18. da Huang W, Sherman BT, Lempicki RA: **Bioinformatics enrichment tools: paths toward the comprehensive functional analysis of large gene lists.** *Nucleic Acids Res* 2009, **37**(1):1–13.
19. Medzhitov R: **Recognition of microorganisms and activation of the immune response.** *Nature* 2007, **449**(7164):819–826.
20. Aderem A, Ulevitch RJ: **Toll-like receptors in the induction of the innate immune response.** *Nature* 2000, **406**(6797):782–787.
21. Sun L, Liu S, Chen ZJ: **SnapShot: pathways of antiviral innate immunity.** *Cell* 2010, **140**(3):436–436. e432.
22. Yu WC, Chan RW, Wang J, Travanty EA, Nicholls JM, Peiris JS, Mason RJ, Chan MC: **Viral replication and innate host responses in primary human alveolar epithelial cells and alveolar macrophages infected with influenza H5N1 and H1N1 viruses.** *J Virol* 2011, **85**(14):6844–6855.
23. Reading PC, Whitney PG, Pickett DL, Tate MD, Brooks AG: **Influenza viruses differ in ability to infect macrophages and to induce a local inflammatory response following intraperitoneal injection of mice.** *Immunol Cell Biol* 2010, **88**(6):641–650.
24. da Huang W, Sherman BT, Tan Q, Collins JR, Alvord WG, Roayaei J, Stephens R, Baseler MW, Lane HC, Lempicki RA: **The DAVID Gene Functional Classification Tool: a novel biological module-centric algorithm to functionally analyze large gene lists.** *Genome Biol* 2007, **8**(9):R183.
25. Fukuyama S, Kawaoka Y: **The pathogenesis of influenza virus infections: the contributions of virus and host factors.** *Curr Opin Immunol* 2011, **23**(4):481–486.
26. Chang JT: **Deriving transcriptional programs and functional processes from gene expression databases.** *Bioinformatics* 2012, **28**(8):1122–1129.

doi:10.1186/1471-2164-13-460
Cite this article as: Shoemaker *et al.*: CTen: a web-based platform for identifying enriched cell types from heterogeneous microarray data. *BMC Genomics* 2012 **13**:460.

Bioinformatics Advance Access published May 10, 2012

Software support for SBGN maps: SBGN-ML and LibSBGN

Martijn P. van Iersel[1,2,3,*], Alice C. Villéger[4,*], Tobias Czauderna[5], Sarah E. Boyd[6], Frank T. Bergmann[7], Augustin Luna[8,9], Emek Demir[10], Anatoly Sorokin[11], Ugur Dogrusoz[12], Yukiko Matsuoka[13], Akira Funahashi[14], Mirit I. Aladjem[15], Huaiyu Mi[16], Stuart L. Moodie[1], Hiroaki Kitano[13,16], Nicolas Le Novère[1], and Falk Schreiber[5,17]

[1]EMBL European Bioinformatics Institute, Hinxton, UK, [2]Netherlands Consortium for Systems Biology (NCSB), The Netherlands, [3]Department of Bioinformatics - BiGCaT, University of Maastricht, the Netherlands, [4]School of Computer Science, Faculty of Engineering and Physical Sciences, University of Manchester, UK, [5]Leibniz Institute of Plant Genetics and Crop Plant Research (IPK), Gatersleben, Germany, [6]School of Mathematical Sciences, Faculty of Science, Monash University, Melbourne, Australia, [7]Control and Dynamical Systems, California Institute of Technology, Pasadena, CA, USA, [8]National Cancer Institute, Bethesda, MD, USA, [9]Bioinformatics Program, Boston University, Boston, MA, USA, [10]Computational Biology, Memorial Sloan Kettering Cancer Center, New York, NY, USA, [11]Institute of Cell Biophysics RAS, Puschino, Russia, [12]Computer Engineering Dept., Bilkent University, Ankara, Turkey, [13]The Systems Biology Institute, Tokyo, Japan, [14]Dept. of Biosciences and Informatics, Keio University, Japan, [15]Dept. of Preventive Medicine, Keck School of Medicine, University of Southern California, Los Angeles, CA, USA, [16]Okinawa Institute of Science and Technology, Okinawa, Japan, [17]Institute of Computer Sciences, Faculty of Natural Sciences III, University of Halle, Germany

Associate Editor: Dr. Trey Ideker

ABSTRACT

Motivation: LibSBGN is a software library for reading, writing and manipulating SBGN (Systems Biology Graphical Notation) maps stored using the recently developed SBGN-ML file format. The library (available in C++ and Java) makes it easy for developers to add SBGN support to their tools, whereas the file format facilitates the exchange of maps between compatible software applications. The library also supports validation of maps, which simplifies the task of ensuring compliance with the detailed SBGN specifications. With this effort we hope to increase the adoption of SBGN in bioinformatics tools, ultimately enabling more researchers to visualize biological knowledge in a precise and unambiguous manner.

Availability & Implementation: Milestone 2 was released in December 2011. Source code, example files and binaries are freely available under the terms of either the LGPL v2.1+ or Apache v2.0 open source licenses from http://libsbgn.sourceforge.net.

Contact: sbgn-libsbgn@lists.sourceforge.net

1 INTRODUCTION

The Systems Biology Graphical Notation (SBGN, Le Novère *et al.*, 2009) facilitates the representation and exchange of complex biological knowledge in a concise and unambiguous manner: as standardized pathway maps. It has been developed and supported by a vibrant community of biologists, biochemists, software developers, bioinformaticians and pathway databases experts.

SBGN is described in detail in the online specifications (see http://sbgn.org/Documents/Specifications). Here we summarize its concepts only briefly. SBGN defines three orthogonal visual languages: Process Description (PD), Entity Relationship (ER) and Activity Flow (AF). SBGN maps must follow the visual vocabulary, syntax and layout rules of one of these languages. The choice of language depends on the type of pathway or process being depicted and the amount of available information. The PD language, which originates from Kitano's Process Diagrams (Kitano *et al.*, 2005) and the related CellDesigner tool (Funahashi *et al.*, 2008), is equivalent to a bipartite graph (with a few exceptions) with one type of nodes representing pools of biological entities, and a second type of nodes representing biological processes such as biochemical reactions, transport, binding and degradation. Arcs represent consumption, production or control, and can only connect nodes of differing types. The PD language is very suitable for metabolic pathways, but struggles to concisely depict the combinatorial complexity of certain proteins with many phosphorylation states. The ER language, on the other hand, is inspired by Kohn's Molecular Interaction Maps (Kohn et al., 2006), and describes relations between biomolecules. In ER, two entities can be linked with an interaction arc. The outcome of an interaction (for example, a protein complex), is considered an entity in itself, represented by a black dot, which can engage in further interactions. Thus ER represents dependencies between interactions, or putting it differently, it can represent which interaction is necessary for another one to take place. Interactions are possible between two or more entities, which makes ER maps roughly equivalent to a hypergraph in which an arc can connect more than two nodes. ER is more concise than PD when it comes to representing protein modifications and protein interactions, although it is less capable when it comes to presenting biochemical reactions. Finally, the third language in the

[*]To whom correspondence should be addressed.

SBGN family is AF, which represents the activities of biomolecules at a higher conceptual level. AF is suitable to represent the flow of causality between biomolecules even when detailed knowledge on biological processes is missing.

Efficient integration of the SBGN standard into the research cycle requires adoption by visualization and modeling software. Encouragingly, a growing number of pathway tools (see http://sbgn.org/SBGN_Software) offer some form of SBGN compatibility. However, current software implementations of SBGN are often incomplete and sometimes incorrect. This is not surprising: as SBGN covers a broad spectrum of biological phenomena, complete and accurate implementation of the full SBGN specifications represents a complex, error-prone and time-consuming task for individual tool developers. This development step could be simplified, and redundant implementation efforts avoided, by accurately translating the full SBGN specifications into a single software library, available freely for any tool developer to reuse in their own project. Moreover, the maps produced by any given tool usually cannot be reused in another tool, because SBGN only defines how biological information should be visualized, but not how the maps should be stored electronically. Related community standards for exchanging pathway knowledge, namely BioPAX (Demir *et al.*, 2010) and SBML (Hucka *et al.*, 2003), have proved insufficient for this role (more on this topic in the discussion section). Therefore, we observed a second need, for a dedicated, standardized SBGN file format.

Following these observations, we started a community effort with two goals: to encourage the adoption of SBGN by facilitating its implementation in pathway tools, and to increase interoperability between SBGN-compatible software. This has resulted in a file format called SBGN-ML and a software library called LibSBGN. Each of these two components will be explained separately in the next sections.

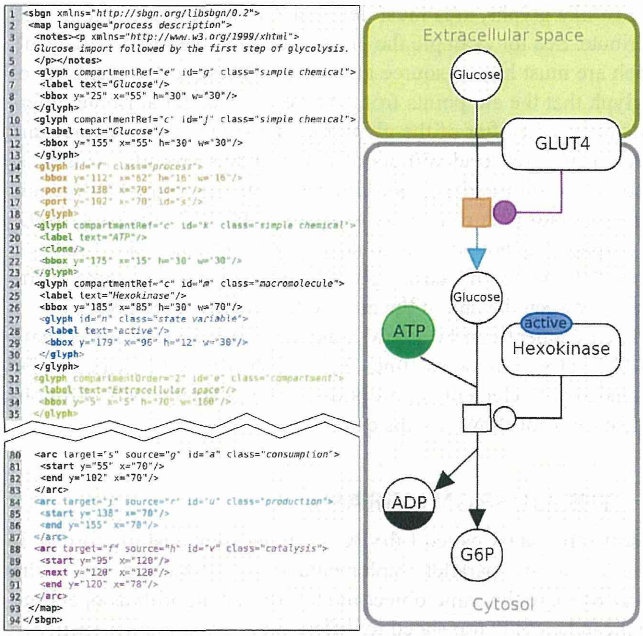
2 THE SBGN-ML FILE FORMAT

SBGN-ML is a dedicated lightweight XML-based file format describing the overall geometry of SBGN maps, while also preserving their underlying biological meaning. SBGN-ML is designed to fulfill two basic requirements:

1. Easy to draw (as a machine) and read (as a human)
2. Easy to interpret (as a machine)

The first set of requirement deals with the graphical aspect of SBGN. It means it should be easy to render an SBGN-ML file to the screen. Therefore, the format stores all necessary information, such as coordinates, to draw the map faithfully, so that rendering tools do not have to perform any complex calculations. Incidentally, this implies the layout of the whole SBGN map has to be expressed explicitly: the size and position of each graphical object and the path of each arc. Various efforts have shown that generating a layout for heterogeneous biological pathways is a computationally hard problem, so a good layout is always worth preserving, if only from a computational perspective. Besides, the choice of a specific layout by the author of a map is often driven by concerns related to aesthetics, readability, or to reinforce ideas of chronology or proximity. This information might be lost with automated layouts. Layout conventions predate SBGN, and are not part of any standard, but they nonetheless play a large role in making it easier for other human beings to understand the biological system being described.

The second requirement encompasses two perpendicular characteristics of SBGN as a language: semantics and syntax. Beyond the picture itself, the format should capture the biological meaning of an SBGN map. Therefore, SBGN-ML specifies the nature of graphical elements (glyphs), following the SBGN terminology (e.g. macromolecule, process, etc.). For example, we can distinguish between a "logic arc" and a "consumption arc" even though they have the same visual appearance. Supporting tools refer to this terminology and draw the glyph according to the SBGN specifications. In terms of syntax, SBGN-ML encodes information on relationships between the various SBGN objects: the glyphs at both ends of an arc, the components of a complex, the members of a compartment and the "decorations" (such as unit of information, state variable) belonging to specific glyphs and arcs. This semantic and syntactic information is essential to a number of automated tasks, such as map validation, or network analysis (as the topology of the underlying biological network can be inferred from the various relationships encoded by the format).



```
1  <sbgn xmlns="http://sbgn.org/libsbgn/0.2">
2   <map language="process description">
3    <notes><p xmlns="http://www.w3.org/1999/xhtml">
4    Glucose import followed by the first step of glycolysis.
5    </p></notes>
6    <glyph compartmentRef="e" id="g" class="simple chemical">
7     <label text="Glucose"/>
8     <bbox y="25" x="55" h="30" w="30"/>
9    </glyph>
10   <glyph compartmentRef="c" id="j" class="simple chemical">
11    <label text="Glucose"/>
12    <bbox y="155" x="55" h="30" w="30"/>
13   </glyph>
14   <glyph id="f" class="process">
15    <bbox y="112" x="62" h="16" w="16"/>
16    <port y="138" x="70" id="r"/>
17    <port y="102" x="70" id="s"/>
18   </glyph>
19   <glyph compartmentRef="c" id="k" class="simple chemical">
20    <label text="ATP"/>
21    <clone/>
22    <bbox y="175" x="15" h="30" w="30"/>
23   </glyph>
24   <glyph compartmentRef="c" id="m" class="macromolecule">
25    <label text="Hexokinase"/>
26    <bbox y="185" x="85" h="30" w="70"/>
27    <glyph id="n" class="state variable">
28     <label text="active"/>
29     <bbox y="179" x="96" h="12" w="38"/>
30    </glyph>
31   </glyph>
32   <glyph compartmentOrder="2" id="e" class="compartment">
33    <label text="Extracellular space"/>
34    <bbox y="5" x="5" h="70" w="160"/>
35   </glyph>

80   <arc target="s" source="g" id="a" class="consumption">
81    <start y="55" x="70"/>
82    <end y="102" x="70"/>
83   </arc>
84   <arc target="j" source="r" id="u" class="production">
85    <start y="138" x="70"/>
86    <end y="155" x="70"/>
87   </arc>
88   <arc target="f" source="h" id="v" class="catalysis">
89    <start y="95" x="120"/>
90    <next y="120" x="120"/>
91    <end y="120" x="78"/>
92   </arc>
93  </map>
94 </sbgn>
```

Fig. 1. An example PD map (right) with the corresponding SBGN-ML code (left). This example shows the import of glucose followed by the first step of glycolysis. The colors used have no special meaning in SBGN, here they merely indicate the relation between each SBGN glyph and its SBGN-ML representation; a process node in orange, a simple chemical (ATP) in green, a production arc in cyan, a catalysis arc in purple, a compartment in yellow and a state variable in blue.

To explain the syntax of SBGN-ML in more detail, consider the example in Figure 1. The figure shows a PD map describing the import of glucose by GLUT4, followed by the first step of the glycolysis. The root element is named "sbgn" (line 1). Below that, there is a "map" element with an attribute indicating that the PD language is used. Below the map element, one finds a series of glyph and arc elements. Each glyph carries a "class" attribute to denote the meaning in SBGN terms. In this example, there is a 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 (line 16-17). 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 (line 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.

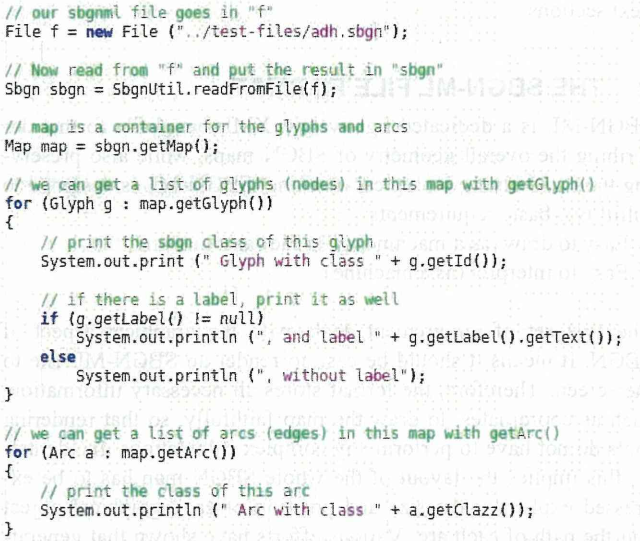
The primary goal of LibSBGN is to simplify the work for developers of existing pathway tools. To reach this goal we followed three design principles. First, we avoided tool-specific implementation details. Implementation artifacts that are specific for one bioinformatics tool would impose difficulties 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 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=Developer_tutorial) aimed at developers who want to include LibSBGN into an existing bioinformatics application.



```
// 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.

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.

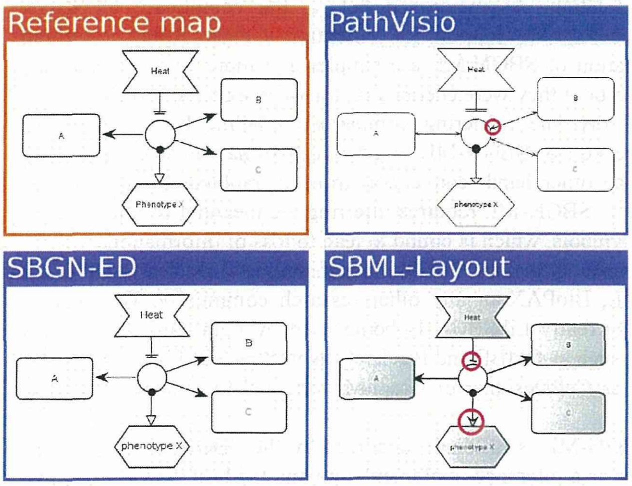


Fig. 3. Rendering comparison. A series of test-cases is rendered by all supported tools in an automated render comparison pipeline. The rendering results are compared to the reference map (top-left), in this case an ER map. A couple of significant differences have been highlighted with red circles. In the PathVisio case (top-right), arrowheads are drawn where none are 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.

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 Fig. 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 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 Fig. 4.

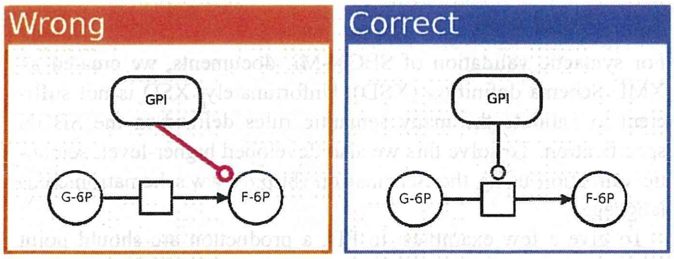


Fig. 4. Typical validator benchmark. This particular example tests the software for rule pd10110: In PD maps, catalysis arcs must point to a process node (not to an entity pool node). In the negative test-case on the left, the enzyme GPI appears to "catalyze" a molecule rather than a reaction. This is 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.

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.

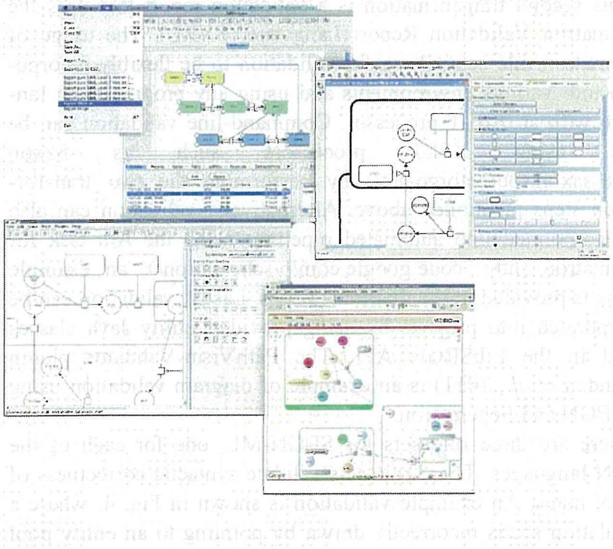


Fig. 5. Screenshots of a number of tools that use LibSBGN. Clockwise, from the top: CellDesigner, SBGN-ED, VISIBIOweb and PathVisio. These tools are able to use SBGN-ML for import, export or both. At the time of writing, for some of these tools a version with SBGN support has not been officially released,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

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.

Secondly, 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.

Thirdly, 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. 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). Authors are grateful for useful feedback from the Path2Models project.

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. *Nature Biotechnology,* 28, 935–942.

Dilek,A. *et al.* (2010) VISIBIOweb: visualization and layout services for BioPAX pathway models. *Nucleic Acids Research,* 38, W150–154.

Funahashi,A. *et al.* (2008) CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks. *Proceedings of the 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.

Le Novère,N. *et al.* (2009) The Systems Biology Graphical Notation. *Nature Biotechnology,* 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.

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

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

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

Molecular Systems Biology 8; Article number 578; doi:10.1038/msb.2012.12
Citation: ***Molecular Systems Biology*** **8: 578**
© 2012 EMBO and Macmillan Publishers Limited All rights reserved 1744-4292/12
www.molecularsystemsbiology.com

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