

# Overview of Design/CPN

© Kurt Jensen  
University of Aarhus  
Computer Science Department  
Ny Munkegade, Bldg. 540  
DK-8000 Aarhus C, Denmark

Phone: +45 89 42 32 34  
Telefax: +45 89 42 32 55  
E-mail: [kjensen@daimi.aau.dk](mailto:kjensen@daimi.aau.dk)

## Abstract

This paper provides an overview of the Design/CPN tool by describing some of the most important facilities. The paper is divided in three chapters which cover the CPN editor (including the syntax check), the CPN simulator and the Occurrence Graph tool, respectively.

The contents of the paper is similar to Chap. 6 of the following book. However, it is shortened and modified to match the newest version of Design/CPN.

K. Jensen: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*. Monographs in Theoretical Computer Science, Springer-Verlag, 1992. ISBN: 3-540-60943-1.

The paper assumes some prior knowledge of Coloured Petri Nets. If you do not have such knowledge, we propose that you first study some of the “Introductory Examples” provided via the Design/CPN WWW pages. It is also recommended to read the introductory chapters of the book mentioned above (in particular Sects. 1.1-1.7, 3.1-3.2, 6.1-6.3 and 7.1-7.5). Before reading the Occurrence Graph chapter it may be a help to look at the “Occurrence Graph Examples” provided via the Design/CPN WWW pages and Chap. 1 one of:

K. Jensen: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods*. Monographs in Theoretical Computer Science, Springer-Verlag, 1994. ISBN: 3-540-58276-2.

Design/CPN WWW pages: <http://www.daimi.aau.dk/designCPN/>

# Table of Contents

## Chapter 1: CPN Editor

Windows and status bar.....	3
Different types of objects .....	4
Why do we need object types?.....	4
CPN objects, auxiliary objects and system objects.....	5
Graphical representation.....	5
Attributes.....	6
Options.....	6
Hierarchy page.....	6
Construction of hierarchical nets.....	7
Move to Subpage.....	7
Substitution Transition.....	8
Replace by Subpage.....	8
Top down or bottom up.....	8
Groups of objects.....	9
Different working styles.....	9
Key and popup regions .....	10
Menu commands .....	10
Different skill levels.....	12
Syntax restrictions.....	12
Syntax checking.....	13
Use of names.....	13
Abstract data base.....	14

## Chapter 2: CPN Simulator

Integration with the CPN editor.....	15
Page instances .....	15
Automatic simulations .....	16
Interactive simulations.....	16
Interactive simulation with manual selection of occurrence sets.....	16
Interactive simulation where Design/CPN selects the occurrence sets.....	19
Observation of interactive simulations.....	20
Graphical feedback during interactive simulations.....	20
Breakpoints in interactive simulations.....	21
Limitations of automatic simulations .....	21
Simulation of timed CP-nets.....	21
Code segments.....	22
Simulation modes .....	23
Simulation menu.....	23
Calculation of enabled bindings.....	23

## Chapter 3: Occurrence Graph Tool

Integration with the CPN simulator.....	25
Construction of occurrence graphs .....	25
Standard report .....	26
Standard queries.....	26
Customised queries .....	27
Temporal logic.....	28
Occurrence graphs with equivalence classes.....	28
Strongly connected components.....	28
Graphical display of occurrence graphs.....	29

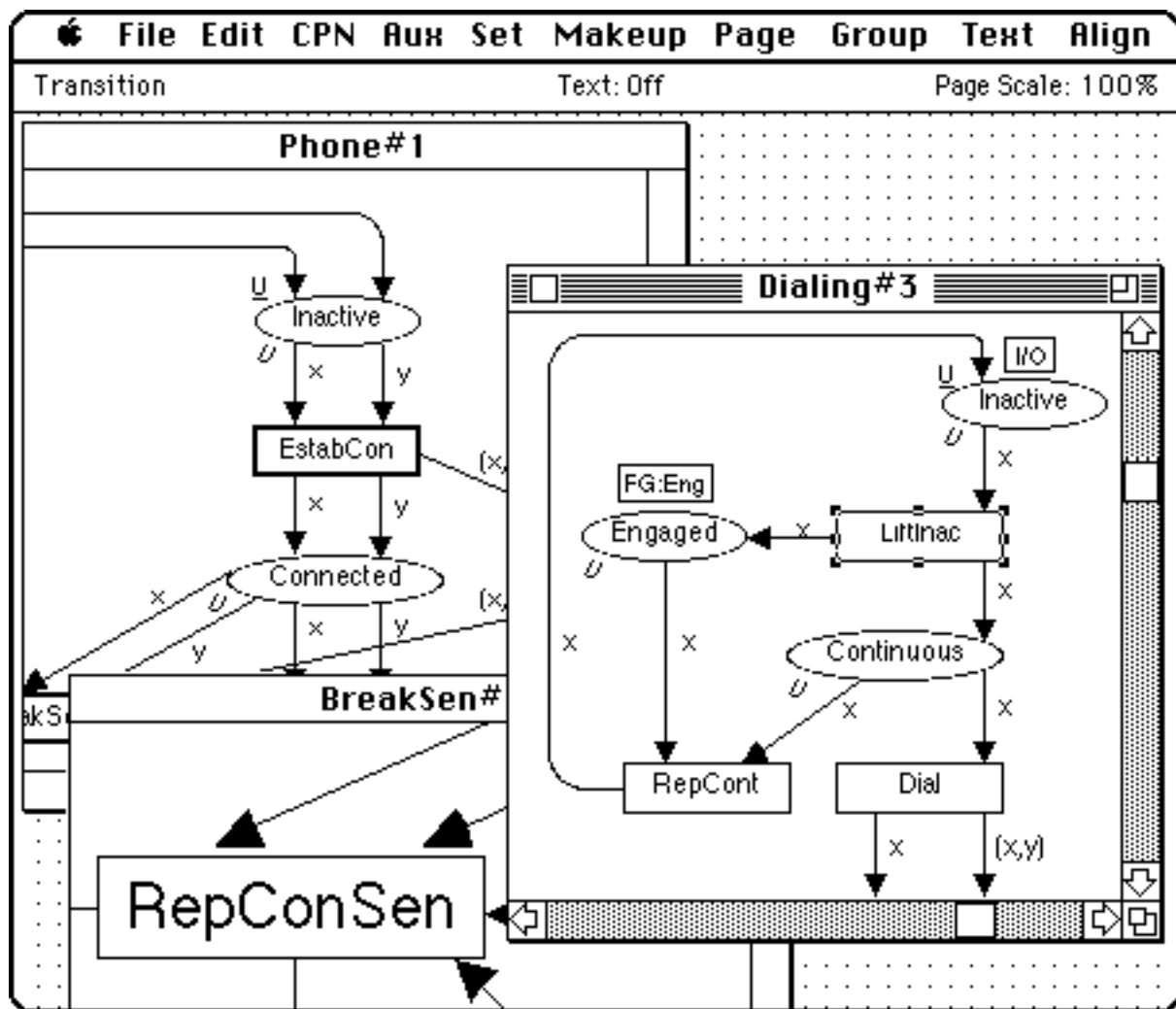
# Chapter 1

## CPN Editor

The CPN editor allows the user to construct, modify and syntax check hierarchical CP-nets – with or without time.

### Windows and status bar

The user works with a high-resolution raster graphical screen and a mouse. It is recommended, but not necessary, to have a large colour screen. The CPN diagram under construction can be seen in a number of windows – where it looks as close as possible to the final output (e.g., obtained by a laser printer). The editor is menu driven and has self-explanatory dialogue boxes.



The user moves and resizes the objects by direct manipulation, i.e., by means of a mouse (instead of typing coordinates and object identification numbers on the keyboard). The CPN editor supports hierarchical CP-nets and this means that each CPN diagram contains a number of pages. A typical screen image looks as shown above.

Each page of the hierarchical CP-net is displayed in its own **window**, which may be open (i.e., visible) or closed. In our example there are three open windows: *Phone#1*, *Dialing#3* and *BreakSen#5*.

Below the menu bar is the **status bar**. It contains information about the present state of the Design/CPN tool. In our example, we are told that the currently selected object *LiftInac@Dialing#3* is a transition. We are not in text mode and the active page *Dialing#3* is displayed with standard scale (i.e., 100%).

### Different types of objects

The pages of a CPN diagram contain a number of graphical objects, e.g., places, transitions, arcs, guards and initialization expressions. Each object has a particular **object type**, which determines the way that the editor interprets the corresponding object. This means that we may draw, e.g., transitions as rounded boxes or icons (or even ellipses) without confusing Design/CPN.

In addition to the object types, Design/CPN distinguishes between nodes, connectors and regions. A **node** is an object which can be created and may exist without being immediately related to other objects. This is the case, e.g., for places, transitions and declaration nodes. A **connector** is an object which always interconnects two nodes, and it cannot exist without these nodes. This is the case, e.g., for arcs. Each arc interconnects a place and a transition. A **region** is an object which is subordinate to another object and cannot exist without that object. This is the case, e.g., for initialization expressions (which are regions of places), guard expressions (which are regions of transitions) and arc expressions (which are regions of arcs). It is also possible to have regions which are regions of other regions.

The object type tells whether a given object is a node, connector or region, and it also tells the possible relationships to other objects. As examples, a transition may have a guard region, but it cannot have an initialization region (because these can only be related to places). The transition may also have one or more arcs connecting it with other nodes (which have to be places).

### Why do we need object types?

The division of objects into nodes, connectors and regions makes it possible for Design/CPN to recognise a CPN diagram as a mathematical graph (and not just as a set of unrelated graphical objects). When the user constructs a connector, he simply identifies the two nodes which he wants to interconnect (by pointing somewhere inside them). Then the system draws the connector and it automatically calculates the position of the connector end points, in such a way that they are precisely on the node borders. This saves time and it makes the produced di-

agram nicer (because the editor can position the end points more precisely than the user). It is also important to notice that the editor automatically redraws the connectors and the regions each time this becomes necessary, for example because a node is repositioned, resized or deleted.

The automatic redrawing described above may seem obvious, but it should be noticed that it only is possible because the CPN editor recognizes the structure of a CPN diagram as a mathematical graph. In most general-purpose drawing tools a CP-net would only be recognised as a set of unrelated objects. This means that the repositioning or resizing of a transition has to be followed by a tedious manual repositioning of the guard, the end points of the arcs, and the arc expressions. Analogously, the deletion of a transition has to be followed by a manual deletion of the guard, the arcs and the arc expressions (and if this is forgotten we get an inconsistent net which, e.g., may have dangling arcs).

### **CPN objects, auxiliary objects and system objects**

The CPN editor also distinguishes between CPN objects, auxiliary objects and system objects. Nearly all objects which we have met so far are **CPN objects**. This means that they have a formal meaning and they influence the behaviour of the CP-net. Places, transitions, arcs, initialization expressions, guards and arc expressions are examples of CPN objects. However, it is also useful to be able to have **auxiliary objects** in a CPN diagram. Such objects have no formal meaning. They are included to make the CP-net more readable, and thus they play a similar role to comments in programming languages. It is possible to create auxiliary objects of many different shapes, e.g., ellipses, rectangles, rounded boxes, polygons, wedges and bit maps. The auxiliary objects may be nodes, connectors or regions, and there are no restrictions on the relationship to other objects. As an example it is possible to create an auxiliary connector between any pair of nodes (e.g., between two places, or between an auxiliary node and a declaration node). All objects may have auxiliary regions. Finally, there are **system objects** which are special objects created by Design/CPN itself. An example is the current marking regions which the CPN simulator creates for each place. These regions contain information about the number and the colour of the tokens which reside on the corresponding place, in the current marking.

### **Graphical representation**

One of the most attractive features of CP-nets (and Petri nets in general) is the existence of a nice graphical representation. It would thus be a pity to put narrow restrictions on this representation, for example by making an editor in which all places and transitions have a fixed shape and size. In our opinion, a good editor must allow the user to draw almost all kinds of CP-nets which can be constructed by a pen and a typewriter.

In Design/CPN it is possible for the user to determine, in great detail, how he wants the CPN diagrams (and the occurrence graphs) to look. This means that the modeller for each object can determine the position and the detailed appear-

ance, e.g., the shape. Connectors can be single headed, double headed or without heads. Nodes and regions can be boxes, rounded boxes, ellipses, polygons, wedges, bit maps or labels.

## Attributes

Each object has its own set of **attributes** which determine, e.g., the position, shape, size, line thickness, line and fill patterns, line and fill colours, and text appearance (including font, size, style, alignment and colour). There are 10–30 attributes for each object (depending upon the object type). When a new object is constructed, the attributes are determined by a set of **defaults**. Each object type has its own set of defaults, and this means, e.g., that place names can be displayed in one font, while transition names are displayed in another. At any time the user can change one or more attributes for each individual object. This is done by means of the Attribute commands (in the Set menu).

It is possible to create a new object by copying an existing object. Then the new object automatically gets the same attributes as the old object.

The user may save a specified attribute value as a new default to apply to new objects of the corresponding object type. When defaults are saved the user determines whether he wants the new defaults to become diagram defaults (i.e., be in effect only for the present diagram), to become system defaults (i.e., be in effect for all new diagrams), or both.

## Options

In addition to the attributes, Design/CPN has a large set of **options** which determine how the detailed operations are performed. An attribute relates to a particular object, while an option relates to the entire CPN diagram. For example there is an option which determines how fast the contents of windows scroll. Another option specifies how duplicate arcs are treated when one or more nodes are merged into a single node (by means of the Merge command).

Options are changed by means of the Option commands (in the Set menu). For options we only have system defaults and no diagram defaults (because a diagram only has one value for each option). Option **defaults** can be saved and loaded in a way which is similar to attribute defaults.

## Hierarchy page

The hierarchical relationships between the individual pages (subnets) of a CPN diagram are shown in a **page hierarchy** graph, which is displayed on a separate page called the **hierarchy page**. This page is automatically updated by the CPN editor – as the user creates and deletes pages (and hierarchical relationships between them). All the objects of the page hierarchy graph are **page objects**, which are a particular class of system objects. Like all other object types, page objects have their own sets of defaults (which determine how they look). The Hierarchy Page Options determines the standard layout of the hierarchy page,

e.g., the spacing between the page nodes. Page objects can be moved and modified, in exactly the same way as all other types of objects. This means that the user can determine in great detail how the page hierarchy looks.

The hierarchy page uses three different line patterns to indicate the status of each page. The page node of the active page is drawn with an unbroken line, while the others page nodes are drawn with two different kinds of shaded line patterns – indicating whether they are open or closed.

The hierarchy page shows the different pages and their hierarchical relationships. However, it is also an active device by which the user can manipulate the pages. As an example, he can open a page by double-clicking the corresponding page node. He can delete a page by deleting the page node and he can remove the relationship between a supernode and its subpage by deleting the page connector/region representing the relationship.

### Construction of hierarchical nets

It is possible to construct the hierarchical relationships of a CPN diagram in many different ways, ranging from a pure **top-down** to a pure **bottom-up** approach. This is done by using the following editor commands:

- Move to Subpage
- Substitution Transition
- Replace by Subpage

#### Move to Subpage

When a page gets too many places and transitions, we can move some of them to a new subpage. This is done by a single editor operation. The user selects the nodes to be moved and invokes the Move to Subpage command. Then the editor

- checks the legality of the selection (all border nodes must be transitions),
- creates the new page,
- moves the subnet to the new page,
- creates the port places by copying those places which were next to the selected subnet,
- calculates the port types,
- creates the corresponding port regions,
- constructs the necessary arcs between the port nodes and the selected subnet,
- prompts the user to create a new transition which becomes the supernode for the new subpage,
- draws the arcs surrounding the new transition,
- creates a hierarchy inscription for the new transition,
- updates the hierarchy page.

As may be seen, a lot of rather complex checks, calculations and manipulations are involved in the Move to Subpage command. However, almost all of these are automatically performed by Design/CPN. The user only selects the subnet, invokes the command and creates the new supernode. The rest of the work is done

by the CPN editor. This is of course only possible because Design/CPN recognizes a CPN diagram as a hierarchical CP-net, and not just as a mathematical graph or as a set of unrelated objects. Without this property the user would have to do all the work by means of the ordinary editing operations (which allow him to copy, move and create the necessary objects). This would be possible – but it would be much slower and much more error-prone.

### **Substitution Transition**

This command turns an existing transition into a substitution transition – by relating it to an existing page. Again, most of the work is done by the editor. The user selects the transition and invokes the command. Then the editor:

- makes the hierarchy page active,
- prompts the user to select the desired subpage; when the mouse is moved over a page node it blinks, unless it is illegal (because selecting it would make the page hierarchy cyclic),
- waits until a blinking page node has been selected,
- tries to deduce the port assignment by means of a set of rules which looks at the port/socket names and the port/socket types,
- creates the hierarchy inscription with the name and number of the subpage and with those parts of the port assignment which could be automatically deduced,
- updates the hierarchy page.

### **Replace by Subpage**

This command replaces a substitution transition by the entire content of its subpage. Also, this operation involves a lot of complex calculations and manipulations, but again all of them are done by the CPN editor. The user simply selects the supernode, invokes the command and uses a simple dialogue box to specify the details of the operation (e.g., whether the subpage shall be deleted when no other supernode uses it).

### **Top down or bottom up**

The three hierarchy commands described above can be invoked in any order.

A user with a **top-down approach** would typically start by creating a page where each transition represents a rather complex activity. Then a subpage is created for each activity. The easiest way to do this is to use the Move to Subpage command. Then the subpage automatically gets the correct port places, i.e., the correct interface to the supernode. As the new subpages are modified, by adding places and transitions, the subpages may become so detailed that additional levels of subpages must be added. This is done in exactly the same way as the first level was created.

In contrast to this, a user with a **bottom-up approach** would start by creating a number of pages with the basic components of the modelled system. Later he would model the more abstract layers and relate these to the existing pages by

means of the command that turns transitions into substitution transitions. In practice, it is very rare to find pure top-down or bottom-up modelling. Most users alternate between the two strategies.

### **Groups of objects**

The CPN editor allows the user to work with **groups** of objects. This means that the user can select a *set* of objects and *simultaneously* manipulate all of these, e.g., change attributes, delete the objects, copy them, move them or align them to each other. Groups can be selected in many different ways, e.g., by dragging the mouse over a rectangular area or by pressing the shift key while objects are being selected. Most commands which can be performed on a single object can also be performed on a group, and this has the same effect as when the command is performed on each group member one at a time. The group concept also applies to the page objects on the hierarchy page. This means that by a single command the user can open, close, scroll or scale a *set* of pages.

All members of a group have to belong to the same page and be of the same category, i.e., be all nodes, connectors or regions. Otherwise, there are no restrictions in the way in which groups can be formed.

The value of the group facility can hardly be overestimated. It has a dramatic impact upon the speed and ease by which editing operations can be performed. Many other drawing tools also recognise groups. However, it is often only delete, drag and align commands which can be used to groups, while all the remaining commands can only be used to one object at a time.

### **Different working styles**

The CPN editor is a flexible tool offering the user a large array of different possibilities. As described above, it is possible to make CPN diagrams which *look* very different. However, it is also possible to *construct* each diagram in many different ways. This means that a working style can be chosen, which is suitable for the needs and temperament of the modeller. One example of this flexibility is the many different ways in which the hierarchical relationships between pages can be constructed.

Another example is the fact that the CPN editor allows the user to construct the individual objects of a CP-net in many different ways. Some users prefer to make the net structure first, i.e., the places, transitions and arcs. Later they then add the net inscriptions, i.e., the CPN regions. They either finish one node at a time or finish one kind of CPN regions at a time. They either type from scratch or they copy text from existing regions. Other users prefer to create templates (e.g., a transition with a guard region and a place with a colour set region and an initialization region). Then the diagram is created by copying the templates until the appropriate number of places, transitions and arcs have been made. During this process the new objects are positioned, and if necessary their inscriptions are modified. When a node is being copied, the editor automatically includes the regions and for a group of nodes also the internal connectors.

Most users prefer to work in a way which is a mixture of those described above. Thus it is extremely important to support as many different working styles as possible – so that they can all be performed in a natural and efficient way. Hence the CPN editor has been designed to allow most operations to be performed in several different ways.

### Key and popup regions

To avoid getting too cluttered CPN diagrams, Design/CPN uses a facility known as key and popup regions. The idea is very simple. Instead of a single region with a lot of information, we have both a key region and a popup region. The **key region** is a region of the object to which we want to attach the information. The key region is small. Usually it only contains one or two characters. Its main purpose is to give access to the **popup region** which contains the actual information. The popup region is a region of the key region, and this means that a repositioning of the key also repositions the popup. A double-click on the key region toggles the visibility of the popup region. Hence it is extremely easy to hide and show large amounts of information. Key and popup regions are used for hierarchy inscriptions. The key contains the letters HS, while the popup contains the identity of the subpage and the port assignment (which may be quite large). In the CPN simulator, key and popup regions are used to display the current marking of each place. Here the key contains the number of tokens, while the popup contains a textual representation of the actual colour values. The latter may be very large. There are applications in which some tokens have a colour which is a list of 50,000 records.

The use of key and popup regions is in many respects similar to the use of popup windows. The difference is that the popup regions are objects of the diagram itself. Thus they can be handled in exactly the same way as all other objects, e.g., copied. An attribute of each key region records whether the corresponding popup region is shown, hidden or missing. The diagram default of this attribute determines the initial state of the popup region. A missing popup region can always be created, with the correct information. This is done simply by double-clicking the key.

### Menu commands

Design/CPN has a number of menus. Each menu contains a set of related commands. In the following sections we give a brief description of the different menus.

The **File menu** contains a command to create new diagrams. Moreover, the user can open, close, save and print existing diagrams. It is also possible to save and load individual pages. This means that a page can be moved from one diagram to another. The contents of text files can be loaded into the texts of nodes, and vice versa.

The **Edit menu** contains the standard undo, redo, cut, copy, paste and clear commands known from the Macintosh concept. For the moment, undo and redo only work for a limited set of commands. There is also a command to get detailed information about an individual object, e.g., how many regions it has.

This **CPN menu** contains commands to create places, transitions, arcs, CPN regions and declaration nodes. There are also commands to create and destroy hierarchical relationships between pages and commands to define fusion sets, specify port nodes and perform port assignments. Finally, there is a command to start a syntax check.

The **Aux menu** contains commands to create auxiliary objects of all different shapes. There are also commands to turn auxiliary objects into CPN objects, and vice versa. Finally, there are commands to start and stop the Standard ML compiler.

The **Set menu** contains commands to change attributes, options and the defaults of attributes and options.

The **Makeup menu** contains commands to select, drag and resize objects (this can also be done by direct mouse manipulation). The shape of objects can be changed and nodes can be merged into other nodes or duplicated. There are commands to hide and show regions and to change the graphical layering of objects. Finally, there are commands to jump to the parent object, oldest child object, next object and previous object (in the region and layering hierarchy). This can also be done by means of the arrow keys.

The **Page menu** contains commands to open, close, scroll and scale pages. In addition, there is a command to create new pages and a command to redraw the page hierarchy graph. The latter is used when the hierarchy page becomes too cluttered, e.g., because the user has made a number of manual changes to the automatic layout maintained by Design/CPN.

The **Group menu** contains commands to select different kinds of groups, e.g., all nodes on a given page, all connectors, all regions, or all members of a given fusion set. There is also a command to reselect the most recent group.

The **Text menu** contains commands to manipulate text. There are commands to search for specified text strings and replace them by others – either in the entire diagram, on a single page, or in the selected objects. It is also possible to scroll to the beginning of a large text and to search for matching brackets.

The **Align menu** contains commands that make it easy to align nodes and regions to each other. There are many different possibilities. The objects may, e.g., be positioned vertically below each other, or in such a way that the upper edges are horizontally aligned. Objects may also be positioned with equal distances between them – on a line or on a circle.

## Different skill levels

Design/CPN can be used at many different skill levels. Casual and novice users only have to learn and apply a rather small subset of the total facilities. The more frequent and experienced users gradually learn how to use the tool more efficiently. All the more commonly used commands can be invoked by means of **key shortcuts**. Many commands have one or more **modifier keys** allowing the user to perform an operation, which otherwise would require several commands.

The user can create a **palette page** from which different objects can be copied as they are needed. The user simply clicks at the desired object, and then at all the positions where he wants a copy of the object. Palette pages are created and modified like all other pages. They simply have an attribute telling that they are palettes. This means that during the modelling, the user can create new palette pages and modify existing palettes. He can also move, size and scale the palettes as he wants. Palettes are useful, for example, when a company or a group of researchers want to impose a common standard for the graphics of their CPN diagrams.

## Syntax restrictions

The CPN editor is syntax directed. This means that it recognizes the structure of CP-nets and automatically prevents the user from making many kinds of syntax errors. This is done by means of a large number of **built-in** syntax restrictions. All the built-in restrictions deal with the net structure and the hierarchical relationships. As an example, it is impossible to make an arc between two transitions, or between two places. It is impossible to give a transition a colour set region, or give a place two colour set regions. It is also impossible to create cycles in the substitution hierarchy, and to make an illegal port assignment, involving nodes which are not sockets or ports, or are positioned on a wrong page.

The CPN editor also operates with **compulsory** syntax restrictions. These restrictions are necessary in order to guarantee that a CPN diagram has a well-defined semantics – and hence they must be fulfilled before a simulation (or other kinds of analysis) is performed. Many of the compulsory restrictions deal with the net inscriptions and thus with CPN ML. As an example, it is checked that each colour set region contains the name of a declared colour set *S*, and that all the surrounding arc expressions have a type that matches *S*. Many of the compulsory syntax restrictions could have been implemented as built-in restrictions. As an example, we could have demanded that each arc expression always has the correct type. This would, however, imply that a colour set cannot be changed without simultaneously changing all the surrounding arc expressions.

The CPN editor has also **optional** syntax restrictions, i.e., restrictions which the user may choose to impose upon himself. As an example, the user may specify that he wants all places to have a non-empty name and that all place names (on a page) must be unique.

## Syntax checking

The possibility of performing an automatic syntax check means that the user has a much better chance of getting a consistent and error-free CPN diagram. This is very useful – also in situations where the user is not interested in performing a simulation or other kinds of machine assisted analysis.

The detailed checking of the expression syntax and the type consistency is performed by the Standard ML compiler. When an error is found, an error message is produced. The message may look as follows:

```
C.11 Arc Expression must be legal
Type clash in:      x : (P ms)
Looking for a:      P ms
I have found a:     U
««135»»
```

The error message tells us that we have an illegal arc expressions. C11 means the 11th kind of compulsory restriction, while ««135»» is a hypertext pointer – which allows the user to jump to the error position (i.e., to the arc with the erroneous arc expression). To perform a hypertext jump, the user positions the cursor inside the hypertext pointer and presses an arrow key.

The syntax check is incremental. When the user modifies a part of a CPN diagram, the CPN editor only rechecks those objects that have been changed – or are related to changed objects. As an example, the change of a colour set means that the initialization expression and all the surrounding arc expressions have to be rechecked. If the place belongs to a fusion set or is an assigned port or socket node, it must also be rechecked that the colour sets are identical and all initialization expressions equivalent.

If the user changes the content of the declaration node, all colour sets are re-declared. This means that the entire CPN diagram has to be rechecked. To avoid using too much turn-around time for such total rechecks, the CPN editor allows the use of a temporary declaration node – in which new declarations can be added to the original declarations without enforcing a total recheck.

## Use of names

Each page has a name and a number. In addition, it is possible (and advisable) to give names to each place and each transition.

Names do not influence the semantics of CP-nets. However, they are used in the feedback information from Design/CPN to the user, e.g., in the page hierarchy graph and in the hierarchy inscriptions. To make the feedback unambiguous, it is recommended to keep names unique, but this is not enforced (unless the user activates an optional syntax restriction). For places and transitions it is sufficient to have names that are unique on each individual page.

Some users tend to have a large number of transitions and places without names. This is no problem for Design/CPN, but it may make it difficult for the user to read the feedback.

### **Abstract data base**

When the user creates a CPN diagram, the editor stores all the semantic information in an abstract data base, from which it can be retrieved by Design/CPN and other analysis programs. The abstract data base was designed as a relational data base, but for efficiency it is implemented by means of a set of list structures – making the most commonly used data base operations as efficient as possible. The existence of the abstract data base makes it much easier to integrate new and existing editors and analysis programs with the Design/CPN tools. For this purpose CPN ML provides three sets of functions. The first set reads the information of the abstract data base, e.g., the colour set of a place. The second set creates pages and auxiliary objects (which have a graphical representation, but no representation in the abstract data base). Finally, the third set converts auxiliary objects to CPN objects (which means that they become included in the abstract data base). The three sets of functions make it easy to write programs which translate CPN diagrams into textual or graphical representations of other Petri net tools, and vice versa.

## Chapter 2

# CPN Simulator

The CPN simulator allows the user to simulate hierarchical CP-nets – with or without time. The simulation may be interactive or fully automatic. Design/CPN is designed to work with complex CP-nets. Many of the industrial applications use CPN models which have more than 100 page instances, each with 5–25 places/transitions. Fortunately, it turns out that a CP-net with 100 page instances usually simulates almost as fast as a CP-net with only a single page instance (when speed is measured in terms of the number of occurring binding elements).

### **Integration with the CPN editor**

The CPN editor and CPN simulator are two different parts of Design/CPN. However, they are closely integrated with each other. In the editor it is possible to prepare a simulation, e.g., set the many options which in detail determine how the simulation is performed. In the simulator it is possible to perform simple editing operations, e.g., modify arc expressions, guards, initialization expressions, code segments and time regions. Furthermore it is possible to change the attributes of CPN objects and add auxiliary objects.

Most modellers use simulation to test the different parts of their CPN diagram as these are constructed. This is similar to a programmer who debugs the individual parts of his program as he writes them. To support this work style, Design/CPN makes it possible to switch reasonably fast between the editor and the simulator. Design/CPN also supports simulation of selected parts of a large CPN diagram, e.g., a one or more new pages, which the user has just constructed and now wants to test. Without such selection possibilities it would be difficult to build large CPN models. The selection is done in Mode Attributes (by means *Prime Page* and *Do Not Include in Simulation*).

### **Page instances**

All instances of a page have the same net structure and the same net inscriptions. However, during a simulation the page instances will have different current markings and different sets of enabled transitions. Hence the CPN simulator must show the individual page instances, and not just the individual pages.

There is a window for each page. In this window the simulator displays all the instances of the page – one at a time. The title bar of the page window specifies the identity of the currently displayed page instance. The identification consists of a non-negative instance number, the page name, and a list containing all

supernodes. The user may use the Switch Instance command to display other page instances. However, during an interactive simulation this is usually not necessary, because Design/CPN automatically displays those instances on which transitions are occurring.

### **Automatic simulations**

Design/CPN supports two different kinds of simulations. An automatic simulation is fast, because everything is done by Design/CPN – without any user involvement and without any kind of graphical feedback on the CP-net. It is only at the moment where the automatic simulation stops (because some specified stop criteria becomes satisfied) that the CPN diagram is updated with the new marking and the new enabling. The results of the simulation can be recorded in a number of different ways. One very easy way is to use the Save Report command which creates a text file listing all the occurred binding elements (provided that the reporting facilities are turned on in General Simulation Options). Another possibility is to write selected results to an output file or to use a number of business charts. Output files and charts are manipulated from code segments (see below).

### **Interactive simulations**

An interactive simulation is slow, the user can choose between enabled transitions and between enabled bindings, he can set breakpoints and he can ask for more or less detailed feedback.

The user can, at any time, switch between interactive and automatic simulation (in General Simulation Options). This means that he can make a few interactive steps, and then start an automatic simulation. When this is finished the current marking and the enabling is updated. The user can investigate these and perhaps decide to do some more interactive/automatic steps.

During an interactive simulation the occurring transitions and bindings may be chosen in two different ways – either by the user or by the CPN simulator. In both cases it is Design/CPN that performs the “hard” work: the calculation of the enablings and the calculation of the effects of the occurring steps. A set of binding elements (selected for execution) is called an **occurrence set**.

### **Interactive simulation with manual selection of occurrence sets**

Design/CPN indicates the current enabling by highlighting those transition instances which have at least one enabled binding element. The highlighting may, e.g., be done by increasing the line thickness of the transition or by changing the line colour. This is determined by the user (in Transition Feedback Options).

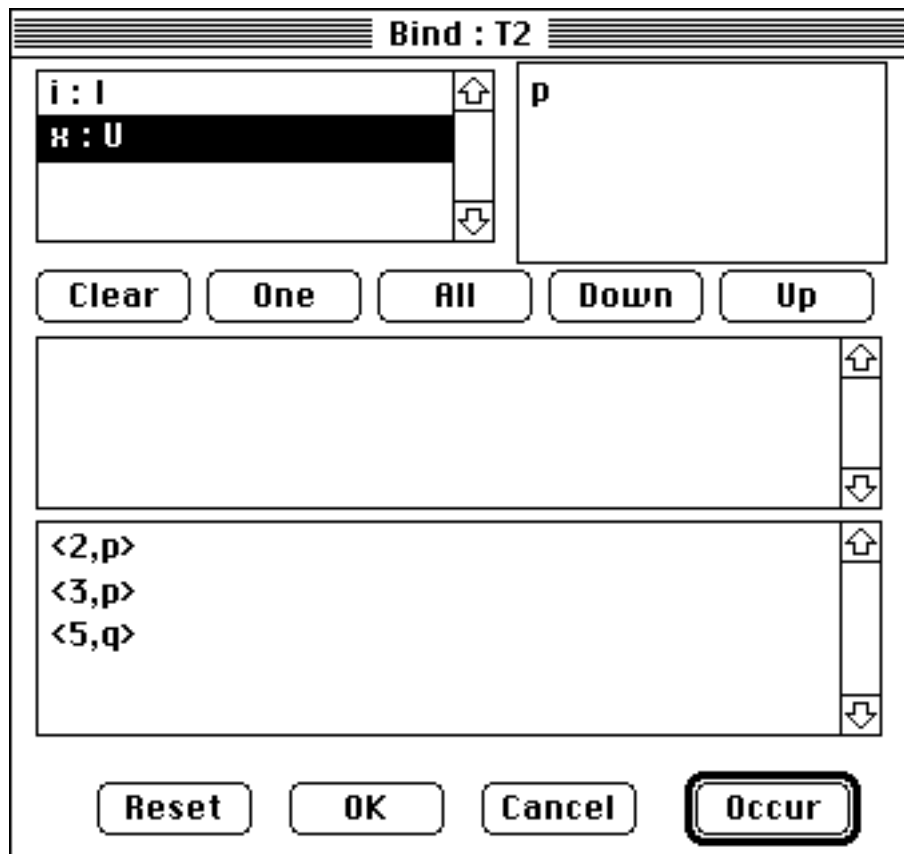
To construct a step the user chooses those binding elements which he wants to include in the step. To include a binding element the user must specify a transition instance and a binding. First he selects a transition instance. This is done by a mouse click on the transition – at the correct page instance. Then the Bind

command is invoked. This displays a dialogue box which may look as shown below.

The title bar contains the name of the transition. The dialogue box has three different parts. The upper part is a **working area** with a list box and a text field. The list box has a line for each variable of the transition. In this case there are two such variables: *i* of type I and *x* of type U. The text box is used to define/inspect the colour value to be bound to the variable selected in the list box. The middle part of the dialogue box contains the **included bindings**, i.e., those binding elements (of the transition instance) which we have already included in the step. In our case, we have not yet included any such bindings, and hence the middle part is empty. The bottom part of the dialogue box contains the **proposed bindings**, i.e., those bindings found by pressing the *All* button (see below).

To define a binding we must specify a value for each variable of the transition. This can be done in several different ways:

- One possibility is to type the desired colour values into the text box in the upper right corner (while the corresponding variable is selected in the list box). In our example, the variable *x* has been bound to *p*. Design/CPN will check that the entered value is of the correct type (if not you will hear a beep).
- A second possibility is to press the *One* button. This will instruct the simulator to make a random choice between the enabled bindings (of the selected transi-



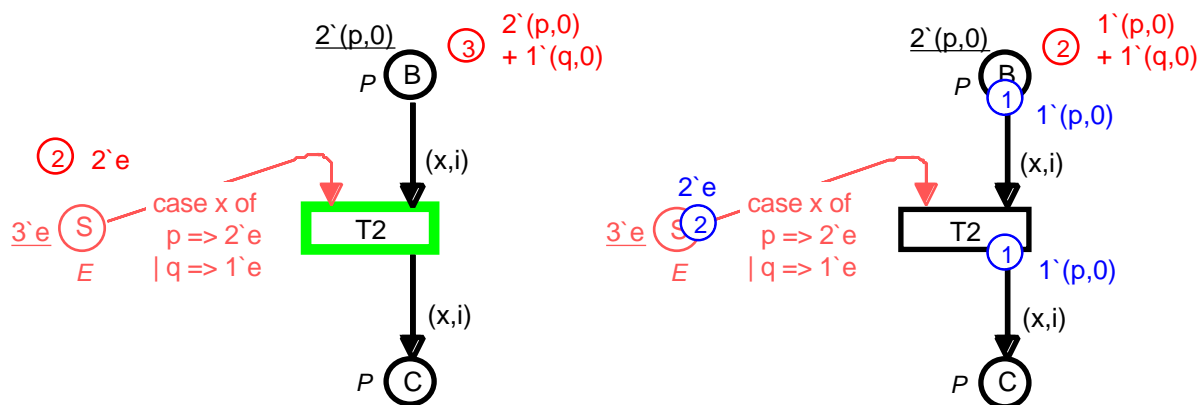
tion instance). The chosen binding will be inserted in the top most part of the dialogue box – where it can be inspected/changed.

- A third possibility is to press the *All* button. This will instruct the simulator to display all the enabled bindings (of the transition instance) into the bottom part of the dialogue box.

The user can move a binding from one part of the dialogue box to another. To do this the user selects the corresponding binding (the binding in the working area is selected when nothing is selected in the two other parts). Then either the *Up* or *Down* button is pressed, and this will move the binding in the desired direction. *Down* moves from the upper part to the middle part, from the middle to the lower, and from the lower to the upper. Analogously, *Up* moves from the lower part to the middle part, from the middle to the upper, and from the upper to the lower part.

Each time a binding element is included in the step, the simulator reserves the input tokens needed by the binding element. This is done by subtracting the tokens from the current marking. If the reservation cannot be made, the binding element cannot be included. This means that it is impossible to construct a step which is disabled in the current marking.

Design/CPN is able to show the detailed effect of the included binding elements. This is done by displaying the set of input tokens and the set of output tokens – in key/popup regions of the corresponding arcs. For a step with transition T2 and binding  $\langle x=p, i=0 \rangle$  you will get the following picture (provided that Interactive Simulation Options is set to Update Graphics During Substeps):



The left-hand side shows the transition before the binding is included, while the right-hand side shows it after the inclusion (but before the occurrence). It should be noticed that Design/CPN also recalculates the enabling. In the right-hand side transition T2 is no longer highlighted. This indicates that we cannot add more binding elements for T2 (unless we delete the one which we have already added). If there had been two additional e-tokens on S, transition T2 would have remained highlighted – and we could either include an appearance of  $(T2, \langle x=q, i=0 \rangle)$  or an additional appearance of  $(T2, \langle x=p, i=0 \rangle)$ .

When the user has specified the desired binding elements for the chosen transition instance, he either presses the *OK* or the *Occur* button. When *OK* is

pressed the simulator remains in the step construction phase. This means that the user can add binding elements for other transition instances. When *Occur* is pressed the constructed step is executed. The new marking and the new enabling are calculated, and after that the user starts the construction of the next step.

It is possible to mix the three ways to specify bindings. As an example, the user may enter colour values for some of the variables – before *One* or *All* is pressed. Then the system only calculates bindings which are consistent with the specified colour values.

The manual construction of binding elements (described above) is rather time consuming. Hence, it is primarily used during the initial design and debugging of the CPN model. There are many shortcuts. As an example, a single key combination makes it possible to include a random binding (for the selected transition instance) without opening a dialogue box. This is useful when a transition instance has only one enabled binding – or when the user does not care which binding is chosen.

### **Interactive simulation where Design/CPN selects the occurrence sets**

As mentioned above the user does not need to choose the binding elements. Instead he can ask Design/CPN to do perform this job – using a random number generator. The selection between pages is fair, in the sense that all pages (with enabled binding elements) have an equal chance to contribute to the step. Analogously, there is a fair selection among the page instances (of a chosen page), among the transitions (of a chosen page instance) and among the enabled bindings (of a chosen transition).

The user may specify how large he wants the individual steps to be. As one extreme he may want each occurring step to have exactly one binding element. As another extreme each occurring step may be demanded to be maximal (in the sense that no other binding element can be added without disabling the step). The last possibility is sometimes referred to as the maximal occurrence rule, and it has been used in different kinds of Petri net models, for example, some of those which have a time concept. Between the two extremes there are many other possibilities. The detailed selection strategy is specified in Occurrence Set Options.

A “page percentage” specifies how eager Design/CPN should be to include binding elements from more than one page. Let us assume that the page percentage is set to 70. Then there is a 70% chance that the CPN simulator will try to include a second page, after which there is a 70% chance that it will try a third page, and so on. When the page percentage is 0, each step will contain binding elements from exactly one page. When the page percentage is 100, each step will contain binding elements from a maximal set of pages (in the sense that it is impossible to include binding elements from another page without disabling the step).

Analogously, there are percentages for page instances, transitions, different bindings and identical bindings. The page instance percentage specifies how eager the CPN simulator should be to include more than one page instance (of each

chosen page). The transition percentage specifies how eager the simulator should be to include more than one transition (of each chosen page instance). Finally, the two binding percentages specify how eager the simulator should be to include different bindings and to include the same binding more than once.

When all five percentages are 100, we have the maximal occurrence rule. When they are all 0, each step contains exactly one binding element. In all other cases random drawings determine the actual size of the step. It should also be noted that the percentages influence each other. As an example, let us assume that all five percentages have a high value. This implies that each included page contributes with many binding elements and hence use many of the available tokens. Therefore it may be impossible to include more than one page (even though we have a high page percentage).

### **Observation of interactive simulations**

For a big model, it is obvious that we cannot arrange the page windows in such a way that they do not overlap. Moreover, it should be remembered that each window only shows one page instance at a time. Hence it is necessary that Design/CPN – during an interactive simulation – rearranges the screen image, so that the user can always see the interesting parts of the CPN diagram. Otherwise, it would be impossible to follow an automatic simulation.

A step may involve several transitions. Then the effects are shown for one transition at a time, and before a transition occurs, Design/CPN automatically makes the following rearrangements:

- The interesting page is made active (and hence displayed on top of all the other windows).
- The interesting page instance is displayed in the window.
- The window is scrolled, to guarantee that the interesting transition is in the visible part of the window (this only happens if Auto Scroll is set in Interaction Options).

By means of one of the Mode Attributes the user can say that he does not want to observe all page instances. In that case, the simulator still executes the transitions of the non-observed page instances, but without rearranging the screen. This means that the user cannot see the transitions of the non-observed page instances (unless they happen to be visible without any rearrangements). By closing a window and making all its page instances non-observable, a lot of graphical updating is avoided, and hence the simulation speed is improved.

### **Graphical feedback during interactive simulations**

It is possible to vary the amount of graphical feedback provided by the CPN simulator (this is done in Interactive Simulation Options). In the most detailed mode the user watches all the occurring transitions. He sees the input tokens, the output tokens, and the current marking. This means that Design/CPN must up-

date and redraw many different graphical objects. This takes time and hence there is a trade-off between information and speed.

The feedback mechanisms are controlled by a number of options. As an example, it is possible to omit the input tokens and output tokens altogether. It is also possible to include them, but with missing popups. The user can then only see the number of input and output tokens. However, he can always stop the simulation and then double-click one of the input/output key regions. This will make the popup visible.

If the user carefully chooses those page instances which he wants to observe and those feedback mechanisms which he wants to be in effect, he can speed up an interactive simulation – often by as much as 100%.

### **Breakpoints in interactive simulations**

The automatic rearrangements guarantee that the occurring binding elements are visible. However, to inspect the simulation in more detail, e.g., to see the colour values of the input and output tokens, it is necessary to be able to stop the simulation. This is done by means of breakpoints, which allow the user to investigate the state of the CPN model.

One kind of breakpoint makes the simulation pause immediately before each transition occurrence. At this time the occurring transition has been made visible and the user can see the involved input and output tokens (as shown above). Another kind of breakpoint implies a pause immediately after each transition occurrence. At this time the current marking of the surrounding places has been updated. Finally, it is possible to have a breakpoint at the end of each step. Breakpoints are specified in Interactive Simulation Options.

### **Limitations of automatic simulations**

Automatic simulations are usually several hundred times faster than interactive simulations. To obtain this it has been necessary to disable many of the features described above. The user can not choose the occurrence sets, he does not get any graphical feedback, and he can not use breakpoints. In an automatic simulation each step contains exactly one binding element (i.e., one transition with one binding). There is only a semi-fair selection between enabled transitions instances (because transitions are tested in a fixed sequence). However, there is a fair selection between the enabled bindings of a single transition instance (unless Fast Automatic is chosen in General Simulation Options).

### **Simulation of timed CP-nets**

To investigate the **performance** of systems, i.e., the speed by which they operate, it is convenient to extend CP-nets with a time concept. To do this, we introduce a **global clock**. The clock values represent the **model time**, and they may either be discrete (which means that all clock values are integers) or continuous (which means that the clock values are reals). In addition to the token colour, we

allow each token to have a **time stamp** attached to it. The time stamp describes the *earliest* model time at which the token can be removed by a binding element.

The execution of a timed CP-net is time driven, and it works in a similar way as the event queues found in many programming languages for discrete event simulation. The system remains at a given model time as long as there are enabled binding elements which are ready for execution. Then the system advances the clock to the next model time at which enabled binding elements can be executed. Each marking exists in a closed interval of model time (which may be a point, i.e., a single moment).

For a timed CP-net we demand that each step consists of binding elements which are both enabled and ready. Hence the possible occurrence sequences of a timed CP-net always form a subset of the possible occurrence sequences of the corresponding untimed CP-net. This means that we have a well-defined and easy-to-understand relationship between the behaviour of a timed CP-net, and the behaviour of the corresponding untimed CP-net. As an example, it is easy to see that the timed net cannot have a dead marking, unless the untimed net has one.

The time delays may depend upon the binding, i.e., upon the colours of the input and output tokens. The delays are specified by means of Standard ML expressions which may use functions (e.g., representing an exponential distribution or random selection from a given interval).

## Code segments

When we simulate a CP-net, it is sometimes convenient to be able to equip some of the transitions with a **code segment**, i.e., a sequential piece of code which is executed each time a binding element of the transition occurs. Code segments can, e.g., be used for the following purposes:

- Data can be read from an input file. This makes it possible to have a simulation model which can be run on different sets of data without changing the initialization expressions. It is also possible to display a dialogue box, in which the user can type the data.
- Data can be written on an output file. This makes it possible to save data, e.g., for later analysis in a spreadsheet program.
- Charts and other forms of graphics can be updated – either during the simulation or at the end of a simulation run. In this way it is possible to make customised feedback representing the simulation results in an immediately comprehensible way. For more details see “Examples of Design/CPN Models” and “Design/CPN Libraries” on the Design/CPN WWW pages.
- A variable of the transition can be determined by the code segment – instead of being bound in the ordinary way.
- Code segments may use reference variables, which can be read and updated. This possibility should be used with great care. It implies that two transitions can influence each other – even when they are totally unrelated (with respect to the net structure).

## Simulation modes

The different simulation modes may be freely combined. This means that a simulation can be:

- interactive or automatic,
- with or without code segments,
- with or without time.

This gives us 8 basic kinds of simulations, each having a large number of variants – due to the other simulation options (which determine, e.g., the amount of feedback and the construction of occurrence sets).

## Simulation menu

Design/CPN has a Sim menu which contains all the commands used during a simulation. The Sim menu replaces the CPN menu – otherwise the menus are the same as in the editor.

We have already described the Bind command. It allows the user to construct occurrence sets, to be used during manual simulation. However, it is not always necessary to start such a construction from scratch. Instead the user can invoke a command that proposes an occurrence set, calculated in the way described above. The proposed occurrence set can be investigated, and perhaps modified, before it is executed.

There are also commands to execute an individual step, to start interactive/automatic simulations and to stop/continue an interactive simulation. The duration of a simulation is determined by a number of **stop criteria** (in General Simulation Options).

The Change Marking command makes it possible to add and remove tokens during a simulation. This is very convenient, e.g., when some of the initialization expressions have been forgotten. The command can also be used to make a detailed test of a transition. The modeller can put different tokens on the input places and then use the *All* button of the Bind command to test that the transition has the expected enabling. There is also a command which allows the user to return to the initial marking, i.e., forget all the steps which have been executed.

Finally, there are commands to save and clear simulation reports, to update charts and to start a reswitch – when, some part of the model (e.g., an arc expression) has been modified). Two commands (in the File menu) allow the user to save and load system states via files. This makes it, e.g., possible to save an interesting marking and then later return to it without having to repeat the steps.

## Calculation of enabled bindings

The generality of the CPN ML language implies that the user can construct syntactically legal CPN diagrams which Design/CPN is unable to simulate – because it cannot calculate the set of enabled bindings. To execute a transition Design/CPN must determine a binding for the transition. To do this it is de-

manded that each variable  $v$  (of the transition) fulfils at *least* one of the following conditions:

- (i)  $v$  appears in an input arc expression which is simple enough to be used to bind  $v$ . This means that the arc expression must be a **pattern**, i.e.:
- a single variable, e.g.,  $v$ ,
  - a tuple, e.g.,  $(v,3)$  or  $(x,v)$ ,
  - a record, e.g.,  $\{\text{sen}=v,\text{rec}=\text{S}(3)\}$ ,
  - a list, e.g.,  $v::\text{tail}$ ,
  - a union, e.g.,  $\text{Floor}(v)$ .

It is allowed to nest tuples, records and lists inside each other, e.g.,  $(x,(v,y))::\text{tail}$ , and it is also allowed to use the same variable repeatedly, e.g.,  $(v,v)$  or  $v::(v::\text{tail})$ . Finally, it is allowed to combine expressions by means of the  $\`$  and  $+$  operations, e.g.,  $2\`(v,x)+1\`(v,y)$ .

A simple arc expression cannot contain functions or other operations. This means that  $f(v)$  cannot be used to bind  $v$ , while  $1\`(f(v),x)+2\`(y,z)$  can only be used to bind  $y$  and  $z$ .

- (ii)  $v$  has a colour set with at most 100 possible values or a “declare ms” clause. Then Design/CPN tries all possible values.
- (iii)  $v$  appears in a guard  $[\dots, \text{ptn} = \text{expr}, \dots]$  or  $[\dots, \text{expr} = \text{ptn}, \dots]$  where  $\text{ptn}$  is a pattern (without  $\`$  and  $+$ ), while  $\text{expr}$  is an expression in which all variables satisfy (i), (ii) or (iii).
- (iv)  $v$  is bound by a code segment..

It is very seldom that the demands in (i)–(iv) present any practical problems. Most net inscriptions fulfil the conditions, and otherwise they can usually be rewritten without changing the semantics.

## Chapter 3

# Occurrence Graph Tool

The Occurrence Graph tool allows the user to construct and analyse occurrence graphs of hierarchical CP-nets – with or without time. An occurrence graph is a directed graph with a node for each reachable marking and an arc for each occurring binding element. Occurrence graphs are also known as state spaces, reachability graphs or reachability trees. With the present version of the Occurrence Graph tool, we have constructed occurrence graphs with more than one hundred thousand nodes and more than one million arcs.

### Integration with the CPN simulator

The Occurrence Graph tool is closely integrated with the CPN simulator. By a single command it is possible to “move” a marking from the Occurrence Graph tool to the simulator. This is very useful, since it allows the user to inspect the marking directly on the graphical representation of the CP-net. It is possible to see the enabled transition instances, investigate their bindings and make simulations. It is also possible to “move” a marking from the simulator to the Occurrence Graph tool – where it becomes a node.

It is possible to make an occurrence graph for part of a large model. This is done in exactly the same way as in the CPN simulator, i.e., in Mode Attributes (by means of *Prime Page* and *Do Not Include in Simulation*).

Occurrence graphs can be constructed with or without time and with or without code segments. Code segments should, however, be used with care. If they have side effects which must be executed in a particular order, it makes no sense to construct an occurrence graph (because this will execute the code segments in a wrong order).

### Construction of occurrence graphs

The construction of the occurrence graphs is totally automatic, and the user can specify whether he wants the entire occurrence graph to be calculated or only a part of it. This is done by means of stop and branching criteria.

The **stop criteria** tell us how large the constructed graph should be. As an example, it is possible to specify that the construction should finish when a certain amount of real time has been used or when a certain number of nodes have been constructed. It is also possible to specify an Standard ML function with a more complex stop criteria, e.g., telling that the construction should stop if a dead marking is encountered.

The **branching criteria** imply that it is possible only to develop some of the immediate successors of a given marking. As an example, the user may specify that he wants the construction to investigate at most two binding elements (for each enabled transition instance).

When a partial graph has been constructed, by using a set of stop criteria or a set of branching criteria, it is later possible to continue the construction (with or without stop and branching criteria).

### Standard report

When an occurrence graph has been constructed it can be analysed in different ways. The easiest approach is to use the Save Report command to generate a standard report providing information about all standard CPN properties:

- Statistics (size of occurrence graph and Scc-graph (see below)).
- Boundedness Properties (integer and multi-set bounds for place instances).
- Home Properties (home markings).
- Liveness Properties (dead markings, dead/live transition instances).
- Fairness Properties (impartial/fair/just transition instances).

The command invokes a dialogue box allowing the user to specify the kind of information which he wants to obtain. This is done by choosing one or more of the possibilities mentioned above.

### Standard queries

To investigate the CP-net in more detail, the Occurrence Graph tool has a large number of query functions, i.e., Standard ML functions providing information about the behavioural properties of specified place instances, transition instances and markings. Below we give some examples of query functions. Each of them is executed in a few seconds (even for large occurrence graph).

- *Reachable* determines whether there exists an occurrence sequence between two specified markings.
- *AllReachable* determines whether all the reachable markings are reachable from each other.
- *UpperInteger* calculates the maximal number of tokens at a specified place instance.
- *UpperMultiSet* calculates the upper multi-set bound of a specified place instance (i.e., the smallest multi-set which is larger than all the markings of the place instance).
- *LowerInteger* calculates the minimal number of tokens at a specified place instance.
- *LowerMultiSet* calculates the lower multi-set bound of a specified place instance (i.e., the largest multi-set which is smaller than all the markings of the place instance).

- *HomeMarkingExists* determines whether the CP-net has any home markings.
- *HomeMarking* determines whether a specified marking is a home marking, i.e., whether it can be reached from all reachable markings.
- *Initial HomeMarking* determines whether the initial marking is a home marking.
- *ListHomeMarkings* returns a list with all those markings that are home markings.
- *HomeSpace* determines whether a specified set of markings is a home space, i.e., whether, from each reachable marking, it is possible to reach at least one of the specified markings.
- *MinimalHomeSpace* returns the minimal number of markings which is needed to form a home space.
- *DeadMarking* determines whether a specified marking is dead, i.e., has no enabled binding elements.
- *ListDeadMarkings* returns a list with all those markings that are dead.
- *TIsDead* determines whether a specified set of transition instances is dead in a specified marking.
- *BEsDead* determines whether a specified set of binding elements is dead in a specified marking.
- *ListDeadTIs* returns a list with all those transition instances that are dead.
- *TIsLive* determines whether a specified set of transition instances is live.
- *BEsLive* determines whether a specified set of binding elements is live.
- *ListLiveTIs* returns a list with all those transition instances that are live.
- *TIsFairness* determines whether a specified set of transition instances is impartial, fair or just.
- *BEsFairness* determines whether a specified set of binding elements is impartial, fair or just.
- *ListImpartialTIs* returns a list with all those transition instances that are impartial.
- *ListFairTIs* returns a list with all those transition instances that are fair.
- *ListJustTIs* returns a list with all those transition instances that are just.

### Customised queries

In addition to the standard queries, it is also possible to make special-purpose queries – testing properties that are particular to the CP-net in question. For this purpose the Occurrence Graph tool has a function called **SearchNodes**. It takes six arguments:

- The first argument specifies the **search area**, i.e., the part of the graph which should be searched. This can, e.g., be the entire graph or a strongly connected component (see below).
- The second argument specifies the **predicate function**. It maps each node into a boolean value. Those nodes which evaluate to false are ignored; the others take part in the further analysis.

- The third argument specifies the **search limit**. It is an integer, and it tells us how many times the predicate function must be true before we terminate the search.
- The fourth argument specifies the **evaluation function**. It maps each node into a value, of some type A. The evaluation function is used to those nodes (of the search area) for which the predicate function is true.
- The fifth argument specifies the **start value**. It is a constant, of some type B.
- The sixth argument specifies the **combination function**. It maps from  $A \times B$  into B, and it describes how the individual results (obtained by the evaluation function) is combined to yield the final result.

It should be noticed that the predicate function, the evaluation function and the combination function are all written by the user. This means that they can be arbitrarily complex.

In addition to SearchNodes there is a function called **SearchArcs**. It takes the same six arguments, and it works in a similar way – except that it searches arcs, instead of nodes. The search functions may seem a complicated. However, they are also extremely general and powerful. This is illustrated by the fact that all the standard queries mentioned above are implemented by means of the search functions. This typically takes 2- 5 lines of Standard ML code.

## Temporal logic

Queries can also be made by means of a CTL-like temporal logic. This is provided via a Standard ML library. It is possible to formulate queries about states, but also queries about state changes (e.g., the occurrence of certain transitions). For more details see the “Design/CPN Libraries” on the Design/CPN WWW pages.

## Occurrence graphs with equivalence classes

It is possible to construct and analyse occurrence graphs with equivalence classes, i.e., graphs where each node represents a class of closely related system states while each arc represents a class of related state changes. Occurrence graphs with equivalence classes are often much smaller than the corresponding ordinary occurrence graphs. Nevertheless they contain the same information and can be used to answer the same kinds of queries. For more details see the “Design/CPN Libraries” on the Design/CPN WWW pages.

## Strongly connected components

A strongly connected component (of a directed graph) is a *maximal* subgraph in which all nodes are reachable from each other. The strongly connected components (SCCs) are pairwise disjoint. For each occurrence graph Design/CPN calculates an **SCC graph**, i.e., a graph with a node for each strongly component and an arc for each occurrence graph arc starting in one component and ending in another.

For cyclic CP-nets we usually have an SCC-graph which is much smaller than the occurrence graph. Often there is only one SCC or a few – while the occurrence graph may have many thousand nodes. By means of the SCC graph a number of behavioural properties can be investigated in a very efficient way – because the SCC graph is small. As an example we can determine whether there are home markings or not by counting the number of terminal SCCs. We can also determine whether a transition instance is live. This is done by checking whether it appears in each terminal SCC.

### **Graphical display of occurrence graphs**

The constructed occurrence graphs are usually large. Hence, it is useless to draw the entire graph. However, it often makes sense to display selected parts of it, e.g., the nearest surroundings of some interesting nodes/arcs – found via queries.

The user specifies the nodes and arcs which he wants the tool to display. This can be done by explicitly listing the desired nodes/arcs. It can also be done by asking for the successors or predecessors of some already displayed nodes. The new nodes are drawn with a default position and with default attributes. The objects of the occurrence graph are ordinary graphical objects, and this means that the user can change the attributes. Usually, he will modify the positions as he adds new nodes – to get a nice layout without too many crossing arcs.

The user determines the text strings to be displayed for the individual nodes and arcs. This is done by specifying two Standard ML functions. One of these maps from nodes into text strings, while the other maps from arcs into text strings. Usually, the first function gives a condensed representation of the marking (of the node), while the second gives a condensed representation of the binding element (of the arc). Each text string is displayed in a region of the corresponding node/arc, and the region can be made visible/invisible by double clicking the node/arc.