glyph with class 'process' (lines 14–18, in orange). Each glyph also carries an 'id' attribute that can be referred from elsewhere in the document, thus storing the network topology (in this case merely the letter 'f' for the sake of brevity). Each glyph must define a 'bbox' or bounding box, which allows the glyph to be placed at the correct position. Its coordinates denote the smallest rectangle that completely encompasses the glyph. Consumption and production arcs connect to process nodes at a so-called 'port' just outside the glyph. 'Port' elements are part of the network topology, so they carry identifiers as well (lines 16 and 17). Another glyph in this example represents the active form of hexokinase (lines 24–31). It carries a label element, which should be positioned in the center of the parent glyph, unless otherwise defined. Hexokinase also contains a sub-glyph for a state variable (lines 27–30, in blue) to indicate that it is the allosterically active form of the enzyme. ATP (lines 19–23, in green) is a simple chemical, and uses a circle as its shape, as opposed to macromolecules that use a rounded rectangle shape. Small molecules often occur multiple times in a map, in which case they must carry a clone marker, a black bottom half. In SBGN-ML this is represented by the 'clone' element (line 21). Cellular compartments are represented by glyphs as well (lines 32–35, in yellow). Entities refer to their surrounding compartment using a 'compartmentRef' attribute.

Just like glyphs, arcs must define a 'class' attribute and an 'id' attribute. See for example the production arc (lines 84–87, in cyan). Each arc must have a source attribute, referring to the identifier of a glyph that the arc points from, as well as a target attribute, referring to the identifier of the glyph that the arc points to. Source and target may refer to identifiers of either glyphs or ports. Arcs must also define start and end coordinates. Arcs can optionally include waypoints for path routing as with the 'catalysis' arc (lines 88–92, in purple). It is not possible to deduce the start and end coordinates from the source and target glyphs, as there may be some white space between the end of the arc and the border of the glyph.

Each element can be freely annotated with notes encoded with valid XHTML elements (lines 3–5). Each SBGN-ML can also be extended with elements in proprietary namespaces to add additional features (not shown in this example).

3 THE LIBSBGN LIBRARY

A software library called LibSBGN complements the file format. It consists of two parallel implementations in Java and C++. The libraries share the same object model, so that algorithms operating on it can be easily translated to different programming languages.

The primary goal of LibSBGN is to simplify the work for developers of existing pathway tools. To reach this goal we followed three design principles. First, we avoided tool-specific implementation details. Implementation artifacts that are specific for one bioinformatics tool would impose difficulties for adoption by others. We sought input from several tool developers into the LibSBGN effort early on.

Second, we do not want to force the use of a single rendering implementation (meaning the software routine that translates from memory objects to screen or graphic format). Early in the development of LibSBGN, it became clear that for most pathway drawing tools, the rendering engine is an integral part that is not easily replaced by a common library. The typical usage scenario is therefore to let LibSBGN handle input and output, but to translate

```
// our sbgnml file goes in "f"
File f = new File ("../test-files/adh.sbgn");

// Now read from "f" and put the result in "sbgn"
Sbgn sbgn = SbgnUtil.readFromFile(f);

// map is a container for the glyphs and arcs
Map map = sbgn.getMap();

// we can get a list of glyphs (nodes) in this map with getGlyph()
for (Glyph g : map.getGlyph())
{
    // print the sbgn class of this glyph
    System.out.print (" Glyph with class " + g.getId());

    // if there is a label, print it as well
    if (g.getLabel() != null)
        System.out.println (", and label " + g.getLabel().getText());
    else
        System.out.println (", without label");
}

// we can get a list of arcs (edges) in this map with getArc()
for (Arc a : map.getArc())
{
    // print the class of this arc
    System.out.println (" Arc with class " + a.getClazz());
}
```

Fig. 2. Example of reading a file using the Java version of LibSBGN. Here an SBGN-ML file named 'adh.sbgn' (included in the LibSBGN source distribution) is read, and some basic information about each glyph in that file is printed to standard output. The complete program can be found as ReadExample.java in the LibSBGN source distribution

to the application's own object model, and display using the application's own rendering engine. Enforcing a common rendering library would hamper adoption of LibSBGN. We instead opted to build a render comparison pipeline to ensure consistency between various renderers (this pipeline is described in more detail in Section 3.2).

Third, we wish to provide optimal libraries for each development environment. For both the C++ and Java versions, code is automatically generated based on the XML Schema definition (XSD). The method of generating code from XSD has reduced the effort needed to keep the Java and C++ versions synchronized during development. The generated Java code plus helper classes form a pure Java library. The alternative possibility, to create a single C++ library and a Java wrapper around that, is not preferable because it complicates multi-platform installation and testing. Our experience with a related project, LibSBML (Bornstein *et al.*, 2008), is that the community has a need for a pure Java library in spite of existing Java bindings for C++, which has led to the development of the pure Java JSBML (Dräger *et al.*, 2011) as an alternative. Although both LibSBML and JSBML are successful projects, the maintenance of two similar projects in different languages is costly in terms of developer time. By generating native libraries for both environments automatically, we hope to avoid that extra cost.

3.1 Code sample

See Figure 2 for an example of usage of LibSBGN in practice. The Java library contains convenient helper functions for reading, writing and validation. In the case of this example the function readFromFile from the SbgnUtil class is used. The source package contains example programs for common operations, and the LibSBGN wiki includes a developer tutorial (see http://sourceforge.net/apps/mediawiki/libsbgn/index.php?title=

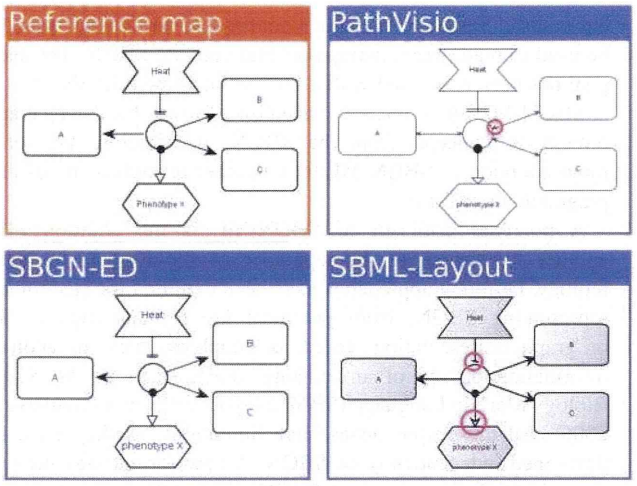


Fig. 3. Rendering comparison. A series of test-cases is rendered by all supported tools in an automated render comparison pipeline. The rendering results are compared with the reference map (top-left), in this case an ER map. A couple of significant differences have been highlighted with red circles. In the PathVisio case (top-right), arrowheads are drawn where none is expected. In the SBML Layout example (bottom-right), the wrong arrowheads are drawn for absolute inhibition and stimulation arcs. Note that these are historical images for illustration purposes, and the highlighted issues have already been fixed

Developer_tutorial) aimed at developers who want to include LibSBGN into an existing bioinformatics application.

3.2 Rendering comparison

We created dozens of test-cases for each of the three languages of SBGN, covering all aspects of the syntax. Each test-case consists of a reference diagram in PNG format and a corresponding SBGN-ML file. To test our software, all SBGN-ML files are automatically rendered by the participating programs, currently SBGN-ED (Czauderna *et al.,* 2010), PathVisio (van Iersel *et al.,* 2008) and SBML Layout (Deckard *et al.,* 2006). The resulting images are viewable side-by-side with the reference map. An example of this can be found in Figure 3.

This pipeline was of tremendous value during development. Typically, an observed difference between a given rendering and the reference diagram could lead to several possible outcomes. Most commonly, the difference indicated a mistake in the participating renderer, which had to be fixed by the author of that software. A second possibility is that the mistake is due to an ambiguity in the interpretation of SBGN-ML. This could lead to a correction in the specification or a clarification in the documentation, so that all involved are in agreement. In several instances, the source of ambiguity was derived not from SBGN-ML but from the SBGN specification. This way, LibSBGN has led to feedback on SBGN itself. A final possibility is that the difference was deemed insignificant. Certain differences in use of color, background shading and line thickness are not meaningful in terms of biological interpretation of the SBGN map. An exception here is differences in layout. As mentioned before, we consider layout valuable to preserve even though it is not semantically significant. This pipeline is now fully automated, and runs automatically, whenever new test-cases are added to the source repository. It can be viewed online at http://libsbgn.sourceforge.net/render_comparison/. We encourage developers of software to contact us to add their tool to the gallery.

3.3 Validation

For syntactic validation of SBGN-ML documents, we created an XML Schema definition (XSD). Unfortunately, XSD is not sufficient to validate the many semantic rules defined in the SBGN specification. To solve this we also developed higher level, semantic validation using the Schematron (http://www.schematron.com) language.

To give a few examples: in PD, a production arc should point from a process towards an entity pool node. It is not allowed to draw the arc in the other direction, or to connect two entity pools directly without an intermediate process (see Figure 4). In ER, outcome glyphs may be drawn on interaction arcs but not on influence arcs. If such a rule were violated, the meaning of the map would be ambiguous or contradictory.

LibSBGN provides functionality for users and developers to validate diagrams against these rules. This validation capability is built using Schematron language which has been previously used for Molecular Interaction Map diagram validation (Luna *et al.*, 2011). Schematron rules are assertion tests written using XPath syntax. Each rule possesses a role to denote the severity of failure, a human-readable message and diagnostic elements to identify the source of the error or warning. Rules in Schematron can be grouped in phases; this feature can be used to denote subsets of rules to be activated during validation. Schematron makes use of XML stylesheet transformations (XSLT) and the validation process occurs in two steps. The first step is the transformation of the rule sets written in the Schematron language to an XSLT stylesheet, and the second step is the transformation of an SBGN-ML file using the XSLT stylesheet from the first step. The product of this second transformation is a validation report that uses the Schematron Validation Report Language (SVRL). The usage of Schematron rule sets allows for validation to be flexibly incorporated into various environments and using any programming language with an XSLT processor. Command-line validation can be done using XSLT processors such as Saxon (http://saxon.sourceforge.net/) by performing the two transformation steps mentioned above. Alternatively, validation can also be incorporated into automated pipelines using the Ant task for Schematron (http://code.google.com/p/schematron/); an example of this is provided in the distributed files. Lastly, validation can be incorporated into projects by using provided utility Java classes found in the LibSBGN API. The PathVisio-Validator plugin (Chandan *et al.*, 2011) is an example of diagram validation using LibSBGN and Schematron.

There are three rule sets for SBGN-ML, one for each of the SBGN languages. These rule sets validate syntactic correctness of SBGN maps. An example validation is shown in Figure 4, where a stimulation arc is incorrectly drawn by pointing to an entity pool node, rather than a process node.

Unfortunately software can have bugs, and if the validation routine does not report any validity errors, this could indicate that either the diagram is indeed correct (true negative), or that there is a bug in the software encoding the rules (false negative). To ensure correctness of the validation rules themselves, we have created

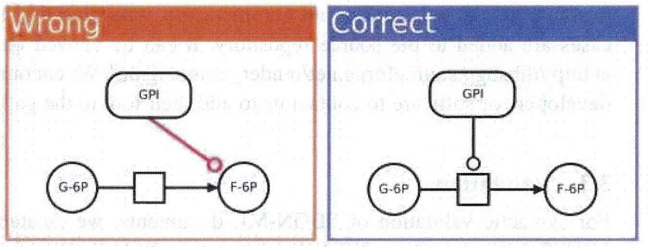


Fig. 4. Typical validator benchmark. This particular example tests the software for rule pd10110: in PD maps, catalysis arcs must point to a process node (not to an entity pool node). In the negative test-case on the left, the enzyme GPI appears to 'catalyze' a molecule rather than a reaction. This is a logical impossibility. The positive test-case on the right shows correctly how the enzyme GPI catalyzes the reaction from glucose-6P to fructose-6P. Taken together, these test-cases help to prevent bugs in the validation software

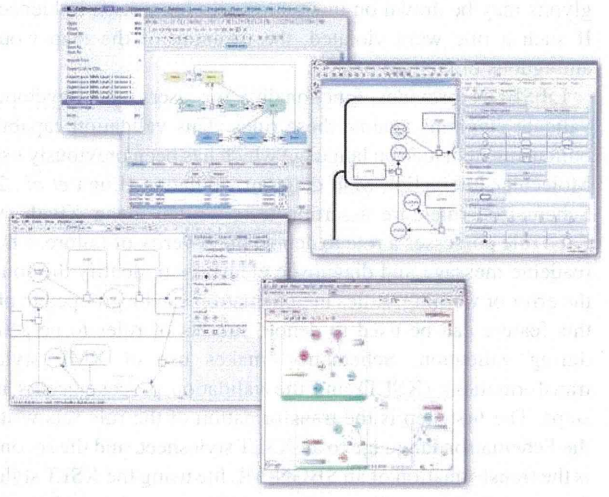


Fig. 5. Screenshots of a number of tools that use LibSBGN. Clockwise, from the top: CellDesigner, SBGN-ED, VISIBIOweb and PathVisio. These tools are able to use SBGN-ML for import, export or both. At the time of writing, for some of these tools a version with SBGN support has not been officially released, but is expected soon

benchmarks for each of them. For each rule there is a positive test-case, for which the rule should pass, and a negative one, for which the rule should fail, similar to the example given in Figure 4.

3.4 Supporting tools

As mentioned earlier, we seek support from a wide community of tool developers. The following tools are already using LibSBGN: PathVisio (van Iersel *et al.,* 2008), SBGN-ED (Czauderna *et al.*, 2010), SBML Layout (Deckard *et al.,* 2006) and VISIBIOweb (Dilek *et al.*, 2010). We are aware of two other tools with LibSBGN support in development: Arcadia (Villéger *et al.*, 2010) and CellDesigner (Funahashi *et al.*, 2008). Desktop applications using LibSBGN are shown in Figure 5.

4 DISCUSSION

We have set out to fulfill the dual goals of simplifying SBGN support as well as standardizing electronic exchange of SBGN. The first goal has been addressed with an open-source software library, which can be used to read, write, manipulate and validate SBGN. The second goal has been addressed with a file format named SBGN-ML.

SBGN-ML fills a pragmatic need for a format that can be mapped directly to concepts from the SBGN specification. We see the rapid adoption of SBGN-ML by a number of tools as proof of the pragmatic need for it.

A potential criticism of SBGN-ML is the addition of yet another file format to the repertoire of file formats in systems biology. Different approaches have been explored for electronically representing SBGN: from graphical file formats such as SVG, or graph representation stored as GraphML files, to additional information on top of an existing model, such as the Systems Biology Markup Language (SBML) layout extension (Gauges *et al.*, 2006). All these approaches have limitations, as they have been developed independently of SBGN. A new format was needed to support all characteristics of SBGN maps (graphics, relationships and semantics). The other formats could be extended to cover these concepts, but at the expense of brevity and clarity.

So we created a new format for the following reasons. First, SBGN-ML focuses on the domain of visualization of SBGN concepts. This sets it apart from existing exchange formats for pathways. BioPAX is a pathway exchange format that occupies the domain of knowledge management, and has close relations to the semantic web. SBML occupies the domain of computational modeling of systems biology. The latter two could be extended to accommodate SBGN concepts, but there is not a straight one-to-one mapping. For example, there is no good equivalent for the AND/OR gates which can be drawn in SBGN. Furthermore, omitted/uncertain processes can be drawn in SBGN but have no direct equivalent in BioPAX.

Second, SBGN-ML is easier to validate against the SBGN specification. As mentioned before, the complexity of SBGN makes software support for validation a must. Rules describing validation of SBGN-ML are simpler and more concise than they would be if they were encoded on top of an existing format.

Third, the rendering comparison pipeline has ensured that conversion of SBGN-ML to graphical formats is straightforward. On the other hand, conversion from a graphical format such as SVG to SBGN-ML requires inferring the meaning of lines, glyphs and symbols, which is bound to lead to loss of information.

Fourth, by making SBGN independent, it is not tied to either the SBML, BioPAX or any other research community. We observe that currently LibSBGN is being used by both BioPAX-oriented tools such as ChIBE and PaxTools as well as SBML-oriented tools such as CellDesigner or GraphML-oriented tools such as SBGN-ED.

SBGN-ML is officially endorsed by the SBGN scientific committee as a reference implementation and the best way to exchange diagrams between applications. It is orthogonal to specific formats used to represent pathways and models such as BioPAX (Demir *et al.*, 2010) and SBML (Hucka *et al.*, 2003), and thus follows the vision of the COMBINE initiative (http://co.mbine.org/about).

In the field of bioinformatics, it occurs all too often that the lack of a feature in an existing piece of software is used to justify the development of a complete new bioinformatics tool, which will in its turn lack features in another area. The end result is the current state of affairs: a balkanization of bioinformatics tools, or in other words, many fragmented tools that integrate poorly. One of the goals of LibSBGN is to improve existing software. LibSBGN could serve

as a model to counter the balkanization trend. We prefer to see the development of software libraries instead of incomplete tools. Libraries, especially if they are open source, can be shared, re-used and adopted by developers.

5 CONCLUSION

The SBGN-ML file format and LibSBGN library provide open-source software support for SBGN maps. They have been adopted by several tools already, and development is ongoing. It is expected that the availability of a community-supported API will significantly expedite SBGN's adoption. We use the word 'Milestone' for versioning purposes—the latest release is Milestone 2, which was released in December 2011.

LibSBGN is primarily focused on exchanging between SBGN software. Other functionalities, such as conversion to other formats, or generating suitable layout, are not currently supported. It is certainly likely that some or all of these functionalities will be added in the future as optional modules. SBGN-ML will likely see the addition of fine-grained graphics specification, support for linking between files, and improved usage of ontologies. Additionally, LibSBGN will see expansion to other programming languages beyond Java and C++, such as for example Javascript.

The SBGN-ML file format is represented as an XML schema (SBGN.XSD). Examples are available as test files (XML, PNG). The accompanying documentation reflects the content of the schema, and clarifies a number of additional rules and conventions (e.g., coordinate system). This set of resources constitutes the SBGN-ML specifications. The LibSBGN library (in C++ and Java) and the file format have been released on Sourceforge, under a dual license: the Lesser General Public Licence (LGPL) version 2.1 or later, and Apache version 2.0.

The development process is an active community effort, organized around: regular online meetings, discussions on the mailing list, and development tools on Sourceforge (bug tracker, SVN repository and documentation wiki). New developers are very welcome.

ACKNOWLEDGEMENTS

The authors thank their individual sources of funding. Authors are grateful for useful feedback from the Path2Models project.

Funding: This work was in part supported by the Biotechnology and Biological Sciences Research Council (BBSRC); the Netherlands Consortium for Systems Biology (NCSB), which is part of the Netherlands Genomics Initiative/Netherlands Organisation for Scientific Research; BioPreDyn which is a grant within the Seventh Framework Programme of the EU, the Intramural Research Program of the NIH, National Cancer Institute, Center for Cancer Research; and the German Ministry of Education and Research (BMBF).

Conflict of Interest: none declared.

REFERENCES

Bornstein,B.J. *et al.* (2008) LibSBML: an API library for SBML. *Bioinformatics*, **24**, 880–881.

Chandan,K. *et al.* (2011) PathVisio-Validator: A rule-based validation plugin for graphical pathway notations. *Bioinformatics*, **28**, 889–890.

Czauderna,T. *et al.* (2010) Editing, validating, and translating of SBGN maps. *Bioinformatics*, **26**, 2340–2341.

Deckard,A. *et al.* (2006) Supporting the SBML layout extension. *Bioinformatics*, **22**, 2966–2967.

Demir,E. *et al.* (2010) The BioPAX community standard for pathway data sharing. *Nat. Biotechnol.*, **28**, 935–942.

Dilek,A. *et al.* (2010) VISIBIOweb: visualization and layout services for BioPAX pathway models. *Nucleic Acids Res.*, **38**, W150–W154.

Dräger,A. *et al.* (2011) JSBML: a flexible Java library for working with SBML. *Bioinformatics*, **27**, 2167–2168.

Funahashi,A. *et al.* (2008) CellDesigner 3.5: a versatile modeling tool for biochemical networks. *Proc. IEEE*, **96**, 1254–1265.

Gauges,R. *et al.* (2006) A model diagram layout extension for SBML. *Bioinformatics*, **22**, 1879–1885.

Hucka,M. *et al.* (2003) The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, **9**, 524–531.

Kitano,H. *et al.* (2005) Using process diagrams for the graphical representation of biological networks. *Nat. Biotechnol.*, **23**, 961–966.

Kohn,K.W. *et al.* (2006) Molecular interaction maps of bioregulatory networks: a general rubric for systems biology. *Mol. Biol. Cell*, **17**, 1–13.

Le Novère,N. *et al.* (2009) The systems biology graphical notation. *Nat. Biotechnol.*, **27**, 753–741.

Luna,A. *et al.* (2011) A formal MIM specification and tools for the common exchange of MIM diagrams: an XML-Based format, an API, and a validation method. *BMC Bioinformatics*, **12**, 167.

van Iersel,M.P. *et al.* (2008) Presenting and exploring biological pathways with PathVisio. *BMC Bioinformatics*, **9**, 399.

Villéger,A.C. *et al.* (2010) Arcadia: a visualization tool for metabolic pathways. *Bioinformatics*, **26**, 1470–1471.

Molecular Systems Biology 8; Article number 578; doi:10.1038/msb.2012.12
Citation: *Molecular Systems Biology* **8: 578**

www.molecularsystemsbiology.com

molecular systems biology

A framework for mapping, visualisation and automatic model creation of signal-transduction networks

Carl-Fredrik Tiger[1,2,8], Falko Krause[2,8], Gunnar Cedersund[1,3,4], Robert Palmér[3], Edda Klipp[2], Stefan Hohmann[1], Hiroaki Kitano[3,5,6,7] and Marcus Krantz[1,2,5,*]

[1] Department of Cell and Molecular Biology, University of Gothenburg, Göteborg, Sweden, [2] Theoretical Biophysics, Humboldt-Universität zu Berlin, Berlin, Germany, [3] Department of Clinical and Experimental Medicine, Diabetes and Integrative Systems Biology, Linköping University, Linköping, Sweden, [4] Freiburg Institute of Advanced Sciences, School of Life Sciences, Freiburg, Germany, [5] The Systems Biology Institute, Tokyo, Japan, [6] Sony Computer Science Laboratories, Inc., Tokyo, Japan and [7] Okinawa Institute of Science and Technology, Okinawa, Japan
[8]These authors contributed equally to this work
* Corresponding author. Theoretical Biophysics, Humboldt-Universität zu Berlin, Invalidenstr. 42, Berlin 10115, Germany. Tel.: +49 30 2093 8389; Fax: +49 30 2093 8813; E-mail: marcus.krantz@biologie.hu-berlin.de

Received 8.7.11; accepted 16.3.12

Intracellular signalling systems are highly complex. This complexity makes handling, analysis and visualisation of available knowledge a major challenge in current signalling research. Here, we present a novel framework for mapping signal-transduction networks that avoids the combinatorial explosion by breaking down the network in reaction and contingency information. It provides two new visualisation methods and automatic export to mathematical models. We use this framework to compile the presently most comprehensive map of the yeast MAP kinase network. Our method improves previous strategies by combining (I) more concise mapping adapted to empirical data, (II) individual referencing for each piece of information, (III) visualisation without simplifications or added uncertainty, (IV) automatic visualisation in multiple formats, (V) automatic export to mathematical models and (VI) compatibility with established formats. The framework is supported by an open source software tool that facilitates integration of the three levels of network analysis: definition, visualisation and mathematical modelling. The framework is species independent and we expect that it will have wider impact in signalling research on any system.
Molecular Systems Biology **8**: 578; published online 24 April 2012; doi:10.1038/msb.2012.12
Subject Categories: metabolic and regulatory networks; computational methods; simulation and data analysis
Keywords: combinatorial complexity; mathematical modelling; network mapping; signal transduction; visualisation

Introduction

All living cells interact with and respond to their environment via the cellular signal-transduction network. This network encompasses all cellular components and processes that are required to receive, transmit and interpret information. Due to its key role in cellular physiology, the signalling network, and several of its subnetworks, have been intensely studied in a range of organisms. However, such networks are highly complex and difficult to analyse due to the so-called combinatorial explosion (Hlavacek *et al*, 2003). This explosion refers to the fact that the specific state of each component is determined by multiple covalent modifications or interaction partners, and that these possibilities rapidly combine to a very large number of possible specific states. Experimental data do not generally distinguish between all these specific states, but instead focus mostly on reactions between pairs of components, usually giving no or limited information on other modifications or interaction partners of the reactants. Hence, there is a discrepancy between the granularity of the empirical data and the highly defined specific states used in most mathematical models. This makes the interpretation and use of empirical data in the context of such model states ambiguous and often arbitrary. These problems pose major challenges for systems biology, as they prevent us from (i) unambiguously describing a network, (ii) visualising it without simplifications or unsupported assumptions and (iii) automatically generating mathematical models from knowledge in data repositories.

Large efforts have been invested in addressing these issues. Signalling systems are commonly visualised through the informal 'biologist's graph' that is simple and intuitive, but lacks the stringent formalism and precision required to meet the three criteria above (exemplified by Thorner *et al*, 2005). The lack of standardised glyphs (defining e.g., mechanism of information transfer and how edges combines to regulate target nodes) makes the information in the 'biologist's graph' ambiguous and difficult to reuse. To address this, the

community has developed the Systems Biology Graphical Notation, SBGN (Le Novere *et al*, 2009). This includes three visual formats; the activity flow diagram, the entity relationship diagram and the process description (or process diagram). The activity flow diagram shares many properties with the 'biologist's graph', but the entity relationship diagram and process description allow precise representations. The process description corresponds to the state transition reaction format used in most models developed by the systems biology community, and which have been standardised in the Systems Biology Markup Language (SBML; Hucka *et al*, 2003). The process description could meet each of the three criteria above but its utility is severely affected by the combinatorial explosion. It is based on a specific state description, which means that, for each component, each possible combination of modifications and interaction partners must be accounted for explicitly. Hence, only very simple systems can be described completely and only very few models include the entire state space (Kiselyov *et al*, 2009) while the vast majority include simplifying omissions. While simplifications are often necessary, the lack of discrimination between arbitrary omissions and exclusions based on experimental evidence is a significant shortcoming. These issues are partially addressed in the entity relationship diagram, or molecular interaction map, which comes in two flavours; explicit and implicit (called heuristic and combinatorial by the author (Kohn *et al*, 2006)). The explicit version requires all specific states to be displayed and hence share the limitations of the process description. In contrast, the implicit version displays only the possible reaction types (or elemental reactions, as we will call them below) and hence largely avoids the combinatorial explosion. The entity relationship diagram represents each component as a single node and reactions in a condensed format. While not as intuitive as the other SBGN formats, it has the advantage of concentrating all information on a given protein and works especially well for simple regulatory circuits, as the concentrated information makes it difficult to trace the order of events in more complex networks. The three SBGN format has complementary strengths, but there is currently no software available for conversion between the three different visualisation formats. However, the SBGN standards are under continuous development and these issues will likely be addressed in the future through the SBGN markup language, SBGN-ML.

Similar efforts on the modelling side have resulted in rule-based modelling and associated visualisation formats (Faeder *et al*, 2005). Briefly, rules are defined as reactions that are valid under a particular set of contingencies, and each reaction is specified for each such contingency set. This means that when a reaction's rate is increased by phosphorylation of one component it will be defined by two rules; one where that component is phosphorylated and one where it is not. While these rules define the entire state space and the system stays subject to the full combinatorial explosion, the rule description has alleviated the combinatorial problem in two respects: (1) the system has been described more compactly and (2) the actualised state space might be significantly reduced by introducing only those states that are actually populated (Lok and Brent, 2005), or by using agent-based stochastic modelling (Sneddon *et al*, 2011). The rule definition format is also a significant step towards the granularity of empirical data, as compared with the abstract-specific states. These advantages are mirrored on the visualisation side by graphical reaction rules, which use the process description format to display individual rules (Blinov *et al*, 2006). Network level visualisation has used either topological contact maps (Danos, 2007) or entity relationship diagrams (Le Novere *et al*, 2009), and these complementary visualisation formats have recently been combined in the extended contact map (Chylek *et al*, 2011). Contact maps have software support, but neither entity relationship diagrams nor extended contact maps can be generated automatically from the rule-based models. Hence, the rule-based format partially addresses the automatic creation of models from data repositories (iii), as it provides the tools to generate mathematical models automatically once the knowledge has been reformulated as rules. However, the rule-based system provides a cumbersome format for (i) unambiguous network description and is not developed for (ii) comprehensive visualisations. Taken together, this raises the question whether graphical- and model-based formats are the most appropriate for stringent network definition, or whether there are more suitable network definition formats that allow both visualisation and automatic model generation.

Here, we present a new framework to describe cellular signal-transduction networks. Our network definition has the same granularity as experimental data, avoids the combinatorial complexity, can be automatically visualised in complementary graphical formats including all three SBGN formats and unambiguously defines mathematical models. The *rxncon* software tool complements the framework by automating visualisation and model creation. The key feature of our framework is the strict separation of elemental reactions (and their corresponding states); which defines the possible signalling events in the network, from contingencies; which describes the contextual constrains on these reactions. Importantly, each elemental reaction corresponds directly to a single empirical observation, such as a protein–protein interaction or a specific phosphorylation. The contingencies define the constraints on these elemental reactions in terms of one or more elemental states, for example, by defining the active state of a protein kinase or the composition of a functional protein complex. Hence, the format directly link model states to empirical observations at the same level of granularity, which pre-empts the need for additional assumptions or extrapolations. Moreover, the separation between reactions and contingencies largely avoids the combinatorial explosion as only combinatorial states with known functional influence are considered. The *rxncon* tool provides automatic export to established visual formats and to two new visualisation methods, which allow compact comprehensive representation. Finally, the framework is stringent and unambiguously defines a mathematical model, and the *rxncon* tool support export to SBML and rule- or agent-based models. This allows coding of models in a format that mirrors empirical data, which can be automatically visualised and which is highly suitable for iterative model building. We illustrate our new approach by conducting the most comprehensive literature survey to date of the complete MAP kinase signalling network of *Saccharomyces cerevisiae*. Taken together, we provide a framework that integrates the three levels of network analysis;

definition, visualisation and mathematical modelling and a supporting software tool for automatic visualisation and export to mathematical models. We expect this to be highly useful for the community and envision a common framework to bridge different standards as well as experimental and theoretical systems biology efforts.

Results

This section describes the architecture of the framework, including its data structure, the different methods of visualisation and how it relates to a mathematical model (Figure 1A). In the first part, we present the results of the methods development and describe the system in detail. In the second part, we present our results using the MAP kinase network. The framework has been implemented in the *rxncon* software tool that is distributed freely under the open source LGPL licence and can be downloaded from www.rxncon.org.

The data structure

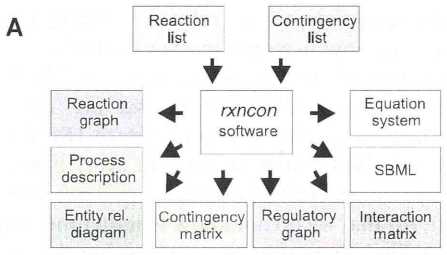
The events in a signal-transduction network can be categorised in four types: (1) catalytic modifications, (2) bindings and interactions, (3) degradation and synthesis and (4) changes in localisation. Due to the limited information on spatial (re)distribution of components, we have focused on types 1–3 here (Table I). However, the framework is fully capable to include localisation reactions and the *rxncon* tool will be upgraded to encompass these in the future. The first step of the network definition is to distil the available knowledge into two distinct categories of information: *what* can happen, and *when* it can happen. The *what*-aspect (referred to as C1, or *elemental reactions*) specifies the possible events, including the event type (1–3 above), and which components and sites that are involved. The *when*-aspect (referred to as C2, or *contingencies*) specifies how the reaction rate is affected by the state of the involved components. For instance, the MAP kinase Hog1 phosphorylate its target Hot1 (C1—'what'; Figure 1B), and this reaction only occurs when Hog1 is phosphorylated on both Thr174 and Tyr176 (C2—'when'). This second category of knowledge therefore represents the causal relationships, or *contingencies*, between the *reactions* characterised in the first class of knowledge. The separation of C1 from C2 allows us to define even large complex networks stringently in a concise format, as exemplified with the yeast MAP kinase network below.

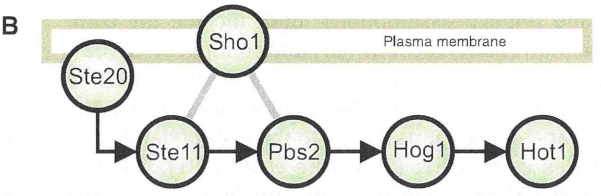
The what-aspects of the knowledge are represented in the *reaction list* (Figure 1C; simplified example). Importantly, we have broken down the reaction network in *elemental reactions*, which change *elemental states*. An elemental state is similar to an empirical observation, such as an interaction between two proteins or a specific modification at a specific site on a specific protein. If a protein has been phosphorylated on two sites, this corresponds to two different elemental states. In other words, the elemental states correspond to overlapping (non-disjoint) sets. This is different from the specific states in ordinary state transition models, but analogous to the macroscopic states used in the works by Conzelmann *et al* (2008) (Borisov *et al*, 2008). An elemental reaction is similarly defined as a two-component reaction that modifies a single elemental state. Note that this precludes lumped reactions and that, for example, a kinase–substrate interaction and phosphorylation must be described by two different elemental reactions. Hence, the reaction list has the same granularity as typical empirical data, which pre-empts the need for assumptions in the mapping process. It also allows us to use the established format for high-throughput data (Stark *et al*, 2006), including specific referencing of each reaction with PubMed identifiers and complemented with additional details such as active domains, subdomains and residues (Supplementary Tables S1 and S2).

The when-aspect of the knowledge is described in the *contingency list* (Figure 1D; simplified example). This list defines the contextual constraints on all elemental reactions. Most contingencies will correspond to the direct effect of single elemental states of the components involved in the particular elemental reaction, but *Boolean states* allow for combinatorial effects and indirect effects in, for example, scaffolds that cannot be directly attributed to a single elemental state in one of the reactants. There are six distinct reaction contingencies; the Effector can be absolutely required (!), positive (K+), completely neutral (0), negative (K−), absolutely inhibitory (x) or of unknown effect (?). These overlap partially with the influences of entity relationship diagrams (Le Novere *et al*, 2011), but distinguish between no effect (0) and no known effect (?). The Boolean states provide a middle layer between reaction contingencies and a combination of elemental states and/or inputs, using either 'AND' or 'OR' to define, for example, large complexes or alternative mechanisms. In addition, *inputs* and *outputs* function as elemental states and reactions, respectively, at the interface between the network and the external environment. Each row in the contingency list contains a Target (elemental reaction, output or Boolean state), an Effector (elemental state, input or Boolean state) and a symbol describing how the Effector influences the Target (Contingency) that is a contingency symbol (!, K+, 0, K−, x, ?) when the Target is an elemental reaction or an output and a Boolean operator (AND, OR) when the Target is a Boolean state. The data structure is illustrated with a simplified version of the Sho branch of the HOG pathway (Figure 1B). The reaction list state that, for example, Hog1 phosphorylates ('P+') Hot1 (Figure 1C; eighth reaction; on the last row), and the contingency list state that this reaction requires ('!') that Hog1 is phosphorylated on both Thr174 and Tyr176 (Figure 1D, last two rows). These states in turn correspond to the reactions six and seven, respectively (Figure 1C). Hence, the reaction and contingency information suffice to describe the network and their separation keeps the description concise and at the granularity of empirical data. Consequently, the data structure addresses the first issue; unambiguous network definition.

Visualising the signal-transduction network

We address the second issue; comprehensive visualisation, with two novel forms of visualisation; the *contingency matrix* and the *regulatory graph*. These also keep reactions and contingencies separate and hence avoid the combinatorial



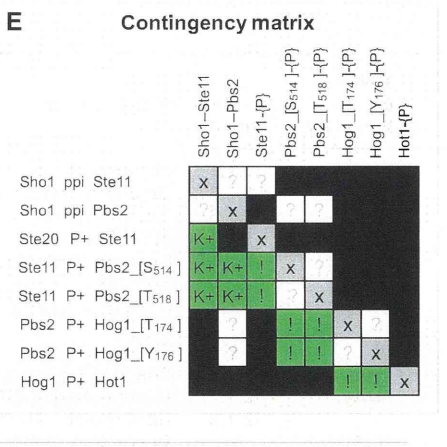


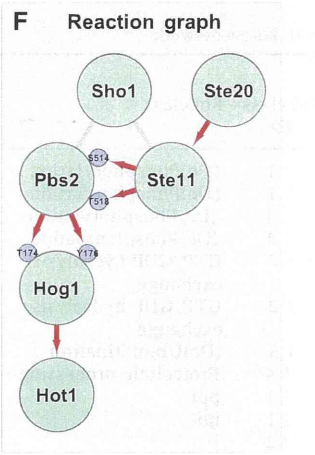
C Reaction list

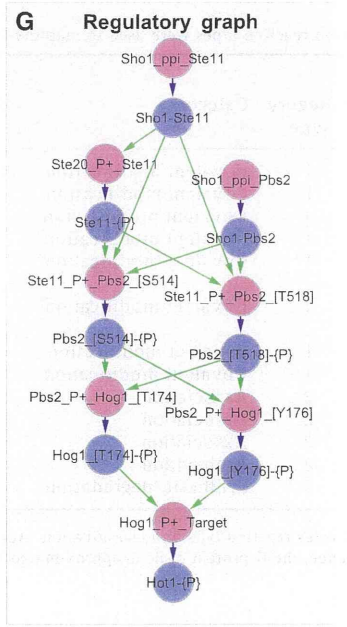
Comp. A	Re. type	Comp. B	Reaction	State
Sho1	ppi	Ste11	Sho1_ppi_Ste11	Sho1–Ste11
Sho1	ppi	Pbs2	Sho1_ppi_Pbs2	Sho1–Pbs2
Ste20	P+	Ste11	Ste20_P+_Ste11	Ste11-P
Ste11	P+	Pbs2_[S514]	Ste11_P+_Pbs2_[S514]	Pbs2_[S514]-P
Ste11	P+	Pbs2_[T518]	Ste11_P+_Pbs2_[T518]	Pbs2_[T518]-P
Pbs2	P+	Hog1_[T174]	Pbs2_P+_Hog1_[T174]	Hog1_[T174]-P
Pbs2	P+	Hog1_[Y176]	Pbs2_P+_Hog1_[Y176]	Hog1_[Y176]-P
Hog1	P+	Hot1	Hog1_P+_Hot1	Hot1-P

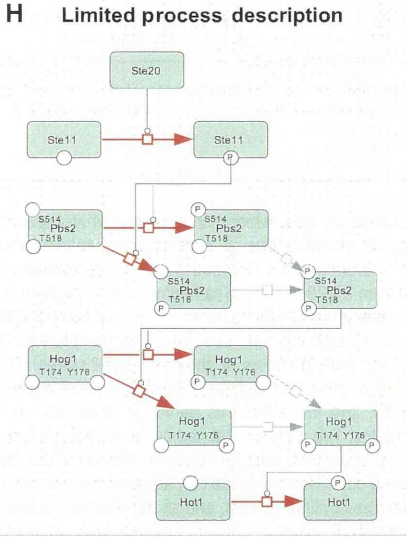
D Contingency list

Target	Cont.	Effector
Ste20_P+_Ste11	+	Sho1–Ste11
Ste11_P+_Pbs2_[S514]	+	Sho1–Pbs2
Ste11_P+_Pbs2_[S514]	+	Sho1–Ste11
Ste11_P+_Pbs2_[S514]	!	Ste11-P
Ste11_P+_Pbs2_[T518]	+	Sho1–Pbs2
Ste11_P+_Pbs2_[T518]	+	Sho1–Ste11
Ste11_P+_Pbs2_[T518]	!	Ste11-P
Pbs2_P+_Hog1_[T174]	!	Pbs2_[S514]-P
Pbs2_P+_Hog1_[T174]	!	Pbs2_[T518]-P
Pbs2_P+_Hog1_[Y176]	!	Pbs2_[S514]-P
Pbs2_P+_Hog1_[Y176]	!	Pbs2_[T518]-P
Hog1_P+_Hot1	!	Hog1_[T174]-P
Hog1_P+_Hot1	!	Hog1_[Y176]-P





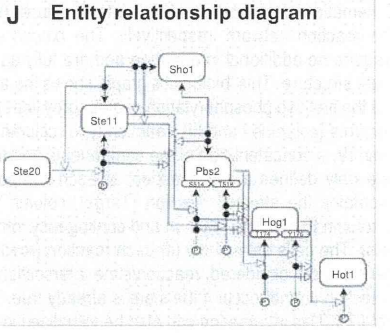




I Interaction/distance matrix

	Sho1	Ste20	Ste11	Pbs2	Hog1	Target
Sho1	–	–	ppi	ppi	–	–
Ste20	–	–	P+	–	–	–
Ste11	ppi	–	–	P+	–	–
Pbs2	ppi	–	–	–	P+	–
Hog1	–	–	–	–	–	P+
Hot1	–	–	–	–	–	–

	Sho1	Ste20	Ste11	Pbs2	Hog1	Target
Sho1	2	–	1	1	2	3
Ste20	2	–	1	2	3	4
Ste11	1	–	2	1	2	3
Pbs2	1	–	2	2	1	2
Hog1	–	–	–	–	–	1
Hot1	–	–	–	–	–	–



explosion and implicit assumptions. Both include the complete information about reactions (C1) and contingencies (C2). This data structure is also well suited for visualisation in entity relationship diagrams or extended contact maps, and the *rxncon* software tool supports export to the entity relationship format (Chylek *et al*, 2011; Le Novere *et al*, 2011). We also provide export to the reaction graph/activity flow diagram and the process description, though neither of these can fully and accurately represent the network as discussed below. Nevertheless, they all provide their unique advantages and can be automatically generated with the *rxncon* tool and the information in the reaction and contingency lists.

The contingency matrix integrates the information in the reaction and contingency lists (Figure 1E). The matrix is spanned by the reactions and their corresponding states (C1) and populated by the contingencies of reactions on states (C2). Each row corresponds to one elemental reaction and each column corresponds to one elemental state. The symbol in each reaction–state intersection specifies how that specific reaction depends on that specific state. Together, one row contains the complete set of rules a reaction follows, and hence describes how it works in every specific state. This is related to a dependency matrix (Yang *et al*, 2010), although the entries in the contingency matrix are more detailed and unambiguous. In the example (Figure 1E), the first row shows that (a) the binding of Sho1 to Ste11 cannot occur if either of the components is already part of such a dimer (column 1), (b) that we do not know whether the prior binding of Sho1 to Pbs2 (column 2) or phosphorylation of Ste11 (column 3) effects the Sho1–Ste11 binding and (c) that the other states appearing in the row are irrelevant for this specific binding reaction—as they do not describe properties of Sho1 or Ste11. The primary advantages of the contingency matrix are that it (1) allows a comprehensive documentation/visualisation of all reactions and dependencies within the network, (2) that it does so without requiring assumptions, (3) that it explicitly defines unknowns and hence gaps in our knowledge and (4) that the matrix constitutes a template from which mathematical models can be derived automatically (see below).

Table I Thirteen reaction types were used to map the MAP kinase network

Reaction	Category type	Category	Subclass ID	Subclass	Modifier or boundary	Reaction type ID	Reaction name
P+	1	Covalent modification	1.1	(De)Phosphorylation	P	1.1.1	Phosphorylation
P−	1	Covalent modification	1.1	(De)Phosphorylation	P	1.1.2	Dephosphorylation
AP	1	Covalent modification	1.1	(De)Phosphorylation	P	1.1.3	Autophosphorylation
PT	1	Covalent modification	1.1	(De)Phosphorylation	P	1.1.4	Phosphotransfer
GEF	1	Covalent modification[a]	1.2	GTP/GDP hydrolysis/ exchange	P	1.2.1	Guanine Nucleotide Exchange
GAP	1	Covalent modification[a]	1.2	GTP/GDP hydrolysis/ exchange	P	1.2.2	GTPase Activation
Ub+	1	Covalent modification	1.3	(De)Ubiquitination	Ub	1.3.1	Ubiquitination
CUT	1	Covalent modification	1.4	Proteolytic processing	Truncated	1.4	Proteolytic cleaveage
ppi	2	Association	2.1	ppi	N/A	2.1.1	Protein–protein interaction
ipi	2	Association	2.1	ipi		2.1.2	Intra-protein interaction
i	2	Association	2.2	i	N/A	2.2	Interaction (non-proteins)
BIND	2	Association	2.3	BIND	N/A	2.3	Binding to DNA
DEG	3	Synthesis/degradation	3.3	DEG	N/A	3.3	Degradation

The table indicates reaction type and classification. Additional details are provided in the 'Reaction Definition' sheet of Supplementary Tables S1 and S2.
[a]For convenience, the G-protein cycle is approximated as a covalent modification by addition/removal of phosphate to/from a basic, GDP-bound form.

Figure 1 Schematic representation of the data structure. (**A**) The input data are the reaction and contingency lists, which contains the 'what-aspects' and 'when-aspects' of the reaction network, respectively. The *rxncon* software uses these lists to create a range of visualisations as well as computational models. These conversions require no additional information and are fully automated. (**B**) A simplified version of the Sho1 branch of the Hog pathway in *S. cerevisiae* will be used to illustrate the data structure. This 'biologist's graph' shows the activating phosphorylation cascade (arrows) from Ste20 to Hot1. Scaffolding and membrane recruitment by Sho1 facilitates the first two phosphorylation events (grey lines). (**C**) The (simplified) reaction list defines the elemental reactions between pairs of components. It includes the two components (columns **I** and **III**), reaction type (column **II**; 'ppi' = protein–protein interaction, 'P+' = phosphorylation; see Table **I** for complete list of reactions), reaction (column IV, a concatenation of the components and the reaction type) and resultant state (column V; protein dimers or phosphorylated states). Note that each elemental state only defines a single aspect of each component's specific state. (**D**) The (simplified) contingency list defines the relationship between states and reactions. It contains the affected reaction (Target, column **I**), the influencing state (Effector, column **III**), and the effect this particular state has on that reaction (contingency, column **II**). (**E**) The reaction and contingency information is summarised in the contingency matrix. The matrix is defined by elemental reactions (rows) and states (columns). The cells define how (if) each reaction (row) is affected by each state (column); that is, the reactions' contingencies on different states. Note that only direct contingencies are considered; reaction/state intersections which do not share components are blacked out. The grey fields ('x') are automatic as states are binary and hence a reaction cannot occur if the state is already true. The green fields ('!'/'K+') are imported from the contingency list, and all other open fields are defined as unknown effect ('?'). This information can also be visualised in a number of graphical forms: The reaction graph (**F**) displays network topology with either components or their domains as functional units. The regulatory graph (**G**) combines the reaction and contingency information to display the causal relationship between the reactions in the network and provides a complete graphical representation of the knowledge compiled in the contingency matrix. The limited process description (**H**) displays the catalytic modifications in the signal-transduction network as state transitions with catalysts but without complex formation (compare Supplementary Figure S1). The interaction and distance matrices (**I**) provide a compact description of network topology and allow calculation of distances between nodes. Finally, the reaction and contingency data can be visualised as an entity relationship diagram (**J**). These visualisations and the equation system for this system, subsystem or your own favourite network defined in the same format can be automatically generated using the *rxncon* software.

The *reaction graph* displays a topological, directed reaction network (Figure 1F). It represents each entity as a single node and each relationship between a pair of entities as a single edge. Edges can be non-directional (e.g., protein – protein interaction), unidirectional (e.g., phosphorylation) or bidirectional (e.g., phosphotransfer). The full reaction graph displays the domains and residues involved in each reaction. The protein parts are independent nodes and defined as neighbours (proteins can have domains or residues, domains can have subdomains or residues, subdomains can have residues). The inclusion of domain information makes the reaction graph similar to the (extended) contact maps (Danos, 2007; Chylek *et al*, 2011). The reaction graph and contact maps are both purely topological and do not include any contextual information, in contrast to the extended contact map which, for example, may show that binding only occurs to phosphorylated residues. We also use a condensed variant that displays only the central node for each component and collapses multiple reactions of the same kind between a pair of components to a single edge, and hence corresponds closely to the activity flow diagram of SBGN (Supplementary Figure S1B; Le Novere *et al*, 2009). The advantages of the reaction graph are (1) the relative simplicity that makes it useful for visualisation of even large networks and (2) that it is suited for visualisation of large-scale data sets within the context of that network (see below).

The *regulatory graph* shows how information is conveyed through the network (Figure 1G). It improves on the reaction graph by including information on causality between the reactions in the network (C2 data). The regulatory graph shows the network's regulatory structure; that is, which reactions (via states) actually influence the rate of other reactions. It is a bipartite graph with the elemental reactions (red) and elemental states (blue) as nodes. Reaction-to-state edges simply show which reactions produce or consume which states. The state-to-reaction edges show which states (products of upstream reactions) affect the dynamics of which (downstream) reactions. These state-to-reaction edges correspond to the symbols in the contingency list, i.e., '!', 'K+', 'K−' or 'x'. The regulatory graph can easily be translated into an influence graph, which can be used for structural analysis of the network (Kaltenbach *et al*, 2011). In contrast to the influence graph or 'story' (Danos, 2007), the regulatory graph strictly separates the effects of reactions (production or destruction of states) and the modifiers (increase or decrease in reaction rates) via distinct edge types. Furthermore, only the (modified) elemental states are displayed and the (the unmodified) complementary source/target state is implicit. Hence, like in the 'stories', cyclic motifs only appear when there is a true feedback in the system. This visualises both the (possible) sequence of events and the feedbacks clearly. However, in contrast to the 'story', the regulatory graph is comprehensive and simultaneously visualises all possible paths or 'stories'. In this example (Figure 1G), the uppermost node pair corresponds to the reaction where Sho1 binds Ste11 (Sho_ppi_Ste11) and the resulting state Sho1–Ste11. The reaction-to-state edge linking these two nodes identifies Sho1–Ste11 as the product of this binding reaction. Note that the source states for this reaction are omitted (i.e., Sho1 not bound to Ste11 and Ste11 not bound to Sho1). The state-to-reaction edge from Sho1–Ste11 to Ste20_P+_Ste11 shows that the phosphorylation of Ste11 by Ste20 is enhanced in the Sho1–Ste11 complex. This reaction in turn produces the state Ste11-{P}, which is required for phosphorylation of Pbs2 on both Ser514 and Thr518. Hence, the information flow can be followed throughout the network as all edges are unidirectional. The main advantages of the regulatory graph are that it (1) allows a comprehensive documentation/visualisation of all reactions and contingencies within the network, (2) that it does so in a very compact format (3) without forcing non-supported assumptions, (4) that it can be used for structural analysis of the network and (5) that it clearly shows the information flow through the network.

Process descriptions are well established and allow visualisation of the information flow and mechanistic detail simultaneously (Kitano *et al*, 2005). They are excellent for representation of small networks which are completely known, but lack of data (of the right granularity) invariably lead to unsupported assumptions. In addition, these diagrams rapidly become very complex, generally forcing *ad hoc* reduction and additional implicit and unsupported assumptions. Therefore, process descriptions do not allow a complete description of the network with the stringency we require. However, the process description can be clear and easy to read, and we generate a limited version which excludes complex formation and hence avoids most of the combinatorial complexity. The difference is highlighted by the upper three nodes in the example (Figure 1H), where Ste20 phosphorylates Ste11. In contrast to full process description, the binding of Ste11 to Sho1, and how this binding would affect the phosphorylation, is not included (compare Supplementary Figure S1). The (limited) process description has several advantages: It (1) is intuitive to read and (2) defines in which internal state(s) an enzyme is active, its substrate and the exact target residue, which (3) conveys the information flow through the pathway, the enzyme–substrate relationships as well as the gaps in our understanding of these aspects.

The information can also be used to generate interaction matrices that specify which components react with which components. These can be rendered at several levels of detail ranging from a complete interaction matrix including protein domains and target residues that defines each interaction type, via condensed interaction matrices with only one row and column per protein that still contains reaction type information (Figure 1I, upper matrix), to numerical matrices that only include information on connection and directionality. We used the latter to calculate the distances within the network to generate a distance matrix (Figure 1I, lower matrix).

Finally, the *rxncon* tool provides export to entity relationship diagrams (Figure 1J). Like the regulatory graph, the entity relationship diagram displays reactions and contingencies separately and hence largely avoids the combinatorial complexity. The entity relationship diagram has the advantage of concentrating all information on a given protein around a central node, which works especially well for simple regulatory circuits. This emphasises the role of each component within the network, in contrast to the regulatory graph which emphasises the information flow through the network. The entity relationship diagram is generated automatically by the