Process model repositories and PNML

K.M. van Hee
R.D.J. Post
L.J.A.M. Somers
J.M.E.M. van der Werf
TU Eindhoven

Abstract

Bringing system and process models together in repositories facilitates the interchange of model information between modelling tools, and allows the combination and interlinking of complementary models.

Petriweb is a web application for managing such repositories. It supports hierarchical process models, represented in a standard format, PNML (the Petri Net Markup Language). Properties can be associated with models, with values supplied manually or by applying tools. This design allows for easy interfacing with tools for modelling and analysis.

Petriweb's first application is a repository for use in research and education that illustrates theoretical properties of Petri nets with examples and counterexamples.

To make the creation of examples as easy and flexible as possible, we created Yasper (Yet Another Smart Process EditoR). Yasper combines the benefits of a generic diagram editor with unique features for Petri net editing, and interfaces with Petriweb and other tools through PNML import/export.

Petriweb's design is closely tied to that of PNML; it can provide good support for PNML's extensibility. PNML is suitable as a mechanism to define document formats for Petri net types, but conflicts between such definitions are still hard to detect; we propose additions to improve this situation.

1 Model repositories for knowledge sharing

Many techniques and tools exist for modelling static and dynamic aspects of business processes and systems, all with their own features and benefits. Therefore, interoperability between tools is often a major concern. It may be necessary to transfer models from one tool to another; for example, if the second tool has different capabilities, runs on a different platform, is more convenient to use or less expensive.

This is addressed by standard document formats, such as GXL [15], XMI [13], BPEL [11], XPDL [30], and PNML [12]. They define which information can be exchanged between tools, and provide a standard syntax to express it. The ideal is depicted in figure 1: tools that read and write the same document format.

However, this does not guarantee full interoperability. Constraints on how elements can be combined and the intended interpretation of the syntax are often described informally, if at all; this may leave

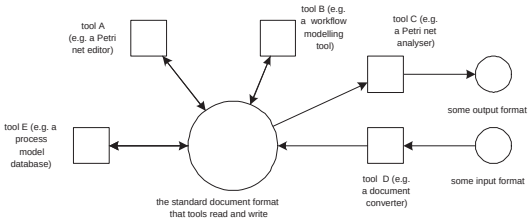

Figure 1: Using a standard document interchange format.

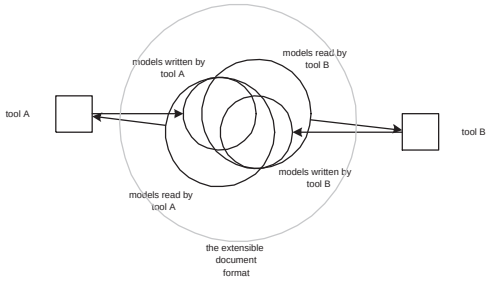

Figure 2: Using an extensible standard format.

the specification ambiguous or incomplete in subtle ways. Problems can be found even when tools completely comply with the document format's specification; in such cases, the specification or the document format itself must be updated, which in turn may require updates to all documents and processing tools. Therefore, standards must undergo careful change management.

To complicate the issue further, two tools may support a common technique with different specific extensions. Extensible document formats allow these to be defined as separate, extended document formats. The situation is illustrated in figure 2: each tool can read only a specific set of models, and usually writes an even further restricted subset, but tools can still interchange models, even when they do not that support the exact same set of extensions. If the extensions are not in conflict, a tool can simply proceed to ignore the extensions it does not understand; but this will not always produce meaningful results. the PNML document format.

An example of such an extensible document format is PNML, the Petri Net Markup Language [12]; it was designed for Petri nets [22], but is also suited for related modelling techniques. A particular type of Petri net – a particular combination of syntactic features – is defined by a Petri Net Type Definition (PNTD). Although it is still under development, PNML is already supported by some tools.

A common interchange format can serve as a simple design contract to discipline the development of new tools. This is especially important when multiple small tools are written to complement existing tools or each other.

However, while well-defined exchange formats are required to make tools interoperable, they are not sufficient. Another important step is to create repositories of models ([16], [20], [14], ...) in which models can be shared, for instance with a publish-and-subscribe mechanism.

Such repositories can serve to provide practical support for a document format, by providing example models, syntax validation services, etc. Tools that support the document format can be interfaced with the repository.

Moreover, repositories can be used to combine and integrate different types of models, that cannot be described with a common document format. In many situations, we require combinations of models of different types to describe a system: for instance, workflow models, organization models and data models. It is not wise to try and cover all this different information with a single document format; there are too many different types of models to consider. However, by combining such models in a repository, we can express relationships between them. For instance, if a Petri net expresses a workflow, its transitions are tasks to be executed by members of the organization, while the data involved appear in a business data model. Whereas an exchange format defines a greatest common divisor between models of different tools, combining them in a repository offers a least common multiple.

The use of repositories allows a less tool-oriented, more model-centric approach to modelling. Instead

of relying on a single tool to do everything, various smaller tools can be used for what they do best.

We are creating such repositories, focusing on process models, for the following two application areas:

- Petriweb, a database of Petri nets for use in research and education
- enterprise modelling with combinations of models

The first application uses a repository to support collections of similar models, namely, Petri nets and related process models, expressed in PNML.

In the second application, we combine and interrelate such process models with other types of models. This ongoing work is not discussed in the present paper.

2 Petriweb: Petri net repository management

The Petriweb application [1] is a web front end to a database with Petri nets. Its purpose is to collect example Petri nets used to illustrate Petri net theory: standard examples, modelling exercises, examples and counterexamples used in proofs. The goal is to advance on the present state of affairs, in which researchers and educators use the first diagram drawing tool or Petri net tool that comes to hand to draw the same illustrations of the same basic material, and reuse very little. Creating good illustrations can be a lot of work. Petriweb aims to develop a repository in which finding a suitable Petri net is less work than creating one from scratch with an ad hoc tool.

Privileged users can upload new PNML documents to the database; an administrator must approve them to make them publicly viewable.

Petriweb mainly contains small, flat, uncolored Petri nets, but it supports hierarchy and allows additional properties to be attached to Petri net elements. All nets are imported and exported in PNML format; basic compatibility with other tools that support PNML, such as the Petri Net Kernel [19], is verified from time to time. Petriweb can even smoothen out incompatibilities between existing PNML variants – see section 2.2.

The static view on Petri nets is not enough. Often, a Petri net must be shown with one or more specific firing sequences. Petriweb delegates the task of generating firing sequences to external tools, but they can be imported into Petriweb in a custom-designed XML format. Firing sequences can be replayed in the built-in browser (figure 5); they can also be employed in certain types of analysis, e.g. process mining [28].

2.1 Properties

The main challenge for Petriweb is the search function. It is obvious how to model a two-place buffer, and somewhat predictable how to model a petrol station, but it is not at all clear how to find such models in a collection of Petri nets. No general assumption can be made on what properties a user would like to use as a basis for search.

Therefore, the Petriweb software leaves the definition of properties open. Arbitrary properties can be attached to Petri nets. They are typed: Boolean, enumerated, integer or string, or a file format. Hand-assigned properties must be filled in by a user when a document is uploaded or approved; nothing

Figure 3: Petriweb: specifying a search

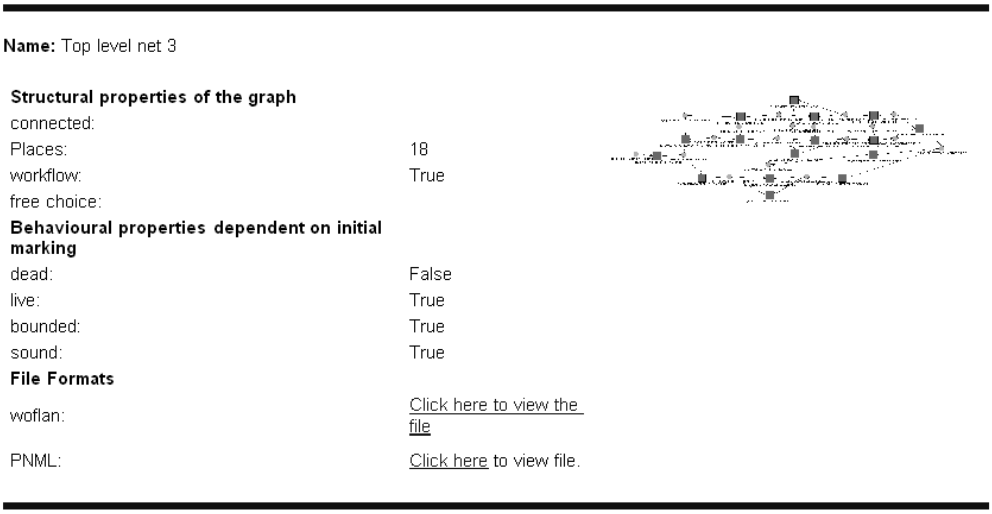

Figure 4: Petriweb: browsing search results

in the system checks that they make any sense. Computed properties have code associated with them that computes a value automatically from PNML. An administrator can delete, add, redefine, and recompute any property. Properties marked as searchable appear on the Petriweb search form (figure 3).

Search results are shown with all their properties, including a diagram sketch (figure 4). The diagram sketch hyperlinks to a full diagram viewer (figure 5), which visualises the Petri net, including hierarchy. Another hyperlink provides the PNML representation.

Most of the properties that users would like to define and use in Petriweb apply to the individual Petri nets and subnets within a PNML document. Therefore, search results are not always whole documents, and properties shown with results or shown in the search form must be attached to nets and subnets. This is true for structural and behavioral properties of a theoretical nature, but just as

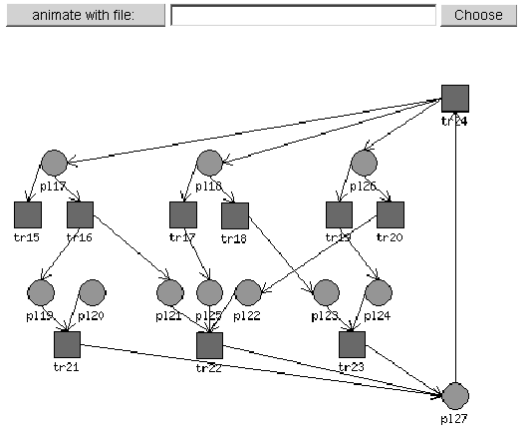

Figure 5: Petriweb: browsing the diagram of a result

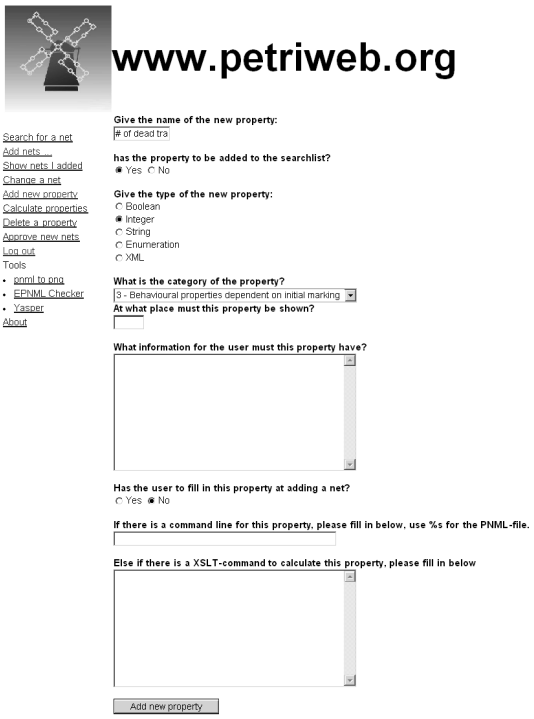

Figure 6: Petriweb: defining a property

well for non-inherent, manually assigned properties, such as a net's stated purpose ("petrol station") or its original author and publication ("Smith & Jones 1984, p. 18").

In order to support this, Petriweb must break down uploaded PNML documents into the smallest constituents for which such properties can be defined. In fact it completely parses the documents and represents the structure of their constituent elements explicitly in the database table structure. This makes the PNML completely regenerable from the database, and allows properties to be associated with literally any element within a net.

This means that a property is, in effect, what in PNML parlance is known as a PNML label: a syntactic construct added to a PNML element type (net, place, page, transition or arc). Although the present version of Petriweb does not support this yet, PNML labels within uploaded PNML documents can be shown as properties in Petriweb, while properties defined within Petriweb can be included as PNML labels into exported PNML. Thus, Petriweb can serve as a tool for defining PNML labels. A corresponding PNTD can be generated automatically.

2.2 Computed properties

When a property is defined, a program can be associated with it to compute its value. To this end the definer provides either a command line to invoke the program in question, or the text of an XSLT [9] stylesheet.

Simple structural properties, such as the number of tokens, whether a net is connected, or whether it is free-choice, are typically defined with XSLT. Such properties are essential in narrowing searches.

More complex, behavioral properties, such as the soundness of workflow nets ([3], [18]) can be computed with specialized external tools, which operate as filters that take a PNML document and

yield one or more property values. In this way, intelligent Petri net processing is incorporated on the server side.

Computed properties can also be used to transform Petri nets into alternative formats. This is most useful when the user has applications installed locally that display or process models in such formats. Useful types of transformations include

conversions to a different form of PNML: for instance, a computed property can flatten structured nets, for the benefit of tools that only accept flat (basic) PNML; another application is to adapt PNML for tools that do not comply with the present PNML standard, such as the Petri Net Kernel and some of our own utilities;

completely different representations of process models: most process modelling and analysis tools have non-PNML native file formats; if a conversion program exists, it can be incorporated as a computed property;

visualizations: e.g. in a vector graphics format such as SVG [25]

The screenshots in figure 3 and figure 4 show the use of computed properties to integrate Woflan [4], a workflow analysis tool. The *woflan* property provides client-side integration: clicking on its value produces a file in native Woflan input, and will feed it to the Woflan application if the user has it installed. Properties such as *workflow*, *live* and *bounded* are the result of server-side integration: their values are computed by a script that invokes the Woflan web service, and the results are stored in Petriweb.

2.3 The range of supported Petri nets

Petri nets feature prominently in our undergraduate computer science curriculum. They are used in courses, student projects, and research. Over time, many users create many Petri nets, and mostly throw them away after one-time use. Petriweb provides the means to collect these efforts and make them available for reuse, where appropriate.

Complex models are created with modelling tools such as the Petri Net Kernel [19], CPN Tools [23] — which support PNML — ExSpect [5], Arena [24], or Protos [21] – which do not. Computed properties (cf. section 2.2) provide a simple mechanism for integration; work to actually interface with some of these tools is in progress.

Our first concern, however, was to support the reuse of relatively simple nets, used as illustrations in course materials, student exercises and research papers. Such nets are often drawn with general-purpose drawing packages. They have uncolored tokens, and appear on a single page, sometimes with a marking. They often exhibit hierarchy in the form of ”inline subnets”, indicated by a named rectangle surrounding part of the net. Inline subnets always act as transitions, in the sense that their cross-border arcs always connect transitions on the inside with places on the outside. They are used often enough to require support by Petriweb and supporting tools.

Modelling tools such as ExSpect [5] and ProVision Workflow Modeller [26] display the contents of subnets on separate pages, as shown in figure 7. The connections and the places they connect to appear twice, once in the context net and once in the subnet. Every place in the context must have a separate reference in the subnet content, while multiple references in a subnet can be made to the same context place. Therefore, the appearances of context places within a subnet are, in effect, references to those places. These places are called the subnet’s pins, and they constitute the subnet’s interface.

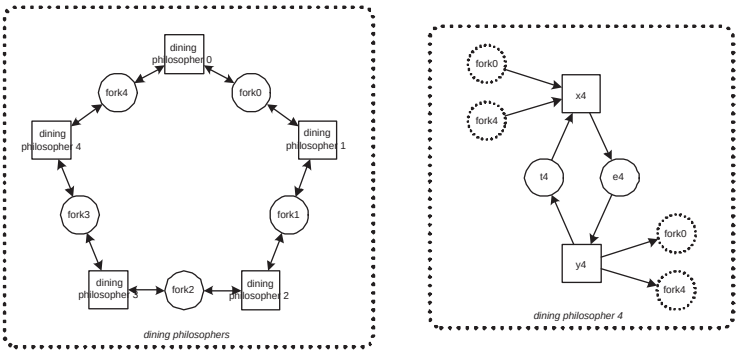

Figure 7: Five dining philosophers, multipage view

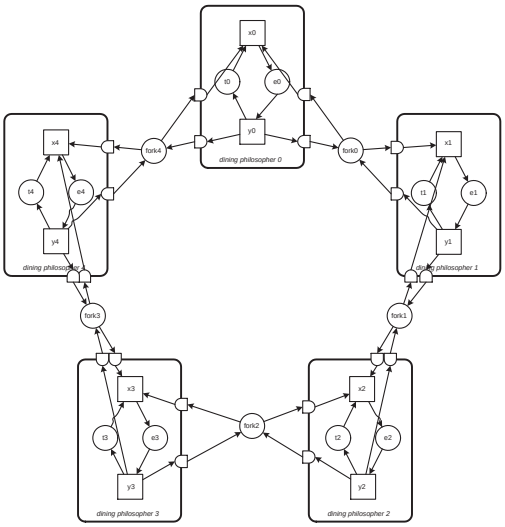

Figure 8: Five dining philosophers, inline subnet view

Inline subnets display hierarchies differently, on a single page. They, too, can be drawn with explicit pins, as shown in figure 8.

Some additional modelling constructs – not illustrated in the diagrams – are used in the repository:

- the inhibitor arc, which prevents a transition from firing if a place is marked
- the biflow arc, a shorthand for a pair of opposite arcs
- the store, a shorthand for a place connected only with biflow arcs, and marked initially
- the XOR block, a shorthand for a subnet with a single place, surrounded by transitions with singleton presets and postsets; it generalizes the XOR splits and joins used in workflow modelling techniques.

2.4 Definition in PNML terms

A PNML-compliant syntax definition for a particular combination of Petri net features is called a PNTD (Petri Net Type Definition).

A PNTD that covers the features discussed above is given on the project website [29]. It expresses XOR blocks, stores, and special arc types by adding an optional PNML label named `type` to transitions, places, and arcs, respectively.

Off-page and inline subnets are defined as a subset of the *structured nets* provided by the PNML standard. Subnets of either type are defined as `<page>`s; pins are `<referencePlace>`s. Defining them in this way maximizes tool interoperability: it reduces the difference between off-page and inline subnets to one of visualization, and allows nets containing such subnets to be manipulated with tools that support arbitrary structured nets.

While defining a PNTD proved straightforward, attempts to actually interchange nets between different tools have demonstrated that defining a PNTD and ensuring that tools comply with it do not warrant interoperability.

2.4.1 Syntactic constraints

The main source of problems is that a PNTD can only be taken as an approximate specification of the PNML syntax allowed. Practically all of the tools we use or develop turn out to assume various additional syntactic constraints, with incompatibilities as a result.

An example is order dependency. While PNTDs allow arcs to precede the definition of the nodes they connect, some tools implement a simple one-pass parsing strategy, and misread documents in which this occurs. It is a simple matter to redefine a PNTD to impose this order dependency; the real issue is that tool authors assume constraints that a PNTD does not define. This could be addressed by stating in the PNTD standard that such assumptions are forbidden.

However, this would require that whenever a syntactic constraint is real, it can be expressed in a PNTD, which is not the case. For instance, the Petri nets described in the previous section all satisfy :

- the source and target of every arc must be within the same net or page as the arc itself
- every reference place must refer to a sibling of the page it is in
- arc types must match with the place type of the place the arc connects to

The formalism used to specify PNTDs, RELAX-NG [7], cannot express these constraints. As a consequence, no PNTD can serve as an exact, application independent definition of syntactic validity for the PNML documents supported by Petriweb.[1]

However, the constraints can be expressed in Schematron [17], a syntax description language based on XPath [6]. The PNTD used to validate documents offered to Petriweb actually includes Schematron rules.[2]

The issue of constraint definition exists for PNML in general. While the third constraint is specific to this PNTD, the first may hold for all Petri nets, and the second is relaxed to the more general

[1] The main alternative for RELAX-NG, XML Schema [8], can express a larger range of constraints.
[2] The full definition can be found in [29].

constraint that references in PNML must not form cycles – a constraint that, as far as we are aware, cannot be expressed in Schematron.

2.5 Extensions for enterprise modelling

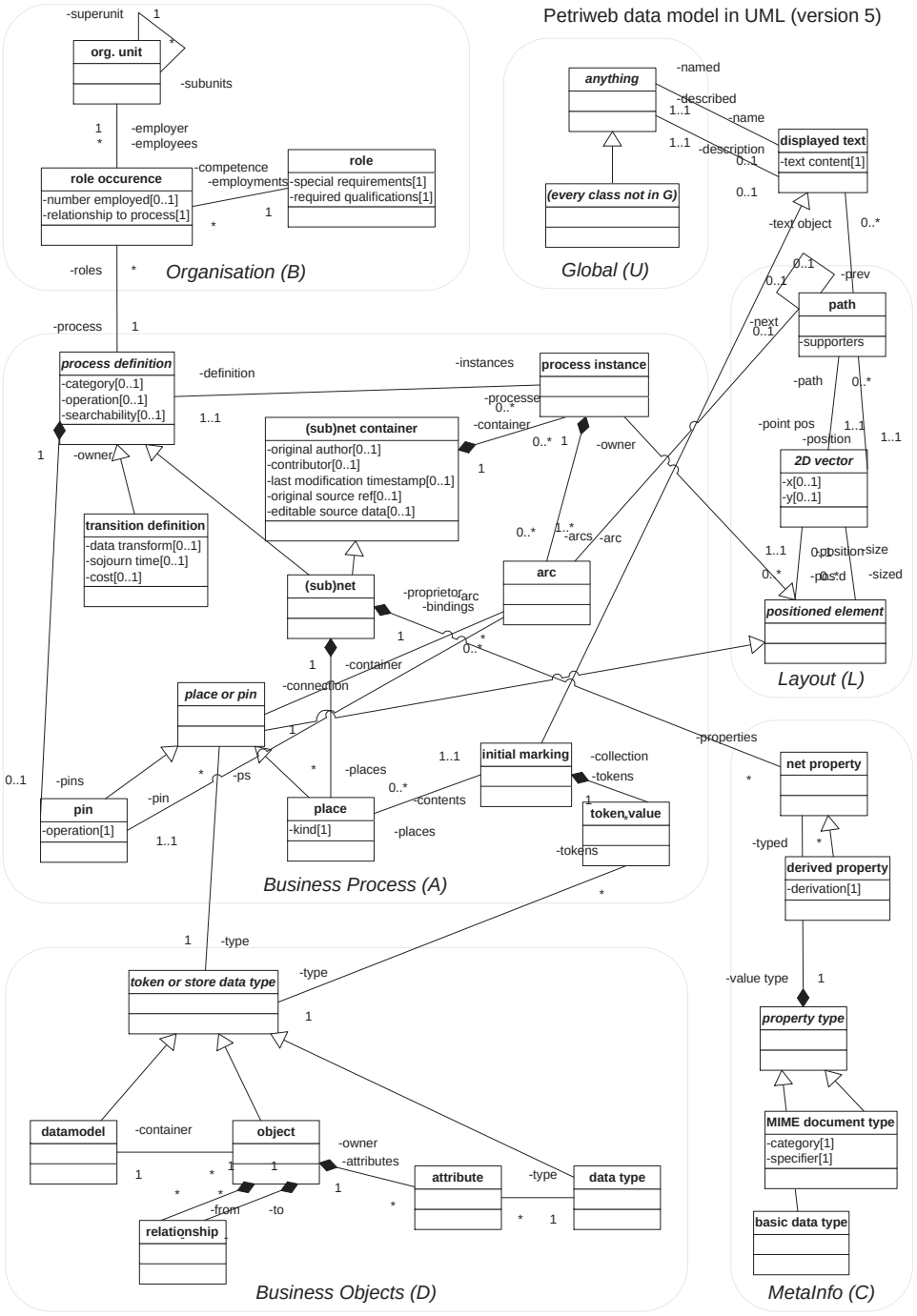

Figure 9: A data model for Petriweb supporting enterprise modelling

The data model implemented in Petriweb is not limited to the features described thus far; it additionally supports the definition and multiple use of subnets, and the integration of process models with business data and organigrams. A full data model is shown in figure 9. We do not support the representation of such information in PNML. However, a separate tool is being developed that already employs this information for modelling and simulation purposes. For instance, persons in the organization can be assigned as possible actors to transitions in the process model.

2.6 Yasper, Yet Another Smart Process EditoR

The nets collected in Petriweb are mainly used as illustrations. Therefore, specialized process modelling tools are not very suitable for creating them: general-purpose drawing packages are easier to install and use, provide more freedom in visualisation, better looking diagrams, and better image format conversion facilities.

One such drawing package is Microsoft Visio™ [10]. It exports diagrams into many image formats and integrates nicely with Microsoft Office™ applications.

Visio provides specific benefits above most other vector drawing packages. First, the look and feel of diagrams is extremely customizable. Second, any user can save sets of customized diagram shapes and fragments in *stencils*, and build diagrams out of them.

In addition, Visio is scriptable. Actions with arbitrary, scripted, effects can be associated with documents, pages, and shapes, with diagram changes and other user interface events. We have employed this facility to develop a full-blown smart Petri net editor: Yasper [2].

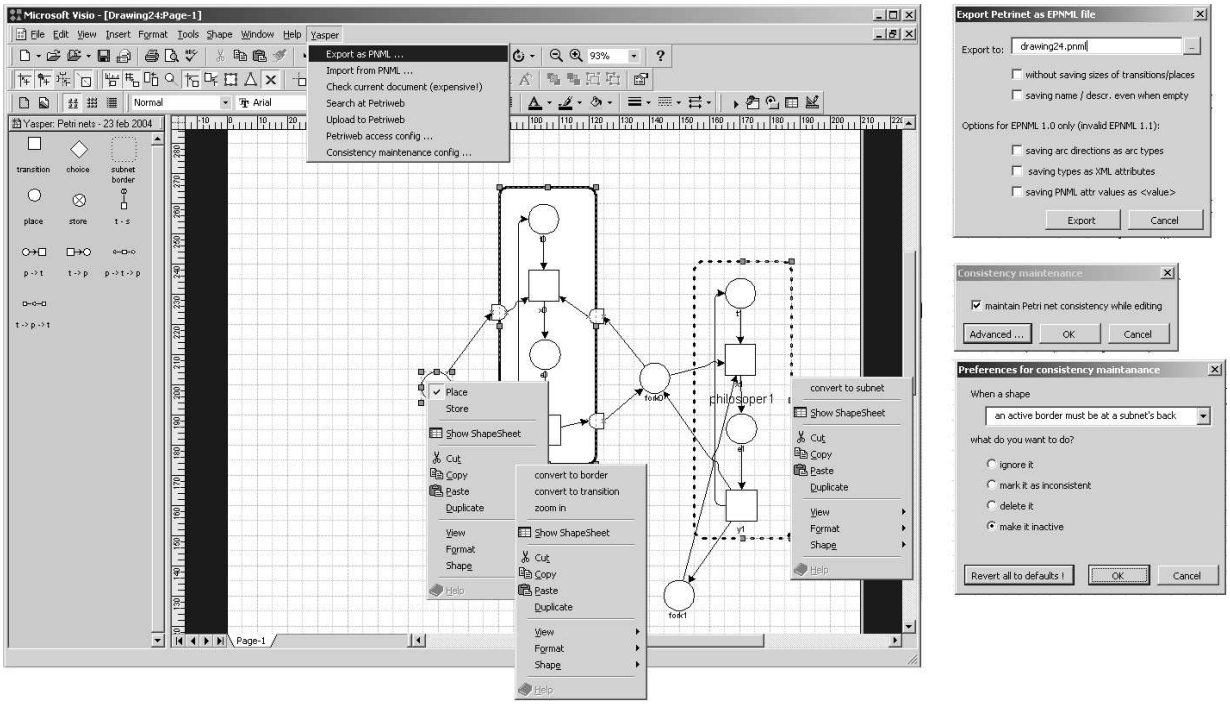

Figure 10: A (combined) Yasper screenshot

Yasper preserves all Visio functionality, and thereby, all benefits already listed. To these, it adds:

- interchange of nets with Petriweb and other Petri net tools
- a single-page view on hierarchical nets (inline subnets)
- easy, arbitrary restructuring of hierarchy
- a correctness-by-construction editing mode, based on node fusion
- optional preservation and restoration of consistency properties
- the creation of token flows

This is realized through the following mechanisms:

- export and import of diagrams in PNML format
- additional special-purpose operations on Petri net shapes
- consistency maintenance on diagrams

The Yasper user interface is shown in figure 10, a combination of screenshots, showing many menu items and dialogues that can actually only appear one at a time.

The Yasper menu offers global configuration options, and PNML document import and export; direct document interchange with Petriweb is an added convenience. Several variants of PNML can be written, an issue we will discuss shortly.

The three menus within the diagram can be invoked on individual shapes, and demonstrate additional shape actions that depend on the shape's type. For example, a place can be turned into a store (a modelling feature from ExSpect); a subnet, as appearing on the left, can be converted to a subnet border, as appearing on the right, and vice versa. Such added functions make Petri net editing much more convenient.

The conversions between subnet borders and subnets prove very convenient to structure and restructure nets. A subnet border is just a rectangle that can be put anywhere in the diagram; a subnet on the other hand drags its contents along with it, and requires connections across its borders to be marked with pins.

This functionality relies on a mapping between the elements and structure of a Visio diagram and those of a Petri net. Yasper conforms with Visio's general method od operation, by supplying a stencil with Petri net building blocks, and regarding all shapes as Petri net elements that have been derived from this stencil. Other shapes, such as additional text elements, are regarded as decoration.

To guarantee that the collections of Petri net shapes used in diagrams actually constitute valid Petri nets, Yasper provides consistency maintenance. Examples of constraints:

- transitions and places do not overlap
- every arc connects a place and a transition
- every shape belongs to the subnet geometrically surrounding it, or to the outer net, otherwise
- shapes are drawn in front of the (sub)net they are part of
- arcs must cross subnets only via pins
- in inline subnets, pins must be placed on the subnet's border

Every constraint is a predicate on shapes with an associated corrective action. Inconsistent shapes can be repaired using the corrective action, deleted, marked as inconsistent, or ignored altogether. Changes can be made with or without user confirmation. Constraints can be checked on the present diagram and/or maintained during editing.

With constraint correction fully enabled, a diagram remains a consistent Petri net at any time. It even becomes impossible to have arcs with loose ends, which means that building blocks can no longer be connected with arcs. However, another constraint forbids overlapping transitions and places, and corrects them by fusing them together. This is a more natural way of working, but not always the most efficient one; for this reason, the user can turn constraint maintenance on and off.

The constraint resolution mechanism allows Yasper to serve as an editor for a multitude of Petri net types, if the differences between them are formulated in terms of constraints. This flexibility comes at a price to the developer: it is a serious task to analyze all interdependencies between constraint resolution rules.

3 Proposals for PNML, its use and definition

Our experiences with PNML prove its practical usefulness.

Most of the Petri nets our applications need to support were already expressible in PNML; additional syntax to support special features proves easy to define with the PNTD mechanism; part of this syntax, and the description of their use, can be moved into PNML's standard conventions document, as described by [12].

Our first proposal is to compare our PNTD with those of others and decide on a standard representation for additional modelling constructs, such as inhibitor arcs. Generally speaking, commonly used features such as inhibitor arcs must become a standardized part of PNML. Currently, there is no well-defined process in effect to achieve this.

Our second proposal is to support, or even require, the inclusion of constraint definitions into PNTDs, as discussed in section 2.4.1. It seems reasonable to adopt a constraint specification language that is already supported by existing syntax checkers, such as Schematron. Constraint specifications not only enable early detection of subtle incompatibilities, they also clarify the meaning of constructs.

A complementary approach is to explicitly recognize, within the PNML standard, the possibility of PNML types that do not have an exact PNTD.

A minor source of problems is caused by differences in interpretation. For instance, PNML does not specify which units of distance to use, so different tools can write positions and sizes to PNML in their own internal units; the need for conversion only becomes apparent when they try to read each other's PNML documents. Several such trivial ambiguities exist. Our proposal is to eliminate them as they are found, by adding disambiguating remarks to the PNML specifications.

Finally, we would like to see a standardized change management process for PNML and PNTDs. A change to a formal syntax definition can break other definitions that depend on it, existing syntax checkers, import/export functions, XSLT stylesheets, and existing collections of PNML models. With proper versioning of all formal documents regarding PNML, changes can no longer slip by unnoticed.

4 Conclusion

Repositories of process models can be developed and successfully integrated with both existing and new modelling and drawing tools by using the PNML language as an interchange format. PNML is quite suitable for this purpose, but its usability would further improve with more finishing polish, such as the further standardization of Petri net type definitions, specification of constraints, and a well-specified change management process.

Our first goal was to simplify the creation, use and reuse of example Petri nets for illustration purposes; the software to support this is largely in place, and future work in this area will mainly concentrate on the practical collection and application of examples. Support for specific modelling features can be added as the need arises.

Our second goal is to use repositories to combine process models with different, complementary views of systems. This work is in an early stage.

References

[1] Petriweb (website), May 2003. www.petriweb.org. Available from: http://www.petriweb.org/.

[2] Yasper, Yet Another Smart Process EditoR (website), September 2003. www.yasper.org. Available from: http://www.yasper.org/.

[3] W.M.P. van der Aalst. Structural Characterizations of Sound Workflow Nets. Computing Science Reports 96/23, Eindhoven University of Technology, Eindhoven, 1996. Available from: http://citeseer.nj.nec.com/vanderaalst96structural.html.

[4] W.M.P. van der Aalst, D. Hauschildt, and H.M.W. Verbeek. A Petri-net-based Tool to Analyze Workflows. In B. Farwer, D. Moldt, and M.O. Stehr, editors, *Proceedings of Petri Nets in System Engineering (PNSE'97)*, pages 78–90, Hamburg, Sept 1997. University of Hamburg (FBI-HH-B-205/97).

[5] Bakkenist Management Consultants. *ExSpect 6.2 User Manual*. http://www.bakkenist.nl/, 1998.

[6] James Clark and Steve DeRose. Xpath, 1999. Available from: http://www.w3.org/TR/xpath.

[7] James Clark, Makoto Murata, and OASIS RELAX NG TC. RELAX-NG, 2003. the website links to all things RELAX-NG. Available from: http://www.thaiopensource.com/relaxng/.

[8] W3C Consortium. XML Schema, 2003. Available from: http://www.w3.org/XML/Schema.

[9] World Wide Web Consortium. http://www.w3.org/TR/xslt, June 2001. Available from: http://www.w3.org/TR/xslt.

[10] Microsoft Corporation. *Developing Microsoft Visio Solutions*. Microsoft Press, Redmond, WA, USA, 2001. Available from: http://www.microsoft.com/visio/.

[11] Francisco Curbera, Yaron Goland, Johannes Klein, Frank Leymann, Dieter Roller, Satish Thatte, and Sanjiva Weerawarana. Business Process Execution Language for Web Services, Version 1.0. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel1/. Available from: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel1/.

[12] J. Billington et al. The Petri Net Markup Language: Concepts, Technology, and Tools. In van der Aalst and Best [27]. Available from: http://citeseer.nj.nec.com/billington03petri.html.

[13] Object Management Group. XML Metadata Interchange (XMI), 1997-2004. http://www.omg.org/technology/documents/formal/xmi.htm. Available from: http://www.omg.org/technology/documents/formal/xmi.htm.

[14] Volker Gruhn and Monika Schneider. Workflow management based on process model repositories. In *Proceedings of the 20th international conference on Software engineering*, pages 379–388. IEEE Computer Society, 1998.

[15] Richard C. Holt, Andreas Winter, and Andy Schürr. GXL: Towards a Standard Exchange Format. Technical Report 1–2000, Universität Koblenz-Landau, Institut für Informatik, Rheinau 1, D-56075 Koblenz, 2000. Available from: citeseer.nj.nec.com/holt00gxl.html.

[16] Matthias Jarke. Process repositories: Models and experiences. In Pericles Loucopoulos, editor, *Entity-Relationship Approach - ER'94, Business Modelling and Re-Engineering, 13th International Conference on the Entity-Relationship Approach, Manchester, U.K., December 13-16, 1994, Proceedings*, volume 881 of *Lecture Notes in Computer Science*, page 314. Springer, 1994.

[17] Rick Jeliffe. The Schematron. An XML Structure Validation Language using Patterns in Trees, 2003. Available from: http://www.ascc.net/xml/resource/schematron/schematron.html.

[18] Ekkart Kindler, Axel Martens, and Wolfgang Reisig. Inter-operability of workflow applications: Local criteria for global soundness. In *Business Process Management*, pages 235–253, 2000. Available from: http://citeseer.nj.nec.com/kindler00interoperability.html.

[19] Ekkart Kindler and Michael Weber. The Petri Net Kernel - An infrastructure for building Petri net tools. *International Journal on Software Tools for Technology Transfer*, 3(4):486–497, 2001. Available from: http://citeseer.nj.nec.com/kindler99petri.html.

[20] Hans W. Nissen and Matthias Jarke. Repository support for multi-perspective requirements engineering. *Information Systems*, 24(2):131–158, 1999. Available from: http://citeseer.nj.nec.com/nissen99repository.html.

[21] Pallas Athena. *Protos User Manual*. Pallas Athena BV, Plasmolen, The Netherlands, 1997.

[22] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[23] Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Sren Christensen, and Kurt Jensen. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In van der Aalst and Best [27].

[24] Rockwell Software. Arena, 2004. http://www.software.rockwell.com/arenasimulation/. Available from: `http://www.software.rockwell.com/arenasimulation/`.

[25] Christian Stehno and Michael Weber. pnml2svg, 2003. see http://parsys.informatik.uni-oldenburg.de/ pep/pnml/. Available from: `http://parsys.informatik.uni-oldenburg.de/~pep/pnml/`.

[26] Avoka Technologies. ProVision Modeling Suite, 2003. www.avoka.com. Available from: `http://www.avoka.com`.

[27] Wil M. P. van der Aalst and Eike Best, editors. *Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands, June 23-27, 2003, Proceedings*, volume 2679 of *Lecture Notes in Computer Science*. Springer, 2003.

[28] Wil M. P. van der Aalst and Kees M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002. Available from: `http://citeseer.nj.nec.com/vanderaalst02workflow.html`.

[29] J.M.E.M. van der Werf and R. Post. *EPNML 1.1 - an XML format voor Petri nets*. TU Eindhoven, November 2003. Available from: `http://www.petriweb.org/specs/pnmldef11.pdf`.

[30] Workflow Management Coalition. Workflow Process Definition Interface – XML Process Definition Languaget, October 2002. Document Number WMFC-TC-1025 – 1.0 final draft.