Markus Nüttgens, Frank J. Rump (Hrsg.)

EPK 2004

Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten

3. Workshop der Gesellschaft für Informatik e.V. (GI)
und Treffen ihres Arbeitkreises „Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)“

06. Oktober 2004 in Luxemburg

Proceedings

Veranstalter

veranstaltet vom GI-Arbeitskreis "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)" der GI-Fachgruppe WI-MobIS (FB-WI) in Kopperation mit der GI-Fachgruppe EMISA (FB-DBIS) und der GI-Fachgruppe Petrinetze (FB-GInf).

Prof. Dr. Markus Nüttgens (Sprecher)
Hamburger Universität für Wirtschaft und Politik
Email: nuettgens@hwp-hamburg.de

Prof. Dr. Frank J. Rump (stellv. Sprecher)
FH Oldenburg/Ostfriesland/Wilhelmshaven
Email: rump@informatik-emden.de

EPK 2004 / Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. Hrsg.: Markus Nüttgens, Frank J. Rump. – Luxemburg 2004

© Gesellschaft für Informatik, Bonn 2004

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

EPML2SVG - Generating Websites from EPML Processes

Jan Mendling, Alberto Brabenetz, and Gustaf Neumann
Department of Information Systems and New Media
Vienna University of Economics and BA
firstname.lastname@wu-wien.ac.at

Abstract: This paper presents an approach to map EPC business process models available in EPML to Scalable Vector Graphics (SVG) and websites. This mapping has been implemented as an XSLT transformation program called EPML2SVG. Such a transformation may serve as a reference for visual tools that use EPML. Furthermore, business process models available in SVG can be used in web presentations and they can be viewed without buying a licence of a business process modelling tool. Moreover, EPML2SVG leverages EPML as an interchange format for EPCs. Websites generated by EPML2SVG contain SVG graphics for each EPC model and an HTML navigation structure based on links to each process models. We discuss design decisions of the program and illustrate the generated SVG by an example.

1 Introduction

Event-Driven Process Chains (EPC) are a wide-spread technique for modelling business processes. EPML (EPC Markup Language) is a tool-neutral interchange format for business process models represented as EPCs. It has been designed to serve as an import and export format as well as an intermediary format for transformations between heterogenous business process modelling tools [MN02, MN03, MN04]. It is related to other standardization efforts like OMG's XML Metadata Interchange (XMI) [Ob03], the Petri Net Markup Language (PNML) [BCvH 03], or WfMC's XML Process Definition Language (XPDL) [Wo02] that all aim to provide a tool-neutral interchange format.

In this paper we address the task of reusing business process models available in EPML and generating websites including Scalable Vector Graphics (SVG) [FJJ03] files from them. There are different motivations for this work. First, a transformation to SVG can serve as a reference model standardizing the visual representation of EPML documents (see [BCvH 03]). Second, SVG graphics of EPC business process models can be integrated into web-based trainings for e.g. SAP courses (see [As02]). Third, an SVG representation can be used within a website to communicate business process models to members of a company without buying a licence of a business process modelling tool like e.g. ARIS Toolset. Finally, the provision of tools is important to leverage EPML as an interchange format for EPCs.

EPML has been designed following the design principles of readability, extensibility, tool

orientation, and syntactical correctness [MN03]. In the context of this work the EPML design principles of readability and tool orientation are especially important. First, readability provides for an easy development of programs that transform EPML code. Second, tool orientation in terms of position information of EPC symbols allows to build graphics from models available in EPML. This paper will present a transformation program that generates websites from EPML files. It is called EPML2SVG. This program is written in XSLT [Cl99], a scripting language specialized on generating new XML documents (i.e. HTML and SVG in this case) from other XML input files (i.e. EPML). Each EPC business process model will be used to build a separate SVG file. The rest of the paper is structured as follows. Section 2 will give an overview of SVG and its principles and advantages. In Section 3 we will present the generation of websites from EPML files. Section 4 will present related research. Finally, Section 5 will give some concluding remarks and an outlook on future research.

2 Scalable Vector Graphics

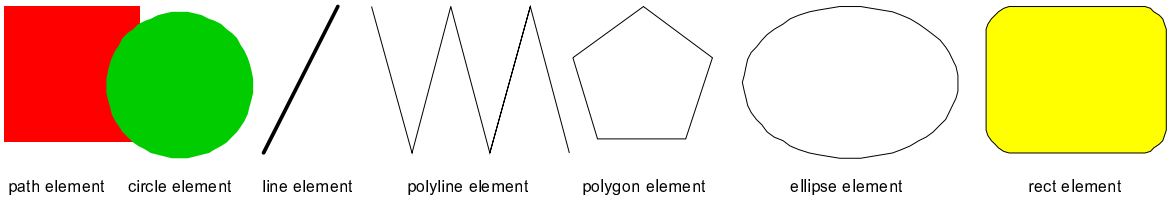
SVG is an XML-based format for two-dimensional vector graphics standardized by the World Wide Web Consortium [FJJ03]. It can contain graphic elements, images, and text. SVG defines a graphic as a set of geometric vector objects that can be grouped, styled, transformed, and composed (see [DHH02] for details). Vector representation provides for a loss-free scaling and storage efficiency. Furthermore, SVG offers dynamic effects like e.g. changing colors, moving text, or zooming. As SVG is based on XML it can be easily manipulated via a simple text editor. Moreover, SVG documents can be combined with other XML and programming technologies. On the one hand, SVG can be included in HTML documents [Pe02] and on the other hand it can include e.g. XLink elements [DMO01] to represent hyperlinks from graphic objects to web resources. Furthermore, text included in SVG graphics can be retrieved and indexed by search engines. A shortcoming is that SVG is not compressed like other image formats[1], because it is text-based. Yet, this is not a problem within the context of the work presented here.

SVG uses the so-called *painter's model* as its rendering concept (see [DHH02]). This term refers to the way an artist paints an oil painting. SVG paints the rendering elements in sequential order. It starts with the first element of the SVG document, let's say a red rectangle, and draws it on the area indicated by its coordinates. Second, a green circle is drawn whose area partially overlaps with that of the rectangle. According to the painter's model, the current element (the green circle) is drawn on top of the already existing drawing. This implies that the rectangle is partially obscured by the circle, unless semi-transparency has been defined for the circle. After that, the next element is drawn, and so on. When the circle has an edge, its area is drawn first, and the edge afterwards. This sequential drawing of elements on top of the current drawing constitutes the painter's model employed by SVG.

Each SVG document has an `svg` root element that includes a `width` and a `height` attribute defining the size of the graphic. The top left point represents the origin of SVG's

[1] For an overview of design criteria for multimedia interchange formats see [Ko92]

coordinate system. The `svg` element may nest multiple graphic elements including `rect`, `circle`, `ellipse`, `line`, `polygon`, `polyline`, `image`, `path`, `text`, and `use`. In the following we will sketch some of their characteristic. For a good introduction to SVG in general refer to [DHH02].




```
<svg width="900" height="300">
   <path d="M50,50 L140,50 L140,140, L50,140Z" fill="red"/>
   <circle cx="170" cy="100" r="50"  fill="green"/>
   <line x1="225" y1="150" x2="275" y2="50" stroke="black" stroke-width="5"  fill="none"/>
   <polyline points="300,50 325,150 350,50 375,150 400,50 425,150" stroke="black" fill="none"/>
   <polygon points=" 480,50 527,84 509,140 450,140 432,84" stroke="black" fill="none"/>
   <ellipse cx="620" cy="100" rx="72" ry="50" stroke="black" fill="none"/>
   <rect x="710" y="50" rx="20" ry="20" width="120" height="100" stroke="black" fill="yellow"/>
   <text x="50" y="180">path element</text>
   <text x="135" y="180">circle element</text>
   <text x="225" y="180">line element</text>
   <text x="320" y="180">polyline element</text>
   <text x="440" y="180">polygon element</text>
   <text x="580" y="180">ellipse element</text>
   <text x="735" y="180">rect element</text>
</svg>
```

Figure 1: Basic elements of SVG.

The most general graphic element of SVG is `path`. In Figure 1 the `path` element is used to draw a red rectangle. The `d` attribute defines the path by giving commands and references to positions. The `M` command moves the current position to the specified (x,y) coordinates. The `L` command draws a line from the current position to the specified position. `Z` can be used to close the path just like in the red rectangle example. See [DHH02] for further commands including cubic bezier curves. Basic shapes like `rect` or `ellipse` are simple shorthands for equivalent paths. Figure 1 illustrates how these shapes are defined: a `circle` by its center coordinates and radius, a `line` by its first and second point coordinates, `polyline` and `polygon` by a sequence of positions, an `ellipse` by its center coordinates and two radii, and a `rect` by its top left corner, width and height, and optionally the radii of elliptic arcs rounding the corners. Furthermore, text can be drawn by defining `text` elements (see Figure 1). The style of these shapes and text elements can be specified by various attributes. These include the definition of a fill color (`fill`), definition of a stroke color (`stroke`), or definition of the stroke width (`stroke-width`). For a complete list of these style attributes see the specification [FJJ03].

Furthermore, SVG permits images to be included using the `image` element. This is similar to the way as images are included in HTML. All elements can be grouped via a `g` element. The whole group can be translated, scaled, rotated, or skewed via a transform command. Finally, the `use` element draws a previously defined `symbol` on a specified positions on the SVG graphic. The SVG viewer internally replaces the `use` by a group element that contains the elements defined in the `symbol` element. The following section will discuss how SVG can be used to render EPC business process models defined in EPML.

3 From EPML to Web Presentations

This section introduces the EPML2SVG program implemented in XSLT. We will first give a short introduction to EPML. Then we discuss architectural alternatives to make EPML business process models available as websites. Afterwards, we present the mapping from EPML to websites implemented in the EPML2SVG program. Finally, we discuss the program. In case of potential confusion of names we use namespace prefixes to identify `svg` and `epml` elements and attributes.

3.1 EPML in Brief

EPML is an XML-based tool-neutral interchange format for EPC business process models [MN02, MN03, MN04]. The `epml` element is the root of every EPML file. It contains among others a `directory` element that can nest further directories and `epc` models. Each of these models is identified by a unique `EpcId` and a `name` attribute. An `epc` element is a container for multiple control flow elements like `event`, `function`, `processInterface`, as well as `and`, `or`, and `xor` connectors, and multiple control flow `arc` elements. Each of these elements is identified by a unique `Id` attribute and a `name` element. The `function` and the `processInterface` element may include `ToProcess` elements. The latter has a `LinkToEpcId` attribute representing a logical pointer to a subprocess of a function or to a subsequent process of a process interface. Each `arc` element has a `flow` element whose `source` and `target` attributes represent the source and the target of the control flow arc. All EPC elements may have a `graphics` element. This element may contain `position`, `fill` (not applicable for arcs), `line`, and `font` visualization information. For control flow elements the `position` element specifies the `x` and the `y` position of the top left corner of a bounding box. Its size is indicated via the attributes `width` and `height`. Control flow arcs may have multiple `position` elements, each representing a point of a polyline. In the most simple case there are two position elements to represent the start point and the end point of the arc. For further details and further syntax elements of EPML we refer to [MN04].

3.2 Static versus Dynamic

Basically, there are two ways to generate websites from EPML files: static and dynamic. For a *static* website architecture all HTML and SVG files are generated in one go. Afterwards, these files can be published in a web directory. This approach has the advantage that it is easy to implement. Furthermore, requesting files from a static website provides a good performance. On the other hand, chances to the original EPML file will require a new run for generating the website. In a *dynamic* setting websites are generated from the original EPML file on request. That has the advantage that chances to the EPML file are immediately available. Yet, generating websites on the fly requires more computing time.

The EPML2SVG program provides for a generation of static websites. In particular, two different kinds of files are generated: HTML files for navigation through the EPC business process models, and SVG files for visualization of individual EPCs (see Figure 2). The EPML2SVG program arranges both kinds of files in an HTML frameset. Each EPML file maps to several HTML files. For one EPML file an HTML frameset is generated displaying a header frame at the top, a navigation frame at the left, and a welcome frame at the right. The navigation frame lists all EPC models of the EPML file and provides hyperlinks to corresponding SVG files to be displayed in the right frame. Furthermore, each `epml:epc` element maps to a separate SVG file whose name is built from its `epml:id` attribute. This naming convention is also used to generate the corresponding hyperlinks.

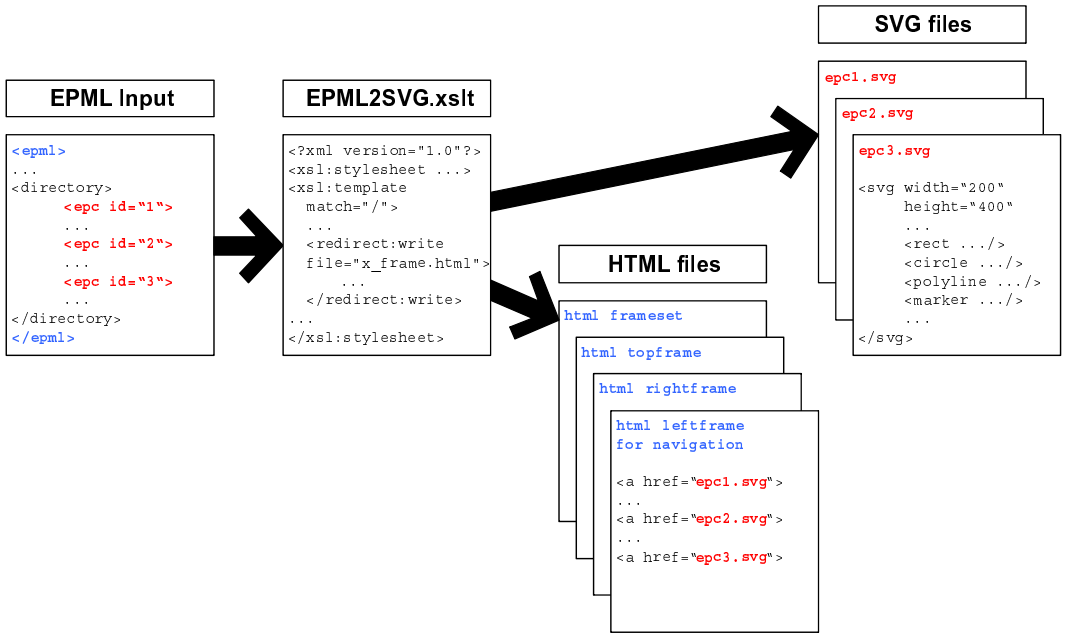
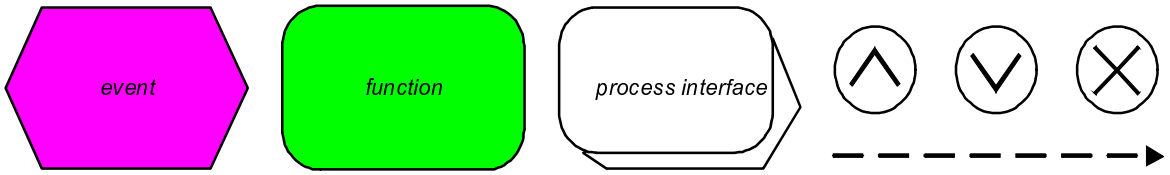


Figure 2: Files generated by the EPML2SVG program

3.3 Mapping from EPML EPCs to SVG

In general, there are two design choices to be taken concerning the representation of SVG elements generated from EPML files: path versus shape, and group versus symbol. The *first choice* is to be made between a generic representation using the `svg:path` element and a shorthand representation via basic shapes like e.g. `svg:rect`. We have decided to use basic shapes whenever possible because it avoids calculation in the transformation program. Consider the `epml:function` element. It is graphically described by its top left corner, and its height and width. This matches exactly the representation of a `svg:rect` element. For transforming it to a `svg:path` all corner coordinates would have to be calculated. The *second choice* is to be made between a group (`svg:g`) and a `svg:symbol` representation. In a group representation each element of an EPC is mapped to a group of SVG elements; e.g. an `epml:function` maps to a group containing a rectangle and a

text label. The symbol representation is applicable for recurring complex sets of graphic elements. They can be defined in an `svg:symbol` which is referenced by a `svg:use` for each position where the symbol is to be displayed. As symbols in SVG cannot be parametrized for different text labels [DHH02], we chose for mapping each element of an EPC to a group of SVG graphic elements.




```
<svg height="80" width="540">
<g id="1" epc="event">
   <polyline transform="scale(0.4)" stroke-width="2" fill="#FF00FF" stroke="black"
      points="350,125 387.5,50 562.5,50 600,125 562.5,200 387.5,200 350,125"/>
   <text transform="scale(0.4)" fill="black" font-style="italic" font-size="12pt" y="125" x="400">event</text>
</g>
<g id="2" epc="function">
   <rect transform="scale(0.4)" fill="#00FF00" stroke="black" stroke-width="2"
      height="150" width="250" ry="20" rx="20" y="50" x="50"/>
   <text transform="scale(0.4)" fill="black" font-style="italic" font-size="12pt" y="125" x="100">function</text>
</g>
<g id="3" epc="processInterface">
   <rect transform="scale(0.4)" fill="none" stroke="black" stroke-width="2"
      height="135" width="222.5" ry="20" rx="20" y="50" x="650"/>
   <polyline transform="scale(0.4)" stroke-width="2" fill="none" stroke="black"
      points="872.5,80 900,143 862.5,200 700,200 675,185"/>
   <text transform="scale(0.4)" fill="black" font-style="italic" font-size="12pt" y="125" x="700">process interface</text>
</g>
<g id="4" epc="and">
   <circle transform="scale(0.4)" fill="none" stroke="black" stroke-width="2"
      height="100" width="100" r="50" cy="100" cx="1000"/>
   <polyline transform="scale(0.4) translate(0,7)" fill="none" stroke-width="5" stroke="black"
      points="970,110 1000,70 1030,110"/>
</g>
<g id="5" epc="or">
   <circle transform="scale(0.4)" fill="none" stroke="black" stroke-width="2"
      height="100" width="100" r="50" cy="100" cx="1150"/>
   <polyline transform="scale(0.4)" fill="none" stroke-width="5" stroke="black"
      points="1120,90 1150,130 1180,90"/>
</g>
<g id="6" epc="xor">
   <circle transform="scale(0.4)" fill="none" stroke="black" stroke-width="2"
      height="100" width="100" r="50" cy="100" cx="1300"/>
   <line transform="scale(0.4)" style="stroke-width:5;stroke:black" y2="130" x2="1330" y1="70" x1="1270"/>
   <line transform="scale(0.4)" style="stroke-width:5;stroke:black" y2="70" x2="1330" y1="130" x1="1270"/>
</g><g id="7" epc="arc">
   <polyline points="950,175 1350,175 " stroke="black" stroke-dasharray="0.25% 0.25% 0.25%"
      fill="none" transform="scale(0.4)" style="marker-end:url(#arrow)"/>
</g>
</svg>
```

Figure 3: EPC elements and SVG code generated by EPML2SVG.

Figure 3 displays EPC icons and their SVG representation generated by EPML2SVG. For each `function`, `event`, `processInterface`, `and`, `or`, `xor`, and `arc` element a `svg:g` group is generated containing icon-specific shape elements. The size and the position of the shape group is determined by the `epml:graphics` subelements of the EPC elements. In order to easily relate SVG shapes to their EPML source code, the EPML2SVG program writes the `epml:id` of the source element to the respective `svg:g` element and attaches an `svg:epc` attribute[2] stating its EPC element type. Accordingly, a `function` with `epml:id="2"` maps to a group element with a `svg:id="2"` and a `svg:epc="function"` attribute (see Figure 3).

[2]Note that this `epc` attribute is syntactically written to the `svg` namespace. Yet, of course, the SVG specification does not define an `epc` attribute for groups. The SVG viewer will ignore it. Accordingly, it is actually a comment.

In the following, we give only some short explanations on the mapping. For further details we refer to the EPML2SVG program[3]. In EPML functions and process interfaces can contain `toProcess` subelements whose `linkToEpcId` attribute represents a reference to another process. These EPML elements are mapped to XLink hyperlinks in the SVG graphic. Such hyperlink allow to open the referenced process graphic via a simple click. Furthermore, `svg:polyline` elements generated from arcs reference a `svg:marker` element. It permits to draw arrow heads for a line by simply referring to a path defined by the marker. For details see the specification [FJJ03]. The rest of the transformation are relatively straight forwards mappings to SVG shapes.

Figure 4 gives a screenshot of an HTML frameset generated by the EPML2SVG program. There are three EPC process models included in the navigation list at the left-hand side. Each contains a hyperlink to an SVG file of the corresponding EPC business process model. The right frame displays the SVG file of such an EPC model.

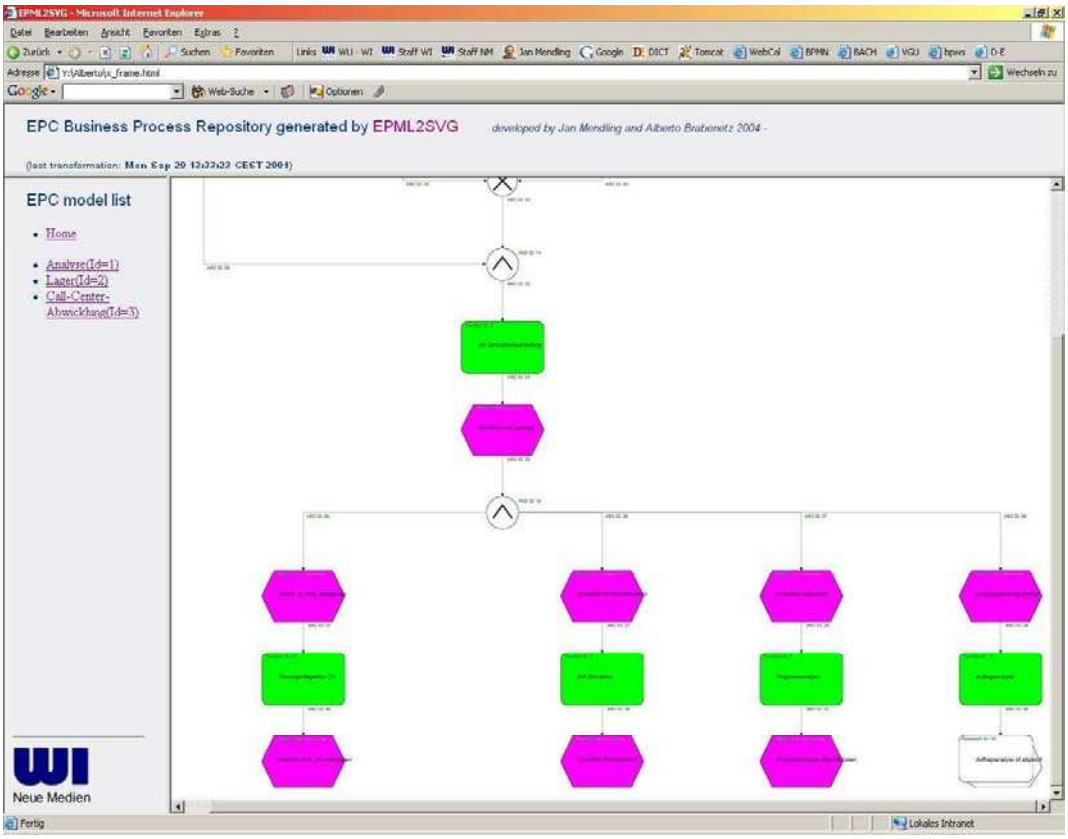


Figure 4: Screenshot of a HTML Frameset generated by the EPML2SVG program.

[3]Information on how to obtain the EPML2SVG program is available on the EPML website (http://wi.wu-wien.ac.at/~mendling/EPML).

3.4 Discussion

Although most of the mappings can easily be implemented using XSLT, there are some challenges regarding texts. SVG does not understand newline characters in text elements. This means that each time there is a newline character in the text label of e.g. an `epml:function` a separate text span at a position below the first part of the string has to be added. In XSLT this has to be programmed by the help of a recursive template that processes the text label as long as there are no more newline characters[4]. If SVG were able to interpret newline characters, the transformation would be much more straight forward.

Another challenge is the automatic layering of process graphs, in particular EPC business process models. The EPML2SVG program is not able to calculate positions when there are no `epml:graphics` element included. Layout and drawing of graphs is a non-trivial problem and subject of a whole research community (see e.g. [BETT99]). The automatic calculation of positions is among others needed when an EPML file has been generated from a language like BPEL4WS [ACD 03] that does not include any graphical information. We expect to address this challenge in the future.

4 Related Research

There have been some academic research projects dealing with transformation of XML business process model data into graphics. In the context of Petri Net Markup Language (PNML) [BCvH 03] a mapping to SVG has been refined. This mapping is implemented in the PNML2SVG transformation script available at [St03a]. This mapping has been defined to provide for a precise description of the graphical presentation of a PNML document. PNML compliant visual tools can check their conformance using this reference model of PNML visualization. A project at TU Chemnitz uses the ARIS Markup Language (AML), the XML interchange format of ARIS Toolset, to generate SVG graphics of business process models [As02]. The purpose of this project is to make EPC business process models available in web-based SAP trainings. In contrast to EPML2SVG it transforms only one EPC model to one SVG file. Beyond that, EPML2SVG generates a whole HTML navigation frameset. The AML Interpreter project at Rostock University also takes AML as an input format [St03b]. The AML Interpreter displays EPC business process models available in AML and offers a transformation to standard graphic formats GIF and JPEG. The motivation of this project is to allow visualization of ARIS process models without having to install ARIS Toolset. This may safe licence cost. There are usually many employees in a company who only sometimes need to view process models. For them a simple viewer like AML Interpreter would be sufficient and no ARIS Toolset installation required. Yet, GIF and JPEG are not vector graphics, but image formats. In contrast, the SVG files generated by EPML2SVG provide for a loss-free scaling of the EPC business process models.

[4] For details on the recursive template solution see the EPML2SVG program available from the EPML website (http://wi.wu-wien.ac.at/~mendling/EPML).

5 Conclusion and Future Work

In this paper we have presented the EPML2SVG transformation program written in XSLT that generates websites including SVG graphic files from EPML documents. This transformation can be used as a reference model when implementing graphical EPC tools. Furthermore, the generated graphics can be used in web training courses on business process-related topics. Moreover, they can be used to publish process models on the intranet of a company without having to buy licences for expensive business process modelling tools. Finally, EPML2SVG leverages EPML as an interchange format for EPCs.

Although the exchange of graphical model information has been recognized as an important issue of interchange model data (see e.g. [MN04, BCvH 03, Ob03]) there is currently no consensus on how representing graphic information in such interchange formats. It will be an interesting topic of future research to identify the pros and cons of directly including SVG code in interchange formats. Furthermore, automatic calculation of positions for EPC business process model elements is a non-trivial task. We expect to address this problem in the context of EPML in the future.

Acknowledgement. The authors would like to thank the anonymous reviewers for there comments on an earlier version of this paper, which helped to improve the presentation of the ideas.

References

[ACD 03] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S.: Business Process Execution Language for Web Services, Version 1.1. Specification. BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems. 2003.

[As02] Assmann, M.: *Generierung webfähiger EPK-Grafiken mittels ARIS, XML und SVG*. TU Chemnitz. http://www-user.tu-chemnitz.de/~maas/projectpage/index2.html. 04.12.2002.

[BCvH 03] Billington, J., Christensen, S., van Hee, K. E., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., and Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In: W. M. P. van der Aalst and E. Best (eds.), *Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands*. volume 2679 of *Lecture Notes in Computer Science*. pages 483–505. 2003.

[BETT99] Battista, G. D., Eades, P., Tamassia, R., and Tollis, I. G.: *Graph Drawing - Algorithms for the Visualization of Graphs*. Prentice Hall. 1999.

[Cl99] Clark, J.: XSL Transformations (XSLT) Version 1.0. W3C Recommendation 16 November. World Wide Web Consortium. 1999.

[DHH02] Duce, D., Herman, I., and Hopgood, F.: SVG Tutorial. In: *Proceedings of the WWW2002 Conference, Hawaii, USA*. http://www.w3.org/2002/Talks/www2002-svgtut-ih/hwtut.pdf. May 2002.

[DMO01] DeRose, S., Maler, E., and Orchard, D.: XML Linking Language (XLink) Version 1.0. W3C Recommendation 27 June 2001. World Wide Web Consortium. 2001.

[FJJ03] Ferraiolo, J., Jun, F., and Jackson, D.: Scalable Vector Graphics (SVG) 1.1. W3C Recommendation 14 January 2003. World Wide Web Consortium. 2003.

[Ko92] Koegel, J. F.: On the Design of Multimedia Interchange Formats. In: *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*. pages 262–271. 1992.

[MN02] Mendling, J. and Nüttgens, M.: Event-Driven-Process-Chain-Markup-Language (EPML): Anforderungen zur Definition eines XML-Schemas für Ereignisgesteuerte Prozessketten (EPK). In: M. Nüttgens and F. J. Rump (eds.), *Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), Trier, Germany*. pages 87–93. 2002.

[MN03] Mendling, J. and Nüttgens, M.: XML-basierte Geschäftsprozessmodellierung. In: W. Uhr, W. Esswein and E. Schoop (eds.), *Proc. of Wirtschaftsinformatik 2003 / Band II, Dresden, Germany*. pages 161 –180. 2003.

[MN04] Mendling, J. and Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In: J. Mendling and M. Nüttgens (eds.), *Proc. of the 1st GI-Workshop XML4BPM - XML Interchange Formats for Business Process Management, Marburg, Germany, March, 2004*. pages 61–79. 2004.

[Ob03] Object Management Group: XML Metadata Interchange (XMI). Specification, Version 2.0. Object Management Group. May 2003.

[Pe02] Pemberton et al., S.: XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). W3C Recommendation 26 January 2000, revised 1 August 2002. World Wide Web Consortium. 2002.

[St03a] Stehno, C.: *PNML2SVG*. Universität Oldenburg. http://parsys.informatik.uni-oldenburg.de/~pep/pnml. 28.02.2003.

[St03b] Stoy, M.: *AML Interpreter*. Universität Rostock. http://www.informatik.uni-rostock.de/~masto/aris/. 01.03.2003.

[Wo02] Workflow Management Coalition: Workflow Process Definition Interface – XML Process Definition Language. Document Number WFMC-TC-1025, October 25, 2002, Version 1.0. Workflow Management Coalition. 2002.