CPN/Tools: A Post-WIMP Interface for Editing and Simulating Coloured Petri Nets

Michel Beaudouin-Lafon, Wendy E. Mackay, Peter Andersen, Paul Janecek, Mads Jensen, Michael Lassen, Kasper Lund, Kjeld Mortensen, Stephanie Munck, Anne Ratzer, Katrine Ravn, Søren Christensen and Kurt Jensen

Department of Computer Science
University of Aarhus
IT-Parken, Aabogade 34
8200 Aarhus N - Denmark
E-mail: cpn2000@daimi.au.dk

ABSTRACT
CPN/Tools is a major redesign of the popular Design/CPN tool from the University of Aarhus CPN group. The new interface is based on advanced, post-WIMP interaction techniques, including bi-manual interaction, toolglasses and marking menus and a new metaphor for managing the workspace. It challenges traditional ideas about user interfaces, getting rid of pull-down menus, scrollbars, and even selection, while providing the same or greater functionality. It also uses the new and much faster CPN simulator. The first internal release of CPN/Tools was made in April 2000 and the first public release is expected in October 2000. CPN/Tools requires an OpenGL graphics accelerator and will run on all major platforms (Windows, Unix/Linux, MacOS).

KEYWORDS: Coloured Petri Nets, graphical editor, Design/CPN, instrumental interaction, OpenGL.

INTRODUCTION
Interaction techniques for desktop workstations have changed little since the creation of the Xerox Star in the early eighties. The vast majority of today's interfaces are still based on a single mouse and keyboard to manipulate windows, icons, menus, dialog boxes, and to drag and drop objects on the screen. While these WIMP interfaces (Windows, Icons, Menus, Pointing) are now ubiquitous, they are also reaching their limits: as new applications become more powerful, the corresponding interfaces become more complex. Some users are at a breaking point and are less and less able to cope with new software releases [14]. Others have begun to actively reject software upgrades and cling to older versions of products (survey of Microsoft users, Business Week, 5 July, 1999).

New interaction techniques, such as toolglasses [4] and marking menus [11], have been proposed to reduce this trade-off between power and ease-of-use. Yet such post-WIMP interaction techniques tend to be developed in isolation, as the focus of a particular research project. As a result, they have not made it into commercial tools even though they have been shown to be significantly more efficient than traditional techniques. CPN/Tools is the first real-size application to combine such advanced interaction techniques into a consistent interface. The goal of this project is two-fold: first, it will provide the CPN community with a new, cutting-edge interface to edit and simulate Coloured Petri Nets; second, it paves the way to a new generation of post-WIMP applications that will take advantage of recent advances in graphical interfaces.

The CPN2000 project
CPN/Tools is a complete redesign of Design/CPN [10], a graphical editor and simulator of Coloured Petri Nets (CPNs) developed at Meta Software (USA) and the University of Aarhus (Denmark) over the past 10 years. Design/CPN has a standard WIMP interface, based on direct manipulation, menus and dialog boxes (Figure 1). It is in use by over 600 organizations around the world, in both academia and industry. Production CPNs can have over a thousand places, transitions and arcs, structured into a hundred modules or more.

The CPN2000 project started in February 1999. We used a highly participatory design process, involving the users throughout the design process [15, 9]. Version 1 of CPN/Tools was released in April 2000 and is in use by a small group of CPN designers for production work. Version 2 is planned for October 2000 and will be released to a selected set of users outside the project.

The CPN/Tools interface uses a combination of traditional, recent and new interaction techniques, e.g. tool palettes, toolglasses, and magnetic guidelines. Integrating these interaction techniques together in a consistent way in a single tool proved quite challenging. To our knowledge, this had never been done before. We wanted to design a system that would strike a better balance between power and simplicity than current WIMP interfaces. This led us to define three design principles: reification, polymorphism

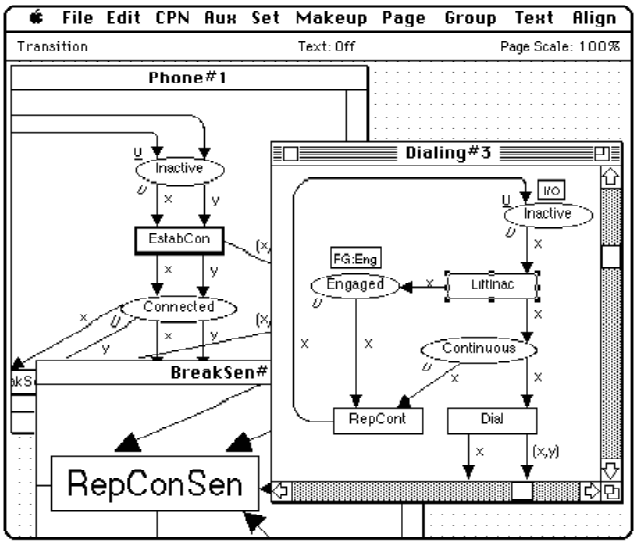

Figure 1: Design/CPN, the predecessor to CPN/Tools

and reuse [1]. Reification states that any entity in the interface should be accessible as a first-class object. Polymorphism states that commands should apply to as many different object types as possible. Reuse states that any output generated by the system and any input to the system should be reusable later, e.g. in the form of macros.

The resulting interface has no menu bars, no pull-down menus, no scrollbars, no dialog boxes and no notion of selection. Instead, it uses a unique combination of floating palettes, toolglasses and hierarchical marking menus, a novel windowing model based on pages and binders, and several new interaction techniques such as magnetic guidelines to align objects and bi-manual interaction to manipulate objects. This interface supports the same or higher level of functionality as the previous Design/CPN application, yet we have empirical evidence [15, 9, 1] that it is both simpler to use and more powerful.

The rest of this article presents the CPN/Tools interface and its design principles, outlines the implementation and gives preliminary performance evaluation data.

THE CPN/Tools INTERFACE

The CPN/Tools interface requires a traditional mouse and keyboard, plus a trackball (or other locator) for the non-dominant hand. For simplicity, we assume a right-handed user, but the mouse and trackball can be swapped for left-handed users. The keyboard is used only to input text and to navigate within and across text objects. The design of the bi-manual interaction follows Guiard's Kinematic Chain theory [7] in which the left hand manipulates the context (container objects such as windows and toolglasses) while the right hand manipulates objects within that context. The exception is direct interaction for zooming and resizing, which, according to Casalta et al. [5], should give both hands symmetrical roles. CPN/Tools incorporates six primary interaction techniques: direct and bi-manual interaction, marking menus [11], keyboard input, floating palettes, and toolglasses [4].

Direct manipulation (i.e. clicking or dragging objects) is used for frequent operations such as moving objects, panning the content of a view and editing text. When a tool is held in the right hand, e.g. after having selected it in a floating palette, direct manipulation actions are still available via a long click, i.e. pressing the mouse button, waiting for a short delay (200ms) until the cursor changes, and then either dragging or releasing the mouse button. Because of the visual feedback, this multiplexing of tools in the right hand is easily understood by users.

Bi-manual manipulation is a variant of direct manipulation that involves using both hands for a single task. It is used to resize objects (windows, places, transitions, etc.) and to zoom the content of a page. The interaction is similar to holding an object with two hands and stretching or shrinking it. Bi-manual interaction could also be used to control the orientation and position of an object. This might be used in the future to control the orientation of our magnetic guidelines (see below).

Marking menus are radial, contextual menus that appear when clicking the right button of the mouse. Marking menus offer faster selection than traditional linear menus for two reasons. First, it is easier for the human hand to move the cursor in a given direction than to reach a target at a given distance. Second, the menu does not appear when the selection gesture is executed quickly, which supports a smooth transition between novice and expert use. Kurtenbach and Buxton [11] have shown that selection times can be more than three times faster than with traditional menus. Hierarchical marking menus involve more complex gestures but are still much more efficient than their linear counterparts.

Keyboard input is used only to edit text. Some navigation commands are available at the keyboard to make it easier to edit several inscriptions in a row without having to move the hands to the mouse and trackball. Keyboard modifiers and shortcuts are not necessary since most of the interaction is carried out with the two hands on the locator devices.

Floating palettes contain tools represented by buttons. Clicking a tool with the mouse activates this tool, i.e. the user conceptually holds the tool in his or her hand. Clicking on an object with the tool in hand applies the tool to that object. In many current interfaces, after a tool is used (especially a creation tool), the system automatically activates a "select" tool. This supports a frequent pattern of use in which the user wants to move or resize an object immediately after it has been created but causes problems when the user wants to create additional objects of the same type. CPN/Tools avoids this automatic changing of the current tool by getting rid of the notion of selection (see below) while ensuring that the user can always move an object, even when a tool is active, with a long click (200ms) of the mouse. This mimics the situation in which one continues holding a physical pen while moving an object out of the way in order to write.

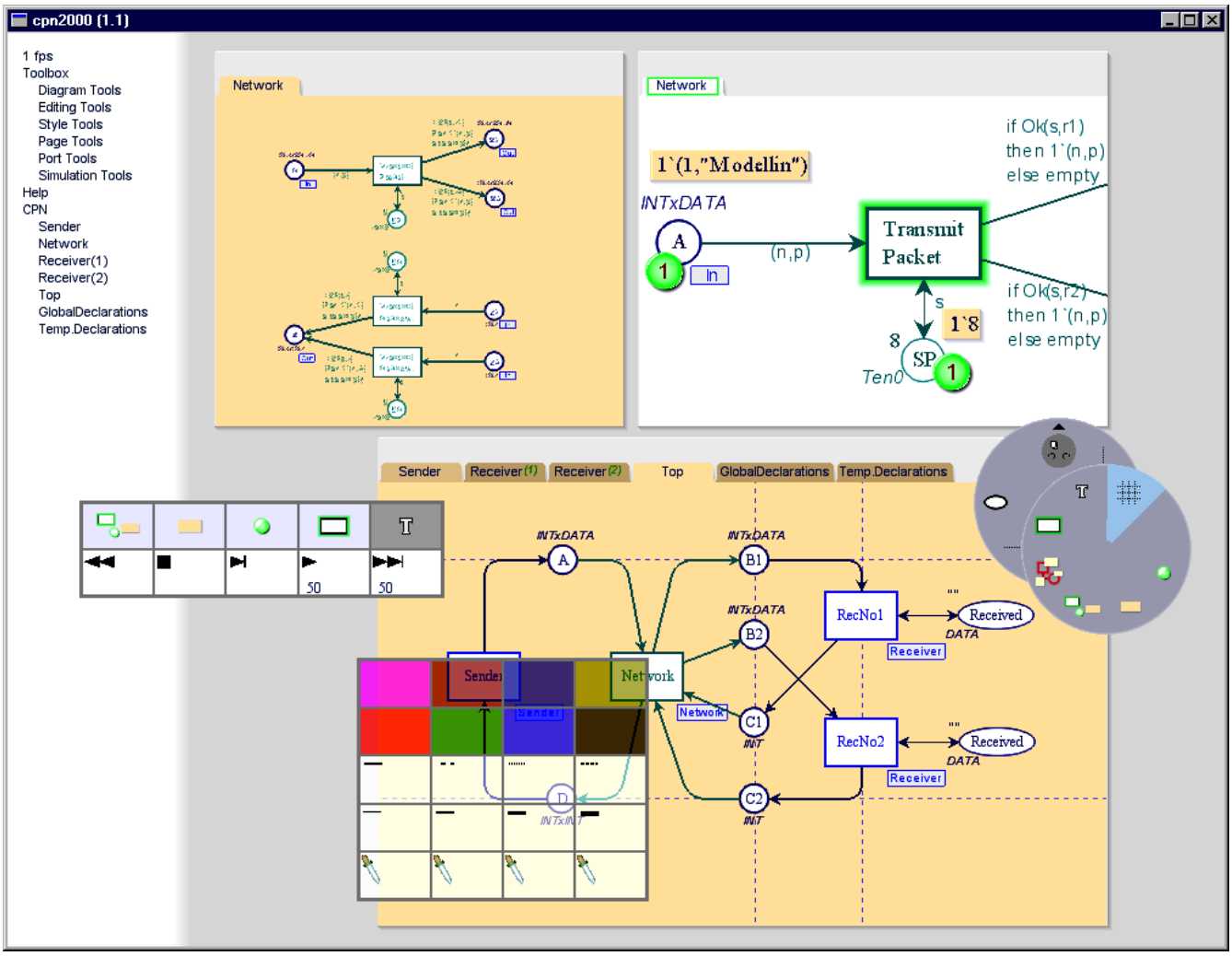

Figure 2: The CPN/Tools interface. The index appears in the left column. The upper-right binder contains a page with the simulation layer active. The upper-left binder contains a view of the same page, at a different scale. The lower binder contains six pages: the top page shows several magnetic guideline (dashed lines). The VCR-like controls to the left belong to the simulation floating palette. The toolglass at the bottom is positioned over objects on the page and is ready to apply any of the attributes shown. To the right, a hierarchical marking menu has been popped up on the page and is ready to accept a gesture to invoke one of the commands displayed.

Toolglasses, like floating palettes, contain a set of tools represented by buttons. Unlike floating palettes, they are semi-transparent and are moved with the left hand. A tool is applied to an object with a *click-through* action: The tool is positioned over the object of interest and the user clicks through the tool onto the object. The toolglass disappears when the tool requires a drag interaction, e.g., when creating an arc. This prevents the toolglass from getting in the way and makes it easier to pan the document with the left hand when the target position is not visible. This is a case where the two hands operate simultaneously but independently.

Since floating palettes and toolglasses both contain tools, it is possible to turn a floating palette into a toolglass and vice versa, using the right button of the trackball. Clicking this button when a toolglass is active drops it, turning it into a floating palette. Clicking this same button on a floating palette picks it up, turning it into a toolglass.

None of the above interaction techniques requires the concept of selection. All are contextual, i.e. the object of interest is specified as part of the interaction. This causes a problem, though, since it is not possible to apply a command to a group of objects as with traditional interfaces. Some features of the interfaces such as magnetic guidelines, described below, reduce the need to work with groups. Nevertheless, Version 2 of CPN/Tools will incorporate additional facilities to create groups, including dynamic groups resulting from a search.

Preliminary results from our user studies [15, 9] make it clear that none of the above techniques is always better or worse. Rather, each emphasizes a different, but common, pattern of use by using a different syntax:

	Mouse (dominant hand)				Trackball (non-dominant hand)		
	Left button short click	**Left button long click**	**Right button click**	**Mouse then trackball**	**Left button click**	**Right button click**	**Trackball then mouse**
Background			Background marking menu				
Binder	Bring to front Move	Bring to front Move	Binder marking menu	Move + pan	Bring to front Move		Resize binder
Page tab	Bring to front Move	Bring to front Move	Page tab marking menu	Move + pan	Bring to front Move		Move + pan
Page background	Apply tool in hand	Pan	Page marking menu	Move (conflicting)	Pan		Zoom page
Objects	Apply tool in hand Select text	Move	Object marking menu	Move	Move		Resize object
Toolglass	Apply cell tool to underlying object	Move underlying object		Apply cell tool to underlying object		Turn toolglass into palette	Apply cell tool to underlying object
Palette	Select tool	Select tool	Palette marking menu	Select tool	Select tool	Turn palette into toolglass	Select tool
Index	Drag (copy of) entry	Drag (copy of) entry			Drag (copy of) entry		Drag (copy of) entry

Table 1: Overview of the commands available according to the input action (down) and the context (across).

- *object-then-command:* point at the object of interest, then select the command from a contextual marking menu;
- *command-then-object:* select a command by clicking a tool in a floating palette, then apply the tool to one or more objects of interest;
- *command-and-object:* select the command and the object simultaneously by clicking through a toolglass or moving it directly.

As a result, marking menus work well when applying multiple commands to a single object. Floating palettes work well when applying the same command to different objects. Toolglasses work well when the work is driven by the structure of the application objects, such as working around a cycle in a Petri net. Table 1 summarizes the available interactions.

The Workspace Manager

Coloured Petri Nets frequently contain a large number of modules. In the existing Design/CPN tool, each module is presented in a separate window and users spend time switching among them. In CPN/Tools we have designed a new window manager to improve this situation: the Workspace Manager.

The workspace occupies the whole screen (figure 2) and contains window-like objects called *binders*. Binders contain *pages*, each equivalent to a window in a traditional environment. Each page has a tab similar to those found in tabbed dialogs (figure 3). Clicking the tab brings that page to the front of the binder. A page can be dragged to a different binder with either hand by dragging its tab. Dragging a page to the background creates a new binder for it. Dragging the last page out of a binder removes the binder from the screen. Binders reduce the number of windows on the screen and the time spent organizing them. Binders also help users organize their work by grouping related pages together and reducing the time spent looking for hidden windows.

CPN/Tools also supports multiple views, allowing several pages to contain a representation of the same data. For example, the upper-right page in figure 2 shows a module with simulation information, while the upper-left page shows the same module without simulation information and at a smaller scale.

Figure 3: Tabs for the pages in a binder

Figure 4: Index

The left part of the workspace is called the *index* (figure 4) and contains a hierarchical list of objects that can be dragged into the workspace with either hand. Objects in the index include toolglasses, floating palettes and Petri net modules. Dragging an entry out of the index creates a view on its contents, i.e. a toolglass, a floating palette or a page holding a CPN module.

Pages and binders do not have scrollbars. If the contents of a page is larger than its size, it can be panned with the left button of the trackball, even while the right hand is using the mouse to, for example, move an object or invoke a command from a marking menu. Getting rid of scrollbars saves valuable space but makes it harder to tell how much of the whole document is being displayed. A future version will use the borders of the page to show what portion of the document is viewed in a non-intrusive, space-saving way.

Resizing a binder and zooming the contents of a page involves direct bi-manual interaction (as described above). Unlike traditional window management techniques, using two hands makes it possible to simultaneously resize and move a binder, or pan and zoom the contents of a page at the same time. Clicking the right button of the mouse on the page tab or on the binder pops up a contextual marking menu with additional commands to close, collapse, expand the page or create a new page with the same content.

Creating and Laying out Objects

Creation tools are accessible via any of the three interaction techniques. The user may select the appropriate object from the floating palette, move to the desired position and click, or use the left hand to move the toolglass to the desired position and click-through with the right hand, or move to the desired location and make the appropriate gesture from the marking menu.

Users of Design/CPN spend a great deal of time creating and maintaining the layout of their Petri net diagrams. The primary technique is a set of *align* commands, similar to those found in other drawing tools. The limitation is that they align the objects at the time the command is invoked, but do not remember that those objects have been aligned. We observed that most users use the same pattern to move an object: They manually select all objects aligned to the object of interest and move them as a group. This dramatically slows down the interaction.

In order to facilitate the alignment of objects, we have introduced horizontal and vertical *magnetic guidelines*. Guidelines are first-class objects that are created in the same way as the elements of the Petri net model, i.e. with tools found in a palette/toolglass or in a marking menu.

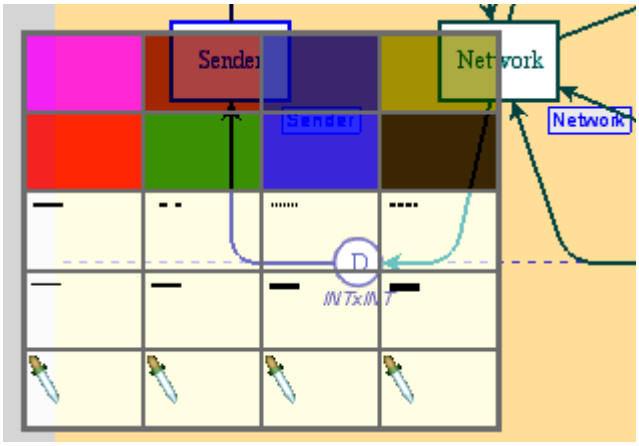

Figure 5: Toolglass for editing attributes

Guidelines are displayed as dashed lines (figure 2) and are magnetic. Moving an object near a guideline causes the object to snap to the guideline. Objects can be removed from a guideline by clicking and dragging them away from the guideline. Moving the guideline moves all the objects that are snapped to it, thus maintaining the alignment. An object can be snapped simultaneously to a horizontal and a vertical guideline.

We have designed, and will implement, additional types of guidelines. For example, rectangular or elliptical guidelines would make it easier to layout the cycles commonly found in Petri nets. We also plan to support spreading or distributing objects over an interval within a line segment, since this is a common layout technique. Adding these new types of guidelines may create conflicts when an object is snapped to several guidelines. One solution is to assign weights to the guidelines and satisfy the alignment constraints of the guidelines with heaviest weight first. Such conflicts do not exist in the current system because only horizontal and vertical guidelines are available.

Editing Attributes

The tools to edit the graphical attributes of the CPN elements are grouped in a palette/toolglass that contains five rows (figure 5): two rows of color swatches, a row of lines with different thicknesses, a row of lines with different dash patterns and a row for user-defined styles. The first four rows are fairly standard and are not described further here.

Tools in the last row correspond to the reification of groups of graphical attributes into styles. Initially, each tool in this row is a *style picker*. Applying this tool to an object copies the object's color and thickness into the tool and transforms the tool into a *style dropper*. Applying a style dropper to an object assigns the tool's color and thickness to that object. Applying a style dropper to the background of the page empties it and turns it into a style picker. If this is done by mistake, the *undo* command restores its previous state. In practice, style pickers and style droppers make it very easy and efficient for users to define the styles they use most often and apply them to objects in the diagram.

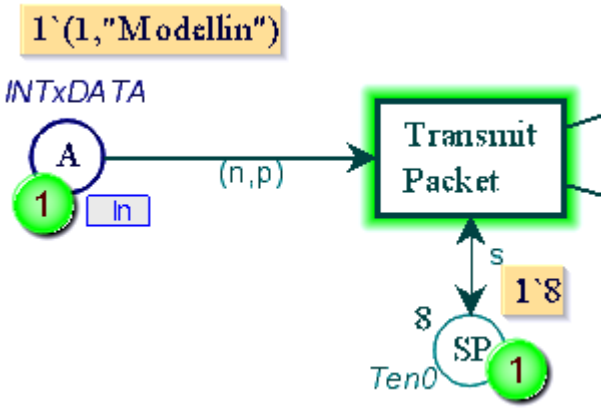

Figure 6: Simulation information

In Version 2 objects will remember which style they belong to (like in, e.g., Microsoft Word) and it will be possible to edit the attributes of a style in the toolglass itself. This will affect all the objects that use this style, saving repetitive editing.

Simulation tools

Once a CPN model has been created, the developer runs simulations to validate it. CPN/Tools uses the new simulator developed by the University of Aarhus CPN group [8], which is up to 1000 times faster than the previous one used by Design/CPN. The simulator runs as a separate process and communicates with the tool over a TCP/IP network connection.

CPN/Tools displays simulation information in a *simulation layer* that can be added to any page via any of the three interaction techniques. When the simulation layer is active (figure 6), the background color of the page changes, the number of tokens are displayed as small green disks, the token colors are displayed as yellow text annotations, and enabled transitions are displayed with a green halo. Each of these types of feedback can be toggled on or off using the tools in the simulation palette or toolglass (figure 7, top row).

Running the simulation involves compiling the net into ML code based on the structure of the net and the text inscriptions. This may result in syntax errors (during compilation) and run-time errors (during the simulation itself). In both cases, error messages are displayed as red "bubbles" next to the location of the error. The object that caused the error has a red halo. Since the error may occur in a page that is not on top, the red halo also appears in the tab of any page that has an error.

CPN/Tools uses a video tape player metaphor to control the simulation (figure 7, bottom row). *Next frame* lets the user select a transition to fire. *Play* randomly fires enabled transitions until a deadlock is reached or the user hits the *stop* button. *Fast-forward* runs the simulation at full speed for a maximum number of steps set by the user, displaying only the final state. *Rewind* resets the net to its initial

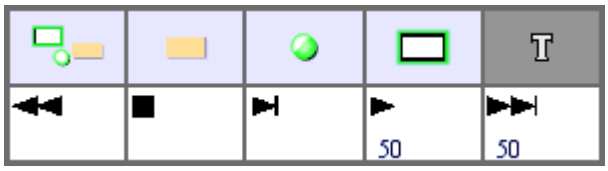

Figure 7: Simulation control panel

state. The *Next frame* command is polymorphic: If applied to an enabled transition, it fires that transition. If applied to a page, it fires a randomly-selected transition within the page. If applied to a binder or to the workspace, it fires a randomly-selected transition within the pages of the binder or the whole model, respectively.

Our user studies showed that users are either interested in the results of the simulation, and thus do not want to change the underlying diagram, or they are interested in editing the diagram and usually do not need the results of the simulation. Therefore, in our design, a diagram cannot be edited in a page while the simulation layer is active, which makes it easier to adjust the location of the simulation feedback. The user can always edit the underlying diagram in a different page with the simulation layer turned off (figure 2).

DESIGN PRINCIPLES

Graphical user interfaces can be broadly defined as consisting of graphical objects and commands. Graphical objects are represented on the screen and commands can be applied to create, edit and delete them. The perceived "ease-of-use" of an interface depends upon many factors, including the effectiveness of the visual representation, the completeness of the command set and the support for efficient patterns of use.

We have developed three principles that address the issues surrounding objects, commands and patterns of use:

- *Reification* extends the notion of what is an object;
- *Polymorphism* extends the power of commands with respect to these objects; and
- *Reuse* provides a way to capture and reuse patterns of use.

Reification

Reification is the process by which concepts are turned into objects. For example, in a graphical editing tool, the concept of a circle is represented as an image of a circle in a tool palette: it is reified into a tool. Reification therefore creates new objects that can be manipulated by the user.

Instrumental Interaction [2] extends the principles of Direct Manipulation [16] by reifying commands into *interaction instruments*. An interaction instrument is a mediator between the user and objects of interest: the user acts on the instrument, which in turn acts on the objects. This reflects the fact that, in the physical world, our interaction with everyday objects is mediated by tools and instruments such as pens, hammers or handles. The menu items, tool buttons, manipulation handles and scrollbars seen in today's user interfaces are examples of interaction instruments. A

scrollbar, for example, is both a visible object on the screen that can be manipulated by the user and also a command the user manipulates to scroll the document.

Another example of reification is the notion of *style*: In a text editor such as Microsoft Word, a style is a collection of attributes describing the look of a text in a paragraph, e.g., the font and margins. The user can create and edit styles and apply them to paragraphs. Styles thus become first-class objects for the user. Instruments may also operate on other instruments. For example, in some user interfaces, menus and toolbar buttons can be reconfigured to tailor the interface: they become objects that can be manipulated by (meta-)instruments.

CPN/Tools uses reification in various ways. All commands, whether in a marking menu, palette or toolglass, are instruments. As described below, this is also reflected in the implementation. Version 2 will support extensive customization by including additional instruments to operate on the instruments in the palettes and toolglasses. Also, the notion of group is reified in various ways. Magnetic guidelines represent a group of aligned objects, reifying the notion of alignment. Styles reify groups of graphical attributes to be applied to other objects. Binders reify groups of pages to be manipulated together.

Polymorphism

Polymorphism is the property that enables a single command to be applicable to objects of different types. Polymorphism allows us to maintain a small number of commands, even as reification increases the number of object types. This property is essential if we want to keep the interface simple while increasing its power.

Most interfaces include the polymorphic commands *cut*, *copy* and *delete*, which can be applied to a wide variety of object types, such as text, graphics, files or spreadsheet cells. *Undo* and *redo* can also be considered polymorphic to the extent that they can be applied to different commands.

Applying a command to a group of objects involves polymorphism at two levels. First, most commands that apply to an object can also be applied to a group of objects of the same type by applying them to each object in the group. Second, such commands can also be applied to a heterogeneous group of objects, i.e. objects of different types, as long as they have meaning for each of the individual object types.

CPN/Tools uses polymorphism in two ways. First, many instruments are polymorphic, i.e., they work with different types of objects. The best example is the "move" instrument used to move objects by direct manipulation: it works with index entries, binders, pages, places, transitions, arc bendpoints, text items, etc. Second, many instruments apply to groups of objects as well as individual objects. In some cases, the effect is to operate on all objects in the group. For example, applying a style or graphical attribute to a guideline affects all the objects attached to the guideline rather than the guideline itself. In other cases, the effect is to select one object in the group and apply the command to it. For example, the "next-frame" simulation tool fires a single transition. When applied to a page or binder, it randomly selects a single transition in the page or set of pages in the binder and fires it.

Reuse

Reuse can involve previous input, previous output or both. Input reuse makes previously-provided user input available for reuse in the current context. For example, the *redo* command lets users repeat complex input strings without having to retype them. Output reuse makes the results of previous user commands available for reuse. For example, *duplicate* and *copy-paste* let users avoid re-creating complex objects they have just created.

Polymorphism facilitates input reuse because a sequence of actions can be applied in a wider range of contexts if it involves polymorphic commands. Reification facilitates output reuse by creating more first-class objects in the interface which are then available for reuse. Thus, for example a Microsoft Word user can create a new style object by reifying the style of an existing paragraph or by duplicating an existing style object, modifying the copy and reapplying it. A more elaborate form of reuse obtains when new styles are created through inheritance from an existing style, which allows changes made in the reused object to be propagated to the edited copies.

Macros, such as those found in Microsoft Excel, illustrate the power of combining these three design principles. The user begins by telling the system to "watch" as a sequence of commands is performed. Reification enables the user to capture the particular pattern of use as a sequence of commands that can be applied as a single new command to a new set of objects.

The current version of CPN/Tools has limited support for reuse. Some commands can be undone/redone, and the style picker/dropper makes it possible to reuse the set of attributes of an object for other objects. Version 2 will incorporate more advanced facilities, including macros that will be available as regular tools in palettes and toolglasses.

IMPLEMENTATION

Implementing CPN/Tools required that we started from scratch: most user interface toolkits do not support our graphical model (e.g. transparency and non-rectangular windows) nor our input model (bi-manual input). We decided to use OpenGL [18] for output and to design and to implement our own input management. Note that the extra hardware for bi-manual input is very inexpensive: any mouse or trackball can be used.

The choice of OpenGL was motivated by several factors. First, the performance of 3D accelerated graphics is increasing faster than Moore's law, thanks to applications such as video games driving the market, and more and more of the rendering costs are off-loaded to the graphics hardware, saving CPU cycles. We anticipate that such hardware, which is already inexpensive, will be standard on

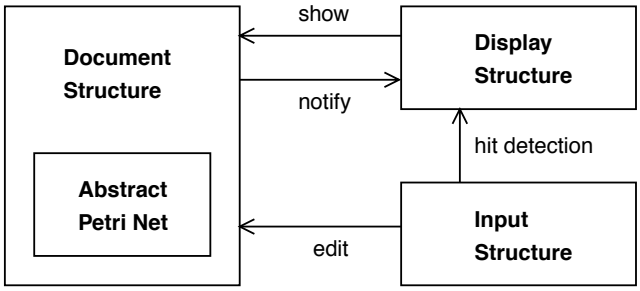

Figure 8: Main components of the architecture

PCs within a few years. Second, OpenGL is a standard API, implemented by many vendors on the three main platforms in use today : Windows, MacOS and Unix. This will make it easier to support Design/CPN on these platforms. Third, OpenGL has a sophisticated graphics model that supports transparency, lighting and texturing. This opens the door to advanced visual effects, which we have only started to use in CPN/Tools.

The system architecture is depicted in figure 8. The Document Structure holds a set of Abstract Documents that represent all persistent data in the system. The Abstract Petri Net is the part that represents the CPN diagrams and interfaces with the CPN simulator. The simulator runs as a separate process and is not covered in this description. The Display Structure is used for rendering the document structure and for hit detection. The Input Structure manages the interaction instruments that edit the documents.

The Document Structure

Abstract documents represent the persistent data of the system, including the diagrams being loaded, the tools and tool palettes, and the configuration of the workspace. This structure must be both extensible and efficient. The file format for storing the data in abstract documents is XML.

An abstract document is a collection of typed nodes. The document maintains a classification tree so that accessing all the nodes of a given type is efficient. The nodes can also be organized in a graph by creating typed associations among them. Any other structure could be superimposed on a document. For example, a spatial index such as in Pad++ [3] could be used to optimize access by the display structure.

Abstract documents generate notifications when nodes are added, removed or changed. The observers of these notifications are the items of the display structure described below as well as other nodes in the document structure. For example, an arc needs to know when its place or transition is moved or destroyed.

The Display Structure

The role of the display structure is to represent what is on the screen. It is used both by the rendering algorithm to update the screen and by the input structure for hit detection. The display structure is a special scene graph, similar to those found in 3D graphics toolkits such as Inventor [17]. Our display structure is slightly more abstract than traditional scene graphs because we separate the document structure from the display structure. Therefore the document structure can be organized to suit the needs of the application while the display structure can be optimized for rendering and hit detection.

The root of the structure is a Canvas, representing the whole screen. The Canvas contains an ordered set of Views, stacked from back to front. Views correspond roughly to windows in traditional window systems. Each view may contain other views, for grouping. Views can be semi-transparent, i.e. the views underneath them may show through, and they can have any shape. This is used to implement toolglasses.

The contents of a View is one or more Scenes, which are stacked like overlapping layers as in the multi-layer model [6]. We use different scenes to represent a single document in order to control which parts of the document structure is visible in a page:

- *main layer*: places, transitions and arcs, no textual inscriptions except the names of places and transitions;
- *text layer*: all text except names of places and transitions;
- *guidelines*: magnetic guidelines for aligning objects;
- *simulation*: tokens and messages related to the simulation
- *annotations*: error messages, help tips, etc.

A Scene can be shared among multiple Views. For example, two pages may display the same diagram by sharing the main layer and the text layer. One page may show the guidelines layer while the other displays the simulation layer, and each page may have its own scale.

Finally, the contents of a Scene is a set of Items. Items are meant to be lightweight objects: they typically contain a reference to the node of the document structure that they represent, and some information used solely by the display and hit detection algorithms.

The Input Structure

The input structure is based on the Instrumental Interaction model [2]. An *interaction instrument* is the association of a physical part (the input device) and a logical part (the representation on the screen). An instrument operates on a node of the document structure called the *target*. When receiving input events, an instrument provides feedback by updating its on-screen representation and sends commands to its target. The target responds to these commands by updating its own state.

Instruments are easy to implement in this framework. For example, it took less than two hours to create a help instrument that works with any object in the interface (including other instruments) and that is available from contextual menus as well as in a tool palette.

Polymorphic instruments such as the move instrument described before are implemented as collections of "concrete" instruments that represent implementations of

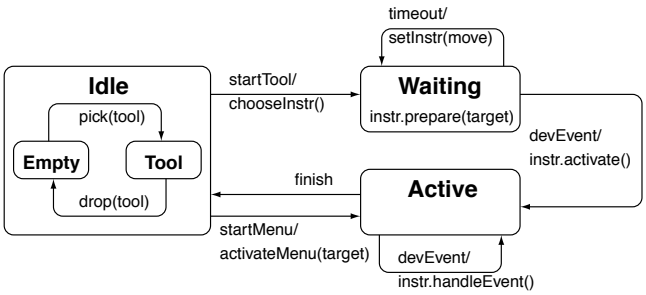

Figure 9: State machine for each hand

the instrument for different target types. When a polymorphic instrument is activated, it looks at the target object type, and activates the concrete instrument that corresponds to that type. For example the move instrument has many concrete instruments for moving places or transitions, bend points of an arc, inscriptions, magnetic guidelines, pages, binders, entries in the index, etc.

Several instruments may be ready for use at any one time. In our design, the user has instant access to up to five instruments at a time:

- the tool selected in a palette and held in the right hand;
- the move/pan tool accessible with a long click with the right hand, or with a short click when no tool is selected;
- the marking menu accessible with the right mouse button
- the toolglass that can be moved with the left hand and clicked-through with the right hand;
- the resize/zoom tool accessible by bi-manual interaction (left button press on the trackball and then click and drag with the mouse).

The input manager uses two identical state machines to manage the activation of instruments, one for each hand (figure 9). The transitions of the state machine are abstract events generated by the input devices or the instrument. The *Idle* state has two sub-states, *Empty* and *Tool*, describing whether a tool has been selected (right hand) or a toolglass has been picked up (left hand). The *Waiting* state is used to distinguish between a long click and a short click: when the 200ms timeout occurs, the current instrument becomes the move instrument. The *Active* state corresponds to the current instrument being used during a click or drag interaction. The instrument interacts with its target by sending it commands. Executing a command normally results in some edit of the document structure, which will be reflected on the screen at the next redisplay. At some point the instrument generates a *finish* event and the state machine is reset.

Since there are two state machines, one for each hand, it is possible to execute parallel actions, e.g., moving an object with the right hand while panning the contents of a page with the left hand. For interactions that involve both hands, such as resizing and zooming, one state machine sets the state of the other to *Active* (if it is *Idle*) and passes it its instrument and target. The two state machines therefore send their respective events to the same instrument.

Empty workspace	50 - 60 fps
One empty page	32 - 36 fps
After loading a 5 modules net:	
One binder with 5 pages	20-23 fps
Two binders with 5 pages each	14-15 fps
Five binders with one page each	11-12 fps
Five binders without text layer	15-16 fps

Table 2: Frame rates measured on our reference platform, in frames per second (fps)

PERFORMANCE EVALUATION

The data reported in this section corresponds to version 1 of CPN/Tools, which is fully functional and has been in use by a small group of members of the CPN group for real work for two months. Our reference platform is a HP/Kayak XW with a 500 MHz Pentium II and a Visualize FX6 graphics card running Windows NT4. We also use a similar PC with a Diamond FireGL 1 graphics card and a SGI Visual PC 320 for testing. CPN/Tools can also run on Unix (SGI O2) and Macintosh, but we need to improve the management of the secondary input device (the trackball) on these platforms.

The system is implemented in Beta [12], a high-level, compiled, strongly-typed object-oriented language based on a single construct, the *pattern*, that unifies classes, objects and methods. The implementation consists of 40000 lines of Beta code.

We have conducted a preliminary evaluation of the memory footprint and display rates of the system. However, we must stress that virtually no optimization has been carried out on this version. The memory footprint is typically between 10 and 12Mb when a medium-size CPN model is loaded, compared to 7.5Mb for the former Design/CPN tool. A break-down of memory usage shows that the Document, Display and Input structures use only 3Mb of the total, plus 1Mb for the texture cache. 2Mb are used by the Beta runtime for garbage collection, and the remaining 4Mb are used by OpenGL and the rest of the Beta run-time. While loading fonts, the Freetype engine may use up to an additional 10Mb.

Table 2 summarizes the frame rates (number of screen redisplays per second) for typical situations on the reference platform. The size of the display window is 1280x1024, and all measurements are done with a semi-transparent toolglass on the screen. As expected, the frame rates are very stable, irrespective of what the user is doing: moving pages, panning, zooming, etc.

The Visual PC 320 gives similar frame rates, while the FireGL 1 is 30% faster. Visually, frame rates of 15 frames per second or more look smooth enough for the interaction techniques we use. A close analysis of the profiling information shows that up to 50% of redisplay time could be optimized by using the hardware more efficiently. Also, profiling data shows that displaying text is time-consuming, as shown by the last two lines in the table above. But our choice of using OpenGL is clearly

successful and the increase in performance of graphics cards will only make things better: the FireGL1 came out less than one year after the Visualize FX6, costs a fraction of its price and is 30% faster!

CONCLUSION AND FUTURE WORK

We have described the interface, design principles and implementation of CPN/Tools and shown how it supports a combination of advanced interaction techniques in a post-WIMP interface. Version 1 is functional and already in use by a small group of users. We are currently working on the next version that will incorporate new features and improvements, based on the same design principles and overall approach.

Version 2 will feature various methods for creating and managing groups of objects. Groups will be specified either explicitly by designating the objects in the group or indirectly through a query, e.g. to find all places that belong to a given color set. The interface will also be customizable: users will be able to compose their own palettes and toolglasses and exchange them with other users. Styles and guidelines will be improved, and context-sensitive help will be available throughout the interface.

We are looking forward to the public release of Version 2 of CPN/Tools to a wider group of users in the Petri Nets community to collect valuable feedback for the next iteration of the design.

ACKNOWLEDGMENTS

We thank the members of the CPN group at the University of Aarhus for their participation in the design and evaluation activities.

This work is supported by the University of Aarhus, the Danish Centre for IT Research (CIT), Hewlett-Packard and Microsoft Research.

REFERENCES

1. Beaudouin-Lafon, M. & Mackay, W. Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces . In *Proc. Conference on Advanced Visual Interfaces*, AVI 2000, Palermo, Italy, May 2000, in press. http://www.daimi.au.dk/~mbl/AVI2000
2. Beaudouin-Lafon, M. Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. In *Proc. Human Factors in Computing Systems, CHI'2000*, ACM Press, 2000.
3. Bederson, B. & Meyer, J. Implementing a Zooming Interface: Experience Building Pad++. *Software Practice and Experience*, 28(10):1101-1135, August 1998.
4. Bier, E., Stone, M., Pier, K., Buxton, W., De Rose, T. Toolglass and Magic Lenses : the See-Through Interface. In *Proc. ACM SIGGRAPH*, ACM Press, 1993, p.73-80.
5. Casalta, D., Guiard, Y. and Beaudouin-Lafon, M. Evaluating Two-Handed Input Techniques: Rectangle Editing and Navigation. *ACM Human Factors In Computing Systems*, CHI'99, Extended Abstracts, 1999, p. 236-237.
6. Fekete, J-D. & Beaudouin-Lafon, M. Using the Multi-layer Model for Building Interactive Graphical Applications. In *Proc. ACM Symposium on User Interface Software and Technology*, UIST'96, ACM Press, p. 79-86.
7. Guiard, Y. Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model. *Journal of Motor Behavior*, 19:486-517, 1987.
8. Haag, T.B. and Hansen, T.R. Optimising a Coloured Petri Net Simulator. Master's Thesis, University of Aarhus (Denmark), December 1994.
9. Janecek, P., Ratzer, A., and Mackay, W. Petri-Nets-In-Use. In *Proc. International Workshop on Coloured Petri Nets*, Aarhus, Denmark, 1999.
10. Jensen, K. *Coloured Petri Nets: Basic Concepts (Vol. 1, 1992), Analysis Methods (Vol. 2, 1994), Practical Use (Vol. 3, 1997)*. Monographs in Theoretical Computer Science. Springer-Verlag, 1992-97.
11. Kurtenbach, G. & Buxton, W. User Learning and Performance with Marking Menus. In *Proc. Human Factors in Computing Systems*, CHI'94, ACM, 1994, p.258-264.
12. Lehrmann Madsen, O., Møller-Pedersen, B. & Nygaard, K. *Object-Oriented Programming in the Beta Programming Language*, Addison-Wesley, 1993.
13. Mackay, W.E. *Users and Customizable Software: A Co-Adaptive Phenomenon*. Ph.D. Dissertation, Massachusetts Instititute of Technology, 1990.
14. Mackay, W.E. Triggers and barriers to customizing software. In *Proc. ACM Human Factors in Computing Systems*, CHI'91, ACM Press, 1991, p. 153-160.
15. Mackay, W., Ratzer, A. & Janecek, P. Video Artifacts for Design: Bridging the Gap between Abstraction and Detail. In *Proc. ACM Conference on Designing Interactive Systems*, DIS 2000, New York, August 2000, in press. http://www.daimi.au.dk/~mackay/DIS2000
16. Shneiderman, B. Direct Manipulation : a Step Beyond Programming Languages. *IEEE Computer* 16(8):57-69, 1983.
17. Strass, P. IRIS Inventor, a 3D Graphics Toolkit. In *Proc. ACM Conference on Object-Oriented Programming, Systems, Languages and Applications*, OOPSLA '93, ACM Press, 1993, p.192-200.
18. Woo, M., Neider, J. & Davis, T. *OpenGL Programming Guide*, Addison-Wesley, 1997.