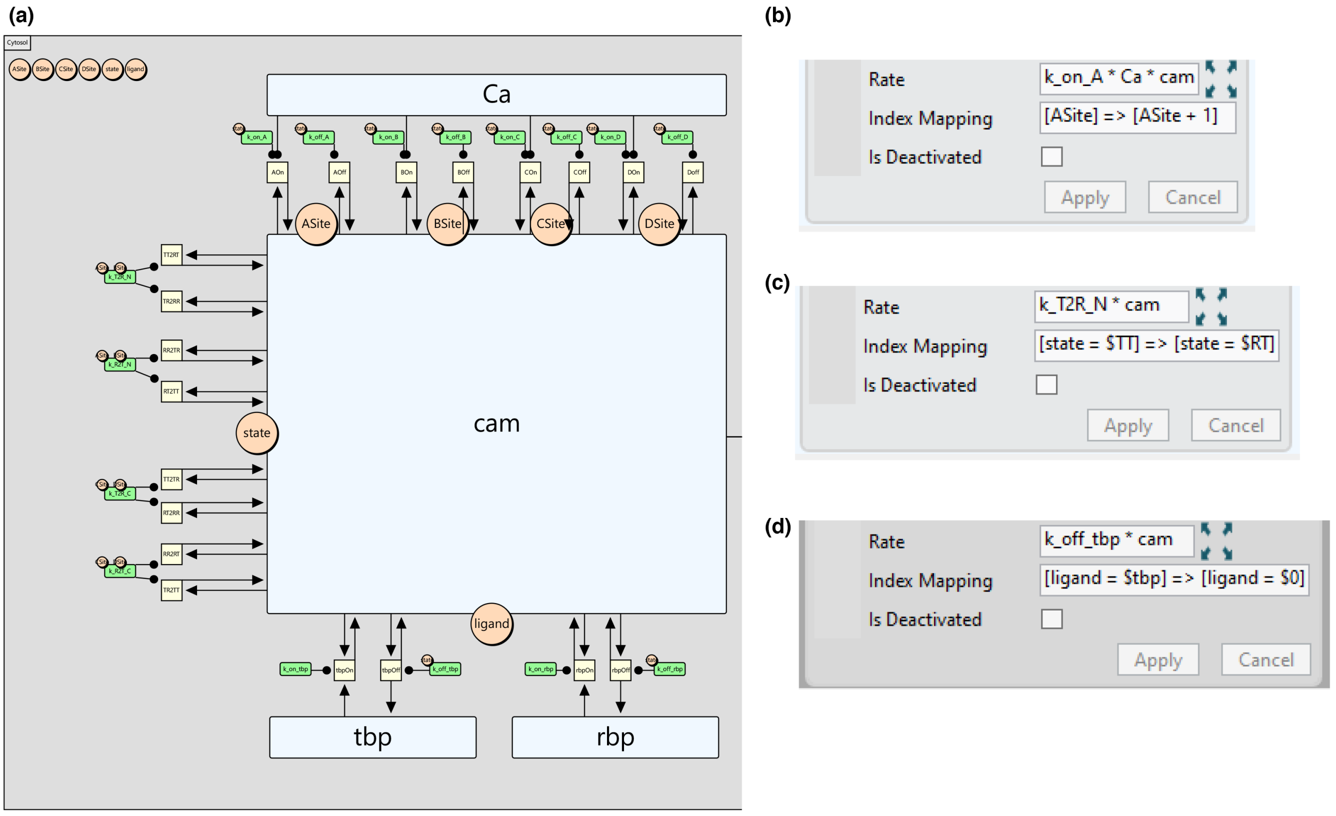
In summary, this case study demonstrates the utility of modules and arrays, two major features of the advanced graphical notation in QSP Designer. Modules are used to present the model at different levels of detail to different audiences. Arrays provide graphical notation that allows automatic model generation for cases when the same model structure is repeated multiple times. The model can be constructed with graphical editor without the need for creating error-prone, ad hoc scripts for generation of multiple ODEs.

CASE STUDY 2: MODELING ALLOSTERIC LIGAND BINDING TO CALMODULIN

Although the model presented in case study 1 illustrates the application of species and parameter arrays in handling models with repeated instances of the same submodel, it does not illustrate the full capability of QSP Designer in creating models with a combinatorial explosion of ODEs. In this case study, we present the model of allosteric binding of two protein ligands to calmodulin molecule. Calmodulin is a calcium-binding protein involved in calcium-regulated cellular processes, such as synaptic plasticity, muscle contraction, cell cycle, and circadian rhythms. It is composed of two globular domains joined by a linker region. Each domain can take alternative conformations, affected by the binding of calcium and target proteins. Here, we reproduce the model of Lai et al.[18] containing four calcium binding sites, one site in each lobe of the calmodulin dimer, two allosteric states, and binding of two protein ligands. Binding and conformational transitions are represented in detailed Monod-Wyman-Changeux framework, leading to a combinatorial explosion of reactions. Due to its ubiquitous presence in cellular signaling, calmodulin is likely to be a part of the QSP Models, but the combinatorial explosion of reactions would necessitate model generation by scripting and prevent concise graphical representation. Here, we show how the advanced graphical notation in the QSP Designer can be used to create concise graphical representation of the model of Lai et al.,[18] which can be used as a module in larger QSP Models.

Figure 5a shows the Biological Process Map of the model. The combinatorial explosion of state variables representing different forms of calmodulin is modeled by one species node with six indices. The indices represent four calcium binding sites (ASite, BSite, CSite, and DSite), one ligand binding site (ligand) and the conformational state. Calcium binding site indices have two values each indicating whether calcium is bound or not at the specific site. The unbound state is indicated by underscore symbol "_" and the bound state is indicated by the letter code of the site (e.g., "A" for site A). The ligand binding site index assumes one of three values: "0," "rbp," or "tbp," representing unbound, RBP-bound, or TBP-bound calmodulin/ligand complexes. Finally, conformational state index assumes values "RR," "RT," "TR," and "TT" corresponding to conformational state names. These six indices define all 192 state variables required to represent all forms of calmodulin molecule defined by conformational changes and calcium or ligand binding. For example, "cam_TR_tbp___B_C__" is a chemical species representing calmodulin in TR conformation, bound to tbp, with calcium bound at B and C, but not A site. During automatic model generation, the QSP Designer creates the ODE for this species and the time profile of the number of the molecules in this specific chemical state is produced during simulation. As an example, automatically generated ODE is shown in Figure 5e.

The use of indices in the calmodulin case study is more advanced than in the case of the T-cell proliferation assay module, where the indices were used solely to generate copies of the model structure, connected only by assignments summing variable values (e.g., sum of all MHC II:T-cell epitope bound complexes). Here, reactions connecting specific forms of calmodulin molecule must be defined. Figure 5b shows the property inspector window with the definition of the calcium association reaction at site A (Aon). The index mapping formula [ASite] = > [ASite + 1] defines reactions from each value of the ASite index to the next value. Because, in this case, the ASite index only has two values, this results in reactions with substrates that are states with index value "_" and products that are states with index value "A." Reactions from index value "A" to the next value are not generated because



(b)

Rate	k_on_A * Ca * cam
Index Mapping	[ASite] => [ASite + 1]
Is Deactivated	☐

(c)

Rate	k_T2R_N * cam
Index Mapping	[state = $TT] => [state = $RT]
Is Deactivated	☐

(d)

Rate	k_off_tbp * cam
Index Mapping	[ligand = $tbp] => [ligand = $0]
Is Deactivated	☐

(e)

$$
\begin{aligned}
\frac{d\,(cam\_RR\_0\_A\_B\_C\_\_ \times Cytosol)}{dt} = {} & -(k\_off\_A\_RR \times cam\_RR\_0\_A\_B\_C\_\_ \times Cytosol)\\
& + k\_on\_A\_RR \times Ca \times cam\_RR\_0\_\_\_B\_C\_\_ \times Cytosol\\
& - (k\_off\_B\_RR \times cam\_RR\_0\_A\_B\_C\_\_ \times Cytosol)\\
& + k\_on\_B\_RR \times Ca \times cam\_RR\_0\_A\_\_\_C\_\_ \times Cytosol\\
& - (k\_off\_C\_RR \times cam\_RR\_0\_A\_B\_C\_\_ \times Cytosol)\\
& + k\_on\_C\_RR \times Ca \times cam\_RR\_0\_A\_B\_\_\_\_ \times Cytosol\\
& + k\_off\_D\_RR \times cam\_RR\_0\_A\_B\_C\_D \times Cytosol\\
& - (k\_on\_D\_RR \times Ca \times cam\_RR\_0\_A\_B\_C\_\_ \times Cytosol)\\
& + k\_off\_rbp\_RR \times cam\_RR\_rbp\_A\_B\_C\_\_ \times Cytosol\\
& - (k\_on\_rbp \times rbp \times cam\_RR\_0\_A\_B\_C\_\_ \times Cytosol)\\
& - (k\_R2T\_C\_C\_\_ \times cam\_RR\_0\_A\_B\_C\_\_ \times Cytosol)\\
& - (k\_R2T\_N\_A\_B \times cam\_RR\_0\_A\_B\_C\_\_ \times Cytosol)\\
& + k\_T2R\_C\_C\_\_ \times cam\_RT\_0\_A\_B\_C\_\_ \times Cytosol\\
& + k\_off\_tbp\_RR \times cam\_RR\_tbp\_A\_B\_C\_\_ \times Cytosol\\
& - (k\_on\_tbp \times tbp \times cam\_RR\_0\_A\_B\_C\_\_ \times Cytosol)\\
& + k\_T2R\_N\_A\_B \times cam\_TR\_0\_A\_B\_C\_\_ \times Cytosol
\end{aligned}
$$

FIGURE 5 Calmodulin. (a) Biological process map. A single species node (cam) is used to represent all the possible states of calmodulin via the attachment of six static indices. Four indices (ASite, BSite, CSite, and DSite) are to represent whether calcium (Ca) is bound at each of four binding sites, one (state) is to represent the overall conformational state and one (ligand) is to represent which ligand (or none) is bound. The values of the calcium sites indices are, for example, "_" and "A." Those of the conformational state index are "RR," "RT," "TR," and "TT." Those of the ligand binding index are "0," "rbp," and "tbp." Reaction nodes representing transitions are automatically expanded out via implicit indexing to generate ODE terms for all relevant pairs of states. (b) The rate and index mapping expressions for the reaction node for one of the calcium binding transitions. (c) The rate and index mapping expressions for the reaction node for one of the conformational state transitions. (d) The rate and index mapping expressions for the reaction node for one of the ligand binding transitions. (e) One of the ODEs automatically generated for the model. ODE, Ordinary Differential Equation.

there is no next value. This formula generates all 95 reactions (rate law terms) of calcium binding at site A to calmodulin species with unbound A site. In this case, the same reactions could also be obtained using an index mapping [ASite = $_] = > [ASite = $A], which uses specific index values rather than specifying an offset. The second order mass action rate law with binding rate constant k_on_A, free calcium and calmodulin (unbound site A) species is assigned to all 95 reactions. Figures 5c,d show two further examples: conformational transition from TT to RT form and TBP dissociation from the ligand binding site.

In summary, the use of indices allows manual graphical creation of the Biological Process Map with four state variable, 20 reaction, 28 parameter, nine assignment, one compartment, and one module symbols which automatically generates the mathematical code of the model with 195 ODEs, 1408 reactions, 125 parameters, and 19 assignments. Whereas complex transitions need to be defined, no coding skills are required and no ad hoc scripts need to be created for model code generation. The user is assisted by syntax and unit consistency checking, minimizing the likelihood of errors. Once the model is generated, it can be simulated through the QSP Designer GUI or exported to code in R, in R with numerical code in C, MATLAB, or Julia. The model workspace can also be copied as a module or imported in a separate file into mechanistic models of biological processes involving calmodulin. Finally, this case study not only demonstrates the model of a molecule common in many pharmacologically relevant biological processes, but also the power of the QSP Designer in modeling complex ligand binding and conformational transition systems frequently of interest in drug development.

CASE STUDY 3: QSP MODEL OF mRNA COVID-19 VACCINES

Following the outbreak of the coronavirus disease 2019 (COVID-19) pandemic, Certara re-purposed their IG Simulator for virtual trials of COVID-19 mRNA vaccines to support dose and dose interval selection in vaccine development projects.[19] As described in case study 1, the IG Simulator was originally created to simulate unwanted immune response to therapeutic protein. Because the basic biology of the humoral immune response is the same regardless of whether unwanted ADA response to therapeutic proteins or desired immunogenicity to a vaccine antigen are modeled, the QSP Model was quickly expanded by developing a vaccine administration module. This effort is described in more detail in our previous article, here, we will use the QSP Model of COVID-19 mRNA vaccines to demonstrate application of the QSP Designer to large scale, QSP Platform models.

The COVID-19 vaccine model is composed of 426 ODEs, 850 rate law terms (reactions), 526 parameters, 15 compartments, and 398 algebraic assignments. The relationships among these mathematical model objects are represented by 2510 visual objects (graph nodes and edges) within the QSP Designer workspace. A model of this size and complexity would be extremely difficult, if not impossible, to represent as one readable diagram which makes hierarchical representation with modules essential. Figure 6 shows the top-level view of the biological process map of the COVID-19 vaccine model, as presented to the QSP Designer user. The modules provide a high-level overview of biological processes included in the model. The model is composed of three major modules: (i) minimal PBPK model for protein antigen, (ii) the model of antibody response, and (iii) the model of lipid nanoparticle (LNP) mRNA PKs and protein expression. Because we use a well-known biologics PBPK model, this part of the model does not need to be presented in detail and the module is collapsed in the top-level view. The LNP mRNA PK model is new and relatively small, and is thus presented in detail. The immune system model is complex, but its biological scope is important to communicate. The top-level view presents compartments and assignment of cellular populations to these compartments. General cell types are marked by images embedded into module symbols. Modules representing biological processes of affinity maturation, antibody distribution, B-cell receptor function, and MHC II/TCR synapse are labeled by text.

All modules in the top-level view are set to “wireless”: the edges connected to the modules are visible only when the user selects the module. When the user browses the map they can examine connections of the module of interest, expand the module, zoom, and examine all details. Figure 6 shows selection of the module representing naïve T-cells in the plasma compartment. The edges representing exit from the lymph node and circulation between vascular and peripheral blood are highlighted upon selection. The user also expands the module to subsequently zoom and examine rate laws describing naïve T-cell behavior. Interactive visualization of edges upon module selection is frequently the only way to represent connections in very large models where the number of connected elements makes construction of clear diagram impossible, due to the unavoidable overlap of graph edges.

Although this is not apparent in the top-level view, the model shown in Figure 6 makes extensive use of indices. The model contains the MHC II pathway module

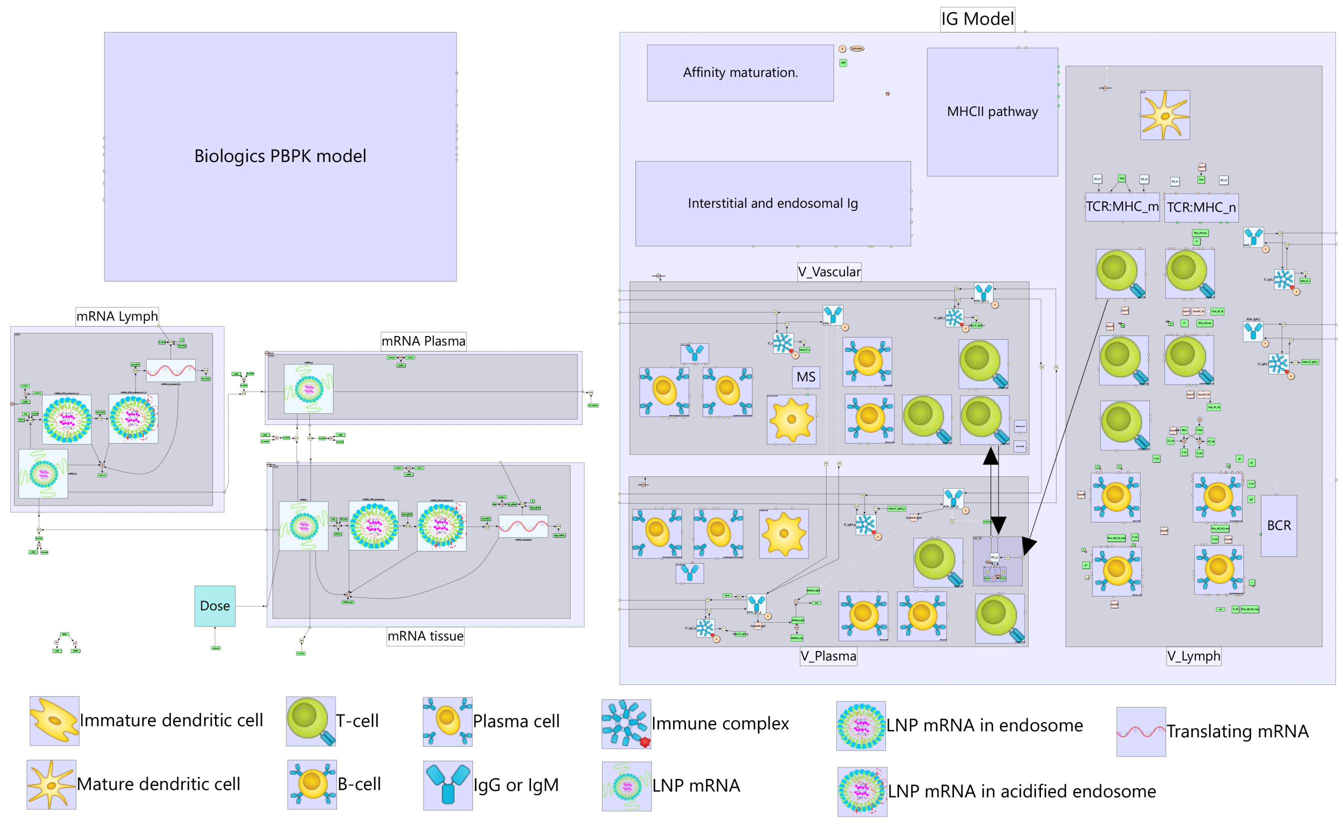


FIGURE 6 QSP model of mRNA vaccines. The map provides overview of biological processes included into the model. The model of 426 ODEs is represented by 2510 visual objects visualizing relationships between state variables, compartments, parameters, and assignments. The diagram of this size cannot be shown with all detail. Therefore, the modules are created in "wireless" mode, where edges connected to module interfaces are shown only when the module is selected. Figure shows expanded and selected module representing naïve T-cells. The user can examine equations governing naïve T-cell dynamics and see connections representing egress from lymph node and circulation between peripheral and vascular blood. Figure legends were also created within the QSP Designer, using empty modules. Thus, the figure shows top level view in user interface, which can be zoomed and expanded to access every detail of the model. The same file is also used for simulation code generation guaranteeing consistency of visual and mathematical representation. PBPK, physiologically-based pharmacokinetic; QSP, Quantitative Systems Pharmacology.

presented in case study 1. The T-cell variables are also indexed, as described in case study 2, to represent the T-cell specific clones. Moreover, additional index is introduced to represent polyclonal antibody response. The B-cells and all antibody species are indexed to represent five classes of antibodies spanning biologically plausible range of antibody affinity.

Figure 7 shows virtual trial results obtained with the internal engine of the QSP Designer. The output was exported to CSV and plotted with R. Population variability was generated by providing individual HLA genotype and PBPK parameters generated by the Simcyp Simulator covariate system. The QSP Designer GUI provides input for a Virtual Population (VPop) file: a CSV file where each row specifies model input vector for an individual subject. This allows easy coupling with other tools that are frequently needed to generate virtual populations for a specific project (e.g., bioinformatics tools for T-cell epitope prediction).

In summary, this case study demonstrates that the QSP Designer is capable of editing, communicating, maintaining, and simulating large scale QSP platform models.

VERIFICATION OF EXPORTED CODE

Whereas the QSP Designer includes simulation and parameter estimation engines it also allows full model code generation in MATLAB, R, C, and Julia to support multiple modeling communities. Exported code can be verified by comparing time profiles calculated by the QSP Designer and exported models. The CSV output of the QSP Designer facilitates comparison of simulation outputs and we include example R scripts in case study distributions (VERIFY.R). Section 3.11 in Data S1 and Appendix B of Data S1 show application of these scripts in case studies 1 and 2. We demonstrate that

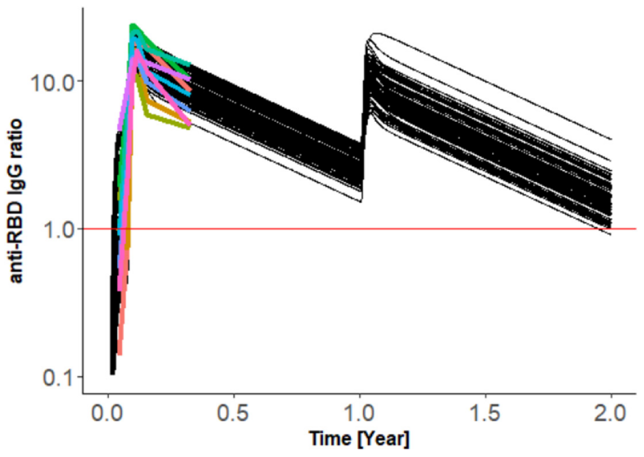


FIGURE 7 Example application of mRNA vaccine model. Plot shows ratio of anti-receptor binding domain IgG to the geometric mean of convalescent serum concentrations, plotted by red horizontal line. The model was calibrated with mRNA-1273 data and used to predict antibody response for third dose administered after 1 year. The black lines show simulation results for 85 virtual patients. Colored lines show clinical data available for first 120 days. A 100 ug dose was given at days 0, 28, and 365.

for two complex QSP models and different parameter inputs simulation of MATLAB, R, and Julia code exports results in time profiles overlapping with simulation in the QSP Designer (root mean square deviations in a range of 1e-8–1e-4). We conclude that simulations with model code exported from the QSP Designer are practically indistinguishable between different code exports and built-in numerical engine. Moreover, if the specific context of use requires verification of code export for specific model, we show that it can be easily implemented and automated for repeated comparison of simulation profiles. We also note that automatic code export offers unique opportunity of demonstrating that the model produces the same results with different ODE solvers, which can be very useful evidence in the technical verification step of model qualification process.

DISCUSSION

A major motivation for the development of the QSP Designer was to create a graphical model editor expanding on functionality available in existing tools as well as providing model code export to multiple mathematical modeling environments. Graphical model building has been widely adopted in engineering and Systems Biology and has already been recognized as an important tool in QSP. SimBiology and MoBi, two popular tools in the field, provide graphical model editors, using bipartite graph notation, where state variables are represented as chemical species connected through reactions representing rate laws. The QSP Designer further expands this capability by introduction of modules and arrays.

We demonstrate, with three case studies, how modules facilitate presentation of the model at different levels of granularity for different expert communities involved in drug development project. The high-level representation can be used to visualize biological interactions, without showing mathematical model details. This is useful when discussing model with biologists and seeking their approval of scope, granularity, and congruence of assumed mechanisms with knowledge of underlying biology. Modules allow representation of interactions in an interaction network graph, rather than bipartite graph format more common in diagrams used in biology literature. The modules also allow “wireless” visualization mode, where interactions are shown only upon selection of the module. This way of presenting the model may be the best option for very large platform models where an informative diagram could not have been created otherwise due to overlapping edges.

When discussion of scope and qualitative assumptions is completed, modules can be expanded to present the model to modeling experts. Here, bipartite graph notation is used with clear distinction between state variables (species), parameters, and rate law terms (reactions). We decided to adopt notation motivated by Extended Petri Nets, where the modifier edge connecting a species to a reaction indicates that the value of state variable is used as a parameter and not changed in the ODE system. This is different from the notation used in SimBiology, where ODE terms increasing and decreasing state variables are added to represent this situation. We believe that notation used in the QSP Designer leads to simpler mathematical representation of the model. The modifier edge is also used to connect parameters to reactions and assignments thus enhancing visualization of parameter dependencies. Although many modelers prefer typing ODEs directly into the mathematical modeling environment, it is our experience that model building in bipartite, species/reaction, graph editor with formal unit dimensionality validation is faster and less error prone. For example, the rate law term is created once as a reaction and then connected to all state variables it modifies, rather than having to be copied multiple times across the code as positive or negative contribution in multiple ODEs. It is also our experience that visualization of parameter dependencies is frequently advantageous for model communication to expert modelers over browsing of model code.

The modules with clearly defined interfaces provide a powerful tool for modelers in the sense of introducing object-oriented concepts into graphical model building. Parts of the model can be created by different team members and encapsulated into modules. Interfaces facilitate re-use of the modules across different models. A team member using a submodel they did not build can focus on understanding inputs and outputs rather than all details. This way of working is further facilitated by allowing import of models from separate workspace files. Import functionality allows individual namespaces in models, which are composed, as well as definition of which variables and parameters can be accessed (equivalent of "public" and "private" in object-oriented programming).

All three case studies used models where mechanistic knowledge dictated combinatorial explosion of state variables and interactions. In the case studies 1 and 3, binding of MHC II receptors produced by six HLA alleles to multiple T-cell epitopes required creation of 6*j (where j is number of epitopes) copies of binding reaction network, all contributing to total number of MHCII:peptide complexes. Case study 2 presents Monod-Wyman-Changeux mechanism of allosteric conformational changes of calmodulin upon binding calcium at three binding sites and binding two ligands at one site which requires construction of 195 ODEs with 1408 rate law terms. It is our experience that these examples represent a frequent, rather than unusual situation (other examples: PBPK models for multiple tissues, compounds and antibody-drug conjugates, and polymerization reactions). The number of reactions in these models makes manual generation of ODEs impractical or impossible and modelers usually apply ad hoc scripting to automatically generate model code. This practice is not only error prone, but also precludes visualization with informative, manually edited diagram. Motivated by these and other examples, we introduced model element arrays into our graphical modeling language. The user can attach an index with a list of values to a species or parameter and the software automatically creates ODEs. Case study 2 shows the additional syntax of reaction definitions that allows creation of arbitrary models using species and parameter arrays to handle combinatorial explosion. The modeler using the QSP Designer can create large numbers of ODEs automatically, without the need for creating an ad hoc script. They can also create an informative model layout, where intuitive index notation helps to visualize multiple occurrences of the same submodel structure. Similar functionality is also available in Colored Petri Nets implemented in Snoopy software, but graphical representation of array indices is not provided. We believe that indices and arrays are a unique novel feature of the QSP Designer significantly facilitating model building and communication in a large number of cases.

Although the QSP Designer includes a computational engine allowing simulation and parameter estimation within its GUI, it can also be used to export model code to R, R with model equations in C, MATLAB, and Julia. We export full model code rather than calls to an API. This means that the code can be used independently of the QSP Designer. This is a different strategy from MoBi, which provides R and MATLAB APIs allowing running model executed in MoBi. Whereas the SimBiology model editor provides MATLAB code very closely integrated with MATLAB, the model cannot be used without the SimBiology toolbox. In addition, contrary to the QSP Designer, SimBiology does not support mathematical modeling environments other than MATLAB. We are not aware of graphical model editors for the QSP Models in Julia. The pumas[20] environment does not provide a model editor. Thus, the QSP Designer provides a unique solution, where the model is built once in a GUI with unit dimensionality verification and exported to multiple mathematical modeling environments. Availability of a full model code, rather than calls to an API allows user full examination of the numerical code as well as more flexibility in adapting this code for use in custom algorithms.

We are aware that use of exported code may necessitate its verification, thus introducing additional step in modeling workflow. We demonstrated above that exported code can be verified by automated comparison with time profiles generated by the stand-alone QSP Designer. It is our experience, that, in many applications, benefits of using exported code by far outweigh the cost of this relatively simple verification step. For example, the modeling team may be interacting with a client who requires the model to be run within their pipeline implemented in a specific modeling language. It is much easier for the modeling team to export the final model to the code, verify it, and distribute it to the client, than to change their modeling environment for the purpose of a specific project. Distribution of the model in open-source modeling language like Julia or R may be attractive for regulators. In this case, demonstration that exported code and the QSP Designer used for development produce the same results, within the qualified range of parameters, adds very little additional work compared to the effort dedicated to other qualification steps. In another scenario, the team including members with experience in different modeling environments may wish to adopt specific toolboxes or legacy code to solve parameter estimation challenges, which are otherwise difficult to solve. The benefit of solving difficult problem with a specific legacy code running an exported model by far outweighs the cost of having to verify that time profiles of exported code and original model are indistinguishable. In the application where the QSP Designer is used solely as a graphical editor of the model, which will be calibrated and simulated in R or Julia, demonstration of equivalence with stand-alone numerical engine, not used in the project, is not necessary. Finally, automatic code export offers unique capability of demonstrating that model produces the same results with different ODE solvers, which can be very useful evidence in technical verification step of model qualification process. We would also like to note that the QSP Designer contains a comprehensive set of features that might be expected in a graph editor. The user can define edit points on the edges and decide whether they are connected with straight or curved lines. The editor contains an undo feature, with a sophisticated queue interface allowing full flexibility with revoking unwanted changes. All textboxes where formulas are edited provide predictive text completion. There is automatic updating of user-defined expressions in response to changes to variable names, as well as simplified, partially automatic transfer of values in arrays on changes to index ranges. There are both quick and advanced search interfaces. Syntax, unit, and other error messages displayed during verification are linked to graph objects and, where relevant, provide error highlighting within associated expressions. The graph and spreadsheet interfaces are also fully linked. Workspace import functionality includes a range of graphical overviews of the graphs formed by imports in workspaces that are themselves imported, etc. Different types of import that allow for one global copy or multiple copies of the same workspace to be used are supported, as well as the ability to inject variables from the importing workspace into the imported workspace. Although many of these features are available in MoBi and SimBiology editors, the QSP Designer provides a more comprehensive, integrated toolbox.

In conclusion, we present the QSP Designer, a QSP model editor with enhanced graphical notation, which enables hierarchical presentation with modules and handling of combinatorial complexity with diagram node arrays. Whereas the software includes a simulation engine its major feature is full model code generation in MATLAB, R, C, and Julia to support multiple modeling communities. We believe that the QSP Designer could be a software of choice for creation of large scale QSP platform models and analysis of these models with bespoke workflows integrating the strengths of multiple mathematical modeling communities.

FUNDING INFORMATION

This work was conducted and funded by Certara UK Ltd.

CONFLICT OF INTEREST STATEMENT

All authors declared no competing interests for this work.

ORCID

Piet H. van der Graaf https://orcid.org/0000-0003-1314-3484

REFERENCES

1. Vicini P, van der Graaf PH. Systems pharmacology for drug discovery and development: paradigm shift or flash in the pan? *Clin Pharmacol Ther*. 2013;93:379-381.
2. Kitano H. Systems biology: a brief overview. *Science*. 2002;295:1662-1664.
3. Gadkar K, Kirouac DC, Mager DE, van der Graaf PH, Ramanujan S. A six-stage workflow for robust application of systems pharmacology. *CPT Pharmacometrics Syst Pharmacol*. 2016;5:235-249.
4. Grady Booch JR, Jacobson I. *Unified Modeling Language User Guide*. Addison-Wesley Professional; 2005.
5. Petri C. Kommunikation mit Automaten. PhD thesis, University of Bonn. 1962.
6. SimuLink. https://www.mathworks.com/products/simulink.html
7. Le Novere N, Hucka M, Mi H, et al. The systems biology graphical notation. *Nat Biotechnol*. 2009;27:735-741.
8. Funahashi A, Morohashi M, Matsuoka Y, Jouraku A, Kitano H. CellDesigner: a graphical biological network editor and workbench interfacing simulator. In: Choi S, ed. *Introduction to Systems Biology*. Humana Press; 2007:422-434.
9. Rohr C, Marwan W, Heiner M. Snoopy-a unifying Petri net framework to investigate biomolecular networks. *Bioinformatics*. 2010;26:974-975.
10. SimBiology. https://www.mathworks.com/products/simbiology.html
11. PK-SIM® and MOBI® for PBPK and Quantitative Systems Pharmacology.
12. Wang YM, Wang J, Hon YY, et al. Evaluating and reporting the immunogenicity impacts for biological products-a clinical pharmacology perspective. *AAPS J*. 2016;18:395-403.
13. Chen X, Hickling TP, Vicini P. A mechanistic, multiscale mathematical model of immunogenicity for therapeutic proteins: part 1-theoretical model. *CPT Pharmacometrics Syst Pharmacol*. 2014;3:e133.
14. Chen X, Hickling TP, Vicini P. A mechanistic, multiscale mathematical model of immunogenicity for therapeutic proteins: part 2-model applications. *CPT Pharmacometrics Syst Pharmacol*. 2014;3:e134.
15. Kierzek AM, Hickling TP, Figueroa I, et al. A quantitative systems pharmacology consortium approach to managing immunogenicity of therapeutic proteins. *CPT Pharmacometrics Syst Pharmacol*. 2019;8:773-776.
16. Franssen LC, Swat MJ, Kierzek AM, et al. Learn-confirm in model-informed drug development: assessing an immunogenicity quantitative systems pharmacology platform. *CPT Pharmacometrics Syst Pharmacol*. 2023;12:139-143.
17. Yogurtcu ON, Sauna ZE, McGill JR, Tegenge MA, Yang H. TCPro: an in silico risk assessment tool for biotherapeutic protein immunogenicity. *AAPS J*. 2019;21:96.
18. Lai M, Brun D, Edelstein SJ, Le Novere N. Modulation of calmodulin lobes by different targets: an allosteric model with hemiconcerted conformational transitions. *PLoS Comput Biol*. 2015;11:e1004063.

19. Giorgi M, Desikan R, van der Graaf PH, Kierzek AM. Application of quantitative systems pharmacology to guide the optimal dosing of COVID-19 vaccines. *CPT Pharmacometrics Syst Pharmacol*. 2021;10:1130-1133.
20. pumas Pharmaceutical Modeling and Simulation. https://juliahub.com/products/pumas/

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article: Matthews RJ, Hollinshead D, Morrison D, van der Graaf PH, Kierzek AM. QSP Designer: Quantitative systems pharmacology modeling with modular biological process map notation and multiple language code generation. *CPT Pharmacometrics Syst Pharmacol*. 2023;12:889-903. doi:10.1002/psp4.12972