COMPUTING
SCIENCE

SONCraft user manual

Bowen Li and Brian Randell

TECHNICAL REPORT SERIES

No. CS-TR-1448 February 2015

TECHNICAL REPORT SERIES

No. CS-TR-1448 **February, 2015**

SONCraft user manual

B. Li, B. Randell

Abstract

This document provides detailed information on how to use SONCraft - an open source tool for the construction and analysis of 'Structured Occurrence Nets'(SONs).

Bibliographical details

LI, B., RANDELL, B.
SONCraft user manual
[By] B.Li and B.Randell

Newcastle upon Tyne: Newcastle University: Computing Science, 2015.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1448)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1448

Abstract

This document provides detailed information on how to use SONCraft - an open source tool for the construction and analysis of 'Structured Occurrence Nets'(SONs).

About the authors

Bowen is working on the EPSRC funded project UNCOVER (UNderstanding COmplex system eVolution through structurEd behaviours). An overall goal of UNCOVER is to develop a rigorous methodology supported by a toolkit based on structured occurrence nets, in order to provide an effective approach to acquiring and exploiting behavioural knowledge of a complex evolving system.

Brian Randell graduated in Mathematics from Imperial College, London in 1957 and joined the English Electric Company where he led a team that implemented a number of compilers, including the Whetstone KDF9 Algol compiler. From 1964 to 1969 he was with IBM in the United States, mainly at the IBM T.J. Watson Research Center, working on operating systems, the design of ultra-high speed computers and computing system design methodology. He then became Professor of Computing Science at the University of Newcastle upon Tyne, where in 1971 he set up the project that initiated research into the possibility of software fault tolerance, and introduced the "recovery block" concept. Subsequent major developments included the Newcastle Connection, and the prototype Distributed Secure System. He has been Principal Investigator on a succession of research projects in reliability and security funded by the Science Research Council (now Engineering and Physical Sciences Research Council), the Ministry of Defence, and the European Strategic Programme of Research in Information Technology (ESPRIT), and now the European Information Society Technologies (IST) Programme. Most recently he has had the role of Project Director of CaberNet (the IST Network of Excellence on Distributed Computing Systems Architectures), and of two IST Research Projects, MAFTIA (Malicious- and Accidental-Fault Tolerance for Internet Applications) and DSoS (Dependable Systems of Systems). He has published nearly two hundred technical papers and reports, and is co-author or editor of seven books. He is now Emeritus Professor of Computing Science, and Senior Research Investigator, at the University of Newcastle upon Tyne.

Suggested keywords

PETRI NET
STRUCTURED OCCURRENCE NETS
SON
WORKCRAFT
SONCRAFT
TOOL

SONCraft user manual

Bowen Li

Brian Randell

Newcastle University,

2 February 2015

Contents

1. Introduction

SONCraft is an open source tool for the construction and analysis of ‘Structured Occurrence Nets’ (SONs) – a Petri Net-based formalism for modelling system activity. The concept of a SON, as its name implies, is based on that of an ‘occurrence net’. Occurrence Nets (ONs) are directed acyclic graphs that represent causality and concurrency information concerning a single execution of a system (of systems). ON diagrams use two types of nodes – conditions (represented by circle) and events (represented by box) – linked by directed arcs that represent causality. Initial and final nodes must be conditions, and though an event can have one or more input and/or output arcs, a condition can have at most one input and/or output arc.

SON models consist of multiple delineated ONs associated together by means of various types of formal relationship, and provide a means of modelling the behaviour of a complex evolving system. Some uses of SONs, e.g. for system verification and system synthesis, involve recording and utilising information about the envisaged behaviour of a system. In other uses, the aim is to analyze records of the past behaviour of an actual system.

This document provides basic information about SONs – fuller, much more formal, accounts of SONs can be found in [1], [2], and [3].

SONCraft is implemented as a Java plug-in to the Workcraft platform, a platform which provides a flexible framework for the development and analysis of Interpreted Graph Models. A detailed Workcraft description and manual can be found in [4].

The present version of SONCraft, and hence this manual, deals with three of the various types of abstractions that have been defined for SONs, namely Communication abstractions (see Section 4.2), Behavioural abstractions (see Section 4.3) and Temporal abstractions (see Section 4.4).

2. Installation

The latest version of SONCraft is available from https://soncraft.codeplex.com/. It is necessary to have a compatible Java Runtime Environment (JRE) version 7 or higher in order to run SONCraft. The standard JRE can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/. There is no automatic installer for SONCraft; to install it, the files from the link archive need to be extracted manually, into (it is suggested) a directory called “SONcraft system”.

2.1 Starting SONCraft

On Windows: SONCraft is started by a workcraft.bat script.

On a Mac or in Linux: After opening a terminal window, the user has first to change directory to that containing the SONCraft software in the directory “SONCraft System”. Then SONCraft can be started by using the command ./workcraft.

3. The Workcraft GUI

A few seconds after starting SONCraft, the Workcraft main framework will appear. The user can create a new *work file* to hold a SON model by selecting **File ->** *Create work*[1]. The graphical interface of the tool is depicted in Figure 1. The generic information for platform interface and settings can be found in http://workcraft.org/. In this section, we mostly focus on the facilities related or designed for the SON plug-in.

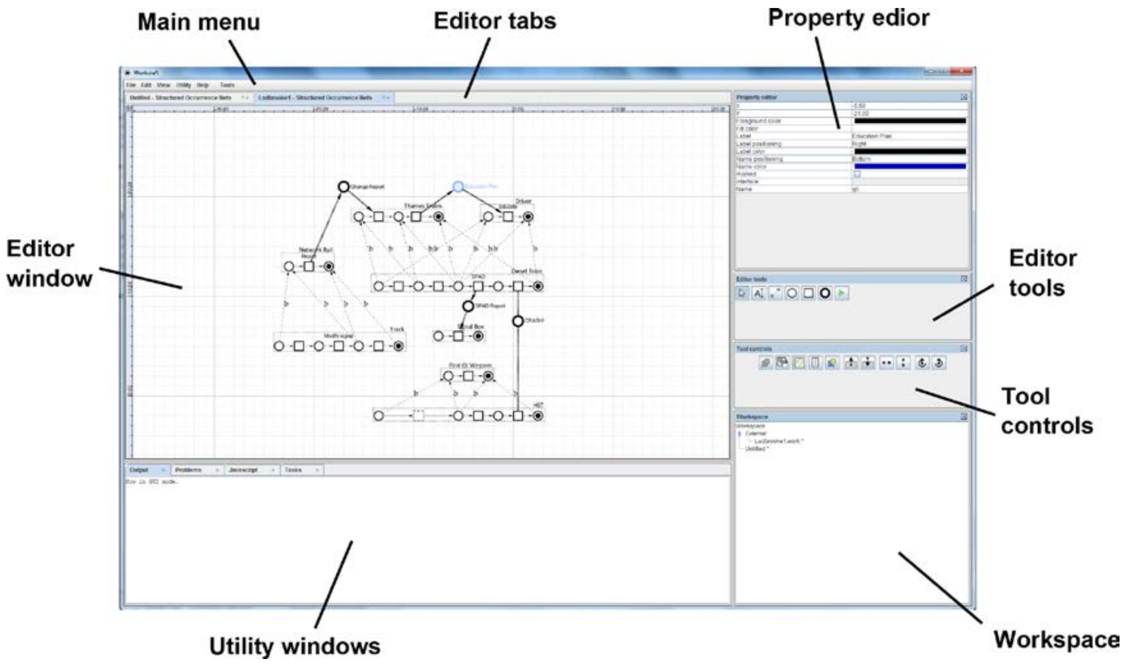


Figure 1. SONCraft interface.

3.1. Main menu

The *Main menu* provides functions to manage, edit and analyse models.

File – provides common operations for managing SON work files (i.e. the containers of SON *models*).

- *Create work*…(CTRL-N[2]) – create a new SON work file, and display this (empty) work file in the Editor window, as the *current* SON model. The name given to this new work file (and model) will be added to the line of Editor tabs. (In fact the Create work command requires the user to indicate whether a SON model or a Petri Net model is to be created – this manual deals just with SON models.)

- *Open work…* (CTRL-O) – looks in the "SONCraft System" directory (the assumption being that this is where work files are held) in order to load a previously-created work file. (If, when trying to save a work file, the user finds that the default directory is a Workcraft directory deep in the Unix file hierarchy she can use the home button to get to the home directory, from where she can navigate to her "SONCraft System" directory.) Such loading causes the chosen SON model to become the current SON model, displayed in the Editor window (possibly replacing some previously displayed SON model), and its name to be added to the line of Editor tabs above the Editor window.

[1] In this manual, main menu commands are are referred to in **bold**, and their sub-menus in *italics.*

[2] Keys on the keyboard are indicated in SMALL CAPS.

- *Open recent…* – load one of the recently opened work files, so displaying its contents as the current SON model.
- *Merge work…* – merge a SON model from an external work file into the current work file.
- *Save work* (CTRL-S) – save the current SON model into a work file (.work format).
- *Save work as…* save the current SON model in a new work file.
- *Close active work* – close the currently active work file.
- *Close all works* – close all the work files.
- *Export…* – export the current work into an external file (e.g. SVG, DOT format).
- *Exit* (ALT-F4) – quit the program.

Edit – provides common operations and settings for editing the current SON model.

- *Undo* (CTRL-Z*)* – undo the last modification to the current SON model.
- *Redo* (CTRL-SHIFT-Z*)*- redo a previously undone modification to the current SON model.
- *Cut* (CTRL-X) – remove selected nodes and connections and put them into the local clipboard.
- *Copy* (CTRL-C) – copy selected nodes and connections into a local clipboard.
- *Paste* (CTRL-V) – paste nodes and connections from the local clipboard into the current SON model.
- *Delete* (DELETE) – remove selected nodes and connections.
- *Select all* (CTRL-A) – select the entire current SON model.
- *Inverse selection* (CTRL-I) – invert the selection.
- *Deselect* – reset selection.
- *Preferences…* – edit global settings for generic and plugin-specific properties.

View – means of controlling what windows are shown in the SONcraft interface. (The relative sizes of the windows can be adjusted by dragging their borders.)

Utility

- *Reset UI layout* – revert the user interface to the original state.

Tools – provides a set of user-friendly analysis tools for SON models.

- *Custom tool -> Reset color to default* – reset all nodes' colours to default.
- *Custom tool -> Reset tokens* – clear any place/channel place tokens.
- *Error tracing -> Enable/Disable error tracing* – i.e. activate/deactivate error tracing function (see Section 6.1).
- *Error tracing -> Reset fault/error state* – reset (simulated) fault/error state to default.
- *Verification -> Check for reachability* – check if a given marking of the current SON model is reachable from the initial states (see Section 7).

- *Verification -> Check for structural properties…* – validate the current SON model's structural correctness (see Section 5).

Help

- *Help contents* (F1)-> table of contents for Workcraft generic information and details of its model plugins.
- *Tips and tricks* -> a collection of hints for efficient use of Workcraft.

3.2 Editor tabs

The *Editor tabs* line shows the names of all of the opened SON models and allows the user to choose which one is to be the *current* model, i.e. is to be displayed in the *Editor Window*, which as its name indicates is where a user can view and edit a model.

3.3 Editor window

The *Editor window* is the place where the current SON model is displayed for editing and simulation. The basic editor controls are as follows:

- Mouse wheel – zoom the model in and zoom out.
- Left click – selects/connects objects, creates new objects.
- Middle button or CTRL + Right button – pan view of the selected area of the model.
- *Maximize window* button – located in the top right hand corner of the Editor window (visible only when just a single work is opened). This button, or the up arrow in the editor tab of each of a set of opened works, can be used to expand the Editor window, so closing all the other windows, such as the Editor tool window, Property editor, etc., so as to show just a single work file, i.e. the current SON model.
- *Restore window* button – The effect of maximize window can be reversed by clicking on the little button (now with a diagonal down arrow) in the top right hand corner of the Editor window.
- *Four small toggle*-buttons – provided in the top-left corner of the Editor window (between the horizontal and vertical rulers). These buttons provide means for controlling the visibility of the grid, the rulers, node names and node labels. They are shortcuts to the corresponding properties of the **Edit**->*Preferences…* (*Show/Hide* the Grid, Labels, Names and Rulers, respectively).

3.4 Editor tools

A user can view and edit a model in the Editor window using one of the *Editor tools* . (At any given moment, those parts of a visible model which are not available for editing are greyed out.)

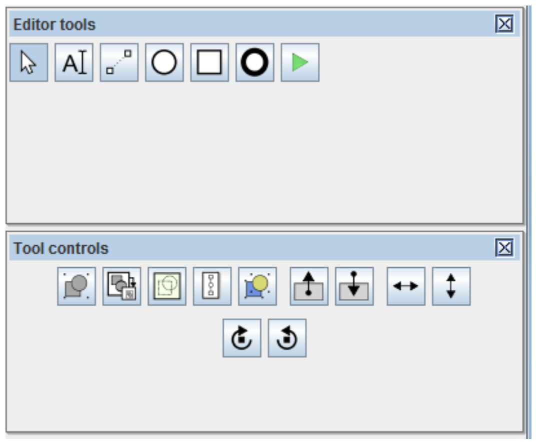


Figure 2. The Select Tool and its Tool Control icons.

- *Select* button (S) – identify the graphical node that is to be moved, connected, etc.
    - Click a graph node to select it. Draw a rectangular area to select several nodes. (If you start drawing the selection rectangle from its top-left corner of the screen and expand it slowly towards the bottom-right corner you will notice that model components become selected only when they are fully inside the selection rectangle. On the other hand, if you start drawing the area selection from its bottom-right corner then the model components become selected as soon as they are touched by the selection rectangle.)
    - Hold SHIFT to include nodes in a selection and CTRL to exclude nodes from a selection.
    - Press CTRL+A to select all nodes (**Edit**->*Select all*). Press the exclamation mark to inverse the selection (**Edit**->*Inverse selection*). Press ESC to reset the selection (**Edit**->*Deselect).*
    - Double click inside a group to enter it (same action as PAGE DOWN). Double-click outside the current group to go one level up (same action as PAGE UP). Use left mouse button or ←, ↑, →, ↓ to move selected nodes.
    - A node will be highlighted (in yellow) while under the mouse cursor if the node is selectable, and will become selected and coloured blue when clicked on, until somewhere else is clicked.
- *Text Note* button (N) AI – place a rectangle ready to contain text wherever appropriate in the Editor window. The default background colour of such a note is yellow. The text in the note turns blue when the note is selected. Double clicking on the note opens it up so that the text can be edited. The user can introduce a new text line when Return is pressed, and can close the note by clicking anywhere outside it. (It is also possible to enter text directly into the label field in the Property editor after a text box has been merely selected, rather than opened. Indeed one can enter the same text into a set of text boxes by selecting them all and typing the required text into the label field in the Property editor.)
- *Connect* button (C) – create SON-based connections between nodes; the choice of connectors available while in this mode – at present just *Causal connection*, or *Behavioural abstraction* (see Figure 3) – is displayed in the Tool control panel.
- *Condition* button (B)○ – create a condition in a SON model.
- *Event* button (E) □ – create an event in a SON model.

- *Channel Place* button ○ – create a channel place in a SON model.
- *Simulation* button ▶ - switch to SON simulation mode (see Section 6).

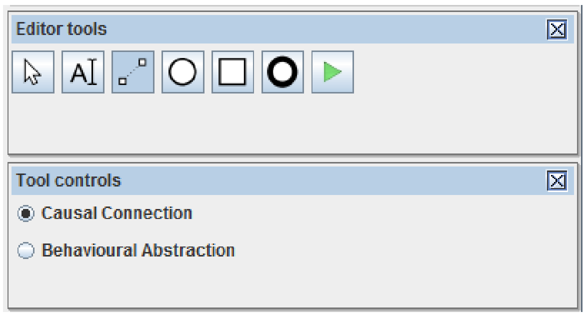
3.5 Tool controls

The *Tool controls* panel provides access to the extended functionality (if any) of a selected tool. The details of usage are considered in the sections below.

Selection controls:

- *Group Selection* button (CTRL+G) – combine the selected nodes into a group.
- *Block Selection* button (ALT+B) – combine the selected nodes into a block.
- *Ungroup Selection* button (CTRL+SHIFT+G) – destroy a group/block.
- *Level Down* button (PAGE DOWN) – enter a (selected) group.
- *Level Up* button (PAGE UP) – leave a group.
- *Flip horizontal* button ↔ – flip the selected objects horizontally.
- *Flip vertical* button ↕ – flip the selected objects vertically.
- *Rotate clockwise* button – rotate the selected objects clockwise.
- *Rotate counter-clockwise* button – rotate the selected objects counter-clockwise.

Connect controls:



(●) Causal Connection
( ) Behavioural Abstraction

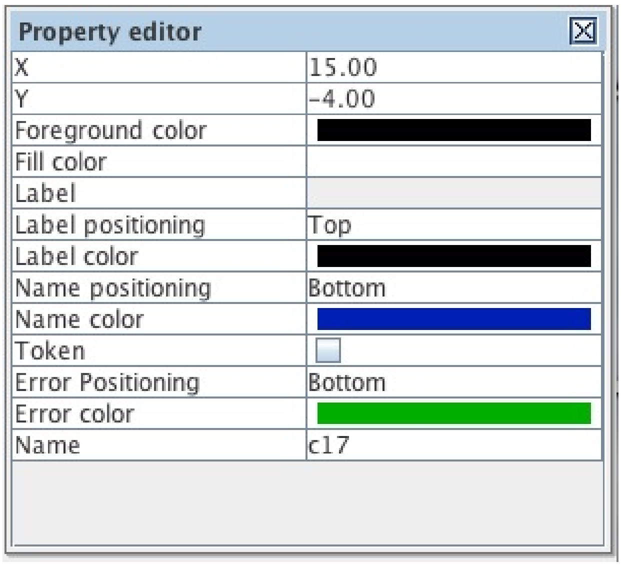
Figure 3. The Connect Tool and its Tool Controls.

- *Causal Connection* – create a causal connection (directed arc).
- *Behavioural Abstraction* – create a behavioural connection

Simulation control – see Section 6.

3.6 Property editor

The *Property editor* window is used to display and change the properties of the currently selected object (condition, event, or group) or connection. For example when a condition has been selected the Property editor window is as shown in Figure 4. This shows the position of the condition, its colourings, its name, label, etc. The (unique) name is automatically assigned when the object is created.



Property editor

X	15.00
Y	-4.00
Foreground color	⬛
Fill color	
Label	
Label positioning	Top
Label color	⬛
Name positioning	Bottom
Name color	🟦
Token	☐
Error Positioning	Bottom
Error color	🟩
Name	c17

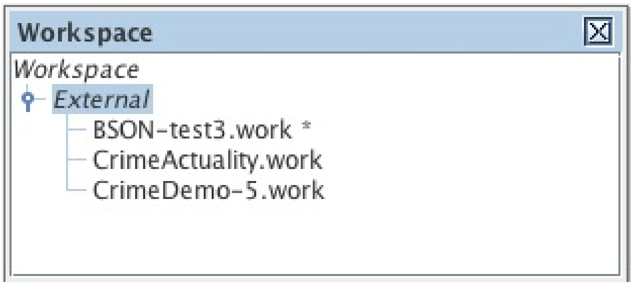
Figure 4. A condition's property editor window.

The label is the field the user is most likely to want to modify; text typed into this field will appear (by default) above the node in the Editor window as soon as RETURN is typed and the user returns to the Editor window. The vertical line symbol can be used to introduce new lines (e.g., aaa|bbb|ccc for a three-line label.) The properties of a given object, such as colouring and positioning information can be changed using the Property editor – the default values for such properties, i.e. for all (subsequently-created) objects of a given type, can be changed in the *Preference settings*, i.e. **Edit** -> *Preferences* -> *Common* -> *Visual*.

Each node in a SONCraft work file has a unique name that is automatically assigned at the time of its creation. The name can be displayed below the node by selecting 'Show component names' in the *Preference settings*.

3.7 Workspace window

The *Workspace window* (Figure 5) lists opened or imported work files (and, using asterisks, indicates which files have been altered but not saved). One can also operate on a work file (delete, save, etc.) by right clicking on its file name in the Workspace window.



Workspace

```
Workspace
External
    BSON-test3.work *
    CrimeActuality.work
    CrimeDemo-5.work
```

Figure 5. The Workspace window.

3.8 Utility window

The *Utility window* has four tabs showing additional information.

- *Output* – this is to provide information on the currently executed task.
- *Problems* – this is to report errors and exceptions.
- *Tasks* – here the progress of the currently run tasks can be monitored.
- *Javascript* – this is console for scripting bulk actions in the Javascript language (for experienced users only).

4. Creating SON Models

4.1. Creating an Occurrence Net Model

As indicated, occurrence nets are directed acyclic graphs used to record dependencies between events in a single execution of a concurrent system. Figure 6 shows an occurrence net example in SONCraft. The user can, by clicking on the *Condition* ○ or *Event* □ button, switch to the chosen mode and create one or more conditions or events, respectively. By clicking on the *Connect* button the user can switch to the Connect mode. In this mode, when the mouse is over a node (i.e. a condition or an event), this node will be highlighted (as though selected). Then, guided by the messages at the foot of the Editor window, the user can draw a connection from this first node to a second node. Multiple such pairs of nodes can be so connected, until the mode is switched to some other mode from connect. If CTRL is held down while clicking on a set of nodes in sequence a whole linear series of connections can be made between these nodes.

In the case of ordinary Occurrence Nets, such a connection must be a *Causal Connection*, and be from a condition to an event, or vice versa – any attempt at linking two conditions, say, will provoke an error message. (When the user has chosen an editor tool, the chosen tool is highlighted, and the tool controls window shows, and is used to control, the operation of the tool, e.g. for the Connect Tool the type of connection, at present either a *Causal Connection* or a *Behavioural Abstraction,* can be chosen – as was shown in Figure 3.)

Note*: Not all user errors are detected and so result only in error messages – those that cannot be imediately detected can be checked for using the Verification Tool (see Section 5).*

Connections are normally straight lines between condition and event nodes. However by double-clicking on a connection the user can introduce a potential corner (shown as a small dot when the connection is selected), that can then be dragged into its chosen position, with its connection lines following it. In fact the connection can be curved by selecting an existing connection, and using *Property editor -> Connection type -> Bezier.*

Conditions and events are automatically named (*c1, c2, c3* etc., and *e1, e2, e3,* etc.) as they are created. These names are by default shown in blue below the conditions and events. The user can use the Property editor to edit various aspects of the currently selected node or connector (or group – see below), e.g. as mentioned earlier, to provide a label, and to indicate the placement and colouring of this label.

Note: *SONCraft provides another form of node name. The hierarchical name space can be viewed by looking at the absolute path of the node name (i.e.* ***Edit*** *-> Preferences -> Common -> Edit -> Name shown with* *absolute paths). For example, in Figure 6 the name of the condition c0 which is inside a group would be shown as g0/c0, indicating that c0 is inside group g0.*

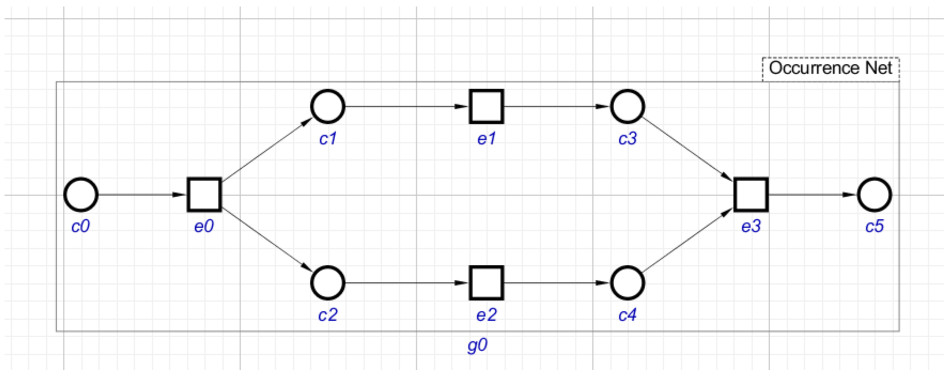


Figure 6. An occurrence net example.

In SONCraft an occurrence net cannot be recognised and analysed until it has been defined, i.e. delineated as a *group*. To do this, use the *Select* tool to select one or more nodes, either by clicking on them or dragging a bounding box around them; the selected nodes turn blue. These selected nodes can then be combined into a group using the *Group selection* . This is the first of the set of buttons that shows in the Tool controls panel when the Select tool has been chosen (Figure 2) . The resulting group is shown using a solid-line box, the entire contents of which will remain blue while the box is selected (see Figure 7). While the box is selected the Property editor can be used to give the group a label, as shown in Figure 6, and the group can for example be moved, or deleted. The box becomes unselected if the user clicks outside it, and becomes selected again if she clicks inside it. (Other selection-related actions are **Edit** -> *Inverse Selection*, **Edit** -> *Select All*, and **Edit** -> *Deselect*. Note: If the user is already inside a group (see Figure 8 below), this group alone will not be coloured blue in response to **Edit** -> *Select All*, since the group is implicitly selected already.)

Text notes can be included with condition and/or event nodes in groups but cannot be in groups on their own – an error report appears if an attempt is made to create such a group.

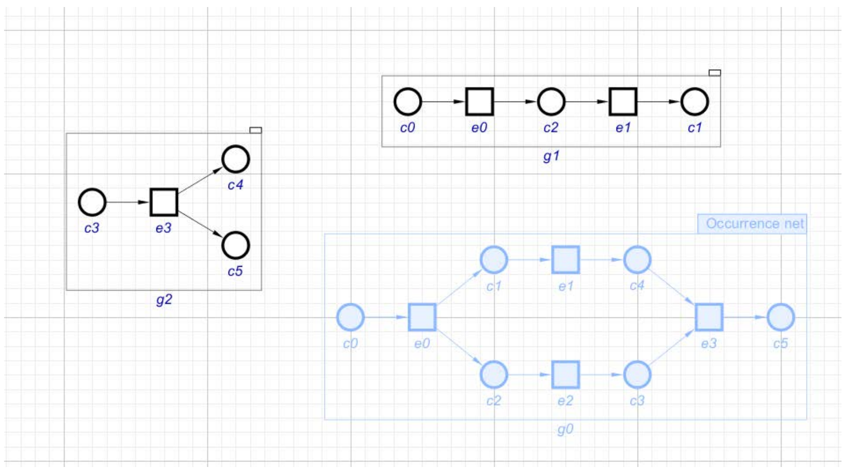


Figure 7. A selected group.

A group can be destroyed, leaving its contents in place but ungrouped, using the *Ungroup selection* button . Other buttons allow the user to flip a selected group horizontally or vertically, or to rotate it.

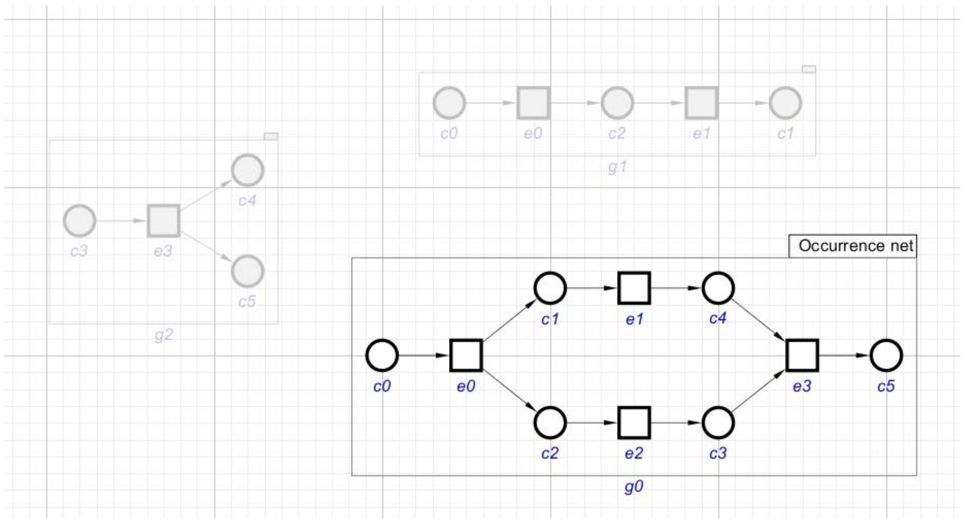


Figure 8. An entered group.

Once a group has been created it will not be possible to edit any of its component within the group without entering the group. The user enters a (selected) group by double-clicking inside it or using the *Level down* button . The result is to remove the blue colour from the group, and to turn everything outside the group grey, and hence uneditable – see Figure 8. The user can leave the group by double-clicking anywhere outside it or clicking the *Level up* button .

Note*: When you have just a set of* _unselected_ *groups (and are not inside a group so none of the groups are greyed out) in the Editor window, you can add nodes to them (apparently) but any such a node, even if situated inside a group boundary, is not in fact part of group it appears to be in. Moreover the user can link such nodes to other such nodes, i.e. ones which are – despite any appearances to the contrary – not in any group. The fact of such a node not being a member of the group that it appears to belong to does however become obvious once the user selects the group (see Figure 9) or moves the mouse cursor over the the group.*

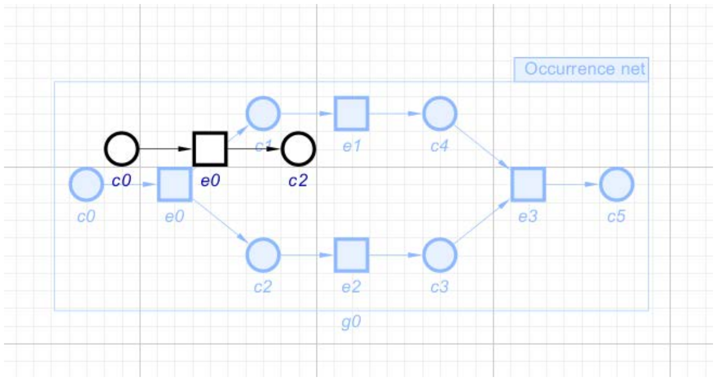


Figure 9. A selected group showing some “orphan” nodes.

In summary, groups are used to structure a SON model. At any moment the user can be in just one group, and so be able to select and then edit the nodes and connectors that form its contents – everything outside the group will be greyed out and uneditable. Blue is used to identify the currently selected nodes or groups.

NOTE*: The present version of SONcraft does* *__not__* *allow groups to be nested.*

4.2. Creating a Communication SON Model

The first and most basic form of Structured Occurrence Net (SON) is the Communication Structured Occurrence Net (C-SON). A C-SON involves two or more Occurrence Nets (ONs) that are connected by one or more *synchronous or asynchronous communications*, so can be used to model an activity that has involved interaction between communicating systems. In SONCraft, any communication between occurrence nets (i.e. groups) must go via a special node named a *channel place*[3]. To create a communication link between two ONs click on the *Channel Place* button ◯ to create a channel place, and then use the *Select* button ▫, to link an event in one ON to the channel place, and then the channel place to an event in another ON. Full details about the channel place concept are given in [5].

A new connection link to or from a channel place will be automatically shown as being of type *asynchronous communication*, represented by a bold dashed line with arrows. The channel place must not be in any group – it can only be linked to events that are in (two different) groups. The connection tool enforces these restrictions, but it does not stop users accidentally making a pair of directed links __from__ a channel place or __to__ a channel place – use of the verification tool (see Section 5 below) will subsequently find this error.

As an example, the C-SON in Figure 10 consists of two interacting occurrence nets where the connection type between events *e0* and *e4* is an asynchronous communication, with the meaning that e0 cannot happen before e4.

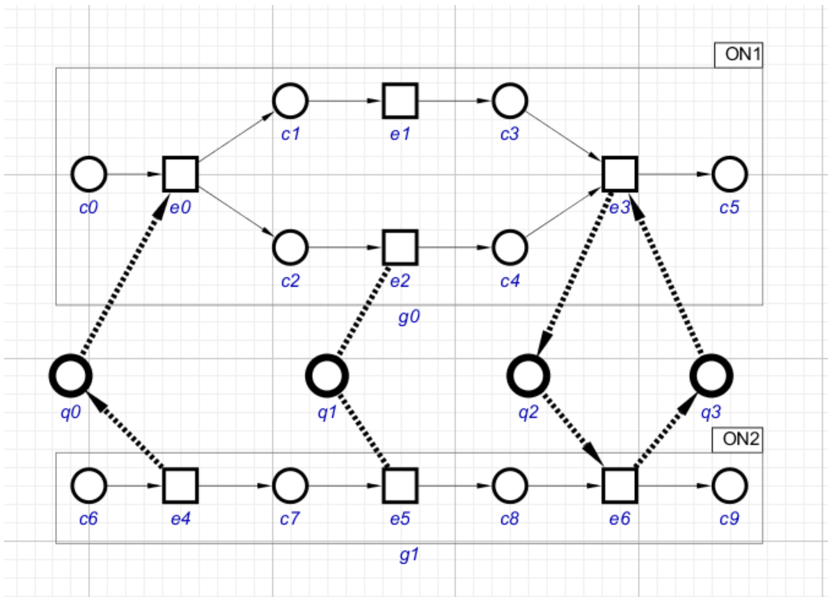


Figure 10. A Communication SON example.

An asynchronous communication can be turned into a *synchronous communication* by double clicking on its channel place. This will remove the arrow heads on the bold dashed lines – see the connection between *e2*

[3] The Channel Place syntactic device has been introduced since [1], which (rather anomalously) showed communication asynchronous or synchronous links as going directly between an event in one ON and one event in another ON.

and *e5* in Figure 10. Double clicking again on the channel place in a synchronous communication will turn it back into an asynchronous communication.

A synchronous connection is in fact logically equivalent to a pair of channel places whose asynchronous communication links form a cycle – see the cycle involving *e3* and *e6*. (The user can use either form when creating a synchronous communication.)

NOTE*: The cycle involving e3 and e6 in Figure 10 does not violate SON's acyclic property due to its execution semantic. The two connected events can only be executed synchronously. The SON Simulator (see Section 6) therefore ensures that both e3 and e6 participate in single step when either is selected by the user for execution, and that both channel places are filled and emptied synchronously, leading to conditions c5 and c9 holding.*

4.3. Creating a Behavioural SON Model

The second form of structuring, behavioural abstraction, allows the activity of an evolving system to be modelled. Thus a Behavioural Structured Occurrence Net (B-SON) gives information about the evolution of an individual system, and the phase of the overall activity that is associated with each successive stage of evolution of this system. (A phase is a fragment of an ON beginning with a global state (or "cut" – see the above model in Figure 11) and ending with a global state which follows it in the causal sense, including all the conditions occurring between these global states. The ON in Figure 11 shows an ON which has been divided into three particular phases by two chosen cuts.) Using this phase concept, a B-SON thus provides a two-level view of execution history: the low level structure provides the details of its behaviours during the different evolution stages represented in the upper (abstract) level view.

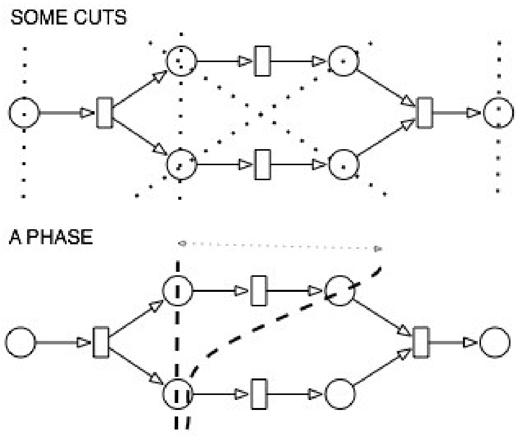


Figure 11. An ON which has been divided into three particular phases by two chosen cuts.

Between them the set of upper level conditions in an upper level ON must be linked to a <u>complete</u> set of phases of the lower level, i.e. to all the initial and final conditions of each phase. And the ordering of the conditions of the upper ON must match that of the phases of the lower ON. The error message "Invalid phase: phase does not reach initial/final state" that results if either of these conditions is not met is rather obscure. (However, subject to these conditions, there can be multiple upper level ONs corresponding to a single behavioural ON, i.e. there can be several different behavioural abstractions of the same detailed activity.)

To create a B-SON in SONCraft, activate *Select* tool and choose the connection type *Behavioural Abstract* in the Connection Tool Controls panel. The connection then is able to provide a link to a condition in the abstract ON from appropriate conditions in the low level ON(s) – see Figure 12.

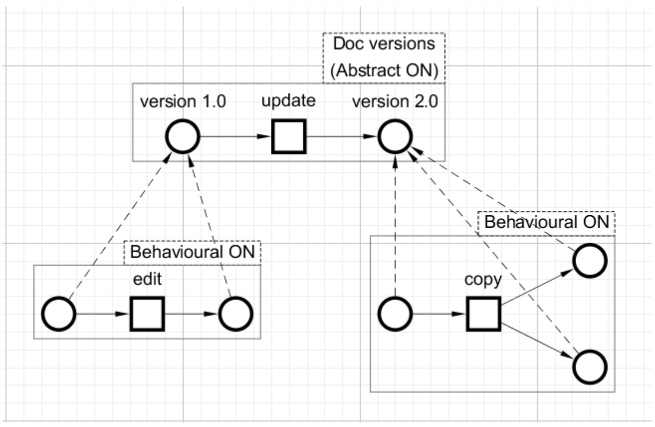


Figure 12. A B-SON example.

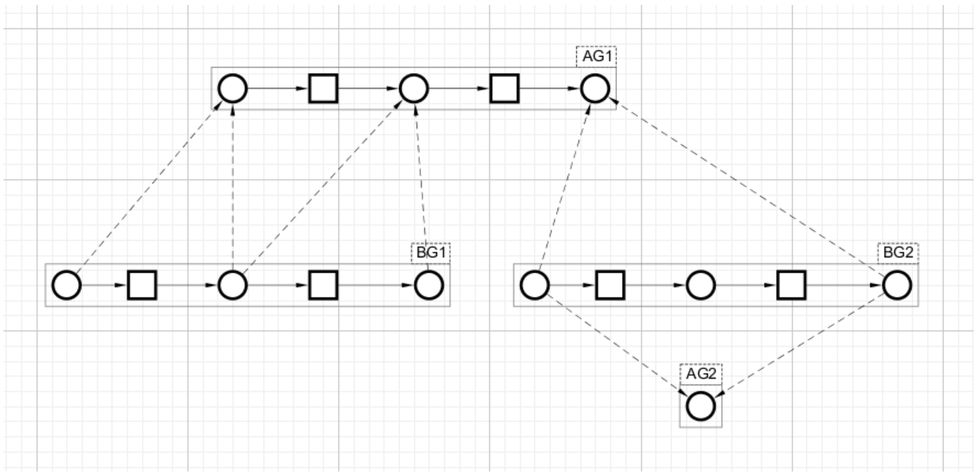


Figure 13. A B-SON showing online evolution, and multiple abstractions.

In Figure 13 BG1 is an activity that has involved an online evolution, and then an offline evolution to a phase of activity BG2, an activity that has two separate abstractions, AG1 and AG2.

Note: *The present version of SONCraft follows the present (somewhat restrictive) set of formal definitions of B-SONs [1], so abstract ONs must be straight line graphs, which can have communication links to other abstract ONs, but cannot themselves have further levels of abstraction "above" them, and or be related to more than one low-level SON for each abstract condition. All of the conditions in an abstract ON must have behavioural links from low level ONs.*

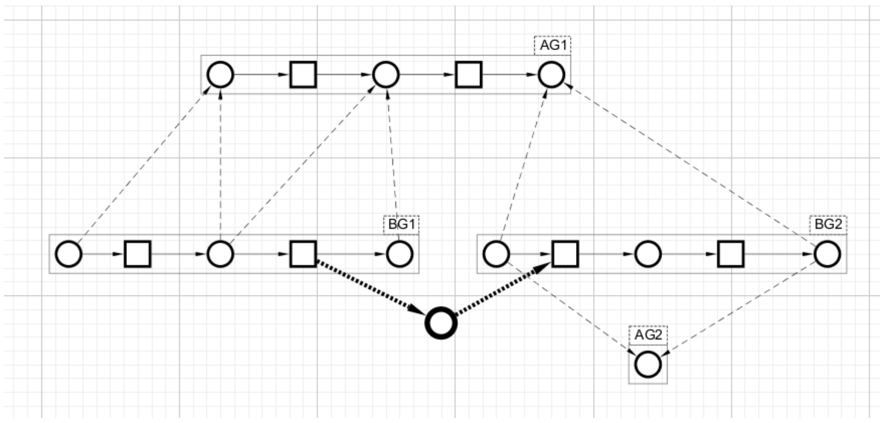


Figure 14. A Behavioural SON involving an abstraction of a C-SON.

Figure 14 includes a SON composed of ONs AG1 and AG2, and shows an abstraction of a C-SON composed of ONs BG1 and BG2. The activity BG1 arises from an evolving system (AG1), that of BG2 from both systems (AG1, AG2). (For example, AG1 may be showing the evolution of a system the behaviour of whose final version is shown by BG2, whereas AG2 is showing which system has recorded the behaviour BG2.)

4.4. Creating a Temporal SON Model

Temporal Structured Occurrence Nets (T-SON), allow the use of temporal abstraction to define atomic actions, i.e., actions that appear to be instantaneous to their environment. Intuitively, a T-SON shows a system abbreviation, i.e. of that part of the behaviour that is hidden by the abstraction (Figure 15). In SONCraft, the user is able to create such an atomic action by using what is termed a *block.*

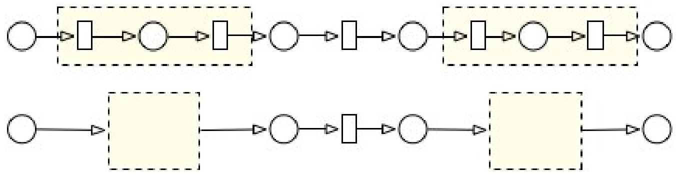


Figure 15. System abbreviation.

The means of defining a block is similar to the group function: use to select a set of nodes and combine them as a block by clicking the *Block* button . One can use to destroy a block (i.e. make the set of nodes editable again). A block is differentiated graphically from a group by having a dotted rather than solid line bounding box, and a shaded interior.

A block can be collapsed by either double-clicking inside it or by changing its *IS Collapsed* property in the Property editor. The behaviour of a collapsed block is regarded as an (abstract) event, where the connectivity between this event and those conditions which survived collapsing (i.e., the block's interface) is inherited from the events that have been collapsed.

Due to the special behaviours of the block, several restrictions apply to block creation (see [1] for more details). Attempted violations of some of these restrictions are recognised and prohibited immediately during the creation operation, i.e., a block's interface must be conditions; a block can involve only conditions and events; and it cannot cross phases. Other restrictions can be detected through the use of verification tool, e.g., the requirement for causal relations between block inputs and outputs; and for acyclicity (see the example below).

As an example, the SON model on the left in Figure 16 contains two (un-collapsed) blocks *b0* and *b1* represented by dotted-line boxes with light green background colour, while the right-hand side figure shows *b1*'s collapsed state. Note that a collapsed block may potentially affect the behaviour of a SON Model. For example, the collapsing of *b1* in Figure 16 (right) has led to an invalid asynchronous cycle in the SON. Such an error can be detected by the structural verification tool (see Section 5).

Note: *The present version of SONcraft does not allow blocks to be nested.*

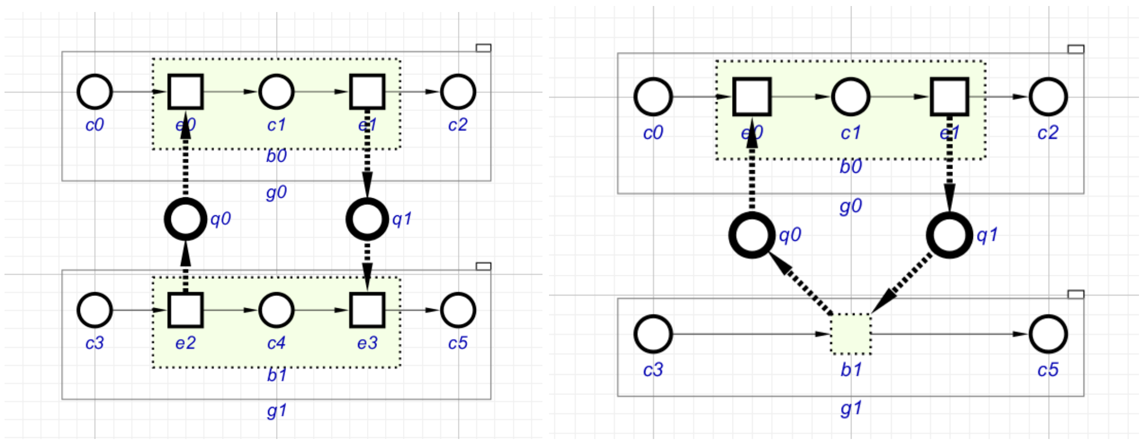
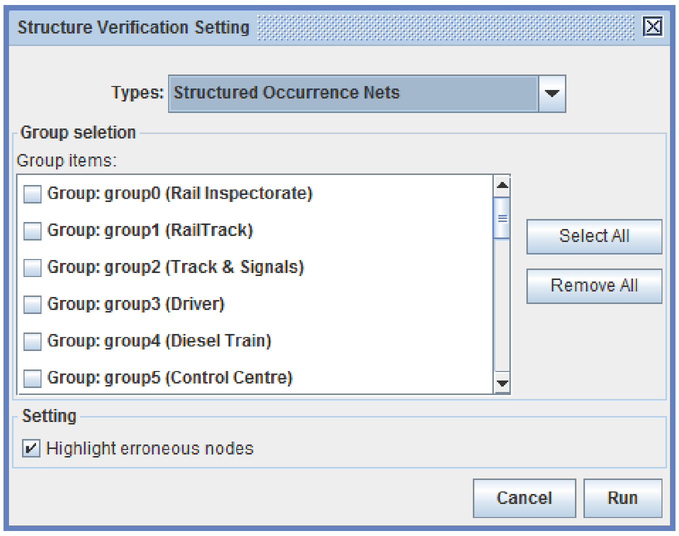


Figure 16. A SON model with two (un-collapsed) blocks (left), and with one of the blocks collapsed (right).

Note: *the SON shown in the right hand side of Figure 16 fails verification because of the cycle. However if one also collapses b0, then what results is a SON with a synchronous connection between b0 and b1, so verification succeeds.*

5. Verifying the correctness of ON and SON structures

SONCraft provides the user with a set of structural verification algorithms that can be used to validate the model. For example, the cycle detection tasks check SONs' acyclic property and the phase correctness tasks are used to verify valid phases in B-SONs. The user can invoke verification by selecting **Tool** -> *Verification -> Check for structural properties*. The verification setting dialog (Figure 17) allows users to specify the particular SON's model type as well as which groups they would like to verify. The results of the verification are detailed in a verification report, and are shown by colouring the SON model (Figure 18). Thus the borders of conditions and events that are implicated in a cycle are coloured red, and invalid nodes (e.g. an initial or final event, of a condition linked to multiple events) have their interiors coloured pink.


Structure Verification Setting

Types: Structured Occurrence Nets

Group seletion

Group items:

- [ ] Group: group0 (Rail Inspectorate)
- [ ] Group: group1 (RailTrack)
- [ ] Group: group2 (Track & Signals)
- [ ] Group: group3 (Driver)
- [ ] Group: group4 (Diesel Train)
- [ ] Group: group5 (Control Centre)

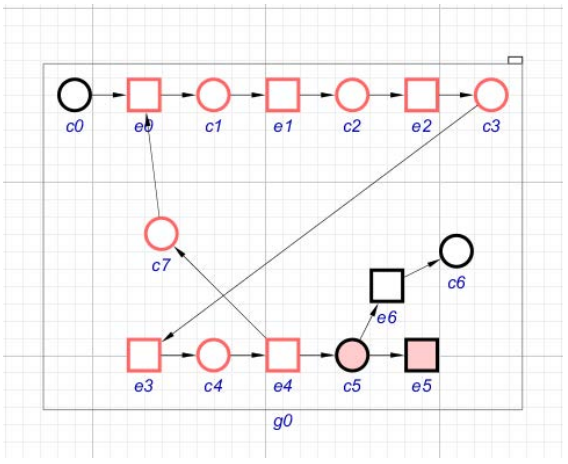
Setting

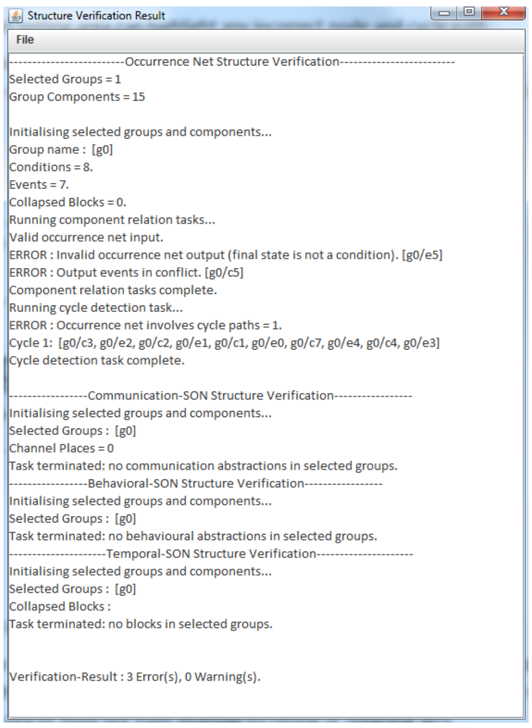
- [✓] Highlight erroneous nodes

Figure 17. Verification Setting dialog.

The *highlight erroneous node* selection in the setting area can highlight any incorrect node and cycle path.

The **File** menu at the top of the Verification Result window (see Figure 18) can be used to *Export* the results to a text file. This will by default be to the Home directory, so the user will need first to navigate to this directory if so wished.

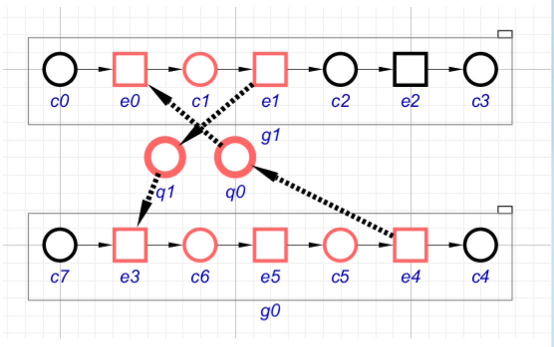


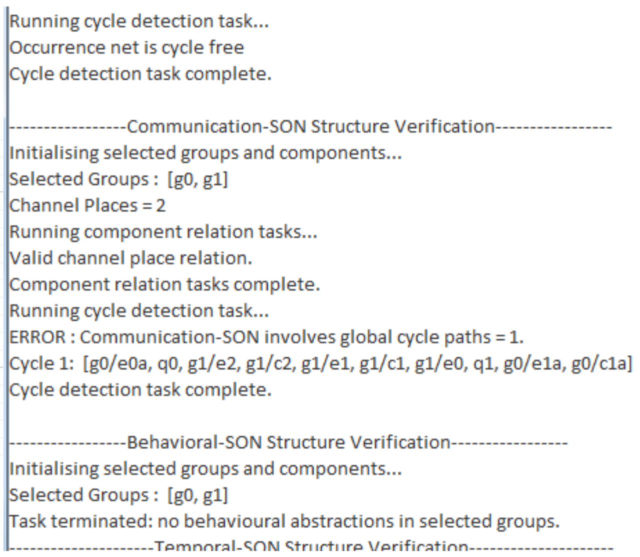


```
-----------------------Occurrence Net Structure Verification-----------------------
Selected Groups = 1
Group Components = 15

Initialising selected groups and components...
Group name :  [g0]
Conditions = 8.
Events = 7.
Collapsed Blocks = 0.
Running component relation tasks...
Valid occurrence net input.
ERROR : Invalid occurrence net output (final state is not a condition). [g0/e5]
ERROR : Output events in conflict. [g0/c5]
Component relation tasks complete.
Running cycle detection task...
ERROR : Occurrence net involves cycle paths = 1.
Cycle 1:  [g0/c3, g0/e2, g0/c2, g0/e1, g0/c1, g0/e0, g0/c7, g0/e4, g0/c4, g0/e3]
Cycle detection task complete.

----------------Communication-SON Structure Verification----------------
Initialising selected groups and components...
Selected Groups :  [g0]
Channel Places = 0
Task terminated: no communication abstractions in selected groups.
----------------Behavioral-SON Structure Verification----------------
Initialising selected groups and components...
Selected Groups :  [g0]
Task terminated: no behavioural abstractions in selected groups.
-------------------Temporal-SON Structure Verification-------------------
Initialising selected groups and components...
Selected Groups :  [g0]
Collapsed Blocks :
Task terminated: no blocks in selected groups.


Verification-Result : 3 Error(s), 0 Warning(s).
```

Figure 18. Result of an attempted ON verification.

Figure 18 shows how the ON verification tool reports (both pictorially and textually) the results of checks for freedom from cycles, caused by arcs within ONs, the requirement for ON fragments to start and terminate in places and not events, and the fact that places must not have multiple incoming or outgoing arcs.





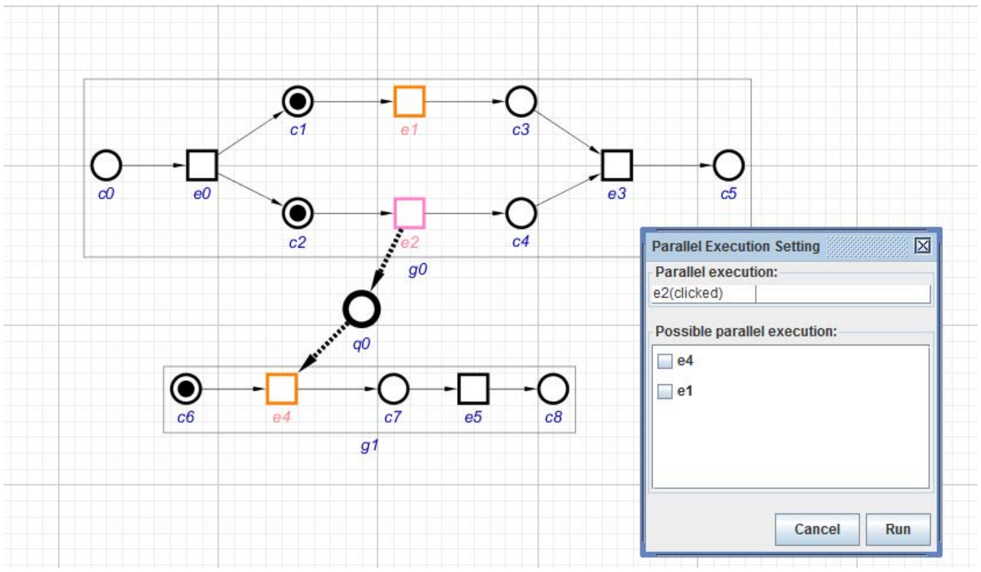
```
Running cycle detection task...
Occurrence net is cycle free
Cycle detection task complete.

----------------Communication-SON Structure Verification----------------
Initialising selected groups and components...
Selected Groups :  [g0, g1]
Channel Places = 2
Running component relation tasks...
Valid channel place relation.
Component relation tasks complete.
Running cycle detection task...
ERROR : Communication-SON involves global cycle paths = 1.
Cycle 1:  [g0/e0a, q0, g1/e2, g1/c2, g1/e1, g1/c1, g1/e0, q1, g0/e1a, g0/c1a]
Cycle detection task complete.

----------------Behavioral-SON Structure Verification----------------
Initialising selected groups and components...
Selected Groups :  [g0, g1]
Task terminated: no behavioural abstractions in selected groups.
--------------------Temporal-SON Structure Verification--------------------
```

Figure 19. Result of an attempted C-SON verification.

In Figure 19 events and conditions that are shown with red borders form a cycle involving asynchronous communications links joining multiple ONs. (**Tools** -> *Set colour to default* can be used to remove these error indications.)

6. The SON simulator

The simulation function in SONCraft can be activated by clicking on the *Simulation* button ▶. The initial marking will be automatically set, i.e. all the input conditions of all the ONs will be filled with black tokens (except for those ONs that are 'waiting' for an event in another ON), and all of the thereby enabled events (including collapsed blocks, i.e. temporal abstractions) will be identified by being coloured orange. (Conditions and events that have not been grouped and made into an ON are not affected by simulation.) The simulation can then be conducted manually by clicking on a succession of enabled events, so causing tokens to move, and event colouring to be updated, and the simulation record augmented. At any stage a simulation can be abandoned, and any tokens and event colouring removed.



Parallel Execution Setting

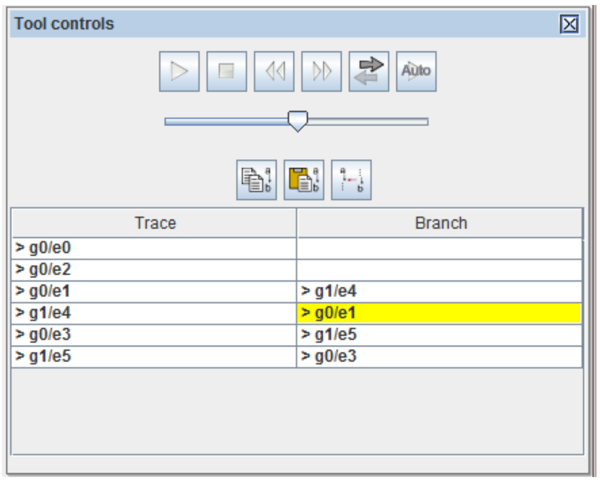
Parallel execution: e2(clicked)

Possible parallel execution:

- ☐ e4
- ☐ e1

Figure 20. Parallel execution setting.

During simulation a dialog box will pop up when an enabled event is clicked and there exist other enabled events. This box can be used to select one or more of these additional enabled events in order to execute them simultaneously with the checked event. As an example, the clicked enabled event in Figure 20 is *e2*, with events *e1* and *e4* shown as two possible further parallel-executed events. When ‘Run’ is clicked the simulation will move on just from *e2*, as *e1* and *e4* have not been selected.



Trace	Branch
> g0/e0	
> g0/e2	
> g0/e1	> g1/e4
> g1/e4	> g0/e1
> g0/e3	> g1/e5
> g1/e5	> g0/e3

Figure 21. Simulation control window.

The simulation tool controls provide the means to analyse and navigate a recorded simulation. There are two sources of data for a simulation record. They are shown in the two columns of the table in the Simulation Tool Controls window (e.g. Figure 21).

Branch (the right hand column) is used to record the firing sequence of events that were executed by explicitly clicking the enabled nodes of the model. The marks ‘>’ and ’<’ represent forward and reverse directions of the simulation respectively. These directions can be switched by using .

Note: *Clicking on any of the editor buttons will cause the simulation record to be deleted.*

Trace (the left hand column) plays the role of a "base sequence of events". Initially, when one starts to use the simulation tool, this column will be blank. However it can be populated from a firing sequence that has been recorded in Branch (the right hand column). This sequence can be moved into Trace using the operation "Merge branch into trace", leaving the Branch table empty.

Note: *Trace can be also filled directly from other tools, e.g., with a firing sequence produced by the reachability tool – see Section 7.*

While there is a firing sequence recorded in Trace or Branch, clicking on one of the entries (in Trace or Branch) causes the simulator to restore the model to its state just before that event has happened, i.e. with the tokens positioned, and the nodes enabled, appropriately.

The user can explore variants of a recorded trace by clicking on enabled events of the model. If one makes the same choice of enabled event(s) as before, nothing will be changed in Trace or in Branch, depending on which column the user selected. The relevant column will only start getting filled in (if the selected start point is from Trace) or replaced (in the case of selecting a start point from Branch) when the new firing sequence diverts from the one already recorded in the table in the Simulation Controls window**.**

In Figure 21, the Trace firing sequence generated from the simulation result of Figure 20, which could have started from the beginning (*g0/e0*), only starts getting recorded, so creating a diverted sequence in Branch, when the user chooses *g1/e4,* instead of g0/e1 – two events which were both enabled. Thereafter, the firing sequence continues to be recorded in Branch as enabled events are selected. When one has such a diverted firing sequence the operation "Merge branch into trace" has the effect of replacing the relevant section of the earlier firing sequence that is recorded in Trace with the new sequence from Branch.

When the simulation button has been selected, and an entry in the branch or trace column chosen, the appropriate simulation tool control panel buttons become active, and provide access to several additional simulation functions, most of which relate to simulation traces:

- *Playback* – automatically playback an existing trace, at a selectable speed. (As each element in the trace is reached its record is turned yellow.)
- *Reset* – reset the simulation to its starting position and clear the trace record.
- *Step backward* – step backward through an existing trace, highlighting (in yellow) the trace record reached.
- *Step forward* – step forward through an existing trace, highlighting (in yellow) the trace record reached.
- *Reverse simulation* – switch to reverse simulation and to showing (for possible subsequent forward simulation) and clear the trace record.
- *Forward simulation* – switch to forward simulation and to showing (for possible subsequent reverse simulation) and clear the trace record.
- *Automatic simulator* – uses maximum parallelism through to the end.
- *Copy* – copy trace or branch to the clipboard.
- *Paste* – reload the stored trace or branch from the clipboard.

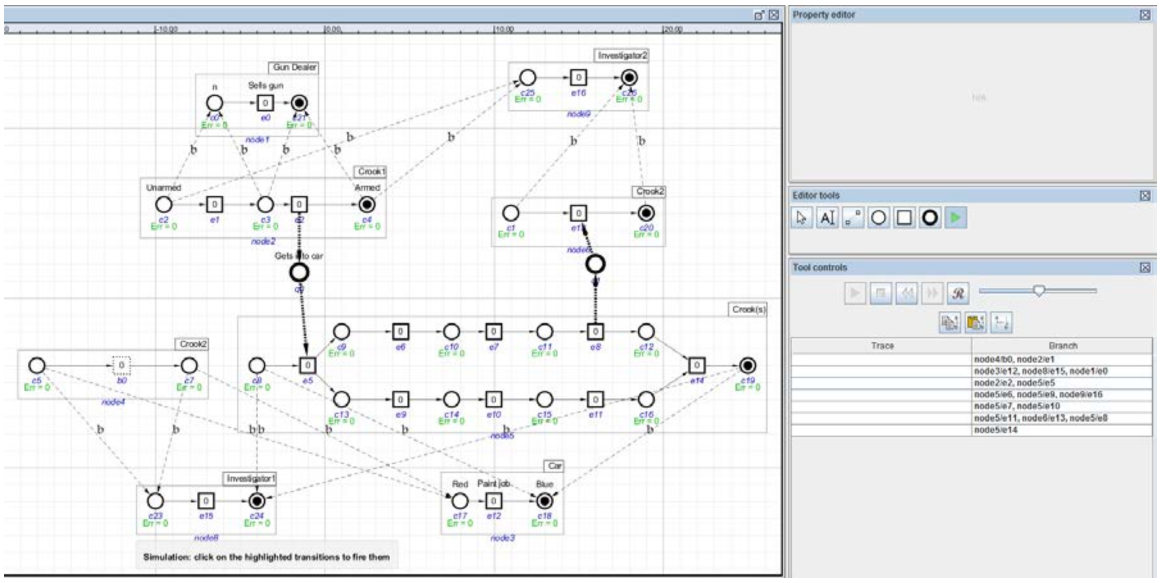
- *Merge* – merge branch into trace.

The slider bar can be used to set the speed at which trace playbacks will occur, and the user can stop and abandon a simulation using the select button.

6.1. Error simulation

Each event has associated with it a Fault bit (visible in the Property editor when the relevant event is selected). This bit can be used to indicate whether the user wishes to regard the event as a faulty one. It can be turned on or off (shown by being checked or unchecked in the Property editor) using **Tool** -> *Error Tracing -> Enable/Disable error tracing*. A collapsed block (representing an abstract event) will be automatically marked with a faulty state when it contains at least one faulty event. If *error tracing* is turned on then the fault status of each event is also indicated in the Editor window, each event being flagged with a “1” or a “0”, “1” indicating a simulated fault. When error tracing is on, an error count is also shown below each condition, set initially to “0”. This count is not visible in a condition’s Property editor window, and cannot be changed manually by the user. Rather it is automatically calculated during simulation to indicate for each condition the number of faults that have been passed on the forward route to the condition. (These error counts are not affected by reverse simulation, or by closing a work and reopening it.)

When a simulation is started, using the *Simulation* button, all the error counts are set to zero, but the fault status of the events is not changed – in fact it becomes unchangeable while simulation is being used. Turning error tracing on and off has no effect on the fault flags and error numbers.



Trace	Branch
	node4/b0, node2/e1
	node3/e12, node8/e15, node1/e0
	node2/e2, node5/e5
	node5/e6, node5/e9, node9/e16
	node5/e7, node5/e10
	node5/e11, node6/e13, node5/e8
	node5/e14

Simulation: click on the highlighted transitions to fire them

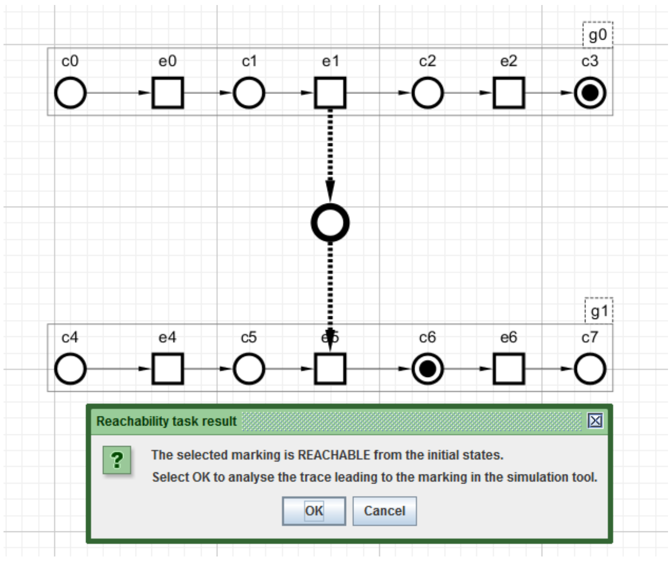
Figure 22. Towards the end of a simulation run.

Tools -> *Error tracing -> Reset fault/error states* is used to re-initialise all numbers to zero.

7. Reachability tool

A SON can be verified for reachability using the Reachability Checking tool. Such analysis establishes whether a given marking (a set of marked conditions and/or channel places) can be reachable at the same time from the initial states. In other words, any marking involving two causally related nodes is not reachable since one of these nodes must happen before the other can.

To use the function, mark some conditions/channel places first by either changing their '*isMarked*' properties in the Property editor or by double clicking on them. Then run **Tool** -> *verification -> check for reachability*. A dialog box will pop up to report the task's result. If the verified marking is reachable, then a request can be made for the trace leading to the marking to be passed to the simulation tool for playback or further analysis (Figure 23).



The selected marking is REACHABLE from the initial states.
Select OK to analyse the trace leading to the marking in the simulation tool.

Figure 23. Reachability task result for a SON model with two marked conditions.

8. Node Grouping

	Valid components	Enter/Edit	Leave	Collapse
Group ▨	○; □; A; ▨	*Double-Clicking;* ⬇	*Double-Clicking outside;* ⬆	*N/A*
Block ▨	○; □; A	*N/A*	*N/A*	*Double-Clicking; Is Collapsed (Property Editor)*

Table 1. Summary of valid component types and general operations for group-based nodes.

9. Printing

To get a printable version of a Work file, export a svg version from **File** -> *Export* -> *.svg*, and open this using Firefox (NOT Safari, on a Mac) or Inkscape.

10. References:

[1] M. Koutny and B. Randell, "Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques," *Fundamenta Informaticae*, vol. 97, no. 1, pp. 41–91, Jan. 2009.

[2] B. Randell, "Occurrence nets then and now: the path to structured occurrence nets," in *Applications and Theory of Petri Nets*. Springer, Berlin Heidelberg, Jun. 2011, pp. 1–16.

[3] B. Randell, *Incremental Construction of Structured Occurrence Nets*. School of Computing Science, Newcastle University. School of Computing Science Technical Report Series 1384, 2013.

[4] I. Poliakov, *Interpreted Graph Models*. PhD thesis, Newcastle University, 2011.

[5] J. Kleijn and M. Koutny, “Causality in structured occurrence nets,” in *Dependable and Historic Computing*, vol. 6875. Springer Berlin Heidelberg, 2011, pp. 283–297.