Textual Interchange Format for High-level Petri Nets*

Regnar Bang Lyngsø and Thomas Mailund

Computer Science Department
University of Aarhus
DK-8000 Aarhus C
Denmark
E-mail: {rblyngso,mailund}@daimi.aau.dk

Abstract. In this paper a text format for High-level Petri Net (HLPN) diagrams is presented.
The text format is designed to serve as a platform-independent file format for the Design/CPN tool. It is consistent with the forthcoming standard for High-level Petri Nets. The text format may also be seen as our contribution to the development of an open, tool-independent interchange format for High-level Petri nets.
The text format will make it possible to move *Design/CPN* diagrams between all supported hardware platforms and versions. It is also designed to be a bridge to other Petri Net tools, e.g., other analysis tools which the user may want to use with *Design/CPN* diagrams. The proposed text format does not address any standardisation for the inscription language used in the diagram. It is, however, possible to extend the format to incorporate such a standardisation.
The text format is designed for the exchange of *Hierarchical Coloured Petri Nets* but the structure is general enough to cope with other High-level Petri Nets as well.
The text format presented here has been implemented as part of *Design/CPN* version 3.1.

1 Introduction

Design/CPN is a widely used tool within the Petri Net community and has been developed for more than 10 years. The tool has been used in many projects in a broad range of application areas [8].

Design/CPN supports Hierarchical Coloured Petri Nets (CP-nets or CPNs) [7] with complex data types (colour sets) and complex data manipulations (arc expressions and guards) - both specified in the functional programming language CPN ML [3]. It also supports hierarchical CP-nets, i.e. net diagrams that consist of a set of separate modules (subnets) with well-defined interfaces.

* This paper is a slightly modified version of: R.B Lyngs, T Mailund: Textual Interchange Format for High-level Petri Nets In K.Jensen (ed.): Proceedings of the Workshop on Practical Use of Coloured Petri Nets and Design/CPN, Aarhus 1998, Department of Computer Science, University of Aarhus, PB-532, 47-63 (also available via: http://www.daimi.aau.dk/CPnets/workshop98/papers/).

1.1 The Reasons for Choosing a Text Format

Until now *Design/CPN* diagrams have been saved in a binary file format, but there are several reasons why we would prefer to use a text format. The current format is a properitary file format which poorly supports the possibility of making diagrams distributed in multiple files. With the text format it should be possible to add some sort of modularisation system making modularisation of diagrams easier. This is not possible with the current file format of *Design/CPN.*

Furthermore the current file format of *Design/CPN* has certain limitations when it comes to moving a diagram between different hardware platforms of the tool. Specifically a user might not be able to save a diagram on one hardware platform (e.g., the Intel family) and load it on another (e.g., Macintosh). Another point is compatibility between different versions of *Design/CPN.* With the binary file format it is more difficult to add and remove features to the language than it is with a text format.

We also hope to make it easier to work independently on different parts of a diagram, and then later on put the pieces together. By adding merging/replacing facilities and some sort of version-control we hope to make *Design/CPN* more attractive for groups working on larger projects. The text format should also make it possible to create libraries of commonly used net structures.

Another desirable feature of a text format is readability, since this will ease debugging and inspection by users.

The advantages of having a widely accepted interchange format supported by different Petri Net tools are obvious and well known. With n Petri net tools, each with its own diagram representation, it would be necessary to have $n^2 - n$ tools to translate between all pairs of languages. By using an intermediate language it would only be necessary with $2n$. Furthermore, *each tool* would only need *one* translator to and from the interchange format, instead of the $n - 1$ to be able to load and save each of the different formats.

To fulfil these requirements the text format must necessarily be expandable in such a way that it matches the capabilities of any format used by any Petri Net tool i.e. a loaded diagram must look and behave exactly as the one that was saved. The text format described here is a proposal for such a common format.

To sum up the goals of the text format, which would be either difficult or impossible to achieve by further development of the current binary file format:

- Modularisation (on file basis) allowing development of libraries.
- Hardware independency.
- Compatibility between different versions of the tool.
- Readable by humans.
- Ease of further development/enhancements of the tool.
- Easier communication between different tools.

1.2 Design Decisions

From the very beginning it was decided that the text format should be independent of the inscription language. This decision was made for numerous reasons.

First of all we wanted it to be easy to implement the text format in an existing tool as we have done with *Design/CPN.*

The considerations mentioned earlier concerning different tools interchanging diagrams are of course true for the inscription language as well. The same benefits can be obtained by sharing an inscription language. We believe however that the inscription language is a far more integrated part of each tool than the graphical layout, thus translation to and from a common inscription language was considered beyond the scope of this text format.

On the other hand it was feasible to implement known standards for both naming conventions and syntax in order to make it easier for humans to read the description of the diagram. To that end we have chosen to use the terminology presented in the current version of the committee draft of the HLPN standard [1]. This means that the entities known in *Design/CPN* as arc expressions, colour regions and guards are called arc annotations, type region and transition conditions, respectively. Furthermore we chose to use the SGML (*Standard Generalized Markup Language*) ISO standard, which will be described in Sect. 3.

The structure of the text format is chosen to be based on the semantic properties rather than the graphical appearance. It is thus considered more important to know that a certain object is a place than it is to know that the object has the shape of an ellipse.

We want the text format to be both general enough to be used by various different tools and specific enough to save the same information about a diagram as the binary format used so far. Different tools need to add different kinds of information to the text format, and later versions of *Design/CPN* will probably add to the format as well.

On the other hand, each tool will need certain information, e.g. graphical attributes such as line thickness or fill pattern, which has no counterpart in some other tools. A simulator need no information about the graphical layout. Thus we cannot expect to get all the information needed by one tool from text format exported by another tool.

The solution to these seemingly conflicting goals, was found to be a very simple structure which could be easily parsed and subsequently weeded for unwanted information. Furthermore, it should be possible to describe a diagram with a minimum of extra information, leaving most of the graphical layout and some of the semantics to defaults. The default shape of a place is an ellipse, the default of a condition is true, etc. In our implementation, when loading a diagram, unknown parts are ignored and missing information is substituted by defaults.

We have implemented routines for loading and saving the text format in *Design/CPN.* The load routines, however, are almost tool independent, and it should be fairly easy to incorporate this module into other tools. For further detail we refer to [9].

2 An Example Diagram

In this section we will give an example of a small diagram described in the text format. A description of the text format will follow in the next section.

Figure 1 shows a small diagram with two places and one transition. Example 1 shows the text description of the same diagram. The text description is slightly edited from text generated by the tool.[1] It is followed by an explanation.

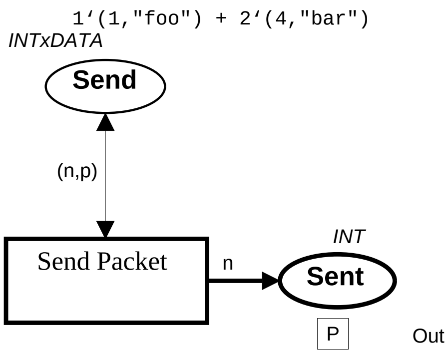

Fig. 1. A simple diagram

Example 1 (Text description of Fig. 1).

```
<!-- File /users/cpn/nets/Sender.int -->
<page id=id10>
  <trans id=id34>
    <text>Send Packet</text>
    <textattr font=Times size=12 just=Centered colour=Lime>
    <lineattr type=Solid thick=2 colour=Yellow>
    <fillattr pattern=White colour=White>
    <posattr x=-644 y=779>
    <box h=134 w=321>
  </trans>

  <place id=id25>
    <text>Send</text>
    <textattr font=Helvetica size=12 just=Centered colour=red bold=TRUE>
    <lineattr type=Solid thick=1 colour=red>
    <fillattr pattern=White colour=red>
    <posattr x=-644 y=1090>
    <ellipse h=84 w=194>

    <type id=id24>
```

[1] A few *Design/CPN* specific attributes have been removed.

```
      <text>INTxDATA</text>
      <textattr font=Helvetica size=9 just=Left colour=red italic=TRUE>
      <posattr x=-722 y=1167>
      <label>
    </type>

    <initmark id=id22>
      <text>1`(1,"foo") + 2`(4,"bar")</text>
      <textattr font=Courier size=9 just=Left colour=red>
      <posattr x=-450 y=1200>
      <label>
    </initmark>
  </place>

  <place id=id7>
    <text>Sent</text>
    <textattr font=Helvetica size=12 just=Centered colour=navy bold=TRUE>
    <port type=out>
    <lineattr type=Solid thick=2 colour=navy>
    <fillattr pattern=White colour=navy>
    <posattr x=-275 y=779>
    <ellipse h=84 w=184>

    <type id=id11>
      <text>INT</text>
      <textattr font=Helvetica size=9 just=Left colour=navy italic=TRUE>
      <posattr x=-260 y=851>
      <label>
    </type>

    <portkreg id=id15>
      <text>P</text>
      <textattr font=Helvetica size=9 just=Centered colour=navy>
      <lineattr type=Solid thick=0 colour=navy>
      <fillattr pattern=None colour=navy>
      <posattr x=-282 y=694>
      <box h=49 w=49>

      <portreg id=id14>
        <text>Out</text>
        <textattr font=Helvetica size=9 just=Left colour=navy>
        <lineattr type=None thick=1 colour=navy>
        <fillattr pattern=None colour=navy>
        <posattr x=-128 y=694>
        <box h=52 w=84>
      </portreg>
    </portkreg>
  </place>

  <arc id=id16 orientation=BOTHDIR>
```

```
71        <placeend idref=id25>
72        <transend idref=id34>
73        <lineattr type=Solid thick=0 colour=navy>
74        <fillattr pattern=Black colour=navy>
75        <seg-conn>
76
77        <annot id=id13>
78          <text>(n,p)</text>
79          <posattr x=-686.15289 y=957.00000>
80          <label>
81          <textattr font=Helvetica size=9 just=Left colour=navy>
82        </annot>
83      </arc>
84
85      <arc id=id19 orientation=TtoP>
86        <placeend idref=id7>
87        <transend idref=id34>
88        <lineattr type=Solid thick=2 colour=navy>
89        <fillattr pattern=Black colour=navy>
90        <seg-conn>
91
92        <annot id=id17>
93          <text>n</text>
94          <textattr font=Helvetica size=9 just=Left colour=navy>
95          <posattr x=-447 y=809>
96          <label>
97        </annot>
98      </arc>
99    </page>
100
```

Line 1 is a comment and has no semantic meaning. Line 2 identifies the start of the page. The page ends in line 99. Lines 3–10 represent the transition *Send Packet.* This is done by means of a `trans` block. The `id` in line 3 is a unique identifier which which it is possible to refer to the transition. Line 4 describes the contents of the text within the transition, i.e. the text "Send Packet". Line 5 describes the format of the text. The font name is Times while the font size is 12. The text is Centered and the colour is Lime. Lines 6–7 describes the borderline and the fill pattern within the transition. The borderline is solid, (i.e. a full-drawn line) with thickness 2 and colour Yellow. The fill pattern is White, which is a non-transparent pattern. The fill colour is White. Line 8 describes the position of the centre of the transition, while line 9 describes the shape of the transition, i.e. box shape together with the width and height of the transition.

Line 12–37 represent the place *Send.* This is done by means of a `place` block. Lines 12–18 play a similar role as lines 3–9 in the `trans` block, but the contents are of course different. It can, e.g., be seen that the text is set in Helvetica, that the text style is boldface (line 14), that the line thickness is 1 (line 15), and that the shape is ellipse instead of box (line 18). The colour of the place is red (lines

14, 15 and 16). It can also be seen that the place has the same x coordinate as the transition which means that the two objects are vertically aligned.

Lines 20–25 describes the type (colour set) of the place. The type block is nested within the place block. This corresponds to the fact that in *Design/CPN* the type is declared in a region under the place object. The text in the type block is smaller (font size 9), left justified and italic (line 22). Line 24 indicates that the type region has the shape of a label (i.e. a box where the size is automatically determined by the text). The initial marking of the first place (lines 27–32) is described similar to the type. The text is however set in Courier font.

The second place of the diagram is described in lines 35–68. It is horizontally aligned to the transition, i.e. they have the same y coordinate (line 9 and 41). It is furthermore shaped as an ellipse. This place is also an output port (line 38), hence it contains a `portkreg` block (port key region), which again contains a `portreg` block (port region). These are *Design/CPN* specific and hence will not be described in any detail.

Lines 70–98 describe the arcs of the diagram. The first arc (lines 70–83) is orientated in both directions (line 70) and connects the *Send* place and the *Send Packet* transition, i.e., the `placeend` (line 71) refers to id *id25* which was the id of the *Send* place (line 12), and the `transend` (line 72) refers to the id of the transition, i.e. *id34*. The shape of the arc is seg-conn, one of the different connector shapes in *Design/CPN* . The line thickness is 0, which is hairline. The annotation (arc inscription) of the arc is found in the `annot` block, lines 77–82.

The second arc (lines 85–98) is very much like the first. There are, however, a few differences. It connects the transition and the *Sent* place. The orientation is from the transition to the place (line 85) and the line thickness is 2 instead of 0 (lines 88 and 73).

3 Description of the Text Format

The format is based on SGML *(Standard Generalized Markup Language)* [6] for several reasons:

SGML

- is designed specifically to support text interchange,
- has a rigorously described structure which makes it easily understood by both computers and humans,
- represents hierarchies,
- sets a standard for communication between different platforms, versions and tools,
- is a well documented, wide spread and accepted standard,
- is the framework upon which numerous well known markup languages are founded, for instance HTML.

3.1 Short Introduction to SGML

In SGML the basic element is the *document element*. It is the root of a tree of elements which as a whole makes up the document's *content*.

As stated in Annex B of the SGML ISO standard [6] a cornerstone of a SGML is the Document Type Definition (DTD for short). The following is a light version of the full DTD which describes the structure of the text format.[2]. It will be followed by an in-depth explanation

```
<!DOCTYPE cpn [

<!-- "Macros" -->
<!ENTITY % cols                "(Black | Maroon | Gray | Olive |
                               Purple | Silver | Red | Teal |
                               Navy | Fucia | Blue | Aqua |
                               Green | Lime | Yellow | White)">
<!ENTITY % arcdirs             "(BOTHDIR | NODIR | PtoT | TtoP">
<!ENTITY % linetypes           "(Solid | Dash | None)">
<!ENTITY % fillpatterns        "(Black | None | Diamonds)">
<!ENTITY % fonts               "(Helvetica | Times | Courier)">
<!ENTITY % textjust            "(Centered | Left | Right)">
<!ENTITY % porttypes           "(In | Out | InOut | General)">

<!-- The basic elements of a net -->
<!--      ELEMENTS      MIN             CONTENT                 -->
<!ELEMENT cpn           O O             page*>
<!ELEMENT page          - -             (arc | place | trans)*>
<!ELEMENT arc           - -             (placeend? &  transend? & lineattr?
                                        & fillattr? & seg-conn? & annot?)>
<!ELEMENT place         - -             (text? & type? & initmark?
                                        & portkreg? & port?)>
<!ELEMENT trans         - -             (text? & textattr? & lineattr? &
                                        fillattr? & posattr? & box?)>

<!--      ELEMENTS      NAME            VALUE           DEFAULT -->
<!ATTLIST page          id              ID              #IMPLIED>
<!ATTLIST arc           id              ID              #IMPLIED
                        orientation     %arcdirs;       #IMPLIED
>
<!ATTLIST place         id              ID              #IMPLIED>
<!ATTLIST trans         id              ID              #IMPLIED>

<!--    Identifiers related to the look of an object and the
        corresponding attributes -->
<!--      ELEMENTS      MIN             CONTENT                 -->
<!ELEMENT posattr       - O             EMPTY>
<!ELEMENT lineattr      - O             EMPTY>
<!ELEMENT fillattr      - O             EMPTY>
```

[2] The full DTD for the *Textual Interchange Format* can be found in the appendix

```
<!ELEMENT textattr      - O             EMPTY>

<!--      ELEMENTS    NAME              VALUE           DEFAULT -->
<!ATTLIST posattr       x               CDATA           #IMPLIED
                        y               CDATA           #IMPLIED
>
<!ATTLIST lineattr      type            %linetypes;
                        thick           CDATA
                        colour          %colours;
>
<!ATTLIST fillattr      pattern         %fillpatterns;
                        colour          %cols;
>
<!ATTLIST textattr      font            %fonts;         #IMPLIED
                        size            NUMBER          #IMPLIED
                        just            %textjust;      #IMPLIED
                        bold            CDATA           #IMPLIED
                        italic          CDATA           #IMPLIED
                        underline       CDATA           #IMPLIED
                        outline         CDATA           #IMPLIED
                        shadow          CDATA           #IMPLIED
                        condense        CDATA           #IMPLIED
                        extend          CDATA           #IMPLIED
                        colour          %colours;       #IMPLIED
>


<!-- Identifers related to the shape of an object -->
<!--      ELEMENTS      MIN             CONTENT                 -->
<!ELEMENT label         - O             EMPTY>
<!ELEMENT seg-conn      - O             EMPTY>
<!ELEMENT box           - O             EMPTY>
<!ELEMENT ellipse       - O             EMPTY>

<!--      ELEMENTS      NAME            VALUE           DEFAULT -->
<!ATTLIST box           x               CDATA          #IMPLIED
                        y               CDATA          #IMPLIED
>
<!ATTLIST ellipse       x               CDATA          #IMPLIED
                        y               CDATA          #IMPLIED
>

<!-- Identifiers related to text -->
<!--      ELEMENTS      MIN             CONTENT                 -->
<!ELEMENT text          - -             CDATA>

<!-- Identifiers related to arcs -->
<!--      ELEMENTS      MIN             CONTENT                 -->
<!ELEMENT placeend      - O             EMPTY>
<!ELEMENT transend      - O             EMPTY>
```

```
90   <!ELEMENT annot         - -             (text? & textattr? & posattr?
91                                           & label?)>
92
93   <!--      ELEMENTS      NAME            VALUE           DEFAULT -->
94   <!ATTLIST placeend      idref           IDREF           #IMPLIED>
95   <!ATTLIST transend      idref           IDREF           #IMPLIED>
96   <!ATTLIST annot         id              ID              #IMPLIED>
97
98   <!-- Identifiers related to places -->
99   <!--      ELEMENTS      MIN             CONTENT                   -->
100  <!ELEMENT type          - -             (text? & textattr? & posattr?
101                                          & label?)>
102  <!ELEMENT initmark      - -             (text? & textattr? & posattr?
103                                          & label?)>
104  <!ELEMENT portkreg      - -             (text? & textattr? & lineattr?
105                                          & fillattr? & posattr? & box?
106                                          & portreg?)>
107  <!ELEMENT portreg       - -             (text? & textattr? & lineattr?
108                                          & fillattr? & posattr? & box?)>
109  <!ELEMENT port          - -             EMPTY>
110
111  <!--      ELEMENTS    NAME            VALUE             DEFAULT -->
112  <!ATTLIST type          id              ID              #IMPLIED>
113  <!ATTLIST initmark      id              ID              #IMPLIED>
114  <!ATTLIST portkreg      id              ID              #IMPLIED>
115  <!ATTLIST portreg       id              ID              #IMPLIED>
116  <!ATTLIST port          type            %porttype;      #IMPLIED>
117  ]>
```

In line 1 of the DTD the type of the document is defined to be cpn. On the lines 4–13 a number of "macros" are defined. This means, for example, that the macro arcdirs in line 8 is expanded to (BOTHDIR | NODIR | PtoT | TtoP) in line 20. Note that a macro substitution is invoked by prepending the macro name with '%' and appending ';'.

In line 17 the Generic Identifier (GI) cpn is defined. Since this GI equals the doctype in line 1 an element of this type is the root of the document (file). It is called a root because a SGML document represents a hierarchical structure. By defining the cpn GI, two so called *tags* are defined. The first one is a start tag which in a document is <cpn>. The second one is </cpn> which is an end tag. These two tags indicate the beginning and the end of the whole document. Generally a piece of text delimited by a start tag and an end tag is called a block. Since nothing but comments can come before the <cpn> tag it might be omitted. Similarly, the </cpn> tag might be omitted since nothing follows it. This is indicated by the two Os just below the MIN in line 16. The Os mean "omit". The first O indicates that the start-tag might be omitted. The second one that the end-tag might be omitted. By writing page* in the content field on the end of line 17 we define that a cpn-block might contain zero or more elements of type page. The '*'-operator means "zero or more".

In example 1 the `<cpn>` and `</cpn>` tags are omitted. In line 2 of example 1 a `page` block starts. This is legal since, as we just saw, `page` is a valid subelement GI of `cpn`. In line 18 of the DTD the GI `page` is defined. Thereby we define that the beginning of a `page` block is identified by the start tag `<page>`. The end of a `page` block is identified by the corresponding end tag `</page>`. The two dashes in line 18 indicate that the start and end tags can't be omitted. The content field at the end of line 18 of the DTD indicates that a page element consists of zero or more elements of type `arc`, `place` or `trans`. The '|'-operator means "or". In example 1 the `page` block contains a `trans` block (in lines 3–10), two `place` blocks (in lines 12–33 and lines 35–68), and two arcs (in lines 70–83 and lines 85–98). This is legal according to the definition in line 18 of the DTD. Notice that it is perfectly legal to have e.g. an `arc` block between two `trans` blocks. In line 2 of example 1 the start tag of the page block also contains the text `id=id10`. This text identifies an attribute. Attributes are only allowed in the start tag. In this case the attribute name is `id` and the attribute value is `id1`. It is defined in line 27 of the DTD that a `page` might have an attribute named `id`. Moreover the value range of this attribute is defined to be `ID`. Line 28 defines that the `arc` GI also can have an attribute whose value range is `ID` (and whose name is also `id`). By defining both value ranges to be `ID` we express that a page and an arc can't have same id value.[3] The `id` attribute is a legal attribute of all GIs representing objects. Thus, in the text format the value of each `id` attribute must be unique.

ID is a SGML reserved keyword. This uniqueness property of ID is defined by the SGML standard. Thus an object has a unique identification within the document. At the end of line 28 the default value of the attribute is defined to be `#IMPLIED`. IMPLIED is also a SGML keyword. That the default is IMPLIED means that it is implementation dependent what to do when processing a `<page>` tag in which the attribute `id` does not appear. In our implementation the page will be created.

The `arc` GI declaration in line 28 of the DTD introduces some new operators for the content declaration. The '&'-operator means that all of the operands must appear. The order in which the operands appear is however not determined. The '?'-operator means zero or one occurrence. This means that an `arc` block can contain at most one of each of the following blocks: `placeend`, `transend`, `lineattr`, `fillattr`, `seg-conn`, and `annot`. The six blocks are not restricted to appear in the listed order.

The `placeend` GI is defined in line 88 of the DTD. The content field contains the SGML keyword EMPTY. This means that a `placeend` has no valid subelement GIs. Hence there is no need for an and tag. In fact an end tag is disallowed in this case. The O indicating that the end tag can be omitted is just a reminder. The `placeend` GI has the attribute named `idref` defined. It is used in example 1 in which the value of the `idref` attribute in line 86 refers to the `id` attribute value declared in the `place` start tag in line 35. The value range of this attribute is IDREF. IDREF is a SGML keyword by which the attribute

[3] Even if the attributes had different names, say `pageid` and `arcid`, the common value range ID would insure uniqueness

value range is defined to be a reference to an attribute value of the type ID. The attribute value referenced must be declared within the document.

In line 43 of the DTD the attributes of the GI `posattr` are defined. The value range of both attributes is CDATA. This means that the attributes `x` and `y` can assume arbitrary character values, which does not conflict with the markup.

In the definition of the `text` GI in line 85 the content is defined to be CDATA. This means that everything within a `text` block is processed and no markup within the `text` is processed. Hence the sequence `<place></place>` is illegal inside a `trans` block whereas it is legal in a `text`. In the `text` block the sequence is not identified as a block but just as a stream of characters.

The rest of the definitions in this light version of the real DTD make use of the constructs discussed above.

For a thorough description of SGML confer with the ISO 8879 standard [6] or [5] which includes an annotated version of the ISO 8879 standard.

3.2 The Text Format as Description of a Petri Net Diagram

We have now presented and explained how the text format might represent a Petri Net diagram. We have also presented the formalism defining the structure of the text format. We now focus on *how* the structure defined by the DTD corresponds to the structure of a Petri Net diagram. We divide the GIs in two categories: the ones where content is allowed, and the ones where content is not allowed, e.g. the content is defined EMPTY. By using GIs of the first type it is possible to make *blocks*. The GIs with empty content do not define blocks.

The Blocks. In the DTD we defined some GIs with non-empty content, e.g. the definition of `page` in line 18. These GIs with non-empty content corresponds to *blocks* in example 1, e.g. the `page` block in lines 2–99 of the example. We have defined a block type called `ignore`. It is used for comments. These comments may extend over several lines. It may encompass other blocks (and other comments). Hence it can be used to ignore a large contiguous part of a document. The rest of the blocks represent five different types of information about the CP-net.

- Actual objects of the CP-net (pages, transitions, places, etc.).
- The text within an object.
- Additional information about a place, transition or arc (place type, transition condition, arc annotation, etc.).
- The text which is part of the additional information.
- Different kinds of defaults.

The first type is represented in example 1 where lines 3–10 represents the transition of Fig. 1. The `text` block in line 4 of example 1 is of the second type. It represents the text within the transition on Fig. 1. An example of the third type is the `initmark` block in lines 27–32. This block represents the initial marking of the *Send* transition on Fig. 1. In *Design/CPN* the initial marking is a region of the place. The text format does not assume that this is the case. If the `id` attribute in line 27 was omitted along with the lines 29–31, the `initmark` block would still be valid according to the DTD. If *Design/CPN* parses a block where the attributes are missing it just creates an initial marking region according to the defaults. The `id` is a valid attribute of the `initmark` GI because in *Design/CPN* an initial marking *is* represented by a graphical object. If a connector was created between the initial marking of the *Send* place in Fig. 1 and one of the arc annotation regions we would need to reference the corresponding blocks in the text format in order to represent the connector. The arc between the transition and the *Send* place on Fig. 1 is represented in lines 70–83 in example 1. To identify which place/transition pair is connected by the represented arc, line 71–72 refers to the ids introduced by the representation of the place and the transition. A connector connecting two object is represented in a similar way.

It is even possible to leave out the `text` block in line 28 of example 1. That block is of the third type according to the block type partitioning above. If it is left out the initial marking is implementation dependent. In *Design/CPN* an initial marking region without any contents is created if such a `initmark` block is parsed

The GIs with no Valid Subelement GIs. A GI which have no valid subelement GIs represents a (named) set of attributes. The attribute sets are primarily used to represent graphical attributes in diagrams. In line 5 of example 1 we have a set of four attributes:

```
<textattr font=Times size=12 just=Centered colour=Lime>
```

In lines 71 and 72:

```
<placeend idref=id25>
<transend idref=id34>
```

we see examples of attribute sets which have only a single attribute each.

In line 31 we have the empty attribute set `<label>`. This set represents the object represented by the enclosing block has box shape and that the size of that object depends entirely on the text within the object.

Representation of Graphical Layout One of the strengths of the Petri Nets is the graphical representation, which is intuitive and easy to understand. To keep this advantage it is essential that the nets are *drawn* in a readable fashion. Much care must be put into creating the graphical layout, and this work we do not want to lose. Thus we need to be able to represent the graphics in the text format.

First of all we need to be able to specify positions. We use a coordinate system with the x-coordinate running from left to right and the y-coordinate bottom-up. The centre of the diagram is at coordinate $(0, 0)$, and the measurement unit used is millimetres and fractions of millimetres. It is not possible to express this requirement in the DTD. In other words, the text format is not described entirely by the DTD.

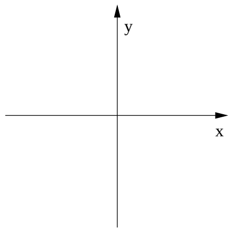

Fig. 2. The coordinates

The position of a object represented by a block is specified in the position attribute set `posattr`. The transition represented in the example is positioned at coordinate $(-644, 779)$ according to line 8 of example 1.

The text format does not make any demands on the shape of net objects. It is thus possible to specify any shape, which a given tool supports, as an attribute set of a block, to keep this information. The transition in the example is box shaped, and the first place has shape ellipse. Line thickness, fill pattern and colours are important. Thus this information is a part of the text format. The first transition in the example has a solid line, yellow coloured, and with line thickness 2pt, as seen in line 6.

4 Benchmarks

When considering a change of the file format used within *Design/CPN* , it is essential that performance is not jeopardised with respect to size and especially speed. To investigate this we have compared the time and space usage when loading and saving two "typical" CP-nets. No systematical benchmarking has been made yet.

Diagram		Binary	Textual	Ratio
Size	uncompressed	161 KB	165 KB	1.02
	compressed	35 KB	16 KB	0.46
Time	save	1 sec	2 secs	2.00
	load	1 sec	5 secs	5.00

Fig. 3. 7 pages, 22 places, 12 transitions

As seen in Figs. 3 and 4, the increase in size of the files is not significant, even though the size increases as much as 61%. This is not important, when considering the disk space available on an average workstation. This is especially true since the size of a diagram saved in the text format can be compressed to

Diagram		Binary	Textual	Ratio
Size	uncompressed	1371 KB	2221 KB	1.61
	compressed	302 KB	137 KB	0.43
Time	save	3 secs	39 secs	13.00
	load	13 secs	73 secs	5.62

Fig. 4. 33 pages, 237 places, 109 transitions

less than what is possible when using the binary format. This is done by using commonly known compression programs.[4]

When it comes to speed, the text format does not match the binary format. This comes as no surprise since it is possible to do a "memory dump" when using the binary format, whereas the diagram is traversed object-by-object when saved and loaded. Experiments indicate that the time used when saving a diagram in the textual format is linear in the number of objects. If modularisation and/or version control is integrated with the text format a substantial decrease in save times is expected. In that case it is only necessary to save the changed parts of a diagram. The same decrease is expected when editing diagrams. The whole diagram is only needed for simulation.

5 Future Work

The text format presented here makes it possible to save and load (parts of) diagrams within *Design/CPN*. When modelling complex systems two additional features would be feasible: version control and a modularisation system.

5.1 Version Control

The text format opens up for libraries of common net structures and distribution of nets over several files, and thus makes it easier for groups to work on different parts of larger projects. Large projects, however, often demand some sort of version control of files.

Common version control systems, e.g. CVS [2], works well on text but, perhaps surprisingly enough, less than perfect on the text format proposed here. Writing diagrams in a text editor will of course work as well as any source code version control, but when editing the diagrams in HLPN tools, as will often be the case, two major problems arise: we have no guaranty of consistency in identifiers, and different tools might export the objects in a different order.

Tools will have different ways of identifying objects internally which might reflect on the exported identifiers. Editing one file in the project can thus change the identifiers in that file making it inconsistent with the other files. This can of course be solved by simply demanding that blocks should be saved with the same

[4] The examples in this paper were all compressed using `gzip` [4].

identifier as they were loaded, forcing the tools to remember these "external" identifiers.

Merging two files is easily done with CVS, but different ordering of the blocks will result in conflicts in most of the file, nullifying the advantages of being able to edit separate copies. A solution could be restricting the order of blocks somehow. A better approach could be to add version control to the text format. With a fine-grained version control system as part of the text format it would be possible for different people to edit different parts of a diagram, perhaps in different tools, and then to merge the changes together, fully automatically. This could be done by adding unique time stamps or version numbers to each block. Merging two files would then simply be a matter of comparing these numbers. Whenever the version numbers differ the user could be prompted to take action, or the newest version would simply be selected.

Such a system could be the next step in making *Design/CPN* more attractive to large project groups.

5.2 Modularisation Management System

For the time being, the only way of dividing a diagram in modules is by saving different modules (subpages) in different files and later loading these separately. This is far from optimal. It would be better if it was possible to let the text format manage modularisation of a diagram. It should be possible to specify some sort of modularisation of the diagram, such that modules can be edited independently *without* having to combine all modules manually afterwards.

One solution is to tag substitution transitions as "modules" and to specify a file name for each of these. When saving the diagram, the subdiagram with the tagged substitution transition as root should be saved in a different file, with the specified file name, and only the file name and the substitution transition should be kept in the original diagram. If the tool discovers a "module" tagged transition during load, it simply loads the subdiagram from the file. Although not essential, this is definitely worth considering.

6 Conclusions

What we have presented here is not just a file format for *Design/CPN* but a framework for representing *High-level Petri Nets*. The benefits of this is that the format is easily changed/enhanced without losing the opportunity to develop a given diagram further in another tool (or another version of a given tool). Furthermore the format makes it possible to split diagrams into different files. This makes it easier for groups to work on the same diagram and also makes it possible to have standard libraries which can be used in different diagrams. The drawbacks of the format are mainly related to the speed for saving and loading a diagram. With current technology these penalties are not considered severe and are believed to become less significant within a short time frame due to hardware development.

We believe that SGML provides a robust framework for representing *High-Level Petri Nets*. The format presented here has proved that it is indeed possible to use SGML to represent *High-level Petri Nets*. The text format will evolve due to future development of *Design/CPN* and hopefully due to input from the Petri Net community.

7 Acknowledgements

We would like to thank the CPN group at University of Aarhus for their feedback to both the text format and this paper. A special thank to Kurt Jensen for his help in structuring this paper.

References

1. Jonathan Billington et al. High-Level Petri Nets – Concepts, Definitions and Graphical Notation. Committee Draft 15909, International Standards Organization and International Electrotechnical Committee, oct 1997. ISO/IEC JTC1/SC7/WG11 N-3.
2. Per Cederqvist. *Version Management with CVS*. Signum Support AB, Box 2044, S-580 02 Linkoping, Sweden, November 1993.
3. CPN ML: Relation with Standard ML. <URL:http://www.daimi.aau.dk/~desgncpn/sml/cpnml.html>.
4. L. Peter Deutsch. RFC 1952: GZIP file format specification version 4.3, May 1996. Status: INFORMATIONAL.
5. Charles F. Goldfarb and Yuri Rubinsky. *The SGML handbook*. Clarendon Press, Oxford, UK, 1990.
6. Information processing: text and office systems: Standard generalized markup language (SGML). ISO standard 8879, International Organization for Standardization, Genève, Switzerland, 1986. Amendment from 1988 exists.
7. Kurt Jensen. *Coloured Petri nets - Basic concepts, Analysis Methods and Practical Use. - Volume 1: Basic Concepts*. Monographs in theoretical computer science, 234 pages, Springer-Verlag, Berlin, 1992.
8. Kurt Jensen. *Coloured Petri Nets - Basic concepts, Analysis Methods and Practical Use. - Volume 3: Practical use*. Monographs in theoretical computer science, 265 pages, Springer-Verlag, Berlin, 1997.
9. Regnar Bang Lyngsø and Thomas Mailund. *Textual Interchange Format for High-level Petri Nets - Developers Guide*, 1998.

Appendix

A Full DTD for *Design/CPN* 3.1

```
<!-- DTD for Textual Interchange Format for Design/CPN ver. 3.1 -->

<!-- Base element -->
<!DOCTYPE cpn [

<!-- Possible shapes for obejcts: rbox is a rounded box -->
<!ENTITY % shape        "(box | ellipse | poly | line | label | wedge | rbox
                        | seg-conn)">

<!-- Legal attributes for objects -->
<!ENTITY % atts         "%shape;? & posattr? & fillattr? & lineattr? &
                        textattr? & flags? & layering?">

<!-- Line types: -->
<!ENTITY % lintyps      "(Solid | Dashed | DotDash | GroupInd |
                        FlashInd | None | Null | Dotted)">

<!-- Colours. These colours corresponds to the standard colours of
     HTML: -->
<!--
Name and RGB value:
    Black  = #000000    Green  = #008000
    Silver = #C0C0C0    Lime   = #00FF00
    Gray   = #808080    Olive  = #808000
    White  = #FFFFFF    Yellow = #FFFF00
    Maroon = #800000    Navy   = #000080
    Red    = #FF0000    Blue   = #0000FF
    Purple = #800080    Teal   = #008080
    Fuchsia= #FF00FF    Aqua   = #00FFFF
-->
<!ENTITY % cols         "(Black | Maroon | Gray | Olive |
                        Purple | Silver | Red | Teal |
                        Navy | Fucia | Blue | Aqua |
                        Green | Lime | Yellow | White)">

<!-- The possible orientations of an arc:
     BOTHDIR = Bidirectional arc: O<->[]
     NODIR = Arc without arrows: O-[]
     PtoT = Arc from Place to Transition: O->[]
     TtoP = Arc from Transition to Place: []->O
-->
<!ENTITY % arcors      "(BOTHDIR | NODIR | PtoT | TtoP)">

<!-- Possible types of fusion sets:
     Global = Global fusion set.
     Page = Page Fusion set.
```

```
     Instance = Instance Fusion Set -->
<!ENTITY % fustyps     "(Global | Page | Instance)">

<!-- Possible orientations of an auxiliary connector: -->
<!-- BOTHDIR = Bidirectional connector.
     NODIR = Nondirectional connector.
     1to2 = Connector from object1 to object2
     2to1 = Connector from object2 to object1 -->
<!ENTITY % orients     "(BOTHDIR | NODIR | 1to2 | 2to1)">

<!-- Possible fill patterns for objects -->
<!ENTITY % pats        "(None | Solid | AlmostSolid | InverseHeavyDotted |
                        Chessboard | InverseOrderedDotted |
                        NarrowVertical | FilledDiagonals
                        | WideVertical | Speckled | RandomDotted |
                        NarrowGrid | Bricks | SparselyDotted |
                        ShortDiagonals | Lightbubbles | PieTop |
                        Buttercups | Waffles |
                        LightDotted | MediumDotted | LightGray |
                        OrderedDotted | NarrowHorizontal |
                        NarrowDiagonal | WideHorizontal | WideDiagonal
                        | DottedBoxes | WideGrid | DiagonalBricks |
                        Holiday | Carrots | DiagonalScales |
                        RoundScales | Mountains | HeavyBubbles |
                        Diamonds)">

<!-- Possible font names. -->
<!ENTITY % fonts       "(Chicago | Courier | Geneva | Helvetica | Monaco
                        | Times | Symbol)">

<!-- The justification of text (left aligned, centered or right
     aligned. -->
<!ENTITY % justs       "(Left | Centered | Right)">

<!-- Possible porttypes: -->
<!-- In = Input Port.
     Out = Output Port.
     InOut = Input/Output Port
     General = General Port -->
<!ENTITY % prttyps     "(In | Out | InOut | General)">

<!-- Possible layering of object -->
<!-- Copy -->
<!-- IOR = Inclusive OR -->
<!-- XOR = Exclusive OR -->
<!-- INV = Inverse the objects covered -->
<!ENTITY % laytyps     "(Copy | IOR | XOR | INV)">

<!-- Possible attributes of objects: -->
<!--
```

```
   posattr = Position attributes
   fillattr = Fill Attributes
   lineattr = Line Attributes
   textattr = Text Attributes
   flags = Various flags (see definition of flags)
   layering = Layering logic of object -->
<!ENTITY % objatts     "posattr | fillattr | lineattr | textattr |
                        flags | layering">

<!-- Objects legal within a page: -->
<!-- place = Place
     trans = Transition
     tempdecl = Temporary Declaration Node
     globdecl = Global Declaration Node
     locdecl = Local Declaration Node
     aux = Auxiliary Object
     aux-conn = Auxiliary Connector -->
<!ENTITY % pageobs     "place | trans | tempdecl | globdecl | locdecl
                        | arc | aux | aux-conn">

<!-- Blocks legal within a place block: -->
<!-- name = Name block.
     type = Type declaration block.
     initmark = Initial marking block.
     aux-endp = Auxiliary endpoint block
     aux-std = Auxiliary region block
     fus-key = Fusion key region block
     portkreg = Port key region block
     mark-key = Marking key region block -->

<!ENTITY % placeob    "name | type | initmark | aux-endp | aux-std |
                       fus-key | portkreg | mark-key">

<!-- Blocks legal within a place block: -->
<!-- name = Name block
     cond = Condition block
     code-key = Code key region block
     log-key = Log key region block
     time = Time block
     aux-endp = Auxiliary endpoint region block
     hier-key = Hierarchy key region block.
     aux-enab = Enabling region. -->
<!ENTITY % transob    "name | cond | code-key | log-key | time |
                      aux-endp | hier-key | aux-enab">

<!-- Elements recognised on outermost level of a cpn-document:
     Information about generator tool.
     One or more ordinary pages.
     Zero or one hierarchy page -->
<!ELEMENT cpn           O O            ((page)* & hierpage? & generator?)>
```

```
<!-- Text blocks are recognised to contain Character Data -->
<!ELEMENT text     - - CDATA>

<!-- Elements and attributes recognised within a page -->
<!ELEMENT page     - -  (text? & (%pageobs; | %shape; | pageattr | fillattr |
                        lineattr | posattr | winattr | modeattr | mult)*)>
<!ATTLIST page         id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a place -->
<!ELEMENT place    - -  (text? & (%placeob; | %shape; | %objatts; | port)*)>
<!ATTLIST place        id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a name block -->
<!ELEMENT name     - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST name         id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a type block -->
<!ELEMENT type     - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST type         id             ID             #IMPLIED>

<!-- Elements and attributes recognised within an initial marking block -->
<!ELEMENT initmark - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST initmark     id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a transition block -->
<!ELEMENT trans    - -  (text? & (%transob; | %shape; | %objatts; | modeattr |
                        subst)*)>
<!ATTLIST trans        id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a condition block -->
<!ELEMENT cond     - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST cond         id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a code key region block -->
<!ELEMENT code-key - -  (text? & (aux-std | code | %shape; | %objatts;
                        | regattr)*)>
<!ATTLIST code-key     id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a code region block -->
<!ELEMENT code     - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST code         id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a log key region block -->
<!ELEMENT log-key  - -  (text? & (aux-std | log | %shape; | %objatts;
                        | regattr)*)>
<!ATTLIST log-key      id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a log region block -->
```

```
<!ELEMENT log      - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST log          id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a time block -->
<!ELEMENT time     - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST time         id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a hierarchy key region block -->
<!ELEMENT hier-key - -  (text? & (aux-std | hierarch | %shape; |
                         %objatts; | regattr)*)>
<!ATTLIST hier-key     id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a hierarchy region block -->
<!ELEMENT hierarch - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST hierarch      id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a fusion key region block -->
<!ELEMENT fus-key  - -  (text? & (aux-std | fusion | %shape; |
                         %objatts; | regattr)*)>
<!ATTLIST fus-key      id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a fusion region block -->
<!ELEMENT fusion   - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST fusion       id             ID             #IMPLIED
                       type           %fustyps;      #IMPLIED
>

<!-- Elements and attributes recognised within an arc block -->
<!ELEMENT arc      - -  (text? & (annot | aux-std | %shape; | lineattr |
                         placeend | transend | textattr | flags | connattr |
                         fillattr | layering)*)>
<!ATTLIST arc          id             ID             #IMPLIED
                       orientation    %arcors;       #IMPLIED
>

<!-- Elements and attributes recognised within an arc annotation block -->
<!ELEMENT annot    - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST annot        id             ID             #IMPLIED>

<!-- Elements and attributes recognised within an local declaration block -->
<!ELEMENT locdecl  - -  (text? & (aux-endp | aux-std | %shape; | %objatts;)*)>
<!ATTLIST locdecl      id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a temporary declaration block -->
<!ELEMENT tempdecl - -  (text? & (aux-endp | aux-std | %shape; | %objatts;)*)>
<!ATTLIST tempdecl     id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a global declaration block -->
<!ELEMENT globdecl - -  (text? & (aux-endp | aux-std | %shape; | %objatts;)*)>
<!ATTLIST globdecl     id             ID             #IMPLIED>
```

```

<!-- Elements and attributes recognised within an auxiliary object block -->
<!ELEMENT aux      - -  (text? & (aux-endp | aux-std | %shape; | %objatts;)*)>
<!ATTLIST aux          id             ID             #IMPLIED>

<!-- Elements and attributes recognised within an auxiliary connector block -->
<!ELEMENT aux-conn - -  (text? & (aux-std | %shape; | lineattr | node1 |
                        node2 | textattr | flags | connattr | fillattr |
                        layering)*)>
<!ATTLIST aux-conn     id             ID             #IMPLIED
                       orient         %orients;      #IMPLIED
>

<!-- Elements and attributes recognised within an auxiliary standard region
     block -->
<!ELEMENT aux-std  - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST aux-std      id             ID             #IMPLIED>

<!-- Elements and attributes recognised within an auxiliary endpoint region
     block -->
<!ELEMENT aux-endp - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST aux-endp     id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a port key region block -->
<!ELEMENT portkreg - -  (text? & (aux-std | portreg | %shape; |
                        %objatts; | regattr)*)>
<!ATTLIST portkreg     id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a port region block -->
<!ELEMENT portreg  - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST portreg      id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a marking key region block -->
<!ELEMENT mark-key - -  (text? & (aux-std | mark | %shape; | %objatts;
                        | regattr)*)>
<!ATTLIST mark-key     id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a marking region block -->
<!ELEMENT mark     - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST mark         id             ID             #IMPLIED>

<!-- Elements and attributes recognised within an auxiliary enabling
     region  block -->
<!ELEMENT aux-enab - -  (text? & (%shape; | %objatts; | regattr)*)>
<!ATTLIST aux-enab     id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a hierarchy page block. -->
<!ELEMENT hierpage - -  (text? & (aux | aux-conn | pagenode | substcon
                        | %shape; | pageattr | fillattr | lineattr |
                        posattr | winattr)*)>
```

```
<!ATTLIST hierpage     id             ID             #IMPLIED>

<!-- Elements and attributes recognised within an hierarchy page node block. -->
<!ELEMENT pagenode - -  (text? & (aux-std | pgmodkey | %shape; | %objatts;)*)>
<!ATTLIST pagenode     id             ID             #IMPLIED
                       pageref        IDREF          #IMPLIED>

<!-- Elements and attributes recognised within a page mode key region
     block -->
<!ELEMENT pgmodkey - -  (text? & (pagemode | aux-std | %shape; |
                        %objatts; | regattr)*)>
<!ATTLIST pgmodkey     id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a page mode region block -->
<!ELEMENT pagemode - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST pagemode     id             ID             #IMPLIED>

<!-- Elements and attributes recognised within a substitution connector block -->
<!ELEMENT substcon - -  (text? & (aux-std | substtag | %shape; | lineattr |
                        sup-page | sub-page | textattr | flags | connattr |
                        fillattr | layering)*)>
<!ATTLIST substcon     id             ID             #IMPLIED
                       orient         %orients;      #IMPLIED
>

<!-- Elements and attributes recognised within a substitution tag block -->
<!ELEMENT substtag - -  (text? & (aux-std | %shape; | %objatts; | regattr)*)>
<!ATTLIST substtag     id             ID             #IMPLIED
                       transref       IDREF          #IMPLIED
>

<!-- Name and version of tool which generated file -->
<!ELEMENT generator     - O EMPTY>
<!ATTLIST generator         tool      CDATA
                            version   CDATA>

<!-- Position attributes: -->
<!-- x : x coordinate in mm
     y:  x coordinate in mm

SGML doesn't allow a definition of numbers which might be negative,
hence the use of CDATA as type for the coordinates.

The coordinate system looks like this:
      ^
      | y
      |
      |
      |      x
--------------\
```

```
      |       /
      |
      |
      |
-->

<!ELEMENT posattr  - O EMPTY>
<!ATTLIST posattr      x              CDATA          #IMPLIED
                       y              CDATA          #IMPLIED
>

<!-- Definition of various shapes.

The attributes:
     h = height (in mm)
     w = width (in mm)

These are declared as CDATA since SGML has no way of defining real
numbers.

Decimal seperator is '.'.

     curv = curvature (0-360)
     stangle = start angle (0-360)
     endangle = end angle (0-360)
     points = sequence of points "(x1,y1)(x2,y2)...(xn,yn)"
-->


<!ELEMENT box      - O EMPTY>
<!ATTLIST box          h              CDATA          #IMPLIED
                       w              CDATA          #IMPLIED
>

<!ELEMENT ellipse  - O EMPTY>
<!ATTLIST ellipse      h              CDATA          #IMPLIED
                       w              CDATA          #IMPLIED
>

<!ELEMENT poly     - O EMPTY>
<!ATTLIST poly         points         CDATA          #IMPLIED>

<!ELEMENT line     - O EMPTY>
<!ATTLIST line         points         CDATA          #IMPLIED>

<!ELEMENT label    - O EMPTY>

<!ELEMENT wedge    - O EMPTY>
<!ATTLIST wedge        h              CDATA          #IMPLIED
                       w              CDATA          #IMPLIED
```

```
                       stangle        NUMBER         #IMPLIED
                       endangle       NUMBER         #IMPLIED
>

<!ELEMENT rbox     - O EMPTY>
<!ATTLIST rbox         h              CDATA          #IMPLIED
                       w              CDATA          #IMPLIED
                       curv           NUMBER         #IMPLIED
>

<!ELEMENT seg-conn - O EMPTY>
<!ATTLIST seg-conn     curv           NUMBER         #IMPLIED
                       points         CDATA          #IMPLIED
>


<!-- Different attributes related to objects: -->
<!ELEMENT fillattr - O EMPTY>
<!ATTLIST fillattr     pattern        %pats;         #IMPLIED
                       colour         %cols;         #IMPLIED>

<!-- Line attributes (type, thichness and colour).
     Thickness is measured in points -->
<!ELEMENT lineattr - O EMPTY>
<!ATTLIST lineattr     type           %lintyps;      #IMPLIED
                       thick          NUMBER         #IMPLIED
                       colour         %cols;         #IMPLIED
>

<!-- Text attributes. The different attributes declared to have
type CDATA may assume values TRUE or FALSE. SGML doesn't allow more
than one attribute in an ATTLIST to have same group.
Hence:
                       bold          (TRUE | FALSE)
is legal, but
                       bold          (TRUE | FALSE)
                       italic        (TRUE | FALSE)
is not. -->
<!ELEMENT textattr - O EMPTY>
<!ATTLIST textattr     font           %fonts;        #IMPLIED
                       just           %justs;        #IMPLIED
                       colour         %cols;         #IMPLIED
                       bold           CDATA          #IMPLIED
                       italic         CDATA          #IMPLIED
                       underlin       CDATA          #IMPLIED
                       outline        CDATA          #IMPLIED
                       shadow         CDATA          #IMPLIED
                       condense       CDATA          #IMPLIED
                       extend         CDATA          #IMPLIED
                       scbar          CDATA          #IMPLIED
```

```
                        size            NUMBER          #IMPLIED
>

<!-- Object flags -->
<!-- All flags attributes might assume values TRUE or FALSE.
     mask - object pickable or not.
     omit - object showable or not
     nobound - object boundary visible or not
     visible - visible or not -->

<!ELEMENT flags    - O EMPTY>
<!ATTLIST flags         mask            CDATA           #IMPLIED
                        omit            CDATA           #IMPLIED
                        nobound         CDATA           #IMPLIED
                        nosizing        CDATA           #IMPLIED
                        visible         CDATA           #IMPLIED
>

<!-- Place end of an arc.
     idref is a reference to the id of the place to connect to.
     endpoint is a reference to the object that the arc physically
     connects to.
     Since only one IDREF is recognised inside an ATTLIST, endpoint
     is declared to be CDATA.
-->
<!ELEMENT placeend - O EMPTY>
<!ATTLIST placeend      idref           IDREF           #IMPLIED
                        endpoint        CDATA           #IMPLIED
>

<!-- Transition end of an arc. Same reasoning as for the placeend -->
<!ELEMENT transend - O EMPTY>
<!ATTLIST transend      idref           IDREF           #IMPLIED
                        endpoint        CDATA           #IMPLIED
>

<!-- First object connected by a connector -->
<!ELEMENT node1    - O EMPTY>
<!ATTLIST node1         idref           IDREF           #IMPLIED>

<!-- Second object connected by a connector -->
<!ELEMENT node2    - O EMPTY>
<!ATTLIST node2         idref           IDREF           #IMPLIED>

<!-- Attributes for a page -->
<!-- name - name of page.
     number - the number of the page (this is NOT the id of the page)
     visbor - borders visible (TRUE/FALSE)
-->
<!ELEMENT pageattr - O EMPTY>
```

```
<!ATTLIST pageattr     name           CDATA          #IMPLIED
                       number         NUMBER         #IMPLIED
                       visbor         CDATA          #IMPLIED
>

<!-- Region attributes: -->
<!-- colorap - Colour as parent
     sizewp - resize with parent
     hoffset - horisontal offset (in mm)
     voffset - vertical offset (in mm)
-->

<!ELEMENT regattr  - O EMPTY>
<!ATTLIST regattr      colorap        CDATA          #IMPLIED
                       sizewp         CDATA          #IMPLIED
                       hoffset        CDATA          #IMPLIED
                       voffset        CDATA          #IMPLIED
>

<!-- Connector attributes -->
<!-- hdwidth - Arrow head width (in points)
     hdheight - Arrow head height (in points)
     txtwidth - Text width (in points)
     txtheight - Text height (in points)
-->
<!ELEMENT connattr - O EMPTY>
<!ATTLIST connattr     hdwidth        NUMBER         #IMPLIED
                       hdheight       NUMBER         #IMPLIED
                       txtwidth       NUMBER         #IMPLIED
                       txtheight        NUMBER       #IMPLIED
>

<!-- Window attributes -->
<!-- open - open on load (TRUE/FALSE)
     width - Width (in pixels).
     height - Height (in pixels) -->
<!ELEMENT winattr  - O EMPTY>
<!ATTLIST winattr      open           CDATA          #IMPLIED
                       width          NUMBER         #IMPLIED
                       height         NUMBER         #IMPLIED
                       xpos           NUMBER         #IMPLIED
                       ypos           NUMBER         #IMPLIED
>

<!-- Mode attributes -->
<!-- Determines some simulation code options. -->
<!ELEMENT modeattr - O EMPTY>
<!ATTLIST modeattr     siminc         CDATA          #IMPLIED
                       observe        CDATA          #IMPLIED
                       propocc        CDATA          #IMPLIED
```

```
                       interact       CDATA          #IMPLIED
                       execcode       CDATA          #IMPLIED
>

<!-- For substitution transitions -->
<!-- subpage - The ID of the corresponding subpage
     portsock - List of port/socket assingments on the form:
       "(port1ID, socket1ID)(port2ID, socket2ID)...(portNID,
       socketID)"
-->
<!ELEMENT subst    - O EMPTY>
<!ATTLIST subst        subpage        IDREF          #IMPLIED
                       portsock       CDATA          #IMPLIED
>

<!-- Port type -->
<!ELEMENT port     - O EMPTY>
<!ATTLIST port         type           %prttyps;      #IMPLIED>

<!-- Superpage end of substitution connectors -->
<!ELEMENT sup-page - O EMPTY>
<!ATTLIST sup-page     idref          IDREF          #IMPLIED>

<!-- Subpage end of substitution connectors -->
<!ELEMENT sub-page - O EMPTY>
<!ATTLIST sub-page     idref          IDREF          #IMPLIED>

<!-- Multiplicity of page -->
<!-- insts - number of instances -->
<!ELEMENT mult     - O EMPTY>
<!ATTLIST mult     insts             NUMBER         #IMPLIED>

<!ELEMENT layering - O EMPTY>
<!ATTLIST layering     type           %laytyps;      #IMPLIED>
]>
```