COMPUTING
SCIENCE

Verification and Simulation Tool for Communication Structured Occurrence Nets

Bowen Li
Verification and Simulation Tool for Communication Structured Occurrence Nets

Bowen Li

Abstract

Occurrence nets (ONs) are acyclic Petri nets recording single system executions. Structured occurrence nets (SONs), composed of multiple ONs together with additional relationships, are intended for portraying the behaviour of complex evolving systems. Such systems generally consist of a large number of sub-systems which may proceed concurrently and interact with each other. In this paper, we describe a tool support for visualisation, verification and simulation of occurrence nets, and of communication structured occurrence nets which are a basic variant of SONs.
Bibliographical details

LI, B.

Verification and Simulation Tool for Communication Structured Occurrence Nets
[By] B. Li


(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1363)

Added entries

NEWCASTLE UNIVERSITY

Abstract

Occurrence nets (ONs) are acyclic Petri nets recording single system executions. Structured occurrence nets (SONs), composed of multiple ONs together with additional relationships, are intended for portraying the behaviour of complex evolving systems. Such systems generally consist of a large number of sub-systems which may proceed concurrently and interact with each other. In this paper, we describe a tool support for visualisation, verification and simulation of occurrence nets, and of communication structured occurrence nets which are a basic variant of SONs.

About the authors

Bowen Li received his MSc in Computer Security and Resilience from Newcastle University in 2011. He is currently a PhD student under the supervision of Prof. Maciej Koutny, in the area of modeling concurrent systems using structured occurrence nets. Other research interests include fault tolerance and human factors.

Suggested keywords

OCCURRENCE NETS
STRUCTURED OCCURRENCE NETS
CONCURRENCY
VERIFICATION
SIMULATION
WORKCRAFT
Verification and Simulation Tool for Communication
Structured Occurrence Nets

Bowen Li
School of Computing Science
Newcastle University
Newcastle upon Tyne NE1 7RU
United Kingdom
bowen.li@ncl.ac.uk

Abstract. Occurrence nets (ONs) are acyclic Petri nets recording single system executions. Structured occurrence nets (SONs), composed of multiple ONs together with additional relationships, are intended for portraying the behaviour of complex evolving systems. Such systems generally consist of a large number of sub-systems which may proceed concurrently and interact with each other. In this paper, we describe a tool support for visualisation, verification and simulation of occurrence nets, and of communication structured occurrence nets which are a basic variant of SONs.

Keywords: occurrence nets, structured occurrence nets, concurrency, verification, simulation, Workcraft.

1. Introduction

The formalism of occurrence nets (ONs) provides means of recording execution histories of concurrent systems. ONs were initially introduced as processes of C/E-Systems [1][9]. Each process unambiguously and explicitly describes the concurrency and causality relations between executed events; more precisely, causally dependent occurrences of events are ordered while their concurrent occurrence is unordered [2]. Since occurrence nets are acyclic, repetitions of the same condition or event are recorded as new elements. Partially ordered sets are therefore suitable as the underlying mathematical structure of occurrence nets.

In structured occurrence nets (SONs) [3], multiple related occurrence nets are combined by means of various special relationships, in particular in order to express dependencies between interacting systems. The concept of a SON arose from the analysis of relationships between system failures, errors and faults [8]. We say that a system failure occurs because system behaviour is unacceptable and deviates from the service it is supposed to deliver. Essentially, a component or design fault in a system is the cause of one of more erroneous transitions within the system. Such an erroneous transition (error) can, by affecting the service provided by system, result in a system failure. In other words, a fault is the cause of an error, and error may lead to failure. Their relation follows a fundamental chain [8]:

\[ \ldots \rightarrow \text{failure} \rightarrow \text{fault} \rightarrow \text{error} \rightarrow \text{failure} \rightarrow \text{fault} \rightarrow \ldots \]
However, tracing such a chain and identifying the provenance of a failure is difficult. One of the major challenges is that of the very high complexity of real-life systems. In particular, systems may be composed of a large number of concurrently-acting sub-systems, and the communication may occur not only among sub-systems themselves, but also with the environment (i.e. other systems or human beings). SONs can intuitively provide a representation of the dependencies and behaviours of such complex systems by using a number of abstract relations. For example, the communication relation enables to directly express asynchronous or synchronous communications between different systems. With suitable tool support, one can then portray a more explicit view of system evolution, involving various types of communication, system upgrades, reconfigurations and replacements, in order to analyse the ‘fault-error-failure’ chain, and to identify the cause(s) of a failure.

In this paper, we present a tool supporting occurrence nets and communication structured occurrence net (C_SON) verification, synthesis and visualisation. A C_SON is one of several types of structured occurrence net, which uses asynchronous and synchronous communications relations to represent separate systems which proceed concurrently and communicate with each other. The tool is implemented in the Workcraft [7] software framework. Workcraft provides a flexible environment and extension points that allow its user to perform rapid prototyping of graph-based models, and to customise the existing functionality in order to implement a new formalism.

The paper is organised as follows. In sections 2 and 3 we provide basic definitions concerning occurrence nets and C_SONs, including properties that play important roles in our later implementation. Section 4 overviews the architecture of Workcraft, and then describes the detailed design and integration of a SON-based tool. Section 5 concludes the paper and discusses future work.

2. Occurrence Nets

In this section, we recall a number of formal definitions from [3] concerning occurrence nets and some of their properties. The selected properties will play a basic role in the verification and simulation functions of our SON-based tool.

Occurrence nets are directed acyclic graphs used to record dependencies between events in a single execution of a concurrent system. One can derive an occurrence net in two different ways: (i) as a process underpinning a run of a standard Petri net, e.g. a Condition/Event system (C/E-system) or Place/Transition net (P/T-net); or (ii) as a direct representation of a system’s execution history (such a system may involve not only computer components, but also components and systems involving people and natural processes, e.g. parties involved in a crime investigation).

**Definition 1.** A triple $\mathcal{ON} = (C, E, F)$ is an occurrence net if the following hold:

- (i) $C \neq \emptyset$ and $E$ are finite disjoint sets of, respectively, conditions (represented by circles) and events (represented by boxes).
- (ii) $F \subseteq (C \times E) \cup (E \times C)$ is the flow relation (represented by directed arcs).
(iii) For all $a,b \in C \cup E$:
\[ a (F') b \Rightarrow \neg (b F' a). \]
(iv) For all $c \in C$ and $e \in E$:
\[ \mid c^* \mid \leq I \land \mid e^* \mid \leq I. \]
\[ \mid c^* \mid \geq I \land \mid e^* \mid \geq I. \]
where $\bullet x = \{ y \mid y F x \}$ and $\bullet^* x = \{ y \mid x F y \}$ for every $x \in C \cup E$.

The set of all conditions with no pre-events ($\mid c^* \mid = 0$) is the initial state, and the set of all conditions with no post-events ($\mid c^* \mid = 0$) is the final state. In general, a state is any set of conditions.

Figure 1 shows an occurrence net (process) corresponding to a partial execution of a C/E system. Note the initial state of the process corresponds to the initial state of the C/E system, and any reachable state of the C/E system state (including the initial one) can be obtained as the final state of some process.

The definition above outlines several restrictions for creating an occurrence net ON:

a) An ON is a finite net in which there exist an initial and final state(s) (see (i)).
b) The flow relation $F$ connects elements of different type, i.e. condition to event or event to condition. Any other connection is invalid (see (ii)).
c) ON is acyclic and so the relation $F^-$ is a partial order (reflexive, anti-symmetric and transitive) (see (iii)).
d) Each condition has at most one pre-event and at most one post-event (see (iv)).
e) Each event at least one pre-condition and at least one post-condition (see (iii)).
The example in Figure 2 shows an invalid occurrence net because: 

- $(C_0,C_1) \in F$ and $(C_1,C_2) \in F$ contradict restriction (b); the cycle $<C_2, E_2, C_3, E_3>$ contradicts (c); there are two post-events, $E_4$ and $E_5$, of condition $C_4$ contradicting (d); and $E_3$ contradicts (e) as events must have pre-conditions and post-conditions.

We will now introduce a number of notions concerning the structure of an occurrence net.

**Definition 2.** Let $ON = (C, E, F)$ be an occurrence net, and $D \subseteq C$.

(i) $D$ is a line if 
\[ \forall a,b \in D : a \ F^+ b \lor b \ F^+ a. \]

(ii) $D$ is a cut if 
\[ \forall a,b \in D : \neg (a \ F^+ b \lor b \ F^+ a). \]

(iii) $D$ is the initial state, denoted by $Init$, if 
\[ \forall a \in D, \ a^* = \emptyset. \]

(iv) $D$ is the final state, denoted by $Fin$, if 
\[ \forall a \in D, \ a^* = \emptyset. \]

Definition 2(i,ii) captures two types of relations between conditions, i.e., concurrency (cut) and causality (line), respectively. Definition 2 (iii,iv) defines the initial and final states of an occurrence net.

The next definition introduces the sequential and step execution of an occurrence net.

**Definition 3.** Let $ON = (C, E, F)$ be an occurrence net, $D_0 \subseteq C$, $G_0 \subseteq E$, $e_i \in E$ and $D_0 = initial state$.

(i) A sequence $\gamma = D_0e_1D_1e_2 ... e_nD_n$ (n$\geq$0) is a sequential execution of $ON$ if, for every $i \leq n$:
\[ \begin{align*} e_i & \subseteq D_{i-1} \text{ and } D_i = (D_{i-1} \setminus e_i \cup e_i^*). \end{align*} \]

(ii) A sequence $\chi = D_0G_1D_1G_2 ... G_nD_n$ (n$\geq$0) is a step execution of $ON$ if, for every $i \leq n$:
\[ \begin{align*} G_i & \subseteq D_{i-1} \text{ and } D_i = (D_{i-1} \setminus G_i \cup G_i^*). \end{align*} \]
Step and sequential executions in the above definition obey the firing rule of Petri nets [5] in which the execution begins at an initial state, and moves successively to another state (which is always a cut) by executing enabled events. In a sequential execution concurrent events are interleaved, while a step execution captures simultaneous execution of events.

The occurrence net in Figure 3 generates three maximal step executions and two maximal sequential executions (i.e. those starting in the initial state and ending in the final state). The depicted one portrays step execution

\[ \chi_1 = \{C1\} \{E1\} \{C2, C3\} \{E3, E2\} \{C4, C5\} \{E4\} \{C6\} \]

in which events \( E2 \) and \( E3 \) occur simultaneously. In this case, one could imagine that a single system with dual-core processor processed two different operations, represented by \( E2 \) and \( E3 \), at the same time. The remaining two executions, in which events \( E2 \) and \( E3 \) executed sequentially, are:

\[ \chi_2 = \{C1\} E1 \{C2, C3\} E2 \{C3, C4\} E3 \{C4, C5\} E4 \{C6\} \]

\[ \chi_3 = \{C1\} E1 \{C2, C3\} E3 \{C2, C5\} E2 \{C4, C5\} E4 \{C6\} \]

2. Communication Structured Occurrence Nets

Communication structured occurrence nets (C\_SON) are a basic case of structured occurrence nets. They use two types of abstract relations to express communication between different sub-systems. A thick dashed directed arc in C\_SON indicates asynchronous communication such as sophisticated buffering or networked communication [6]. A thick dashed line is used to represent synchronous communication. Figure 4 illustrates an example of C\_SON involving two systems which first communicate asynchronously and then synchronously.

![Fig.4. A communication structured occurrence net.](image)

**Definition 4.** A tuple \( C\_SON = (ON_1, ..., ON_k, \kappa, \sigma) \) is a communication structured occurrence net if the following hold:

(i) \( ON_i = (C_i, E_i, F_i) \) for \( i = 1, ..., k \) are disjointed occurrence nets \( k \geq 1 \).

(ii) \( \kappa \) and \( \sigma \) are respectively asynchronous (represented by thick dashed arcs) and synchronous (being symmetric, and represented by thick dashed lines) communication relations satisfying:

\[ \kappa, \sigma \subseteq (E_m \times E_n) \text{ where } k \geq m, n \text{ and } m \neq n. \]

(iii) For all \( a, b \in C_i, E_i : \)

\[ a [F_i^* (\kappa \cup \sigma)^* F_i^*] b \implies \neg b [F_i^* (\kappa \cup \sigma)^* F_i^*] a. \]
In the above (and in Definition 5): \( C = C_1 \cup \ldots \cup C_k \), \( E = E_1 \cup \ldots \cup E_k \) and \( F = F_1 \cup \ldots \cup F_k \).

The initial state of a \( C_{\text{SON}} \) is the union of the initial states of the \( ON_i \)'s, and the final state of \( C_{\text{SON}} \) is the union of the final states of the \( ON_i \)'s.

There are several properties we can derived directly from definition 4:

a) All restrictions in Definition 1 concerning occurrence nets, such as finiteness, acyclicity and partial order are satisfied for each single occurrence net in a \( C_{\text{SON}} \) (see Definition 4(i)).

b) The \( \kappa, \sigma \) relations representing asynchronous and synchronous communication connect events coming from different occurrence nets (see Definition 4(ii)).

c) Communication structured occurrence nets are an acyclic model, thus cycles involving relations \( F, \kappa \) and \( \sigma \) are invalid (see Definition 4 (iii)).

The net in Figure 5 satisfies Definition 4(i, ii), but not Definition 4(iii), because there exists a cyclic path \(<E_1, C_2, E_2, F_2, B_2, F_1>\).

![Figure 5. Not a \( C_{\text{SON}} \): contains a cycle path.](image)

The execution for each single occurrence net in \( C_{\text{SON}} \) is similar to the step execution in definition 3 (ii). The only new requirement is the precedence order of the events connected by the \( \kappa \) and \( \sigma \) relations. For the events \( e, f \) connected by asynchronous relation (from \( e \) to \( f \)), either event \( e \) occurs before \( f \), or \( e \) and \( f \) are executed in the same step (i.e. simultaneously). Under the synchronous communication relationship, events \( e \) and \( f \) have to be executed simultaneously.

**Definition 5.** Let \( C_{\text{SON}} = (ON_1, \ldots, ON_k, \kappa, \sigma) \) be a communication structured occurrence net, \( D_0 \subseteq C \), \( G_i \subseteq E \), and \( D_0 \) be initial state of \( C_{\text{SON}} \).

A sequence \( \chi = D_0 G_1 D_1 G_2 \ldots G_n D_n \quad (n \geq 0) \) is called a step execution of \( C_{\text{SON}} \) if the following hold:

(i) For every \( i \leq n \): \( *G \subseteq D_{i-1} \) and \( D_i = (D_{i-1} \setminus *G_i) \cup G_i^* \).

(ii) \( (e, f) \in \kappa U \sigma \) and \( f \in G_i \) implies \( e \in G_1 \cup \ldots \cup G_i \).

Following the definition, the \( C_{\text{SON}} \) depicted in Figure 4 generates two maximal step executions:

\[
\chi_1 = \{B1, C1\} \{F1, E1\} \{B2, C2\} \{E2, F2\} \{B3, C3\}
\]

\[
\chi_2 = \{B1, C1\} \{F1\} \{B2, C1\} \{E1\} \{B2, C2\} \{F2, E2\} \{B3, C3\}
\]
In this case, the occurrence of event $E_1$ never precedes $F_1$ since they are connected by the $\kappa$ relation. Moreover, events $E_2$ and $F_2$ connected with the $\sigma$ relation are always executed in the same step.

3. A SON-based Tool

This section introduces a tool providing the functions of representation, manipulation and analysis of occurrence nets and C_SONs. The tool has been developed and implemented in the Workcraft software framework [7] that serves as a specialized working environment for the design of Interpreted Graph Models.

4.1 Workcraft

Workcraft is a software framework that supports the visualisation, verification and synthesis of graphs based formalisms such as STG and Petri nets. The toolkit is a plug-in driven architecture providing extensive graph-based model analysis and visualisation facilities.

![Fig 6. Workcraft architecture](image)

Figure 6 shows the tool architecture of Workcraft which consists of three major parts:

- **Core framework**: the framework core, which connects with services components and other service, is in charge of the Workcraft start-up and shut-down processes. When the program is starting up, the configuration manager will be started for loading configuration files, followed by the initialising of the plug-in manager and start-up scripts, which provide the information about existing plug-ins and additional start-up logic respectively. In the shut-down process, the framework core requests the configuration manager to save the current configuration variables.

- **Plug-in manager**: The responsibility of the plug-in manager is to scan and categorise all registered plug-in modules. A reflection mechanism is used to load and inspect the
module interfaces dynamically in order for them to be instantiated and initialized. The manager maintains a list of existing plug-ins that may be provided to some other parts of the framework.

*Services:* A set of services is provided for program process or user functionality such as: the configuration manager; the visualisation service which supports visual editor for the node types defined the drawing and transformation functions; and the GUI service which provides window toolkit involving user interface, workspace, visual model creation and editor.

### 4.2 The SON-based Tool

The SON-based toolkit is an open-source plug-in deployed in Workcraft for the verification, simulation and visualisation of occurrence nets and communication structured occurrence nets.

The architecture depicted in Figure 7 shows a detailed view of the integration between the Workcraft framework and the SON plug-in. The SON model interface is defined in a plug-in module in order to be discovered and registered by the plug-in manager during program initialisation. The module also defines a set of SON analysis tools that provide the functionality of editing, verification and simulation. The SON visual node types, i.e., condition, event, and communication relations, are implemented in the visualisation user service. Furthermore, the GUI user service contains a simulation panel supporting dynamic process execution.

*Visualisation:* The SON plug-in defines a series of visual node types allowing user to create and edit SON-based models in a Workcraft workspace. A property editor is implemented to support various visual node editing operations, such as setting node label, color, token, position etc. Auxiliary editing operations are inherited from the framework, which include the operations of controlling the viewport via panning and zooming, selecting and moving individual nodes, and choosing the nodes to be connected. Figure 8 shows a C_SON model created by the SON plug-in. The property
editor and editor tools (on the right side of the workspace) support model creation and edition functions, respectively.

Verification: SON verification tools provide correctness checking for SON-based models (currently for occurrence nets and C_SON). The model correctness criteria follow the restriction rules introduced in section 2 and 3. A relation checking function focuses on the correctness of connection status for each node in a model. Such correctness concerns include, for example, the initial and final state must be condition; each conditions has no more than two linked events; the asynchronous and synchronous relations in C_SON are connected by two events coming from different occurrence nets, etc. Avoiding cycles is another crucial criterion in SON verification. The implementation of the cycle checking function is based on the DFS algorithm which provides cycle detection. In addition, the toolkit supports a partial verification function that allows user to restrict the checking scope. For the C_SON in Figure 4, the program is able to verify the correctness between (E1, F1) and (C3, B3) instead of checking the complete model every time.

Simulation: Two types of occurrence net execution (sequential and step) and one C_SON execution (step) are implemented by the simulation part of the SON plug-in. The formalizations of each kind of execution are presented in Definitions 3 and 5. The toolkit supports two main functions: A basic execution path function aims to traverse and output all possible paths. The simulation result in Figure 9 corresponds to the SON step execution paths in Figure 4, in which the occurrence of event F1 either precedes E1 or is executed simultaneously with E1; the synchronous relation between E2 and F2 indicates that the execution order is always simultaneous.
In the simulation part, there is a function which supports the analysis of the maximal execution paths. The functionality aims to check whether the paths contain a specific node set. In Figure 10, an occurrence net contains three possible step execution paths and there is only one path that contains a condition set \(\{C_3, C_4\}\). The presence of such a function allows the user to filter out important execution paths generated by huge SONs.

5. Conclusion

In this report, we first presented the background for using occurrence nets and SONs. In Section 2, we provided a formal definition of occurrence nets and the conditions which are used as the correctness criteria in the later verification implementation. The sequential execution in occurrence nets represents only non-deterministic execution of concurrent events. However, step executions additionally support simultaneous execution of events. In Section 3, we formally defined communication structured occurrence nets and outlined conditions which they need to satisfy. The simulation of C_SONs requires considering asynchronous and synchronous communications relations. Section 4 outlined a prototype tool which can be used to create, edit and analyse SON-based models. The tool has been implemented in the Workcraft framework. Currently, it supports basic occurrence nets and C_SON. The development of suitable support for other kinds of structured occurrence nets [3], as well as the optimization of the existing plug-ins, are left for further work.
6. Acknowledgement

This report is based on MSc dissertation [4]. The author would like to thank Maciej Koutny and Brian Randell for their suggestions and comments.

References

4. B. Li: Verification and Simulation Tool for Communication Structured Occurrence Nets. MSc dissertation, School of Computing Science, Newcastle University (2011)