

SNIFF: An Input/Output Library for Design/CPN

Christoph Maier * Daniel Moldt and Heiko Rölke †

Abstract

In our group several projects used and use Coloured Petri nets as their main technique for modeling. Design/CPN has been one of the tools to edit and simulate the Petri net diagrams. However, Design/CPN has to be considered as a closed tool. To enlarge its functionality the desire is to open the tool for its environment. This is achieved by a library of functions which are implemented using Design/ML and Mimic allowing selective import and export of single or all pages of a Petri net diagram. Its input/output format is a human-readable text file. The text has a well defined syntax and maintains the nets structure and its content. Hence the interface is public other tools can use the diagrams produced by Design/CPN and Design/CPN can use the output of other tools.

1 Introduction

Today's software development needs special and powerful tools which support the underlying concepts and techniques. In the area of specification and prototyping (Coloured) Petri nets are a well known formalism for which the tool Design/CPN [Design/CPN 1993b] has specifically been developed, besides some others. Building such tools is a very expensive task as can be seen in all parts of computer science. Therefore existing tools are adopted or extended to fit the specific needs within projects.

In Hamburg one of the main techniques used within projects were Coloured Petri Nets [Jensen 1992]. Especially in the context of prototyping projects [Beckmann 1993] as well as object-oriented Petri net projects [Becker und Moldt 1993; Maier 1996] the need for tool support arose. The tool Design/CPN has been used since its first distribution. However, especially the introduction of new concepts and structuring rules for Coloured Petri nets diagrams required to use an open tool. Some former projects in Hamburg showed that the required man-power to build an own special version of a Petri net tool would have been expensive with respect to time and resources. Therefore Design/CPN has been extended

*Forschungsinstitut für Angewandte Software-Technologie (FAST) e.V., Arabellastr. 17, D-81925 Muenchen, cma@fast.de

†Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Theoretische Grundlagen der Informatik, Vogt-Kölln-Str. 30, D-22527 Hamburg, {moldt,3roelke}@informatik.uni-hamburg.de

by some functions which are now bundled in the SNIFF library¹. SNIFF allows to connect Design/CPN to the output of other tools, e.g. diagrams which contain Coloured Petri nets transformed from class diagrams (for class diagrams see e.g. [Fowler und Scott 1997]). Also some extensions were necessary due to some mistakes, gaps, or design flaws within Design/CPN. The following list provides some functionality requirements for Design/CPN which influenced our design decisions for the SNIFF architecture.

- Within the prototyping projects many cycles of editing and simulating the nets occurred. During that process the internal representation got somehow corrupted without being recognized by the modelers. In the later process this turned out to be a reason for crashing of the program. Due to the unreadable file format the large nets had to be rebuilt by hand. Therefore a reasonable backup facility was required.
- The binary file format of Design/CPN did not allow to modify the net diagrams with other tools, e.g. changing some values for whole nets using emacs for textual replacement of regular expressions.
- The import and export of the whole net or some parts were very limited. Desirable features are to select pages or to blind out attributes of diagrams, e.g. graphical attributes for an external analysis tool.
To load other diagrams only loading of IDEF diagrams was available. However, in some projects it was of interest to load nets which were generated by other tools, e.g. object-oriented Coloured Petri nets as defined in [Maier 1996; Moldt 1996].
- A library of Design/CPN diagrams could not be build due to the insufficient import/export functionality. Independent development of parts of a Petri net e.g. by different people was not supported. Models build separately could not be merged, except for the restricted use of loading and saving one single diagram page. This implied many difficulties for teamwork.
- There was no pretty printer for the net diagrams. Modelers have to ensure themselves the use of the same attributes for the same kind of element within one Design/CPN diagram.
- Porting diagrams between different platforms (Machintosh and Sun) was insufficient.
- For some diagrams the concepts of place refinement was of interest, even when these diagrams should not be simulated, the automatic consistency was desired. Unfortunately only the versions before Design/CPN version 2 supported this. Diagrams already drawn using version 2.0 had to be redrawn by hand. Therefore a support for different versions was desirable. The same is true when moving from version 3.xx to version 2.xx.

¹SNIFF stands for "Structured Net Interchange File Format"

First of all **SNIFF** is the name for the library given here. At the same time it is the name of a project which runs now for several years besides our main activities. The phrase **SNIFF** is for obvious reasons strongly related to the **SNIFF** grammar and the **SNIFF** textual format.

Design/CPN

Design/CPN integrates three kinds of Petri net tools: editor, simulator, and analyzer (see [WWW-Design/CPN 1998b]): "The CPN Editor supports construction, modification and syntax check of CPN models. The CPN Simulator supports interactive and automatic simulation of CPN models. The Occurrence Graph Tool supports construction and analysis of occurrence graphs for CPN models (also known as state spaces or reachability graphs/trees). All three parts can be used for large hierarchical CP-nets - with or without time."

Functionality

A Petri net is called a diagram within the context of Design/CPN. A diagram is the combination of all net components that can be accessed at the same time. The diagram is divided into single pages, each shown in a window of its own. The pages can be connected by substitution transitions and fusion places. Main elements of the net structure are places, transitions and arcs. In Design/CPN diagrams (and therefore in the **SNIFF** grammar) all these elements can be further specified by regions and auxiliary regions: places need a colour region, transitions may have a guard region or a code region, arcs should have an arc expression etc. Hierarchy information for the already mentioned substitutions and fusions are located in such regions, too, as well as simulation specific and other information. For a deeper introduction to Design/CPN see [Design/CPN 1993b] or [WWW-Design/CPN 1998a].

Design/ML

Design/CPN is based upon the functional programming language ML (see [Design/CPN 1993a; Harper 1986; Paulson 1991]). ML is not only used in code regions of more complex transitions, but it is also possible to execute any program without leaving the tool. The language itself is extended mainly by colors of places and variables to satisfy special Petri net needs. Furthermore an extensive set of functions is provided. These functions have been used intensively for **SNIFF**.

Mimic

Mimic is an extension to Design/CPN which allows to build simple graphical user interfaces using only functions from the built in Design/ML library. The mouse is supported for user interaction. The functions can be used within the edit-mode of Design/CPN. Mainly the selection possibilities for the mouse device are used within **SNIFF**. For more details about Mimic see [Rasmussen und Singh 1995] and [WWW-Mimic 1998].

After this introduction section 2 explains basic ideas, architecture, functionality, grammar, and examples files of **SNIFF** and closes with a discussion of gained experience. The general conclusion and outlook is followed by **SNIFF**'s grammar and an example in the appendices.

2 Textual Output for Design/CPN: SNIFF

To open Design/CPN to its environment, general import and export functions have to be implemented. These can be used in different contexts. Therefore the provided library can be used to backup, to overcome the problem of corrupted diagrams, to import from other tools, to export to other tools, to build Coloured Petri net diagram libraries, to pretty print Coloured Petri net diagrams, e.g. according to project standards, and to use selective filters. As the conceptual basis the Structured Net Interchange File Format (SNIFF) is used.

2.1 Basic Idea

Hence the internal representation of the diagrams is hidden, the Design/ML functions are used to access the internal representation to build an SNIFF internal representation which is then exported in ASCII format using a reasonable, human-readable notation. Generally the Design/CPN tool becomes an open tool, hence the exported ASCII text diagrams have a public interface, the SNIFF grammar. Once we have such an internal SNIFF representation and the exported diagrams, we can manipulate the Coloured Petri net diagrams to implement the functional requirements mentioned above.

The SNIFF text diagram representation² can be written to a text file, reread from a file and converted again to a Design/CPN diagram. It is important to notice that this includes the hierarchical structure of Design/CPN diagrams, like substitution transition or fusion place relations in the same way as all other regions. Separate im- and export functions are available which can be used independently, e.g. to filter diagram output. To allow for flexibility, parameter files are used for filter and default options. The non-terminals in the grammar are easily changeable keywords, allowing to adopt the output file, e.g. to new evolving international standards.

The textual description of a net allows the access to a net by external programs. These programs may further process an existing net, but also generate a new net description which can be converted to a Design/CPN net. This can be graphically edited and simulated as usual.

2.2 Architecture of SNIFF

SNIFF is realised as a function library. There are four major function blocks. Figure 1 shows the functions and which functions are used for what purposes. A Coloured Petri net is applied as the description technique for the architecture.

One function block converts a Design/CPN diagram to SNIFF's internal representation (*read diagram*), another converts the internal representation to text file format (*write text*). The two remaining function blocks handle the other direction (*read text*) and (*write diagram*). The figure is explained by describing a scenario for exporting a Design/CPN diagram to SNIFF's text format. The small dashed box annotated with **Export** indicates the

²A direct backup of the net to a text file would also be possible, but we prefer a modular approach.

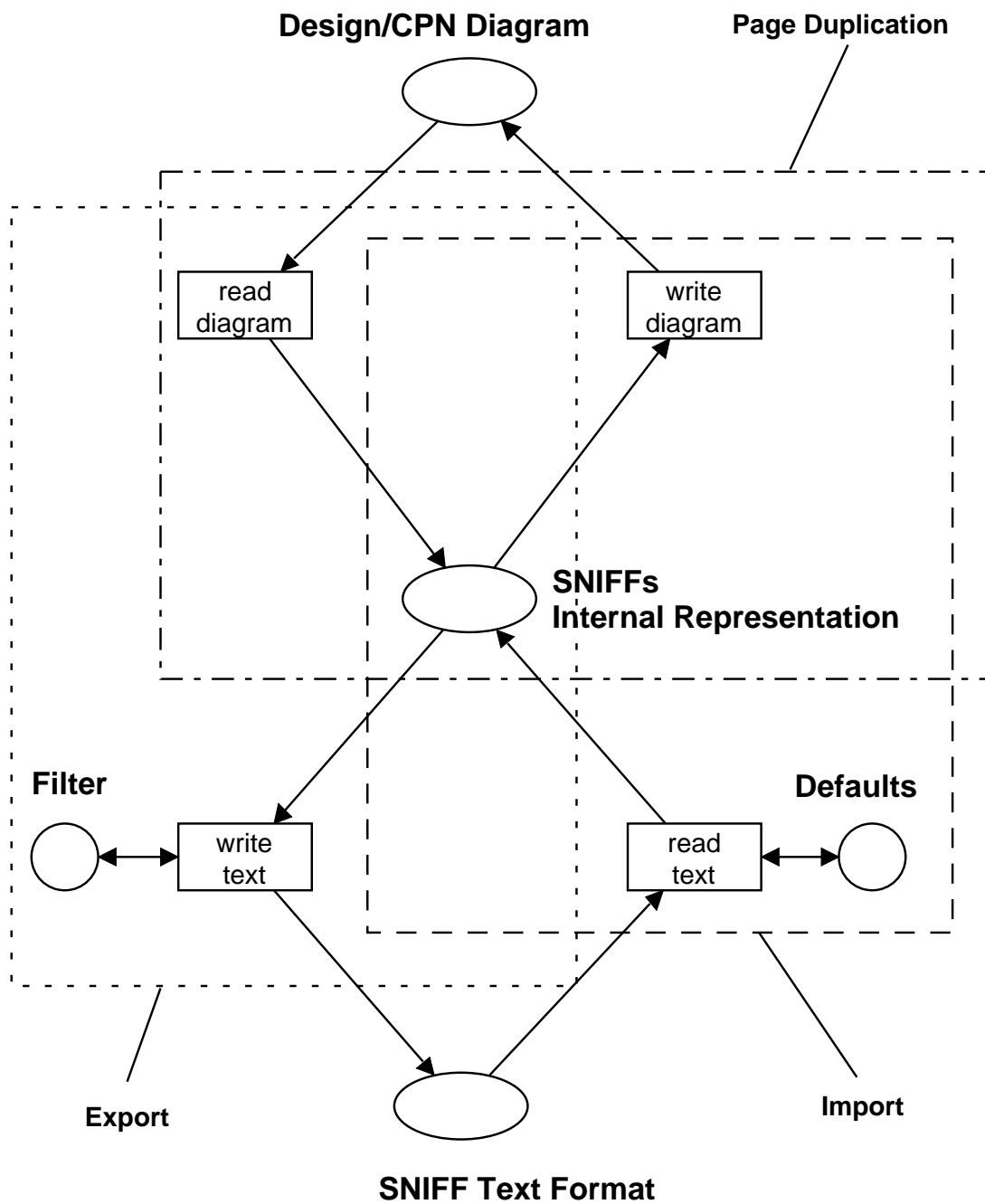


Figure 1: SNIFF's Architecture

involved parts.

Imagine a Design/CPN diagram lying as a token on the place³ at the top of the figure. The upper left transition takes this token and generates a new token of the type '*Internal Representation*' and puts it on the place in the middle. The token represents the whole description of the net in a SNIFF specific format. Now there are two ways to go: The upper right transition and the lower left transition are activated as well. This means that one may call the function for producing a text format diagram or for generating new pages in the given Design/CPN diagram. (This would be the page duplication function, indicated by the dotted dashed box in figure 1.) To achieve our goal of exporting the diagram we choose the *write text* transition.

Our example scenario is simplified in terms of some adjustment possibilities. There is a text file influencing in detail which diagram elements and even which attributes are taken over from the internal representation to the text format diagram and which are dropped. In the figure it is shown as a side condition to the *write text* condition (place **Filter**). Another text file (place **Defaults**) controls the transformation of the internal representation back to a Design/CPN diagram. Attributes may be overridden by default values.

This functionality is not realized as a monolithic function but is done via function composition: A function's result is the next function's parameter. This allows for a flexible application of the SNIFF functions.

2.3 Functionality

SNIFF is started by a function call in an auxiliary box. Its outer appearance is a kind of graphical user interface. The user will see only the usual Design/CPN graphical interface, e.g. the file menu to select a backup file. To provide easy expandability the advanced user may call some of SNIFF's functions directly in any combination with other Design/ML functions.

The ordinary user gets a simple menu containing the following topics:

- 1 → Save Net to Text
- 2 → Restore Net from Text
- 4 → Save selected Pages
- 5 → Restore selected Pages
- 7 → Duplicate Pages
- 9 → Exit

The selection results in the direct execution of the function. Selection one and two are related to the whole diagram. After selecting them the user is asked for the name of the new backup file or for an already existing one respectively. By choosing selection four or five one gets the file menu as well as a new window with a list of all pages of the diagram. Now the mouse can be used to select any subset of pages for further processing. This page list is being examined for integrity and automatically supplemented by all pages which are

³So this place's colour is '*Design/CPN Diagram*'.

below the selected pages in the substitution hierarchy. Duplication of pages only needs a selection of pages, copies of these pages, including the subordinated pages, are integrated into the present diagram. The creation of the new pages takes care of already existing pages and their numbers. If necessary, the new pages get assigned new pages numbers.

The menu was implemented using Mimic for Design/CPN [Rasmussen und Singh 1995]. All other parts of SNIFF only use functions available in Design/ML. No external function library is needed for execution of SNIFF. So every user of Design/CPN can easily utilize SNIFF, too.

2.4 Grammar and Textual Format

The major goal of mapping a diagram is a complete description of all net contents to ensure the possibility of a one to one copy. Furthermore it is desirable to be able to translate the internal representation to a text file without too much effort. The text file itself should maintain the structure and should be human readable. This is because simple changes ought to be feasible in an ordinary text editor by hand. e.g. to correct corrupted diagrams. These requirements again influence the internal representation which in large parts resembles the text format. The internal representation is implemented as an ML datatype.

The functionality of SNIFF is closely connected to the internal structure of Design/CPN. Thus a central feature is a unique integer as ID assigned to each page, object, arc, and attribute. These IDs are used whenever a graphical object is referenced e.g. socket-port assignment. SNIFF processes diagrams page by page: The internal representation is a list of pages. Each page contains several fixed attributes and a list of its elements. As an analogy each element contains fixed attributes and a list of its subordinated elements like auxiliary regions, hierarchy informations, code segments and so on. The subordinated elements always contain a fixed number of fields. The integer constants for some attributes like `'shape=2'` could easily be replaced by symbolic constants for which one can find reasonable names in the Design/ML handbook [Design/CPN 1993a]. The complete SNIFF grammar can be found in appendix A. All non-terminals of the grammar are constants within the program code and are read in from a text file when starting SNIFF. Modification for keywords is therefore easy. The structure of the grammar requires more effort. Section 2.5 provides an introduction to this data structure by discussing two examples.

2.5 Examples

First a very simple diagram is presented as an example, which consists only of a single page (see figure 2). This page shows the three main elements of all Petri nets: place, transition, and arc. The arc is directed from place to transition. Beside the figure SNIFF's text output is shown. Each page block is encapsulated by a begin-end. Fixed attributes such as name or page number are followed by the blocks for the page's content. Places (the first block) and transitions (the last block) are somewhat similar in their attributes. The arc's block looks a little bit different because an arc needs extra fields for its orientation and its style. The example net has been scaled up so that its parts fit the text counterparts.

```

begin
  name=/*New*/;
  OID=4;
  subtyp=/*Simple*/;
  pagenr=1;
  visible=1;
  coord=[0,0,533,776];
  begin
    OID=7;
    typ=/*Place*/;
    subtyp=/*Simple*/;
    name=/**/;
    shape=2;
    visuals=[0,2,0,1];
    coord=[~15,~268,120,60];
  end
  begin
    OID=10;
    typ=/*Arc*/;
    subtyp=/*Simple*/;
    name=/**/;
    shape=20;
    visuals=[0,2,1,1];
    pointlist=[~15,~268,~15,65];
    coord=[7,8];
    begin
      orientation=1;
      connprops=[6,12,20,64,32];
    end
  end
  begin
    OID=8;
    typ=/*Trans*/;
    subtyp=/*Simple*/;
    name=/**/;
    shape=1;
    visuals=[0,2,0,1];
    coord=[~15,65,120,60];
  end
end
end

```

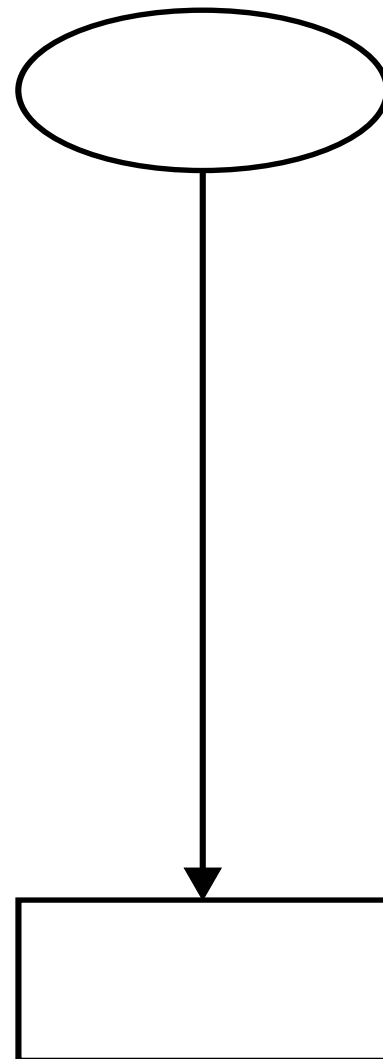


Figure 2: Example 1 for SNIFF text file and Petri net diagram

Appendix B contains an example for a Coloured Petri net which shows the two putative seasons of Hamburg (from the point of view of some pessimists). It exemplifies how regions are assigned to nodes and some other essential elements of Design/CPN diagrams. Both places have a name and a colour region. The 'Summer' place has an initial marking. Transitions are named, they do not have any guards or code regions. The arcs are inscribed with simple arc expressions. In appendix C the textual representation as produced by the export function of SNIFF can be found.

2.6 Discussion

Both examples are rather simple Coloured Petri nets. The size of textual representations of more complex nets prohibits larger examples here, however, the examples give an idea of the textual representation, its advantages and disadvantages. The benefits of SNIFF have already been mentioned above. It can be added, that the usual Design/CPN feature of saving a simulation state is still preserved, hence SNIFF is an add-on. The option to modify Design/CPN to use SNIFF as one save/load option in the file menu is independent from this.

However, it must be mentioned that the strong closeness to Design/CPN causes some problems, which are discussed in the following. The special tailoring to Design/ML with the specific Design/CPN functions allows no portation to other tools which use other platforms, even if the design of SNIFF tried to avoid this. The implementation had to obey runtime requirements.

The grammar has been designed some years ago without the aim to fit any upcoming standard for High-Level Petri nets (HLPN). However, it should be possible to adopt it. The import-functions of SNIFF has to be adopted to be able to drop other information which is not present in usual Design/CPN diagrams. E.g. an extension of Design/CPN by test-arcs in a newer version would also require an appropriate representation in the internal data structure. For new functions there will be internal Design/CPN functions which could be easily integrated into the SNIFF functions. While new features, not present in Design/CPN had to be dropped or transformed by special functions.

The direct use of integers e.g. for the shape of an object (for transitions usually: `shape = 1`, for places usually: `shape = 2`), can be replaced by defined textual constants such as `shape = rectangle` or `shape = ellipse`, if this is desired. Here the internal values are used.

One problem using the Design/ML functions is that they do not provide all informations. Sometime values are returned e.g. as `unknown`, even if the behavior of Design/CPN clearly shows that it is known. This caused many time consuming work arounds. Even if the design could mainly be kept, the implementation had to be more complex than necessary otherwise.

The speed of the save and load functions is reasonable and at a linear speed compared to the overall number of objects, arcs, and attributes. However, burdened with a fixed amount of setup time. The loading of a file can be directly watched, hence the nets are build up using the usual Design/ML functions. The appearance is quite impressive and,

due to the rapidly drawn nets, does not disturb the user while waiting for the completion of the diagram, since the progress can be directly followed.

The last versions of the SNIFF library have been tested on Sun and Linux platforms only. Only very few problems were encountered between the different version. E.g. the hierarchy page of the Sun version has a bug, which prevents the preservation of its graphical structure, while the Linux version does not have this problem. The Machintosh platform was only used at the beginning of the SNIFF project. Mainly the insufficient performance of the available Machintosh computers prevented the portation for the newer Design/CPN versions to those machines. Also there were some minor problems with the compatability of the ML versions. Furthermore, the graphical appearance on the Machintosh/Sun platforms can be different. The colours used at a Machintosh have to be mapped to black/white attributes such as dotted lines. This caused the provision of the text files for filters and defaults as can be seen in figure 1.

Altogether a new tool has been created which gives Design/CPN users the possibility of using their diagrams in a more widespread way. Figure 3 gives an idea of SNIFF's possibilities. The ASCII text allows to provide Design/CPN diagrams to other tools.⁴ They have to provide an import and an export function. Due to the SNIFF text format the exchange between other tools also becomes possible, if they stick to the format. What was of special importance for us, was the possibility to allow external generation of Coloured Petri nets, especially for class diagrams which have a considerable amount of components when described as Colured Petri nets. What is interesting at this point is that Design/CPN itself can be used as the generating tool, hence it can draw arbitrary diagrams, which do not have to be Coloured Petri nets. The Design/ML allows the expanding or transformation of the other diagrams, for which again SNIFF functions can be used. This has been used in the SIMON⁵ project [Moldt 1996] and worked quite well. The generation of special Petri nets is especially important for net formalisms with a high degree of administrative overhead like pages for class diagrams or message schedulers in object-oriented Petri nets.

3 Conclusion and Future Plans

SNIFF has been used within several projects. Due to the difficult access to the internal representation several versions had to be build to cope with the available internal Design/ML functions. The availability of SNIFF has widened the range of possible applications considerably, covering the before mentioned features. It would be interesting to see how the interchange format for Design/CPN could be used in the context of the evolving international standard (see [WWW-PetriNetStandard 1998]).

At the Computer Science department of the University of Hamburg SNIFF is at present used within projects for diagram merging, for backups as well as for importing automatically generated net descriptions into Design/CPN. These external functions for generating textual Coloured Petri nets outside Design/CPN are still at prototype level.

⁴XPN in figure 3 stands for all other tools.

⁵SIMON stands for *SIMulator for Object-oriented Petri Nets*

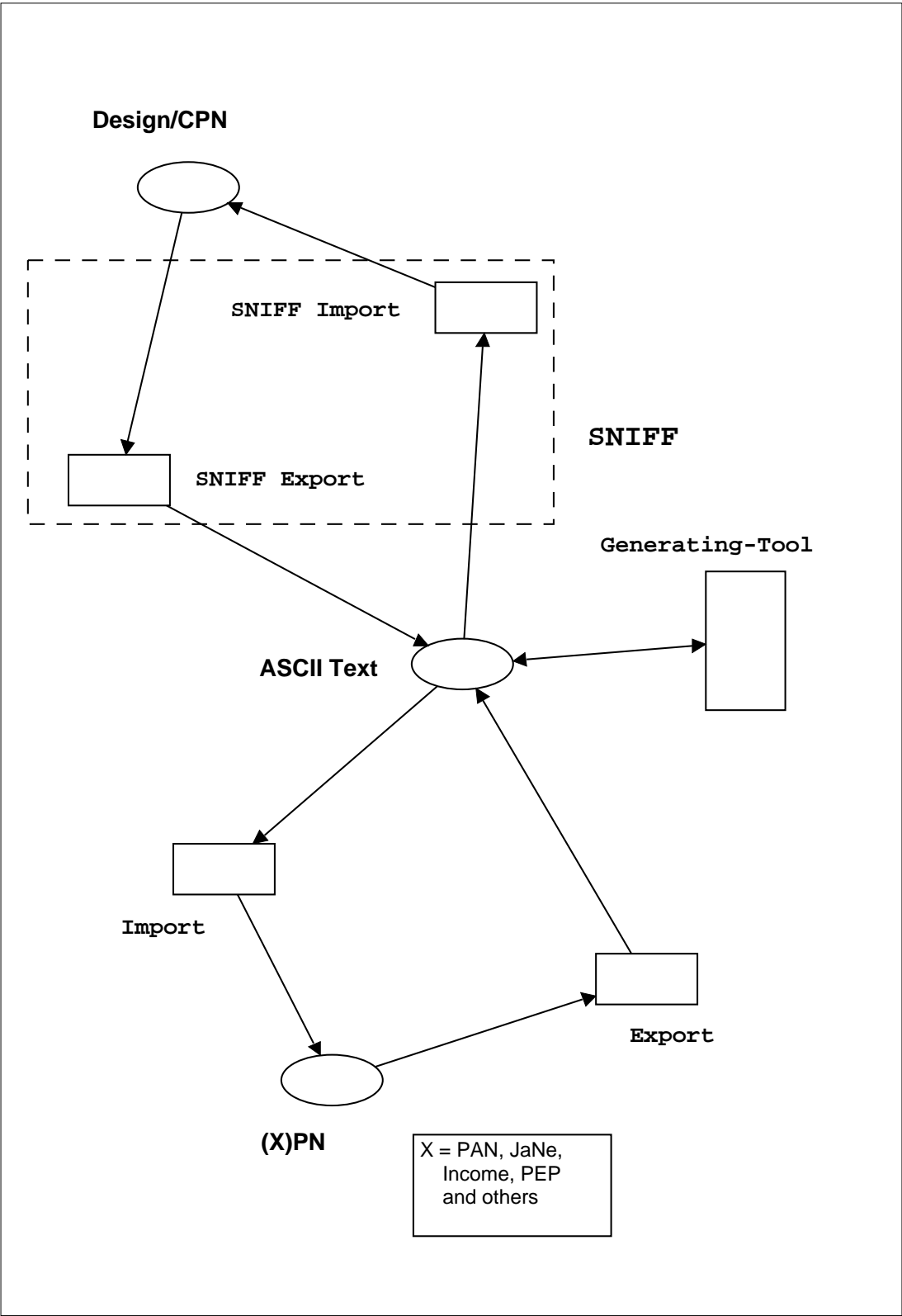


Figure 3: Possible use of SNIFF

A Grammar of SNIFF's text file output

```
net ::= <page>*

page ::= begin
      name           = <string>;
      OID            = <int>;
      subtyp         = <ObjSubType>;
      pagenr         = <int>;
      visible        = <int>;
      coord          = <int list>;
      { <element>* }
    end

element ::= <object> | <arc>

object ::= begin
        name           = <string>;
        OID            = <int>;
        typ            = <ObjType>;
        subtyp         = <ObjSubType>;
        shape          = <int>;
        visuals        = <int list>;
        fontsize       = <int>;
        coord          = <int list>;
        { begin <attribute>* end }
      end

arc ::= begin
     name           = <string>;
     OID            = <int>;
     typ            = Arc;
     subtyp         = <ObjSubType>;
     shape          = <int>;
     visuals        = <int list>;
     pointlist      = <int list>;
     coord          = <int list>;
     begin
         { <attribute>* }
         orientation = <int>;
         connprops   = <int list>;
     end
  end
```

```

        end

attribute      ::= <attr_sort> = <string> with
                OID           = <int>;
                parentid     = <int>;
                shape         = <int>;
                visuals       = <int list>;
                fontsize      = <int>;
                coord         = <int list>;
                end

ObjType        ::= Unknown | DefBox | Trans | Place

ObjSubType     ::= Simple | Subtr | SubPl | Glob | Temp | Loc
                | In | Out | InOut | Gen | Sub | Inv
                | GlobFus | PageFus | InstFus

attr_sort      ::= name | region
                | namereg
                | color | initmark | port
                | globalfusion | pagefusion | instfusion
                | guard | codeseg | time | hierarchy |
                | expr

string         ::= /* <ident> */

int            ::= {~} <digit>+

int list       ::= [] | [ <int> {, <int>}+ ]

```

B Design/CPN Petri Net Example

Seasons in Hamburg

```
color token = with tok;  
  
var toc : token;
```

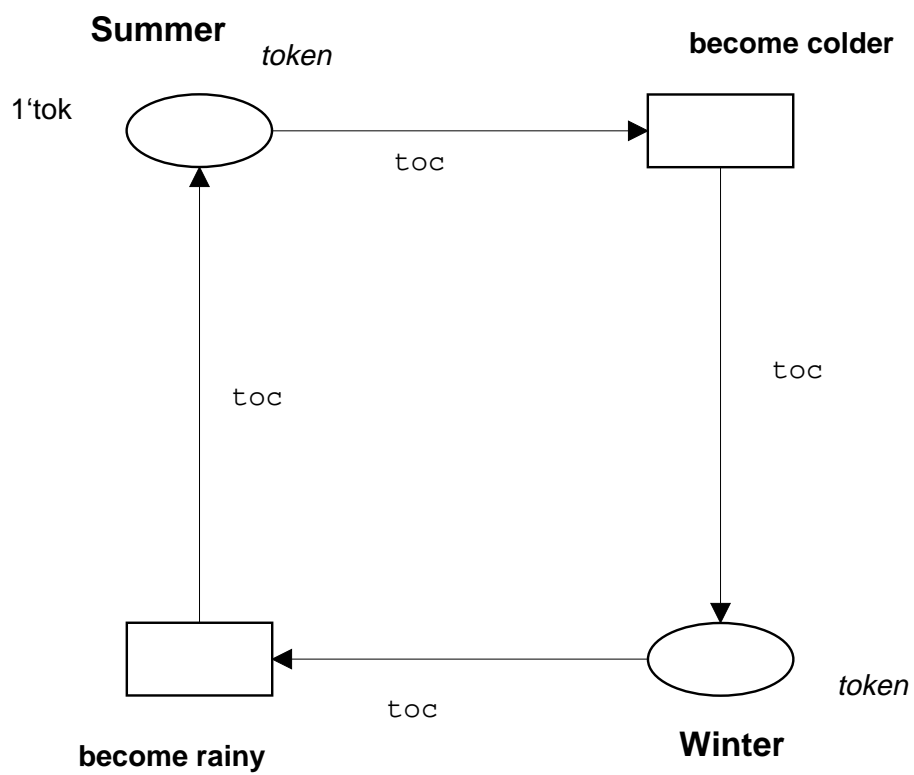


Figure 4: A more "complex" net

C Design/CPN Petri Net Example as SNIFF text file

```
begin
    name=/*New*/;
    OID=18;
    subtyp=/*Simple*/;
    pagenr=1;
    visible=1;
    coord=[0,0,533,776];
begin
    OID=36;
    typ=/*Unknown*/;
    subtyp=/*Simple*/;
    name=/*Seasons in Hamburg*/;
    shape=1;
    visuals=[0,1,0,1];
    fontsize=18;
    coord=[~145,~347,188,23];
end
begin
    OID=34;
    typ=/*DefBox*/;
    subtyp=/*Glob*/;
    name=/*color token = with tok;
        var toc : token;
        */;
    shape=1;
    visuals=[0,1,0,1];
    fontsize=14;
    coord=[106,~301,204,116];
end
begin
    OID=21;
    typ=/*Trans*/;
    subtyp=/*Simple*/;
    name=/**/;
    shape=1;
    visuals=[0,1,0,1];
    coord=[~124,64,60,30];
begin
    namereg=/*become rainy*/ with
        OID=28;
        parentid=21;
        coord=[~134,104,86,16];
        shape=1;
        visuals=[0,0,0,1];
        fontsize=12;
end
end
end
begin
    OID=2;
    typ=/*Trans*/;
    subtyp=/*Simple*/;
    name=/**/;
    shape=1;
    visuals=[0,1,0,1];
    coord=[91,~154,60,30];
begin
    namereg=/*become colder*/ with
        OID=30;
        parentid=2;
        coord=[121,~190,92,16];
        shape=1;
        visuals=[0,0,0,1];
        fontsize=12;
end
end
end
begin
    OID=5;
    typ=/*Place*/;
    subtyp=/*Simple*/;
    name=/**/;
    shape=2;
    visuals=[0,1,0,1];
    coord=[91,64,60,30];
begin
    namereg=/*Winter*/ with
        OID=28;
```

```

        parentid=5;
        coord=[97,99,52,18];
        shape=1;
        visuals=[0,0,0,1];
        fontsize=14;
    end
    color=/*token*/ with
        OID=38;
        parentid=5;
        coord=[154,75,36,16];
        shape=1;
        visuals=[0,0,0,1];
        fontsize=12;
    end
end
end
begin
    OID=7;
    typ=/*Place*/;
    subtyp=/*Simple*/;
    name=/**/;
    shape=2;
    visuals=[0,1,0,1];
    coord=[~124,~154,60,30];
    begin
        namereg=/*Summer*/ with
            OID=26;
            parentid=7;
            coord=[~141,~196,63,18];
            shape=1;
            visuals=[0,0,0,1];
            fontsize=14;
        end
    end
    color=/*token*/ with
        OID=37;
        parentid=7;
        coord=[~84,~185,36,16];
        shape=1;
        visuals=[0,0,0,1];
        fontsize=12;
    end
end
initmark=/*1'tok*/ with
    OID=39;
    parentid=7;
    coord=[~189,~163,33,16];
    shape=1;
    visuals=[0,0,0,1];
    fontsize=12;
end
end
end
begin
    OID=25;
    typ=/*Arc*/;
    subtyp=/*Simple*/;
    name=/**/;
    shape=20;
    visuals=[0,0,1,1];
    pointlist=[~124,64,~124,~154];
    coord=[21,7];
    begin
        expr=/*toc*/ with
            OID=60;
            parentid=25;
            coord=[~101,~45,27,15];
            shape=1;
            visuals=[0,0,0,1];
            fontsize=12;
        end
        orientation=1;
        connprops=[4,8,20,1,1];
    end
end
begin
    OID=24;
    typ=/*Arc*/;
    subtyp=/*Simple*/;
    name=/**/;
    shape=20;
    visuals=[0,0,1,1];
    pointlist=[91,64,~124,64];
    coord=[5,21];
    begin
        expr=/*toc*/ with
            OID=66;
            parentid=24;

```


References

- [Becker und Moldt 1993] ULRICH BECKER AND DANIEL MOLDT. Object-Oriented Concepts for Coloured Petri Nets. In IEEE, editor, “Conference Proceedings, IEEE International Conference on Systems, Man and Cybernetics”, volume 3, pages 279–286, Le Touquet, Frankreich (17.–20. October 1993). IEEE.
- [Beckmann 1993] NICOLE BECKMANN. Prototyping mit gefärbten Petrinetzen. Studienarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Germany (November 1993).
- [Design/CPN 1993a] “Design/CPN Handbook: CPN ML – CPN Palette; Part 1, Version 2.0”. Cambridge, MA, USA (1993).
- [Design/CPN 1993b] Meta Software Corporation, Cambridge, MA, USA. “Design/CPN Handbook Version 2.0” (1993).
- [Fowler und Scott 1997] MARTIN FOWLER AND KENDALL SCOTT. “UML Distilled – Applying the Standard Object Modeling Language”. Addison-Wesley, Reading, Massachusetts (1997).
- [Harper 1986] ROBERT HARPER. Introduction to Standard ML. LFCS Report Series ECS-LFCS-86-14, University of Edinburg (1986).
- [Jensen 1992] KURT JENSEN. “Coloured Petri Nets: Volume 1; Basic Concepts, Analysis Methods and Practical Use”. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin Heidelberg New York (1992).
- [Maier 1996] CHRISTOPH MAIER. Darstellung von Konzepten der objektorientierten Modellierung und Programmierung mit Petrinetzen. Studienarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Germany (May 1996).
- [Moldt 1996] DANIEL MOLDT. “Höhere Petrinetze als Grundlage für Systemspezifikationen”. Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Germany (August 1996).
- [Paulson 1991] LAWRENCE C. PAULSON. “ML for the Working Programmer”. Cambridge University Press, Cambridge, England (1991).
- [Rasmussen und Singh 1995] JENS L. RASMUSSEN AND MEJAR SINGH. “Mimic/CPN A Graphic Animation Utility for Design/CPN”. Computer Science Department, Aarhus University, DK-8000 Aarhus C, Denmark, 1.5 edition (1995).
- [WWW-Design/CPN 1998a] “<http://www.daimi.aau.dk/designCPN/>” (1998).
- [WWW-Design/CPN 1998b] “<http://www.daimi.aau.dk/PetriNets/tools/db/designcpn.html>” (1998).
- [WWW-Mimic 1998] “<http://www.daimi.aau.dk/designCPN/libs/mimic/>” (1998).
- [WWW-PetriNetStandard 1998] “<http://www.daimi.aau.dk/PetriNets/standard/>” (1998).