SONCraft: A Tool for Construction, Simulation and Analysis of Structured Occurrence Nets

Bowen Li, Brian Randell, Anirban Bhattacharyya, Talal Alharbi, and Maciej Koutny
School of Computing, Newcastle University
Newcastle upon Tyne NE1 7RU, UK
{bowen.li2, brian.randell, anirban.bhattacharyya, talal.alharbi, maciej.koutny}@ncl.ac.uk

***Abstract*—This paper presents SONCraft - an open source tool for editing, simulating, and analysing Structured Occurrence Nets (SONs), which is a Petri net-based formalism for portraying the behaviour of complex evolving systems. The tool is implemented as a Java plug-in within the Workcraft platform, which is a flexible framework for the development and analysis of Interpreted Graph Models. SONCraft provides an easy to use graphical interface that facilitates model entry, supports interactive visual simulation, and allows the use of a set of analytical tools. We give an overview of SONCraft functionality and architecture.**

***Index Terms*—SONCraft, system modelling and analysis, structured occurrence nets, complex evolving systems**

I. Introduction

The concept of Structured Occurrence Nets (SONs) [8], [17] is an extension of that of Occurrence Nets (ONs) [4]. ONs are directed acyclic graphs that represent causality and concurrency information concerning a single execution of a system. The SONs formalism has been introduced to facilitate the portrayal and analysis of the behaviours, and in particular failures, of *complex evolving* systems. Such systems are composed of a large number of sub-systems which may proceed concurrently and interact with each other and with the external environment while their behaviour is subject to modification by other systems. Examples include a large hardware system which suffers component break-downs, reconfigurations and replacements, a large distributed system whose software is being continually updated (or patched), a gang of criminals whose membership is changing, and an operational railway system that is being extended. (In these latter cases we are regarding crimes and accidents as types of failure.) The underlying idea of a SON is to combine multiple related ONs by using various formal relationships (in particular, abstractions) in order to express dependencies among the component ONs. By means of these relations, a SON is able to portray a more explicit view of system activity, involving various types of communication between subsystems, and of system upgrades, reconfigurations and replacements than is possible with an ON, so allowing one to document and exploit behavioural knowledge of (actual or envisaged) complex evolving systems.

Communication Structured Occurrence Nets (CSONs) are a basic variant of structured occurrence nets that enable the explicit representation of synchronous and asynchronous interaction between communicating subsystems. (The original occurrence net concept provided no means of distinguishing the events and states that resulted from different subsystem's activities.) A CSON is composed of a set of distinct component ONs representing separate subsystems. When it is determined that there is a potential for an interaction between subsystems, an asynchronous or synchronous communication link can be made between events in the different subsystems' ONs via a special element called a channel place[1].

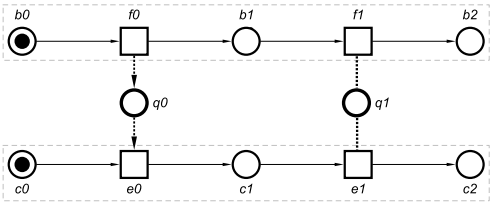

Fig. 1. A CSON with two interacting occurrence nets.

Figure 1 shows a CSON which consists of two interacting occurrence nets connected via two channel places (portrayed graphically by bold circles). The thick dashed connection between events f_0 and e_0 is an asynchronous communication, which means that e_0 cannot happen before f_0. Events f_1 and e_1 are connected via a single channel place with undirected arcs, indicating the two connected events can only be executed synchronously and behave as a transaction. Such communications entail token instantaneously picked up by the other party via channel place and provide possibility to execute multiple events in a single step.

A second variant of structured occurence nets, Behavioural Structured Occurrence nets (BSONs), conveys information about the evolution of individual systems. They use a two-level view to represent an execution history, with the lower level providing details of its behaviours during the different evolution stages represented in the upper level view. Thus a BSON gives information about the evolution of an individual system, and the phases of the overall activity are used to represent each successive stage of the evolution of this system.

Figure 2 shows a simple example of a BSON portraying an (off-line) system update. The upper level represents a

[1]Communication relations were represented by a directed dashed line between two events in the original definition of CSONs [8]. The notion of a channel place, which was introduced later [7], is a more flexible means of representing such relations.

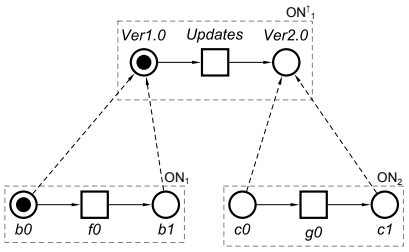

Fig. 2. A BSON example portraying (off-line) system update.

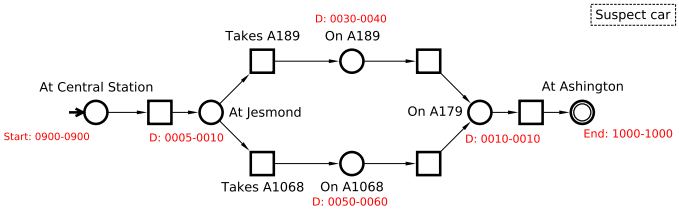

Fig. 3. The modelling of alternate scenarios with time information using Alternative and Time SONs.

version change caused by a system update event — in effect it shows the behaviour of the "updater", be this a person or an automated system. The lower level represents the detailed behaviour of the system before and after the update. The dashed lines between the two levels are used to capture the some new causal dependencies between the two types of behaviours. Intuitively, it represents the 'happened before' relationship of the events.

A third variant, Temporal Structured Occurrence Nets (TSONs), allows the use of temporal abstraction to define atomic actions, i.e. actions that appear to be instantaneous to their environment. The effective use of atomicity can significantly reduce the cognitive complexity of structured system behaviours, by defining "abbreviated" parts of systems, whose detailed behaviours are collapsed and hidden within a lower level and replaced by simple symbols in an upper level representation.

The concept of the three basic SON variants mainly concerns the modelling of single executions of a system, and there is lack of any direct way of representing the possibility that a system might rise to possible alternative behaviours. The idea of adding alternates to SONs initially arose from [18] and was further extended in [9] for the propose of modelling and analysing system activities, e.g. major (cyber) crimes or accidents, that are likely to give result in a mass of contradictory or uncertain evidence regarding the actual activity. In an Alternative SON, multiple SONs (scenarios) can be merged into a single structure by 'gluing' common states together. So a state shared by more than one scenario can branch to multiple events with each corresponding to a different scenario; the different branches can subsequently merge from their respective end events to a single state, with the result that different scenarios share the same state. Therefore, the representation of such structure is more flexible than branching processes [6] since in such processes two branches outgoing from a condition will never meet again.

Figure 3 shows an example of Alternative SONs involving two scenarios. The model describes the behaviours of a suspect car. We assume that there is no further information captured after "At Jesmond", where there are two roads that the driver can potentially take to Ashington. In order to assist with the analysis of which road the driver did actually take, the hypotheses can be represented using the alternative behaviours "Takes A189" and "Takes A1068".

In [5], we introduced time information to the SONs concept. This was motivated by the fact that time is another commonly required and indeed often unavoidable feature of complex investigations of various types of system "Failure"; in crime or accident investigations, for example, it is important to establish the sequence in which events have occurred (i.e. the chain of events) and to establish the time durations between events. This was in order to help to determine causes and their effects, e.g. by eliminating infeasible hypothesized scenarios from a conjectured criminal act. We also allowed for uncertainty involving time - specifically, uncertainty about the time of occurrence or the duration of an event, and the time at which a state comes into existence, or how long it lasts.

The visual editing of SONs, their extensions, w.r.t. time and alternatives, verification, simulation, and analysis are the functionalities supported by the SONCraft toolkit. The toolkit is implemented as a plug-in module within Workcraft [15], a system that provides a flexible framework for the development and analysis of Interpreted Graph Models (IGM) [14]. The framework is built using a plugin-based architecture and supports run-time scripting, which makes it easily extensible to new IGM-based formalisms, and to the provision of support for their analyses and verification. It also provides a GUI environment that facilitates model entry and supports interactive visual simulation, together with convenient "single-click" verification. So far several modules have been implemented and supported by the platform, including Structured Occurrence Nets (SONCraft), Petri nets, and other Petri net based formalisms, e.g. STGs [21] and CPOGs [12]. A detailed Workcraft description and manual can be found in [3].

In the remainder of this paper we give an overview of SONCraft functionality and architecture. The present version of SONCraft deals with the three types of SON variants mentioned above, i.e. CSONs, BSONs, and TSONs. In Section II, we present the key functionality offered by SONCraft. Section III describes the tool architecture and the way in which SONCraft is integrated with the Workcraft framework. Section IV provides a comparison with other tools for SONCraft. Section V provides installation information. Section VI concludes the paper and outlines ongoing and future work. An associated technical report [10] contains formal definitions, properties, and the algorithms that are used in the implementation.

II. Functionality

This section presents an overview of the major features provided by SONCraft.

A. SONCraft overview

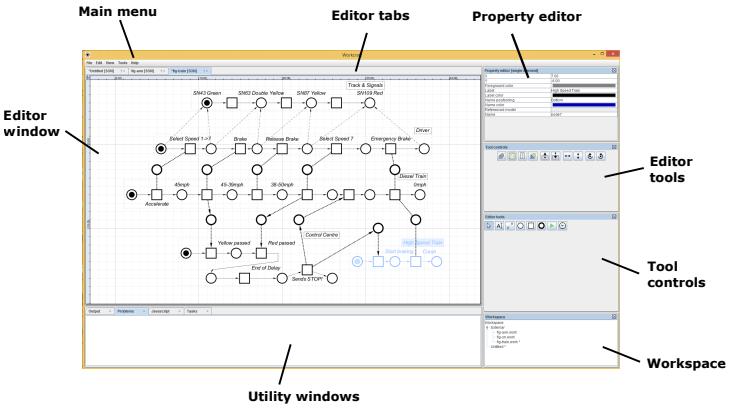

Fig. 4. SONCraft interface.

The graphical interface of SONCraft is depicted in Figure 4. The *Main menu* provides functions to manage, edit and analyse models. For example, the *Tools* menu provides a set of analysis tools for checking models, and there is a vector graphics export function in the *File* menu. (All SON models shown in Section I were imported directly from SONCraft.) The *Editor tabs* line shows the names of all of the opened models and allows the user to choose which one is to be displayed in the *Editor window*. The latter is the place for the user to create, edit and simulate a SON model.

SONCraft defines a series of graphical nodes and connection types. These are displayed in the *Editor tools* panel that allows the user to create, edit SON-based model and also save the models as XML format. The *Property editor* panel at the top-right of Figure 4 is used to support various visual node-editing operations, e.g. to change the label, colour, or position of a node. The *Tool controls* panel provides access to the extended functionality of a selected tool. For example, when the connection tool is activated, the user is able to switch between various connections in order to construct different types of SON abstractions. The *Workspace window* lists opened or imported SON model files. One can also operate on a SON model file (delete, save, etc). The *Utility window* is used for showing additional information concerning the progress of currently executed tasks, verification results, and information about any errors that may have occurred during execution.

B. Editor tools

SONCraft offers a set of editor tools for constructing and editing SON models. Some generic tools for editing models are directly inherited from the Workcraft framework, including selection, text note, flip horizontal, flip vertical, etc. Other tools for constructing SON models, defined specifically in SONCraft, are as follows:

- The *SON component toolkit* contains a group of buttons for creating SON-based components in the editor window. The toolkit contains condition, event, and channel place creators, where the first two creators are used for constructing ONs, and the latter is for CSON construction.
- The *Connection tool* is used for creating relations between nodes. The tool provides several connection types that can be chosen to construct different abstractions. The tool also offers a basic relation validation facility, based on the SON definition. Any invalid user operation, e.g. connecting two conditions, will trigger an error message. However, not all user errors are detected at this stage; those that cannot be detected immediately can be checked using the verification tool – see the section C below.
- The *Group tool* allows the user to combine a set of nodes (in particular, conditions, events, and ON relations) into a group. In SONCraft, an ON is not recognised as such until it has been delineated as a group. Thus, a SON model in SONCraft is generally composed of a set of groups representing component ONs, and the relations (abstractions) between the groups' components.
- The *Block tool* creates atomic actions in TSONs. Similarly to the group tool, a block in SONCraft is implemented as a container holding a set of user-selected components. It can be collapsed into a single node causing its components to be hidden.

C. Structural analysis tool

The structural analysis tool provides the user with a set of structural verification algorithms that can be used to validate a model. It is important to verify the correctness of structure before further analysis, otherwise the results are likely to be incorrect. The verification criteria follow from the formal definitions and properties introduced in [8], [10]. The tool consists of two sub-checkers:

The *relation property checker* deals with the relation-based correctness of a SON model. The checking includes conflict-freeness, phase decomposition, component ONs disjointness, and advanced connection type checking. The *acyclic property checker* focuses on the acyclicity condition of SONs. The verification of such a property comes down in practice to searching strongly connected components (SCC) in a SON model. The checker applies Tarjan's algorithm [20] as a core engine to compute maximal SCCs. Depending on the variant of SONs involved, a specific filter is used in order to obtain the desired results.

D. Simulator

SONCraft offers a built-in simulator for ONs, CSONs, BSONs, and TSONs. The underlying semantics of SON-based simulation follows the firing rules presented in [10]. When the function is activated, the initial marking is then automatically set, and all enabled events highlighted. The simulation can then be conducted either manually or automatically, by firing

a succession of enabled events, causing tokens to move, event highlighting to be updated, and the simulation record augmented. Simulations can be performmed either forwards, to investigate what further activity was caused, or backwards, to investigate the cause(s) of a given situation.

The simulation tool controls provide the means to analyse and navigate a previously recorded simulation. There are two sources of data for a simulation record: a 'branch' records the firing sequence of events that were executed by explicitly clicking the enabled nodes of the model, and 'traces' are automatically generated from the external tools, e.g. the reachability tool. Thus one is able to generate simulation records from different executions in order to perform a comparison. The panel also provides access to several additional simulation functions, most of which relate to the simulation traces or branches (see Figure 5).

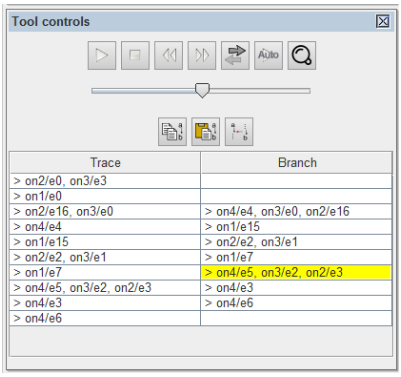

Fig. 5. Simulation control panel.

The SON simulator provides a simple failure analysis function, called error tracing. When the failure analysis function is on, each event has an associated *fault bit*, a '1' or a '0'. This bit can be used to indicate whether one wishes to regard the event as faulty, with '1' indicating a simulated fault. An error count is also shown in the editor window below each condition, and is set initially to '0'. This count 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 that condition.

E. Scenario generator

The *scenario generator* aims to generate particular scenario in a SON model with alternative behaviours. The tool applies the basic SONs execution semantics to simulate the model automatically using maximum parallelism. Nodes with alternative behaviours will be recorded as different scenarios. The tool also provides a set of easy to use features including save, remove, and display records. The generated scenario can be used in other analysis tools, such as time consistency and reachability checking. Figure 6 shows the interface of scenario generator. The "save list" column displays all saved scenarios, and the "scenario" column shows the details of the selected scenario.

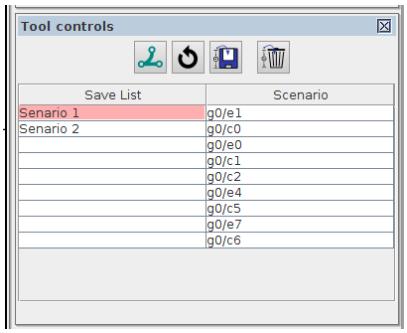

Fig. 6. Scenario generator with two saved scenarios

F. Time mode

The *time mode* was originally developed to display of the time information for basic SON model and offers related analysis algorithms (see [5] for details). To improve the solver, we enhanced its functionalities to additionally support for analysing time information in Alternative SONs (as shown in Figure 3). A *time property setting tool* allows for specifying the time information of a given node in a SON model. Users can either manually or automatically set the information. For each manually input value (which can take the form of a set of bounds) the tool verifies whether or not the value is well-defined (e.g. the start time must be earlier than the end time). The tool also estimates time bounds for nodes with unspecified time intervals. Depending on the user selection different strategies will be chosen based on forward or backward DFS (Depth First Search) and the time information on the path.

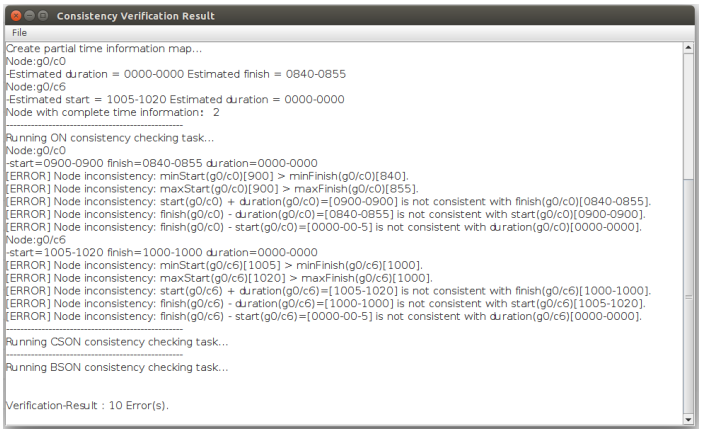

Fig. 7. Time consistency result of the scenario "Takes A1068" in Figure 3.

A *time consistency checking tool* provides consistency verification for the time value and bounds information that is

specified. Such checking aims to find any 'gap' or 'overlap' between successive times. The tool provides a user interface for additional settings. For example, the user can either choose to verify the time information of a particular scenario or of a single node. Figure 7 shows a consistency verification result of the scenario "Takes A1068" displayed in Figure 3. The result shows a number of errors regarding the inconsistency between the start and end time of the scenario. For example, the estimated time that the car was at Ashington based on presented information is between 8:40-8:55. This is calculated by performing a forward DFS and accumulating the time information on the path. However, such a value is inconsistent with the specified time 10:00.

G. Reachability Tool

SONCraft provides a tool for checking the reachability problem. This Reachability Checker is used to establish, e.g. whether a given marking can be reached from the initial marking. The main idea of the algorithm is to compute all the causal predecessors of required nodes (i.e. markings) and check that none of the required nodes is consumed by their causal predecessors. If the marking is reachable, a request can be made for the trace that leads to the marking to be passed to the simulation tool for playback or further analysis.

III. Tool Architecture

SONCraft is written in JAVA making it accessible on all platforms for which there exists a JVM. The architecture depicted in Figure 8 shows a detailed view of the integration between the Workcraft framework and SONCraft.

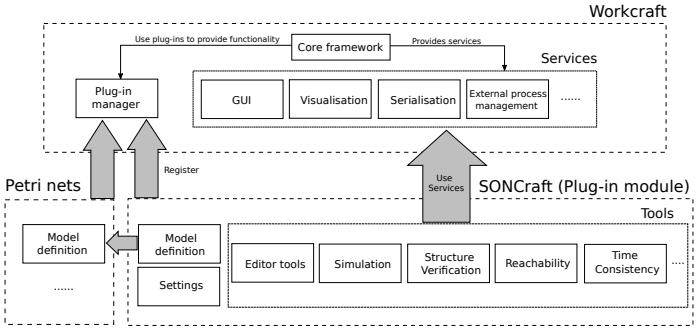

Fig. 8. Tool architecture.

A. Workcraft architecture

The Workcraft framework consists of the following three parts:

The *Core framework* is in charge of the initialisation of Workcraft, managing plug-ins and provision of common services to the plug-ins. When the program starts up, services such as the configuration manager and the framework GUI are initialised. This is followed by the initialising of the plug-in manager, which provides the facility for loading all existing plug-ins. On shut-down, Workcraft saves the configuration of the framework; it restores it on the next start-up.

The *Plug-in manager* is responsible for scanning and loading all plug-in modules which have been registered to the manager. A plug-in module is a related collection of plug-ins that together implement a specific functionality, for instance the SONs module.

The *Services* are fully managed by Workcraft and accessible to the plug-ins. The GUI service provides the facilities for creating editor, tool, and information windows. A number of advanced GUI capabilities, such as the multiple document interface and full-screen mode, are also supported. The Visualisation service facilities provide editing functions for the node types defined by any model, for instance, drawing, transformation, and auxiliary editing operations. The Task management service is responsible for executing all external process tasks – it maintains the list of all running tasks and uses a separate thread for their parallel execution.

B. SONCraft integration

SONCraft is deployed in the Workcraft framework as an individual plug-in module. There are three main components inside the module:

The *Model definition* component describes the basic features of a SON model. The component is divided into mathematical and visual models in order to avoid mixing unrelated responsibilities. Both models extend the Workcraft Petri-nets plugin with a number of additional SON-based node and connection types. The mathematical model describes all the semantics concerning model integrity — it keeps information such as connection types, node names, etc. The visual model provides all the information concerning model visualization. The model classes defines how to draw and present the model, keep information such as colors, position, etc.

The *Settings* component records default properties of a SON model and stores them in a configuration XML file. The Workcraft start-up process loads the stored settings and allows other components to read their configuration variables.

The *Tools* component consists of all the editor and analysis tools described in Section II. The implementation of each built-in component tool uses the interfaces provided by Workcraft services and other components. In particular, the development of each tool's graphical interfaces rely on the GUI and visualisation services for node placement, trace table creation, etc. The implementation of the various algorithms are based on the mathematical model defined in the model definition component.

IV. Comparison with Other Tools

Currently, SONCraft is the only tool that supports of modelling and analysing systems using SONs. There are existing tools for modelling concurrency and alternatives in a system's behaviour based on occurrence nets. POD [16] relies on labelled partial orders and occurrence nets to build up concurrent and causal relations on the activities of an event log for the process discovery. Workcraft's CPOG plug-in implements Conditional Partial Order Graphs [12] - a model that can be used for graph-based representation for complex

concurrent systems, whose behaviour could be thought of as a collection of many partial order scenarios. However, none of these tools facilitate the explicit structuring and modelling of system evolution or communication.

Numerous time extensions have been proposed for Petri nets, and they are well supported by tools such as the CPN Tool [19]. However, most of them only consider time information as an abstraction at the level of system specification, and research on attaching time information to the level of system behaviour is limited. Moreover, there is lack of research on, and tool support for, modelling uncertainty, consistency checking, and computation missing time information.

V. Availability

The latest version of SONCraft is available from the Workcraft homepage [3]. It is necessary to have a compatible Java Runtime Environment (JRE) version 7 or higher in order to run the platform and SONCraft. There is no automatic installer for the platform; to install it, the files from the link archive need to be extracted manually. A comprehensive online SONCraft user manual can be found in [1], and [2] provides a tutorial showing how to use SONCraft for modelling and analysing crimes and accidents.

VI. Conclusion and future work

In this paper, we introduced the SONCraft tool-kit for the construction, checking, simulation and analysis of SONs. SONCraft provides an easy to use graphical interface that enables the user to construct models easily and quickly, and offers a powerful simulator and a set of analytical tools. The SONs concept and the SONCraft are now being extended to include further facilities, as described below. An interesting and practically important property of the SONs and its application is support for associating probability estimates. Such an idea has been addressed in [11]. Briefly, a user will be able to annotate particular nodes and relations with some form of probability estimates, which indicate the current degree of accuracy of their representation. This probability information can guide the analysis of the likelihood of different scenarios. Integrating SONCraft with a probabilistic analyser such as [13] is a possibility.

In Time SONs, associating time information with SON execution semantics and its implementation are left for future work. (The determination of whether an event is enabled not only be determined by marked states, but also by the consistency of the time information provided by the event and its input states.)

To date, SONCraft has been assessed and experimented with using a variety of manually-entered artificial trial models, the ongoing work includes integrating the system with an existing large-scale DBMS-based crime investigation support system. Such integration will, we expect, provide the most practical means of assessing the effectiveness of the system's facilities for structuring and hence coping with, realistic large and complex evolving models.

Acknowledgment

The authors would like to thank Danil Sokolov and Stanislavs Golubcovs for their helpful advice. The work is supported by UK EPSRC EP/K001698/1 UNderstanding COmplex system eVolution through structurEd behaviouRs (UNCOVER) project.

References

[1] *SONCraft online user manual*, https://workcraft.org/help/son_plugin/.

[2] *SONCraft Tutorial*, https://workcraft.org/tutorial/modelling/son/start/.

[3] *Workcraft homepage*, http://workcraft.org.

[4] E. Best and R. Devillers, "Sequential and concurrent behaviour in petri net theory." *Theoretical Computer Science*, vol. 55, no. 1, pp. 87–136, 1987.

[5] A. Bhattacharyya, B. Li, and B. Randell, "Time in structured occurrence nets," in *Proceedings of the International Workshop on Petri Nets and Software Engineering 2016*, 2016, pp. 35–55.

[6] J. Engelfriet, "Branching processes of petri nets," *Acta Informatica*, vol. 28, no. 6, pp. 575–591, 1991.

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

[8] 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.

[9] B. Li, "Visualisation and analysis of complex behaviours using structured occurrence nets," Ph.D. dissertation, School of Computing Science, University of Newcastle, 01 2017.

[10] B. Li, B. Randell, and M. Koutny, "Soncraft: A tool for construction, simulation and verification of structured occurrence nets," School of Computing Science, Newcastle University, Tech. Rep. CS-TR-1493, 2016.

[11] M.Koutny and B.Randell, "Structured occurrence nets: incomplete, contradictory and uncertain failure evidence," School of Computing Science, Newcastle University, Tech. Rep. CS-TR-1170, 2009.

[12] A. Mokhov and A. Yakovlev, "Conditional partial order graphs: Model, synthesis, and application," *Computers, IEEE Transactions on*, vol. 59, no. 11, pp. 1480–1493, Nov 2010.

[13] M.Z.Kwiatkowska, G.Norman, and D.Parker, "PRISM 4.0: verification of probabilistic real-time systems," in *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, 2011, pp. 585–591.

[14] I. Poliakov, "Interpreted graph models," Ph.D. dissertation, School of Electrical, Electronic and Computer Engineering, Newcastle University, 2011.

[15] I. Poliakov, V. Khomenko, and A. Yakovlev, "Workcraft–a framework for interpreted graph models," in *Applications and Theory of Petri Nets*. Springer Berlin Heidelberg, Jun. 2009, pp. 333–342.

[16] H. Ponce de León, C. Rodríguez, and J. Carmona, "POD - A tool for process discovery using partial orders and independence information," in *Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2, 2015.*, 2015, pp. 100–104.

[17] 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.

[18] B. Randell and M. Koutny, "Structured occurrence nets: Incomplete, contradictory and uncertain failure evidence," School of Computing Science, Newcastle University, Tech. Rep. CS-TR-1170, Sep. 2009.

[19] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, and K. Jensen, "Cpn tools for editing, simulating, and analysing coloured petri nets," in *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets*, ser. ICATPN'03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 450–462.

[20] R. Tarjan, "Depth first search and linear graph algorithms," *SIAM Journal on Computing*, 1972.

[21] A. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli, "A unified signal transition graph model for asynchronous control circuit synthesis," in *Proceedings of the 1992 IEEE/ACM International Conference on Computer-aided Design*, ser. ICCAD '92. Los Alamitos, CA, USA: IEEE Computer Society Press, 1992, pp. 104–111.