

Time in Structured Occurrence Nets

Anirban Bhattacharyya, Bowen Li, and Brian Randell

School of Computing Science, Newcastle University
Newcastle upon Tyne, United Kingdom

{anirban.bhattacharyya, bowen.li, brian.randell}@ncl.ac.uk

Abstract. This paper presents a new tool-supported formalism based on collections of related timed occurrence nets, namely, timed structured occurrence nets (timed SONS) for the modelling and analysis of causally-related events and concurrent events with uncertain or missing time information in evolving systems of systems. The application domain of timed SONS includes accident and crime investigations. A global discrete time model is used to support consistent causal reasoning about a system and time intervals are used to capture uncertainty about time values. We define the timed SONS notation and conditions for checking the consistency of time information, present algorithms of linear computational complexity for estimating missing time intervals using default duration intervals and redundant time information, and describe the facilities provided by the SONCraft tool.

Keywords: timed structured occurrence nets, time intervals, constraint propagation, algorithms, tools, SONCraft

1 Introduction

The concept of a structured occurrence net (SON) [7, 14, 12] is an extension of that of an occurrence net [2] – a directed acyclic graph that represents causality and concurrency information about a single execution of a system. SONS were created in order to characterise the behaviour of *evolving systems of systems*. Representing time information about such systems is also important. The SON concept originated from a general investigation of *failure analysis*, the problem of identifying the faults that might be the causes of an identified computer system failure. However, we believe that SONS are potentially of wide applicability, and two of the areas that we have been considering are accident and crime investigation. For example, in a criminal investigation, constructing a timeline of a crime for each suspected party is helpful in organising the evidence into a cohesive presentation for a court of law. However, in many cases, the time information available about an incident is not precise or is incomplete. For instance, it may not be possible to give an exact time (e.g. 9am) at which a robbery occurred, but it may be possible give time bounds for the robbery (e.g. 9am to 10am). Petri net-based research on uncertainty and computation of time information is limited. Therefore, the contribution of this paper is a new tool-supported formalism (timed SONS) that is based on collections of related timed occurrence

nets and is designed for modelling and reasoning about *causally-related events and concurrent events with uncertain or missing time information* in evolving systems of systems.

The rest of the paper is organised as follows: SONS are briefly described in Section 2, and the notation of timed SONS (based on discrete time intervals) is given in Section 3. Conditions for checking the consistency of time intervals are defined in Section 4, and algorithms for estimating and for increasing the precision of time intervals using default duration intervals and redundant time information are given in Section 5; the algorithms are of linear computational complexity in the number of nodes in the SON. Support for timed SONS is provided by the SONCraft tool, which is described in Section 6. Related work is briefly reviewed in Section 7.

2 Structured Occurrence Nets

In this section, we first introduce the concept of occurrence nets, and then recall from [7, 15] several notions based on the structuring of occurrence nets.

2.1 Occurrence nets

An occurrence net (ON) is a directed acyclic graph used to record dependencies between events in a single execution of a concurrent system. A standard ON consists of three basic elements: *conditions* (denoted by C), *events* (denoted by E), and a binary *flow relation* (denoted by F). Each tuple of the flow relation represents an arc of the ON from a source condition to a destination event, or from a source event to a destination condition; the source node (condition or event) is termed an *input* of the destination node (event or condition respectively), and the destination node is termed an *output* of the source node. For a given node n , the set of input nodes of n is denoted by $\bullet n$, and the set of output nodes of n is denoted by $n\bullet$. The *initial state* of an ON consists of the conditions with empty inputs, and the *final state* of an ON consists of the conditions with empty outputs. Each condition of an ON has at most one input event and at most one output event, and each event of an ON has at least one input condition and at least one output condition. A global state of an ON is a maximal set comprising pairwise concurrent conditions [7]. A *phase* is a fragment of an ON that begins with a global state and ends with a causally-related subsequent global state and includes all the causally-related events and conditions between the two global states. Figure 1 shows an ON that has been divided into two phases by three chosen global states.

Occurrence nets were originally introduced as processes of running PT-nets [3]. Each process unambiguously and explicitly describes the causality and concurrency relations between executed events; more precisely, causally dependent occurrences of events are ordered, whereas concurrent occurrences of events are unordered. It is also possible to derive an occurrence net from historic data (e.g. in log files) in order to represent directly a system's execution history [13].

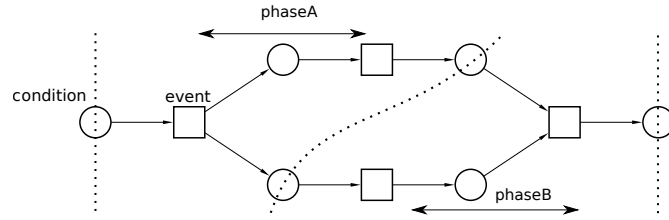


Fig. 1. An occurrence net divided into two phases by three global states.

However, the generality of causally-related events and concurrent events enables them to model not only computing systems and their histories, but also components and systems involving people and natural processes, for example, parties involved in a crime investigation – one of the several application areas we have studied.

Since occurrence nets are acyclic, repetitions of the same condition or event are recorded as new nodes. Partially ordered sets are suitable as the underlying mathematical structure of occurrence nets.

2.2 Communication SONs

A standard Petri net represents an asynchronous relation, and does not provide means to synchronise different transitions directly. Communication structured occurrence nets (CSONs) are the basic variant of structured occurrence nets that can express synchronous (as well as asynchronous) interaction between communicating systems. Thus, the CSON concept is an extension of the concept of an occurrence net. Intuitively, a CSON model combines multiple related occurrence nets into a single structure by letting them communicate via two special relationships, namely, synchronous and asynchronous communication.

The original definition of CSONs represented a communication relation by a directed or undirected dashed line between two events [7]. Subsequently, the notion of *channel place* was introduced [6], which results in a more flexible representation of synchronous communication, see Figure 2. In a synchronous communication, the two communicating events must be executed simultaneously. In an asynchronous communication, the two events can be either executed simultaneously or the sending event executes before the receiving event.

2.3 Behavioural SONs

A behavioural structured occurrence net (BSON) represents the activity of an *evolving* system by representing the evolution of the system at different levels of abstraction.

Using the phase concept, a BSON provides a two-level view of execution history: the structure at the lower level provides the details of the system’s abstract behaviour represented at the upper level. The behavioural relations (denoted by

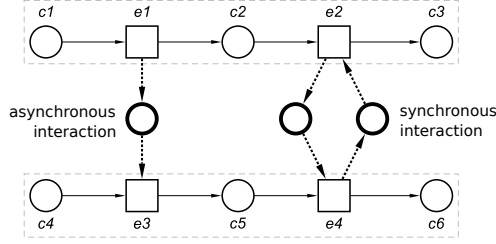


Fig. 2. A CSON with two interacting occurrence nets.

β and graphically represented by dashed lines) between the two levels express their consistent dependencies. Figure 3 shows an example of a BSON representing a system undergoing an (online) update. The upper level provides the evolution information concerning system version change caused by an update event. The lower level provides a detailed behaviour of the system. The behavioural relations between the two levels are used to capture the relationships between the two types of behaviour. In this case, the first half of the system behaviours (phaseA in Figure 1) belongs to the pre-update state a_0 , and the second half of the system behaviours (phaseB in Figure 1) belongs to the post-update state a_1 .

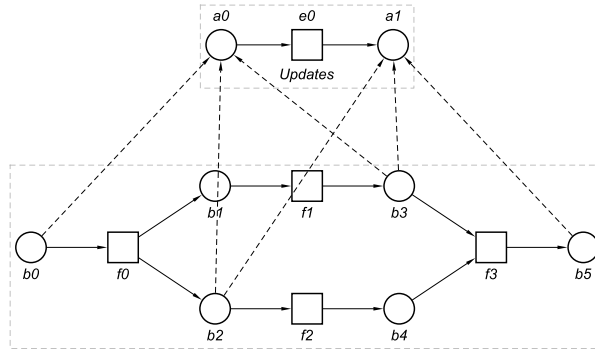


Fig. 3. A BSON portraying system (online) update.

In order to capture causal dependencies between events at different levels of a BSON, we use a special relation $causal(e)$ in BSONs that is formally defined in [8]¹.

Intuitively, $causal(e)$ is the relation representing the directed graph of events that either directly or indirectly cause e or are caused by e . For the BSON in Figure 3, we have:

$$causal(e_0) = \{(f_0, e_0), (f_1, e_0), (e_0, f_2), (e_0, f_3)\}$$

¹ We write $before(e)$ instead of $causal(e)$ in [8]; the change to $causal(e)$ is for greater clarity.

which indicates that f_0 and f_1 happen before e_0 , and e_0 happens before f_2 and f_3 .

3 Time Model

In both criminal and accident investigations, it is important to establish the order in which events have occurred (i.e. the ‘chain of events’) and to establish the duration between events in order to determine causes and their effects, and thereby eliminate infeasible hypothesized scenarios and suspects from the investigation and if possible identify the real culprits of a crime or the actual causes of an accident. The notion of a global time enables different investigators to order a given set of events consistently (i.e. in the same order), which facilitates their cooperation. Therefore, the SONS used in an investigation should have a global time model.

Uncertainty is a common and unavoidable feature of investigations, in particular, uncertainty about the time of occurrence of an event, or the duration of the event, or the time at which a state comes into existence, or how long the state lasts. Fortunately, this uncertainty is often bounded. Such uncertainty should be modelled and taken into consideration when making causal inferences during investigations.

Global time. A global clock with a fixed origin and a fixed granularity implements a global time model. A physical clock has a fixed granularity, and (therefore) cannot order two events with a non-zero separation that is less than the clock granularity (unless the events occur on different sides of a clock tick). Hence, arbitrarily close timestamps of a dense global clock cannot be verified empirically in general. On the other hand, a global clock with a fixed granularity greater than or equal to that of a physical clock supports empirical verification of event ordering. The fixed origin of the global clock supports the correct ordering of events using timestamps.

The use of a fixed granularity for the global clock implies that integers can be used to represent time values, which simplifies computation. The use of different levels of abstraction (in BSONs) requires time abstraction, that is, coarser granularities of time corresponding to higher levels of abstraction, which can be implemented using clocks with larger units of integer-valued time that correspond to higher abstraction levels. The time unit of the base level of abstraction of a SON can be chosen such that the duration of each event is zero, which facilitates the computation of missing time values in an investigation. Therefore, the duration of a node resulting from an abstraction is the maximal sum of the durations of its states at the base level of abstraction such that no two states are pairwise concurrent.

Modelling uncertainty. An integer interval is a simple way of representing a time value or a duration and the bounds on its uncertainty. Thus, the start time of a state, the finish time of the state, the start time of an event, the finish time of the event, the duration of the state, and the duration of the event (and the bounds on their respective uncertainties) can each be represented using an

integer interval. Certainty about a time value or a duration can be represented by making the two endpoints of their respective intervals identical.

For example, the occurrence of an instantaneous event at 9.00am can be represented by the interval $[0900, 0900]$. A state known to have started sometime between 9.00am and 12.30pm can be represented using the interval $[0900, 1230]$. An event known to have occurred at any time before 12.30pm can be represented using the interval $[-\infty, 1230]$, where $-\infty$ denotes an arbitrarily low integer bound. An event known to have occurred at any time after 12.30pm can be represented using the interval $[1230, \infty]$, where ∞ denotes an arbitrarily high integer bound. An unknown time can be represented by the interval $[-\infty, \infty]$.

Similarly, there are five possibilities for durations of states and of events, and an unknown duration can be represented by the interval $[0, \infty]$.

4 Time Information and its Consistency

We assume that each node of a SON (i.e. condition, event, or channel place) has a start time (T_s) and a finish time (T_f), and that each time value has bounded uncertainty represented by a specified time interval ($[T_{s,l}, T_{s,u}]$ and $[T_{f,l}, T_{f,u}]$ respectively). We also assume the node has a duration (D) with bounded uncertainty represented by a specified duration interval ($[D_l, D_u]$), see Figure 4.

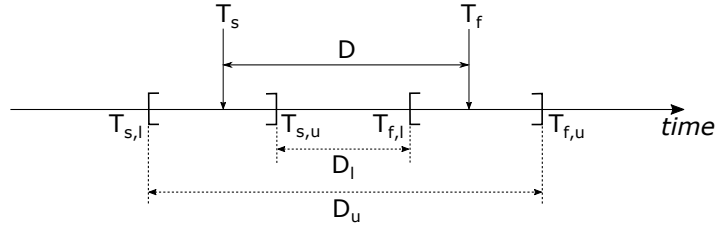


Fig. 4. Relationship between the unknown time and duration values of a node and their known bounds on the global timeline.

4.1 Notation

Let n be a node of a SON. The time information for n is defined as follows.

The start and finish times of n are denoted by T_s^n and T_f^n respectively.

The start time interval of n represents the bounded uncertainty about the value of T_s^n , and is denoted by:

$$I_s^n \triangleq [T_{s,l}^n, T_{s,u}^n]$$

where $T_{s,l}^n$ and $T_{s,u}^n$ are the lower and upper bounds respectively on the start time of n . I_s^n is well-defined if and only if the following inequality is satisfied:

$$T_{s,l}^n \leq T_{s,u}^n \quad (1)$$

The finish time interval of n represents the bounded uncertainty about the value of T_f^n , and is denoted by:

$$I_f^n \triangleq [T_{f,l}^n, T_{f,u}^n]$$

where $T_{f,l}^n$ and $T_{f,u}^n$ are the lower and upper bounds respectively on the finish time of n . I_f^n is well-defined if and only if the following inequality is satisfied:

$$T_{f,l}^n \leq T_{f,u}^n \quad (2)$$

We assume the start time of n is at, or before, the finish time of n , which is expressed by the restriction: $T_s^n \leq T_f^n$. In order to ensure consistency with this restriction, the start and finish time intervals of n must satisfy the following inequalities:

$$T_{s,l}^n \leq T_{f,l}^n \wedge T_{s,u}^n \leq T_{f,u}^n \quad (3)$$

The duration of n is denoted by D^n . The duration interval of n represents the bounded uncertainty about the value of D^n , and is denoted by:

$$I_d^n \triangleq [D_l^n, D_u^n]$$

where D_l^n and D_u^n are the lower and upper bounds respectively on the duration of n . I_d^n is well-defined if and only if the following inequality is satisfied:

$$0 \leq D_l^n \leq D_u^n \quad (4)$$

In this paper, if the node n is clear from the context, we will omit the superscript n .

4.2 Time consistency

Time consistency in linear ONs. In a linear ON, each event has exactly one input condition and one output condition, and each condition has at most one input event and at most one output event.

We assume that for any two directly connected nodes (i.e. a condition ending in an event, or an event that starts a condition), the finish time of the source node is equal to the start time of the destination node. Therefore, we have the following:

$$\forall n_1, n_2 \in (E \cup C) ((n_1, n_2) \in F \implies I_f^{n_1} = I_s^{n_2}) \quad (5)$$

Let n be a node in a linear ON. The information with respect to the start time, finish time, and duration of n is defined to be *node consistent* if and only if the following three conditions are satisfied:

$$[T_{s,l} + D_l, T_{s,u} + D_u] \cap [T_{f,l}, T_{f,u}] \neq \emptyset \quad (6)$$

$$[T_{f,l} - D_u, T_{f,u} - D_l] \cap [T_{s,l}, T_{s,u}] \neq \emptyset \quad (7)$$

$$[\max(\{0, T_{f,l} - T_{s,u}\}), T_{f,u} - T_{s,l}] \cap [D_l, D_u] \neq \emptyset \quad (8)$$

Examples of node intervals that satisfy the conditions for node consistency are shown in Figures 5 – 8, in each of which the entire rectangle represents the calculated interval and the yellow rectangle represents the intersection. The specified start time and duration intervals of node n in combination bound uncertainty about the finish time of n , and Condition (6) verifies the bounds are consistent (i.e. overlap) with the specified finish time interval of n . Similarly, the specified finish time and duration intervals of n in combination bound uncertainty about the start time of n , and Condition (7) verifies the bounds are consistent with the specified start time interval of n . Condition (8) verifies that the bounds on uncertainty about the duration of n determined from the specified start and finish time intervals of n are consistent with the specified duration interval of n . Condition (8) handles two cases, namely, the case where the uncertainty is such that the start and finish time intervals overlap and can be identical, when the condition evaluates to $[0, T_{f,u} - T_{s,l}] \cap [D_l, D_u] \neq \emptyset$, and the case where the two intervals are disjoint, when the condition evaluates to $[T_{f,l} - T_{s,u}, T_{f,u} - T_{s,l}] \cap [D_l, D_u] \neq \emptyset$.

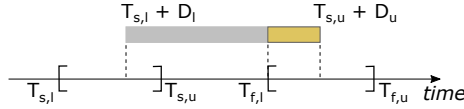


Fig. 5. Example of node intervals that satisfy condition (6).

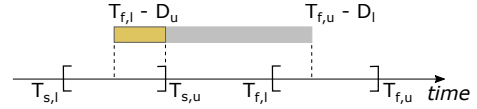


Fig. 6. Example of node intervals that satisfy condition (7).

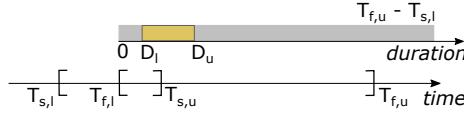


Fig. 7. Example of overlapping start and finish time intervals of a node that satisfy condition (8).

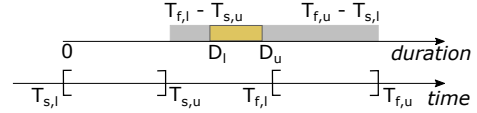


Fig. 8. Example of disjoint start and finish time intervals of a node that satisfy condition (8).

A linear ON is defined to be *time consistent* if and only if for all nodes n in the ON, n is node consistent and the flow relation F of the ON satisfies Condition (5).

For example, consider the linear ONs shown in Figure 9, produced using the SONCraft tool. The time interval shown above each arc (prefixed by ‘T:’) represents the finish time interval of its source node as well as the start time interval of its destination node, and the duration interval of a node is prefixed by ‘D:’. The absence of a time or a duration interval of a node indicates that the information is unspecified, that is, $[0000, 9999]$ (SONCraft represents $-\infty$ as 0000 and ∞ as 9999). Using Condition (6) above, we can see that the time information of event e_1 in (a) is inconsistent, because its estimated finish time

interval is $[T_{s,l}+D_l, T_{s,u}+D_u] = [0910, 1020]$, and its specified finish time interval is $[1030, 1100]$, and the two intervals do not intersect. In contrast, event e_1 in (b) is node consistent.

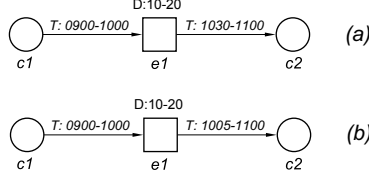


Fig. 9. Two linear ONs with time information.

Time consistency in concurrent ONs. In a concurrent ON, each event has at least one input condition and at least one output condition, and each condition has at most one input event and at most one output event.

We assume that for any two directly connected nodes, the finish time of the source node is equal to the start time of the destination node (as in linear ONs). An event starts if and only if all its input conditions are satisfied, and its output conditions are satisfied if and only if the event finishes. We assume there is no delay in the occurrence of the event. Therefore, the finish time of the input conditions must be the same as the event's start time, and the start time of the output conditions must be the same as the event's finish time. Therefore, we have the following definition.

Let e be an event in a concurrent ON. The time information of e is defined to be *concurrently consistent* if and only if the following conditions are satisfied:

$$\forall c \in \bullet e (I_f^c = I_s^e) \quad (9)$$

$$\forall c' \in e \bullet (I_s^{c'} = I_f^e) \quad (10)$$

$$e \text{ is node consistent} \quad (11)$$

Thus, for any event e with multiple inputs and outputs, verifying its concurrent consistency consists of verifying that the finish time intervals of $\bullet e$ are equal to the start time interval of e , that the start time intervals of $e \bullet$ are equal to the finish time interval of e , and that e is node consistent, that is, the start time, finish time, and duration intervals of e satisfy Conditions (6), (7), and (8).

A concurrent ON is defined to be *time consistent* if and only if for all conditions c in the ON, c is node consistent, and for all events e in the ON, e is concurrently consistent.

Time consistency in CSONs. In CSONs, communication between events is represented using channel places – such places behave identically to conditions. In asynchronous communication, the sending event e executes either before the receiving event e' , or e and e' execute simultaneously, and the two events are connected through an *asynchronous channel place* that records information about the communication using a condition. In synchronous communication, the two communicating events execute simultaneously and are connected through two

synchronous channel places that record the communication information using conditions and have the same timing characteristics as the events.

Formally, let q be a channel place and let e, e' be the input and output events of q respectively. The time information of q is defined to be *a/synchronously consistent* if and only if the following conditions are satisfied:

$$I_f^e = I_s^q \quad (12)$$

$$I_s^{e'} = I_f^q \quad (13)$$

$$q \text{ is node consistent} \quad (14)$$

Figure 10 shows how time information in a CSON can reveal the behaviour of events during asynchronous communication. In (a) the events f_0 and e_0 have the same start and finish time intervals, which indicate that the two events execute simultaneously. In (b) the time intervals indicate that f_0 executes earlier than e_0 .

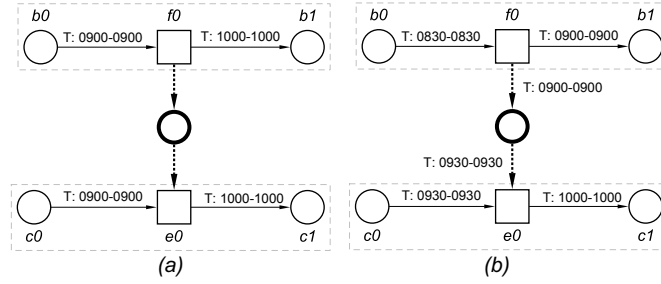


Fig. 10. Two CSONs with different runs of f_0 and e_0 .

Time consistency in BSONs. The verification of time consistency in BSONs involves verifying time consistency between occurrence nets at different levels of abstraction using the behavioural (β) and *causal* relations. For simplicity, we assume the different abstraction levels have the same time origin and granularity.

Given a BSON, let *causalU* be the binary relation consisting of the causally-related pairs of events of the BSON that is defined as follows:

$$causalU \triangleq \bigcup_{e \in \mathbf{E}} causal(e)$$

where \mathbf{E} is the set of events in the ONs of the BSON. The time information of *causalU* is defined to be time consistent if and only if the following condition is satisfied:

$$\forall (g, h) \in causalU (T_{s,l}^g \leq T_{s,l}^h \wedge T_{s,u}^g \leq T_{s,u}^h) \quad (15)$$

For all conditions $c_i, c'_i \in \mathbf{C}$ (\mathbf{C} is the set of conditions in the ONs of the BSON) such that $(c_i, c'_i) \in \beta$ and c_i belongs to the initial state of a lower level ON

of the BSON, the following condition must be satisfied:

$$I_s^{c_i} = I_s^{c'_i} \quad (16)$$

Moreover, for all conditions c_t, c'_t such that $(c_t, c'_t) \in \beta$ and c_t belongs to the final state of a lower level ON of the BSON, the following condition must be satisfied:

$$I_f^{c_t} = I_f^{c'_t} \quad (17)$$

For example, Figure 11 portrays a system undergoing an ‘offline modification’. The behaviour of the system is represented by two disjoint occurrence nets, since the situation portrayed is that of a modified system restarting its activities from some given initial state, rather than continuing from the state reached before the system modification started. The behaviour of such offline modification is reflected in the correspondence between time intervals in the two levels, as shown in the figure. The finish time interval of the pre-modified system (c_3) and the start time interval of the post-modified system (c_4) are identical to those of their corresponding upper level conditions.

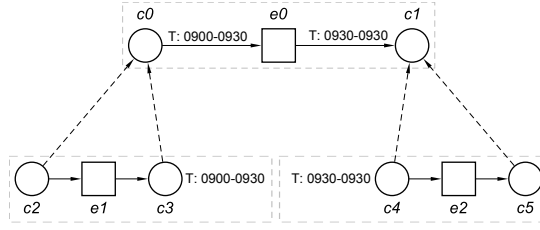


Fig. 11. Example of a BSON portraying system (off-line) update.

5 Computation of Time Intervals

Investigations of crimes and accidents typically encounter situations where information is missing, or is unavailable, or is unknown. In such cases, it is often required to estimate the information that would have filled the gaps. Furthermore, in cases where *complete* time information is available (i.e. the start time, finish time, and duration intervals of all nodes are specified) the precision of the information can perhaps be increased. In the following, the estimated value of a quantity X is denoted by \tilde{X} , and all specified information is assumed to be consistent using the conditions defined in Section 4.

For a given node, the estimations of I_s , I_f , and I_d are defined as follows:

$$\text{Let } \tilde{I}_s \triangleq [\tilde{T}_{s,l}, \tilde{T}_{s,u}] \text{ and } \tilde{I}_f \triangleq [\tilde{T}_{f,l}, \tilde{T}_{f,u}] \text{ and } \tilde{I}_d \triangleq [\tilde{D}_l, \tilde{D}_u] \quad (18)$$

In situations where complete time information is available for a node, the precision of the information can be increased using the following equations:

$$[\tilde{T}_{s,l}, \tilde{T}_{s,u}] = [T_{f,l} - D_u, T_{f,u} - D_l] \cap [T_{s,l}, T_{s,u}] \quad (19)$$

$$[\tilde{T}_{f,l}, \tilde{T}_{f,u}] = [T_{s,l} + D_l, T_{s,u} + D_u] \cap [T_{f,l}, T_{f,u}] \quad (20)$$

$$[\tilde{D}_l, \tilde{D}_u] = [\max(\{0, T_{f,l} - T_{s,u}\}), T_{f,u} - T_{s,l}] \cap [D_l, D_u] \quad (21)$$

The start time, finish time, and duration intervals of a node collectively contain redundant information. Therefore, a missing interval of the node can be estimated if the other two intervals are specified, as shown below:

$$[\tilde{T}_{s,l}, \tilde{T}_{s,u}] = [T_{f,l} - D_u, T_{f,u} - D_l] \quad (22)$$

$$[\tilde{T}_{f,l}, \tilde{T}_{f,u}] = [T_{s,l} + D_l, T_{s,u} + D_u] \quad (23)$$

$$[\tilde{D}_l, \tilde{D}_u] = [\max(\{0, T_{f,l} - T_{s,u}\}), T_{f,u} - T_{s,l}] \quad (24)$$

In situations where a time interval and the duration interval of a node are missing, we assume it will be possible to use a default duration interval as an estimate based on statistics of durations of similar events or conditions that have occurred in the past, for example, the minimum and maximum duration of a telephone call, or the minimum and maximum duration of a train journey from London to York. Hence, the missing time interval of any node can be estimated using the default duration interval, the specified time interval, and Equation (22) or (23). If a node has a type, then the default duration interval of the node can be regarded as part of the node's type information.

In situations where both time intervals of a node are missing, it is necessary to use a specified time interval of another node. We now describe algorithms for estimating missing time intervals of nodes in a SON using two approaches: the first approach is to estimate the intervals of an individual node, the second approach is to estimate the intervals of all the nodes of the SON.

5.1 Computation of time intervals of a node

The estimation of unspecified time intervals of a node involves traversing a SON structure using its causality relations, including the flow relations of its ONs, asynchronous and synchronous communications in CSONs, and *causal(e)* relations in BSONs. Algorithm 1 gives the structure of function *causalPostset*, which obtains the causal output neighbours of a given node n . Line 3 deals with flow relations: all nodes that are the outputs of n in the ON containing n are added to the result *Postset*. Lines 4 – 5 deal with CSON relations: the channel places contained in the outputs of n together with the output events of the channels (located in a different ON from that of n) are added to *Postset*. Lines 6 – 8 concern *causal(e)* relations in BSONs: for any *causal(e)* relation with domain $\{n\}$, the range is added to *Postset*.

To determine the computational complexity of Algorithm 1, let k be the total number of nodes in the SON containing the node n . The total number of node additions to *Postset* is less than k , and the total number of tests of node equality to n is less than k . Therefore, the computational complexity of *causalPostset* is $O(k)$.

Algorithm 1 (Causal postset)

```

1: function causalPostset(Node  $n$ )
2:  $Postset := \emptyset$ 
3: add  $n^\bullet$  to  $Postset$  // output of  $n$  in ON containing  $n$ 
4: for all channel place  $q$  such that  $n$  is the input of  $q$  do
5:   add  $q$  and its output event to  $Postset$  // a/sync output of  $n$ 
6: for all  $(e, f) \in causalU$  do
7:   if  $e = n$  then
8:     add  $f$  to  $Postset$ 
9: return  $Postset$ 

```

A SON is essentially a directed acyclic graph, and (therefore) traversal of a SON can be performed in two directions. Function *causalPreset* obtains the causal input neighbours of a given node n ; the structure of the function is given in [4]. The computational complexity of *causalPreset* is $O(k)$.

causalPostset and *causalPreset* are both used in estimating the unspecified time intervals of an individual node. Algorithm 2 describes the structure of procedure *estimateFinish*, which computes the finish time interval of a node n using the causal functions and is outlined below. The algorithm assumes the SON containing node n is represented by an acyclic structure.

1. Given a node n with unspecified finish time interval, perform forward breadth-first-search (BFS) using the *findRightBoundary* procedure to identify the nodes with a specified finish time interval that are nearest to n along paths beginning at n ; if no such node exists on such a path, the final node of the SON on the path is used.
2. Using the identified nodes, perform backward BFS using the procedure *backwardBFSTimes* to calculate unspecified duration and time intervals of the nodes causally-related to n (Lines 27 – 28, 32 – 33) or to recalculate the intervals to increase their precision (Lines 34 – 35), until node n is reached.
3. A count is kept (in *visits*) of the number of times a node is visited during the backward BFS traversal (Line 31) in order to handle multiple paths of unequal length between two nodes and to ensure that the time and duration intervals of a node are fully calculated exactly once (Lines 37 – 44).

Notice that the *estimateFinish* procedure can fail to compute the finish time interval of n if the computation involves an empty intersection between intervals (i.e. the SON contains inconsistent time information) or if the BFS is unable to find a node with a specified finish time interval on a path from n (i.e. the SON contains insufficient time information). The computational complexity of *estimateFinish* is $O(k)$, since the number of operations on a node is bounded by a constant and each node of the SON is fully processed at most once.

Figure 12 shows the use of *estimateFinish* to compute the finish time interval of the initial node of an ON. The ON contains two specified time intervals:

$I_f^{c_1} = I_s^{c_3} = [2001, 2005]$ and $I_f^{c_2} = [2004, 2008]$, and we assume the duration interval for each node is $[0001, 0001]$. To estimate the finish time interval of c_0 ,

Algorithm 2 (Estimates finish time interval of a node using causal relation)

```

1: procedure estimateFinish(Node  $n$ )
2:    $RBoundary := \emptyset$  // nearest right nodes of  $n$  with specified finish time intervals
3:    $RNeighbourhood := \{n\}$  // nodes on paths from  $n$  to  $RBoundary$  nodes
4:    $findRightBoundary(n, RBoundary, RNeighbourhood)$ 
5:    $backwardBFSTimes(n, RBoundary, RNeighbourhood)$ 

6: procedure findRightBoundary(Node  $n$ , Set  $Boundary$ ,  $Neighbourhood$ )
7:    $Working := \{n\}$  // nodes used for forward boundary searching
8:   while  $Working \neq \emptyset$  do
9:      $NextWorking := \emptyset$  // nodes with unspecified finish time intervals
10:    for all  $m \in Working$  do
11:      if  $causalPostset(m) = \emptyset$  then
12:        add  $m$  to  $Boundary$ 
13:      else
14:        for all  $nd \in causalPostset(m)$  do
15:          add  $nd$  to  $Neighbourhood$ 
16:          if  $nd.finish.specified$  then
17:            add  $nd$  to  $Boundary$ 
18:          else
19:            add  $nd$  to  $NextWorking$ 
20:        remove  $m$  from  $Working$ 
21:     $Working := NextWorking$ 

22: procedure backwardBFSTimes(Node  $n$ , Set  $Boundary$ ,  $Neighbourhood$ )
23:    $Working := Boundary$  // nodes used for backward estimation of time intervals
24:   while  $Working \neq \{n\}$  do
25:      $NextWorking := \emptyset$  // nodes with unspecified time intervals
26:     for all  $m \in Working$  do
27:       if  $\neg m.duration.specified$  then
28:          $m.duration := I_d^{default(typeof(m))}$ 
29:       for all  $nd \in causalPreset(m) \cap Neighbourhood$  do
30:         add  $nd$  to  $NextWorking$ 
31:          $nd.visits := nd.visits + 1$ 
32:         if  $\neg nd.finish.specified \wedge m.finish.specified$  then
33:            $nd.finish := m.finish - m.duration$  // Equation (22)
34:         else if  $m.finish.specified$  then
35:            $nd.finish := nd.finish \cap (m.finish - m.duration)$  // Equation (19)
36:       for all  $nd \in NextWorking$  do
37:         if  $nd.visits = |causalPostset(nd)|$  then
38:           for all  $ndout \in causalPostset(nd)$  do
39:             if  $\neg ndout.start.specified \wedge ndout.finish.specified$  then
40:                $ndout.start := nd.finish$ 
41:                $ndout.duration := ndout.duration \cap (ndout.finish - ndout.start)$ 
42:               // Equation (21)
43:              $nd.visits := 0$ 
44:           else
45:             remove  $nd$  from  $NextWorking$ 
46:      $Working := NextWorking$ 

```

the forward search is used to find the right boundary, that is, $\{c_2, e_1\}$. Then the backward BFS is performed to calculate intervals iteratively until reaching the initial node c_0 . Notice that during the iteration, e_0 cannot be added to the working set until both its output conditions c_1 and c_2 have been visited.

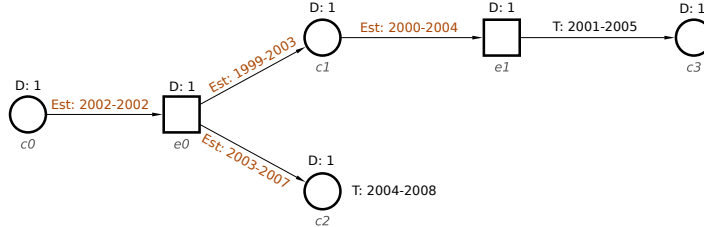


Fig. 12. Estimating the finish time interval of a node.

Estimation of the start time interval of a node n is done in the reverse way to *estimateFinish* using the procedure *estimateStart* (not shown in this paper). The structural similarity of the two procedures implies their computational complexity is the same, that is, $O(k)$.

5.2 Computation of time intervals of a SON

Procedure *estimateSONTimesBFS* estimates the unspecified time and duration intervals of an entire SON and increases the precision of the specified intervals through intersection using the principle that *each interval of a SON must be able to affect the computation of every other interval*. The structure of the procedure is given in [4]. Basically, a temporary *pre-initial* input node is created for the initial nodes of the SON and a temporary *post-final* output node is created for the final nodes of the SON to enable concurrent nodes to affect each other indirectly. This construction is often used in solving problems of directed acyclic graphs, for example, the maximum flow problem [10]. The finish time interval of the pre-initial node is calculated using the procedure *estimateFinish* described earlier, the SON is traversed going forward using BFS until the *post-final* node is reached, after which the SON is traversed going backward using BFS until the *pre-initial* node is reached. The computational complexity of *estimateSONTimesBFS* is $O(k)$.

Figure 13 shows a CSON with one specified time interval. The pre-initial and post-final nodes of the SON are respectively connected to both initial conditions and both final conditions. The algorithm first estimates the finish time interval [0958, 0958] of the pre-initial node, then uses the interval to estimate time information for the entire SON. Notice the time granularity in this example is 24-hour-clock/mins.

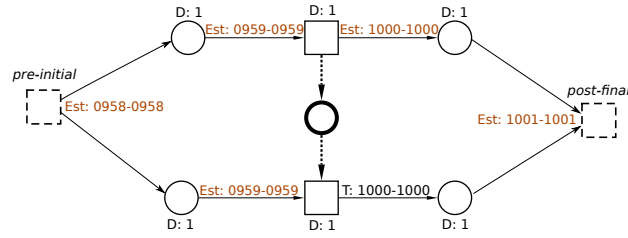


Fig. 13. A CSON with pre-initial and post-final nodes (all duration intervals are assumed to be $[0001, 0001]$).

6 Implementation

We have implemented timed SONs and their analysis algorithms in SONCraft – an open source tool for the construction and analysis of structured occurrence nets. In this section, we provide an overview of the major features of SONCraft, then present the implementation of the time-based tools.

6.1 SONCraft

SONCraft supports the visual editing, simulation, analysis, and verification of structured occurrence nets. The tool is implemented as a Java plug-in within the Workcraft platform, which provides a flexible framework for the development and analysis of interpreted graph models. Detailed descriptions of SONCraft and Workcraft and their manuals can be found in [11, 9]. The present version of SONCraft handles three types of SON structure, namely, CSONs, BSONs, and temporal abstractions [7]. A single SON can incorporate multiple instances of each type of structure. The major features of SONCraft are as follows.

Graphical user interface: an intuitive and easy to use graphical interface for editing, simulating, and analysing SON-based models, see Figure 14.

Simulator: a built-in simulator that can handle multiple ONS, CSONs, and BSONs in a SON. Simulations can be conducted either manually or automatically by firing a succession of enabled events, causing states to evolve, event colouring to be updated, and the simulation record to be augmented.

Analysis tools: a set of analysis tools is integrated into SONCraft. The *structural property tool* provides several 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. *Error tracing* is a failure analysis facility integrated with the simulator. The facility is used to analyse how a ‘faulty’ event passes through and affects the system states. The *reachability tool* is used for verifying the reachability property of a SON. The analysis establishes whether or not given states are mutually concurrent.

6.2 Time visualisation

The *time mode* in SONCraft enables the display of time information of a SON model, as shown in the screenshot in Figure 15. Notice the editor window in the

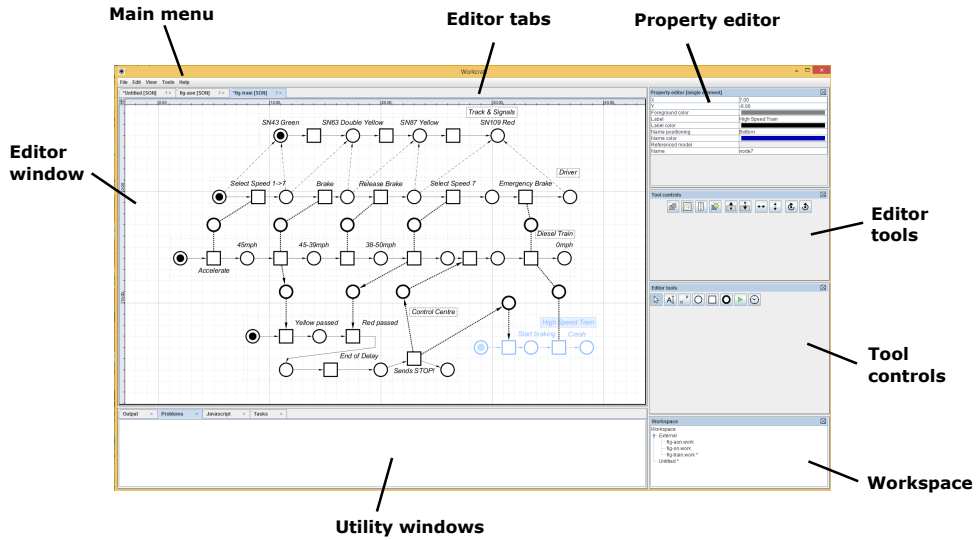


Fig. 14. SONCraft interface.

figure has been maximized for presentation purposes here, thereby minimizing all other windows (e.g. the editor tool window and the property editor window) in order to show only a single work file (i.e. the current SON model). In this mode, an initial condition is represented by a circle with a thick small arrow, and a final condition is represented by a double circle (inspired by the state representation used in finite state machines). The time representations of different node types are displayed differently because of the amount of visual space they occupy. Thus, rather than display all three intervals (i.e. start, finish, and duration) for every node, we simplify the representation by merging and displaying some intervals on arcs. More precisely, the time interval displayed on each arc indicates the finish time interval of its source node as well as the start time interval of its destination node. Thus, each non-initial and non-final node shows only its duration value (see condition ‘Ver 0.2’). However, there is no input arc for an initial condition and no output arc for a final condition; so, some of their time information is displayed directly against the node. For example, in Figure 15, the start time interval [2000, 2000] of the initial condition ‘Ver 1.0’ is displayed directly next to the node.

6.3 Time property setting tool

The *time property setting tool* shown in Figure 16 is an interface for specifying time information of a given node in a SON model. The *time granularity* panel at the top of the interface currently offers two granularities: year/year and 24-hour-clock/mins. Different granularities have their own time and duration bounds as well as arithmetic. For example, the time and duration bounds of the 24-hour-clock/mins granularity are 0001-2400 and 0000-0060 respectively.

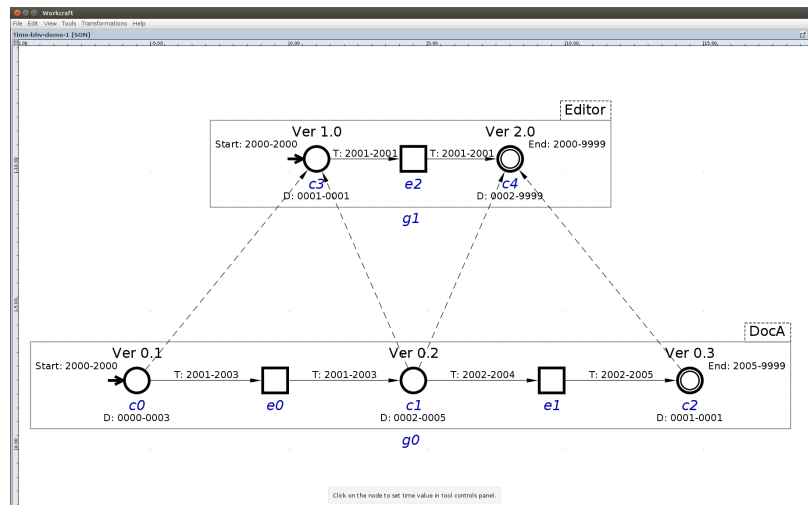


Fig. 15. Timed SON visualisation.

Users can either manually or automatically set time information for a selected node in the *time value* panel. For each manually input interval, the tool verifies whether or not the interval is well-defined. The verification criteria are based on Conditions (1) – (4). Moreover, for each input interval, checking of its time or duration bounds is performed according to its time granularity. The tool also provides efficient time estimation for nodes with unspecified time intervals. Depending on the user selection, different algorithms (e.g. Algorithm 2 or entire SON estimation) can be used for a calculation.

The 'Tool controls' dialog box is shown with the following settings:

- Time Granularity:** T:year D:year, T:24-hour D:mins
- Time value:**
 - Start time interval: 2012 - 2015
 - Finish time interval: 0000 - 9999
 - Duration interval: 0000 - 9999
- Buttons: Estimate..., Clear

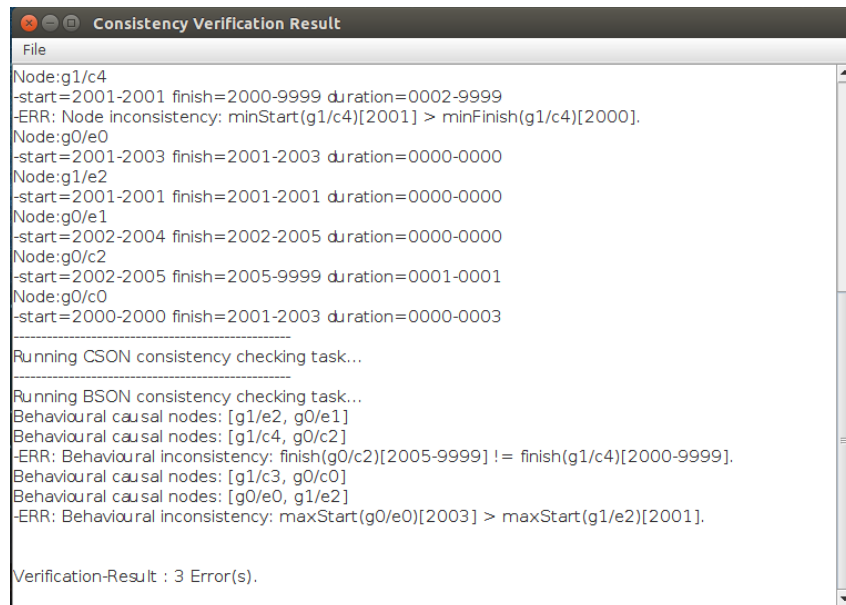
Fig. 16. Time property setter.

6.4 Time consistency checking tool

The *time consistency checking tool* provides consistency checking for the time information that is specified. The tool encodes the conditions and equations given in Sections 4 and 5, and provides a user interface for additional settings. The interface is divided into the following three (sub-)panels.

Two granularities are present in the *time granularity* panel. Causal consistency checking can be activated using the *causal consistency* panel. The facility implements the *sonConsistency* function in [4] and aims to verify the nodes with incomplete time information using causal relations. The panel also includes means of specifying the default duration setting that is used in time estimation. The *user settings* panel has an option to request the highlight of time inconsistent nodes (if any) in the editor window after the verification.

Figure 17 shows a consistency verification result of the SON displayed in Figure 15. The result shows three time inconsistency errors. For example, the first error message involves condition c_4 and shows that its start time lower bound (2001) is greater than its finish time lower bound (2000), that is, the condition can cease to hold before it starts to hold.



```

File
Node:g1/c4
-start=2001-2001 finish=2000-9999 duration=0002-9999
-ERR: Node inconsistency: minStart(g1/c4)[2001] > minFinish(g1/c4)[2000].
Node:g0/e0
-start=2001-2003 finish=2001-2003 duration=0000-0000
Node:g1/e2
-start=2001-2001 finish=2001-2001 duration=0000-0000
Node:g0/e1
-start=2002-2004 finish=2002-2005 duration=0000-0000
Node:g0/c2
-start=2002-2005 finish=2005-9999 duration=0001-0001
Node:g0/c0
-start=2000-2000 finish=2001-2003 duration=0000-0003
-----
Running CSON consistency checking task...
-----
Running BSON consistency checking task...
Behavioural causal nodes: [g1/e2, g0/e1]
Behavioural causal nodes: [g1/c4, g0/c2]
-ERR: Behavioural inconsistency: finish(g0/c2)[2005-9999] != finish(g1/c4)[2000-9999].
Behavioural causal nodes: [g1/c3, g0/c0]
Behavioural causal nodes: [g0/e0, g1/e2]
-ERR: Behavioural inconsistency: maxStart(g0/e0)[2003] > maxStart(g1/e2)[2001].

Verification-Result : 3 Error(s).

```

Fig. 17. Time consistency checking result.

7 Related Work

Petri net-based research on uncertainty, consistency checking, and computation of time information is limited. However, there is research on genealogies (e.g. [5] and [16]) and on temporal logics (e.g [1]) that addresses these issues. In [5], a mathematical relaxation method (of exponential complexity in the number of dates) is used to adjust iteratively the endpoints of intervals containing unknown dates of birth, marriage, and death until the endpoints finally stabilise. The restrictions on the dates are encoded in the algorithm, and their parameters are set manually. In [16], the restrictions on the dates determine intervals, which are intersected in order to determine the final endpoints. The parameters of the restrictions are determined by statistical analysis of the input data. In [1], an algebra of relations between temporal intervals is developed with a method (of quadratic complexity in the number of intervals) for determining the relation (one of thirteen) between any two intervals. However, none of the research models abstraction of events or of states or models communication.

8 Concluding Remarks

This paper has presented the notion of a timed SON in order to model and reason about causally-related events and concurrent events with uncertain or missing time information in evolving systems of systems. Discrete intervals have been used to capture uncertainty about time values. Conditions have been defined to verify the consistency of time information. Algorithms have been presented that are based on the use of default duration intervals and redundant time information and are of linear computational complexity in the number of nodes in a SON in order to estimate missing time intervals and to increase the precision of user-specified intervals. Finally, the facilities provided by the SONCraft tool have been described.

Future work will extend timed SONS to handle multiple scenarios so that different hypotheses about the cause of an accident or the chain of events of a crime can be modelled and investigated. Probabilities will be added to support multiple scenarios, and different time granularities will be represented to support modelling at different levels of abstraction. Furthermore, we hope to use timed SONS to model and analyse cyber crime, problems involving ‘big data’, neurological processes, dynamic reconfiguration of real-time software, and hardware design, in order to demonstrate the generality and exploit the full potential of the formalism. SONCraft is being extended with a multipage facility that will enable very long ONS to be represented and abstracted, and allow different ONS of a SON to be stored in different files.

Acknowledgements

This work is funded by the EPSRC project UNderstanding COMplex system eVolution through structurEd behaviouRs (UNCOVER). The authors acknowl-

edge the help and advice given by Victor Khomenko and Maciej Koutny, and thank the anonymous reviewers for their comments.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11), 832–843 (1983)
2. Best, E., Devillers, R.: Sequential and concurrent behaviour in petri net theory. *Theoretical Computer Science* 55(1), 87–136 (1987)
3. Best, E., Fernández, C.: *Nonsequential Processes: A Petri Net View*, vol. 13 of EATCS monographs on Theoretical Computer Science. Springer-Verlag (1988)
4. Bhattacharyya, A., Li, B., Randell, B.: Time in structured occurrence nets. Tech. Rep. CS-TR-1495, School of Computing Science, Newcastle University (May 2016)
5. Brox, V.: Date Estimation in Lineage-Linked Databases. BSc dissertation, Newcastle University School of Computing Science (2000), <http://homepages.cs.ncl.ac.uk/brian.randell/Genealogy/Brox/dissertation/dissertation.html>
6. Kleijn, J., Koutny, M.: Causality in structured occurrence nets. In: *Dependable and Historic Computing*. vol. 6875, pp. 283–297. Springer Berlin Heidelberg (2011)
7. Koutny, M., Randell, B.: Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques. *Fundamenta Informaticae* 97(1), 41–91 (Jan 2009)
8. Li, B., Koutny, M., Randell, B.: SONCraft: A tool for construction, simulation and verification of structured occurrence nets. Tech. Rep. CS-TR-1493, School of Computing Science, Newcastle University (Jan 2016)
9. Li, B., Randell, B.: SONCraft user manual. Tech. Rep. CS-TR-1448, School of Computing Science, Newcastle University (Feb 2015)
10. Nazer, B., Gastpar, M.: Computation over multiple-access channels. *Information Theory, IEEE Transactions on* 53(10), 3498–3516 (2007)
11. Poliakov, I.: Workcraft homepage, <http://workcraft.org>
12. Randell, B.: Occurrence nets then and now: the path to structured occurrence nets. In: *Applications and Theory of Petri Nets*. pp. 1–16. Springer Berlin Heidelberg (Jun 2011)
13. Randell, B.: Incremental construction of structured occurrence nets. Tech. Rep. CS-TR-1384, School of Computing Science, Newcastle University (May 2013)
14. Randell, B., Koutny, M.: Failure: their definition, modelling and analysis. In: *Theoretical Aspects of Computing—ICTAC 2007*. pp. 260–274. Springer (Sep 2007)
15. Randell, B., Koutny, M.: Structured occurrence nets: Incomplete, contradictory and uncertain failure evidence. Tech. Rep. CS-TR-1170, School of Computing Science, Newcastle University (Sep 2009)
16. Wilson, R.: Date range propagation in genealogical databases. In: *Family History Technology Workshop* (2012)