Separation of Style and Content with XML in an Interchange Format for High-level Petri Nets

Thomas Mailund
Department of Computer Science
University of Aarhus
e-mail: mailund@daimi.au.dk

Kjeld H. Mortensen
Department of Computer Science
University of Aarhus
e-mail: khm@daimi.au.dk

Abstract

Style sheets have been proposed by the World Wide Web Consortium (W3C) as a means for separating presentation and content in World Wide Web documents, and they are now widely used and generally popular. In this paper we use the same idea for an XML interchange format proposal for high-level Petri nets. There are several benefits of using this design principle: It is easier to exchange content-only for non-graphics tools, and alternative styles can conveniently be replaced to make a new graphical appearance. The ideas presented in this paper are illustrated by means of examples.

1 Introduction

Since the founding of W3C in 1994, they have promoted the use of style sheets on the Web. Several recommendations have been made such as CSS1 and CSS2 [2, 4, 5], and XSL [3] is under preparation. The main idea is essentially to separate presentation and content. Often it is sufficient to have only a few small centralised style sheets in order to control the rendering of thousands of pages in a Web site. In this paper we expore this generally accepted design technique in order to separate presentation from content for Petri net diagrams.

CSS is currently the prevailing technology supported by both Netscape Navigator and Microsoft IE and is therefore a well-tested technology. The alternative is XSL [3]. The main difference between CSS and XSL is that CSS is used to annotate an existing document tree (cf. DOM [6]), while XSL is used in the context of transforming one tree into another.

2 Interchange Format with Style Sheets

Most high-level Petri net tools can agree on the underlying mathematical model of a Petri net: a set of places, a set of transitions, a set of arcs connecting places and transitions, and some annotations describing data types and data [1]. It is quite another matter when it comes to the graphical layout of nets. Different tools have different graphical attributes that can be associated with the net elements.

To be able to interchange models between different tools, we would like a minimal format containing only the information found in the standard for high-level nets, and a more flexible format for describing graphical attributes. The latter should be extendable to include tool-specific graphical information.

2.1 Separating Content from Style

Style sheets are used on WWW to separate typesetting of HTML from the actual HTML text. We propose a similar scheme for separating the graphical layout of nets from the net structure.

We use a "core" description, which describes the mathematical structure of the net, and a style sheet describing graphical attributes of nodes and arcs. To these two descriptions we add a layout description to the format. The layout description contains information about coordinates of nodes and the bend points or bezier curve of arcs. This information is not relevant for tools with no graphics, and thus should not be part of the core description. Nor does it belong in the style sheet, since layout describes information about specific elements and not classes of elements. If we added layout information to style sheets it would not be possible to share styles between several models, which is definitely something that we want to be able to do.

There are two ways of using style sheets. The first approach, taken by CSS, is setting values for graphical attributes for elements, depending on the type of elements and possibly their context. This way, a tool can load a net and render it by looking up graphical attributes in the style sheet. This is shown in Figure 1.

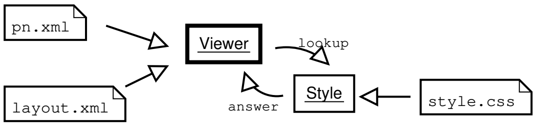

Figure 1: *Architecture of an XML/Style Sheet viewer.*

The second approach is tree transformation, which is the one taken by XSL [3, 9]. In this approach the style sheet describes a set of transformation rules that enables tools to transform one tree into another. This can be used to translate an interchange format description (XML file or DOM object) into a tool specific description. This approach is shown in Figure 2. This approach is the most flexible and most powerful, but it has certain drawbacks. Since the style sheet provides a transformation from one format to another, tools using the same style sheets will need to to agree on the target format. This can limit the interchange of style sheets, though not the interchange of the net descriptions.

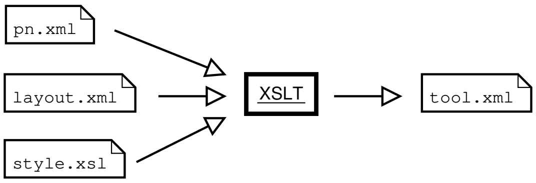

Figure 2: *Interchange format used with an XML/XSL transformation.*

In this paper, for simplicity, we look only at the first approach, CSS. Although CSS is primarly targeted for text, and as a consequence is less well suited for our purpuses, we will use CSS-syntax to describe styles in order to make the presentation of ideas easier to understand without introducing *ad hoc* notions.

2.2 Examples of the Technique

We will examine the use of the XML/Style Sheet technique for Petri nets through a small example: a flexible manufacturing system. Manufacturing systems consist of the three classes of elements: materials, machines, and transportation. Materials enter the system, are then processed in a number of manufacturing cells by the machines in the cells, and are transported between the cells by the transportation system.

If we abstract away from the actual processing taking place in the individual machines, the different cells can all be modeled by a generic machine model. In Figure 3 such a model is shown. Material enters the cell on the Input place. When a machine becomes available the item is taken, processed, and put on the place Output.

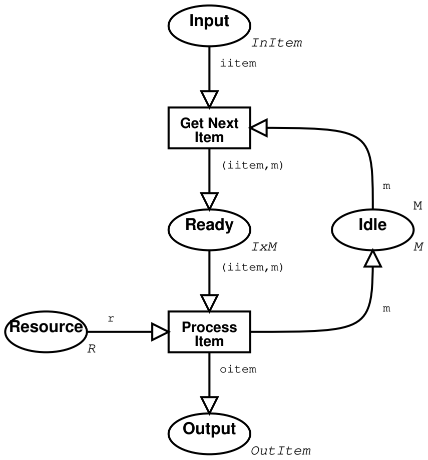

Figure 3: *CPN model of a machine which can be used in a manufacturing cell.*

The XML file describing the mathematical structure of the machine net will contain information about the places and their types and initial markings, the transitions and their guards, and the arcs and their expressions. To be able to refer to net elements, we associate a name with each. The name of an element is not strictly part of the mathematical model of a net, but nevertheless an important part of the model and should therefore be part of the core description of the net rather than part of a layout or style description. The XML file describing the layout will contain information about the placement of the nodes and bend points of the arcs.

The name we use has no special meaning in XML terms, so to refer to a specific element in a given XML document we use the ID attribute of the element. An ID attribute has special meaning in XML documents where it is used to uniquely identify an element. We can refer to an element from inside the document using IDREF attributes, and from outside the document using XPointers [8].

Assume that the mathematical structure of a net and the layout of the same net are separated into two documents. In the document describing the net structure, we can use ID attributes to name the places and transitions, and we can associate arcs with their endpoints using IDREFs. In the layout document we can associate coordinates with nodes in the first document using XPointers.

The valid IDs are restricted and cannot contain e.g. white spaces. It would not be reasonable to put this restriction on names, so we need both a name and an ID for elements, even though the IDs uniquely identify the elements.

Parts of the description of the mathematical model of the net in Figure 3 and the corresponding layout description is shown in Figure 4.

Here, in the layout description, the only data associated with arcs is bend points. The endpoints can be derived from the coordinates of the nodes that the arc connects. If we allow styles to specify the dimension of nodes, it would not make sense to keep information about end points in the layout description. An end point that is set to point at the edge of a node under one style will no longer point at the edge of the node if we change the dimensions of the node, as shown in Figure 5.

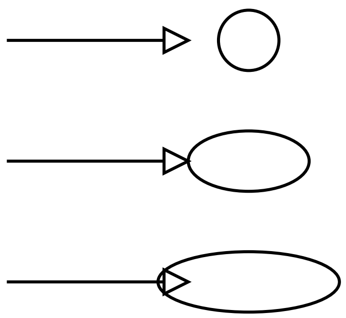

Figure 5: *A fixed end point and a node with changing dimensions.*

With the net description from Figure 4, we can supply a style sheet that describes how places, transitions, and arcs should be shown. For example that places should be shown as ellipses and transitions as boxes, and the name of a transition should be displayed with a Courier 11pt font, while the name of a place should be displayed with a Courier 12pt. In CSS syntax this would look like Figure 6.

There is one problem though, there is nothing in the description of the net that lets us distinguish between different classes of elements. We cannot display places modelling states of machines differently from places modelling buffered items. We have no way of

```
place { shape: ellipse; }
place name { font-size: 12pt; }

transition { shape: box; }
transition name { font-size: 11pt; }
```

Figure 6: *Simple style sheet in CSS syntax. The shape of places and transitions is specified, as well as the font size of name labels. The rest is left as defaults. The font size of name labels is set different for place names and for transition names.*

identifying net elements related to, say, material flow. To be able to do that, we need to attach additional information to the net description. We can do this by adding "class" attributes to the tags in the net description, as shown in Figure 7.

Notice that we are not attaching any information about the appearance or layout of the net elements. We simply add a classification. Such classifications will, in nature, be model dependent. They do not add any information to the mathematical model, rather they provide an interpretation of the particular elements in the given model. The classification of elements could therefore be separated from the description of the net structure, as is the layout. However, we believe that this information belongs in the core description of the model, since, in real life, the classification of the elements in a given model is just as important as the pure mathematical structure.

```
<place id="input" class="material-flow">
  <name>Input</name>
  <type>InItem</type>
</place>

<arc id="a1">
  <placeend idref="input" class="material-flow">
  <transend idref="get-next-item">
  <expr>iitem</expr>
</arc>

<transition id="get-next-item" class="material-flow">
  <name>Get Next Item</name>
</transition>
```

Figure 7: *Entities from the mathematical description of the machine model, extended with class information.*

Once we have associated a class attribute to a net element, we can use this information when displaying the net. If, say, we want to highlight the flow of materials in the manufacturing system model, we could set a style that displayed "material-flow" elements with thicker lines, as shown in CSS syntax in

```
<pn>
  <place ID="input">
    <name>Input</name>
    <type>InItem</type>
  </place>

  <arc ID="a1">
    <placeend IDREF="input">
    <transend IDREF="get-next-item">
    <expr>iitem</expr>
  </arc>

  <transition ID="get-next-item">
    <name>Get Next Item</name>
  </transition>

  <place ID="idle">
    <name>Idle</name>
    <type>M</type>
    <initmark>M</initmark>
  </place>

  <arc ID="a2">
    <placeend IDREF="idle">
    <transend IDREF="get-next-item">
    <expr>m</expr>
  </arc>

  ...

</pn>
```

```
<layout>
  <node ref="machine.pn#input">
    <x>10.0</x>
    <y>0.0</y>
  </node>

  <conn ref="machine.pn#a1"/>
     <!-- no bend points -->

  <node ref="machine.pn#get-next-item">
    <coord>
      <x>10.0</x>
      <y>10.0</y>
    </coord>
  </node>

  <node ref="machine.pn#idle">
    <coord>
      <x>20.0</x>
      <y>20.0</y>
    </coord>
  </node>

  <conn ref="machine.pn#a2">
    <coord>
      <x>20.0</x>
      <y>10.0</y>
    </coord>
  </conn>

  ...

</layout>
```

Figure 4: *The XML description of the mathematical structure of the machine from Figure 3 is shown on the left. This description contains the minimal description needed to describe the mathematical structure of the net. On the right is shown the layout description of the same net. The layout description contains the information needed to render the net on a screen or page.*

Figure 8. The machine model, with this style sheet, is shown in Figure 9.

```
.material-flow { line-thickness: 4pt; }
```

Figure 8: *Style sheet setting line-thickness for all material flow elements.*

2.3 Using Style Sheets with Reusable Modules

Manufacturing systems come in many flavors, from bottling systems to car factories. Still, such diverse systems have much in common, and a generic component, as the machine model, can be reused in many instances of manufacturing system models.

Importing components created for one model into another model can be time consuming. This is true even when the component satisfies all syntactical con-

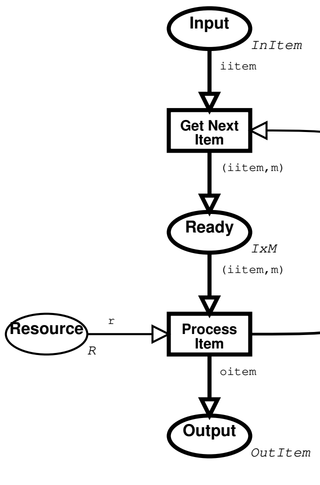

Figure 9: *The machine model rendered with thick-line material flow.*

straints imposed on it from the new environment it is inserted into. The classification of net elements is usually implicit in the graphical attributes of the elements, e.g., in a manufacturing system materials are associated with, say, blue, thick lines, while the transportation is shown with red, dotted lines. The graphical attributes used for classification is usually not the same for different models, even when the models have the same classifications. For example, a fluid bottling system model might use the attributes described above for identifying material flow and transportation system, while a car factory model, which also contains material flow and transportation system elements, uses different attributes to display the two classes. If someone imports a component created for the bottling system model, into the car factory model he will have to change the graphical attributes of the elements, such that they reflect the classification in the factory model rather than the bottling model.

When using style sheets for displaying nets, we get rid of this problem. Then, it is simply a matter of importing a net, created for another model, and displaying it in the style of the current model. This way components from heterogenous models can be combined into a homogeneous looking model simply by rendering the components with a common style sheet. This does require, of course, that the different models agree on the classification, e.g. that, say, material flow is marked with the class attribute "material-flow" in both models.

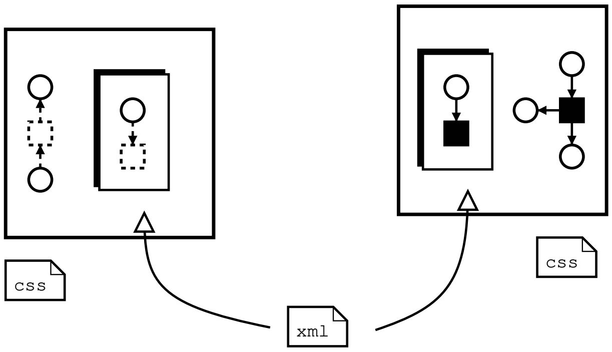

Figure 10: *Using the machine component in different manufacturing system, models, with different style sheets. The imported component is displayed in the same style as the rest of the net.*

3 Concluding Remarks

In this paper we suggest to separate presentation and content for a high-level Petri net interchange format. We used CSS examples to illustrate the main benefits of such a design technique. However CSS is more suitable for presentation of text rather than vector graphics such as Petri nets. Hence as a future activity we propose to design a special purpose style language in the context of the interchange format activity.

References

[1] Jonathan Billington et al. High-Level Petri Nets — Concepts, Definitions and Graphical Notation. Technical report, ISO/IEC JTC1/SC7/WG11 N-3, Oct 1997. Committee Draft 15909.

[2] W3C, http://www.w3.org/Style/CSS/. *Cascading Style Sheets.*

[3] W3C, http://www.w3.org/Style/XSL/. *Extensible Stylesheet Language (XSL).*

[4] W3C, http://www.w3.org/TR/REC-CSS1. *Cascading Style Sheets, level 1, W3C Recommendation*, Dec 1996.

[5] W3C, http://www.w3.org/TR/REC-CSS2. *Cascading Style Sheets, level 2, W3C Recommendation*, May 1998.

[6] W3C, http://www.w3.org/TR/REC-DOM-Level-1. *Document Object Model (DOM) Level 1 Specification, W3C Recommendation*, Oct 1998.

[7] W3C, http://www.w3.org/TR/REC-xml. *Extensible Markup Language (XML) 1.0, W3C Recommendation*, Feb 1998.

[8] W3C, http://www.w3.org/TR/xptr. *XML Pointer Language (XPointer), Working Draft*, Dec 1999.

[9] W3C, http://www.w3.org/TR/xslt. *XSL Transformations (XSLT), W3C Recommendation*, Nov 1999.