COMPUTING SCIENCE

Structured Occurrence Nets: A formalism for aiding system failure prevention and analysis techniques

M. Koutny, B. Randell

TECHNICAL REPORT SERIES

No. CS-TR-1120 September, 2008

TECHNICAL REPORT SERIES

No. CS-TR-1120 September, 2008

Structured Occurrence Nets: A formalism for aiding system failure prevention and analysis techniques

Maciej Koutny and Brian Randell

Abstract

This paper introduces the concept of a `structured occurrence net', which as its name indicates is based on that of an `occurrence net', a well-established formalism for an abstract record that represents causality and concurrency information concerning a single execution of a system. Structured occurrence nets consist of multiple occurrence nets, associated together by means of various types of relationship, and are intended for recording either the actual behaviour of complex systems as they communicate and evolve, or evidence that is being gathered and analysed concerning their alleged past behaviour. We provide a formal basis for the new formalism and show how it can be used to gain better understanding of complex fault-error-failure chains (i) among co-existing communicating systems, (ii) between systems and their sub-systems, and (iii) involving systems that are controlling, creating or modifying other systems. We then go on to discuss how, perhaps using extended versions of existing tools, structured occurrence nets could form a basis for improved techniques of system failure prevention and analysis.

Printed and published by the University of Newcastle upon Tyne,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

KOUTNY, M., RANDELL, B.

Structured Occurrence Nets: A formalism for aiding system failure prevention and analysis techniques
[By] M. Koutny, B. Randell

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2008.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-1120)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-1120

Abstract

This paper introduces the concept of a `structured occurrence net', which as its name indicates is based on that of an `occurrence net', a well-established formalism for an abstract record that represents causality and concurrency information concerning a single execution of a system. Structured occurrence nets consist of multiple occurrence nets, associated together by means of various types of relationship, and are intended for recording either the actual behaviour of complex systems as they communicate and evolve, or evidence that is being gathered and analysed concerning their alleged past behaviour. We provide a formal basis for the new formalism and show how it can be used to gain better understanding of complex fault-error-failure chains (i) among co-existing communicating systems, (ii) between systems and their sub-systems, and (iii) involving systems that are controlling, creating or modifying other systems. We then go on to discuss how, perhaps using extended versions of existing tools, structured occurrence nets could form a basis for improved techniques of system failure prevention and analysis.

About the author

Maciej Koutny obtained his MSc (1982) and PhD (1984) from the Warsaw University of Technology. In 1985 he joined the then Computing Laboratory of the University of Newcastle upon Tyne to work as a Research Associate. In 1986 he became a Lecturer in Computing Science at Newcastle, and in 1994 was promoted to an established Readership at Newcastle. In 2000 he became a Professor of Computing Science.

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

Suggested keywords

FAILURES,
ERRORS,
FAULTS,
DEPENDABILITY,
JUDGEMENT,
OCCURRENCE NETS,
ABSTRACTION,
FORMAL ANALYSIS

Structured Occurrence Nets: A formalism for aiding system failure prevention and analysis techniques

Maciej Koutny and Brian Randell

School of Computing Science
Newcastle University
Newcastle upon Tyne, NE1 7RU
United Kingdom
{maciej.koutny,brian.randell}@ncl.ac.uk

Abstract. This paper introduces the concept of a 'structured occurrence net', which as its name indicates is based on that of an 'occurrence net', a well-established formalism for an abstract record that represents causality and concurrency information concerning a single execution of a system. Structured occurrence nets consist of multiple occurrence nets, associated together by means of various types of relationship, and are intended for recording either the actual behaviour of complex systems as they communicate and evolve, or evidence that is being gathered and analysed concerning their alleged past behaviour. We provide a formal basis for the new formalism and show how it can be used to gain better understanding of complex fault-error-failure chains (i) among co-existing communicating systems, (ii) between systems and their sub-systems, and (iii) involving systems that are controlling, creating or modifying other systems. We then go on to discuss how, perhaps using extended versions of existing tools, structured occurrence nets could form a basis for improved techniques of system failure prevention and analysis.
Keywords: failures, errors, faults, dependability, judgement, occurrence nets, abstraction, formal analysis.

1 Introduction

The concept of a failure of a system is central both to system dependability and to system security, two closely associated and indeed somewhat overlapping research domains. Specifically, particular types of failures (e.g., producing wrong results, ceasing to operate, revealing secret information, causing loss of life, etc.) relate to, indeed enable the definition of, what can be regarded as different attributes of dependability/security: respectively reliability, availability, confidentiality, safety, etc. The paper by Avizienis et al. [1] provides an extended (informal) discussion of the basic concepts and terminology of dependability and security, and contains a detailed taxonomy of dependability and security terms. Our aims in this present paper are: (i) to improve our understanding — in part by formalising — of the concept of failure (and error and fault) as given by [1]; (ii) to reduce (in fact by uniting the apparently different concepts of 'system' and 'state') the number of base concepts, i.e., concepts that the paper uses without explicit definition; and (iii) to provide a basis for an investigation of possible improved

techniques of system failure prevention and analysis. The paper is a greatly extended version of [21], providing proofs for the various results that were merely indicated in our earlier paper, together with several new concepts, definitions and supporting results.

Complex real systems, made *up* of other systems, and made *by* other systems (e.g., of hardware, software and people) evidently fail from time to time, and reducing the frequency and severity of their failures is a major challenge — common to both the dependability and the security communities. Indeed, a dependable/secure system can be regarded as *one whose (dependability/security) failures are not unacceptably frequent or severe* (from some given viewpoint).

We will return shortly to the issue of viewpoint. But first let us quote the definitions of three basic and subtly-distinct concepts, termed 'failure', 'fault' and 'error' in [1]:

> 'A system *failure* occurs when the delivered service deviates from fulfilling the system function, the latter being what the system is *aimed at*. An *error* is that part of the system state which is *liable to lead to subsequent failure*: an error affecting the service is an indication that a failure occurs or has occurred. The *adjudged or hypothesised cause* of an error is a *fault*.'

Note that errors do not necessarily lead to failures — such occurrences may be avoided by chance or design. Similarly, failures in a component system do not necessarily constitute faults to the surrounding system — this depends on how the surrounding system is relying on the component. These three concepts (respectively an event, a state, and a cause) are evidently distinct, and so need to be distinguished, whatever names are chosen to denote them. The above quotation makes it clear that judgement can be involved in identifying error causes, i.e., faults. However it is also the case that identifying failures and errors involves judgement (not necessarily simple adherence to some pre-existing specification) — a critical point that we will return to shortly.

A failure can be judged to have occurred when an error 'passes through' the system-user interface and affects the service delivered by the system — a system being composed of components which are themselves systems. This failure may be significant, and thus constitute a fault, to the enclosing system.

Thus the manifestation of failures, faults and errors follows a 'fundamental chain':

$$\ldots \rightarrow \text{failure} \rightarrow \text{fault} \rightarrow \text{error} \rightarrow \text{failure} \rightarrow \text{fault} \rightarrow \ldots,$$

i.e.,

$$\ldots \rightarrow \text{event} \rightarrow \text{cause} \rightarrow \text{state} \rightarrow \text{event} \rightarrow \text{cause} \rightarrow \ldots.$$

It is critical to note that this chain can flow from one system to: (i) another system that it is interacting with; (ii) a system which it is part of; and (iii) a system which it creates or sustains.

Typically, a failure will be judged to be due to multiple co-incident faults, e.g., the activity of a hacker exploiting a bug left by a programmer. Identifying failures (and hence errors and faults), even understanding the concepts, is difficult. There can be uncertainties about system boundaries, the very complexity of the systems (and of any specifications) is often a major difficulty, the determination of possible causes or consequences of failure can be a very subtle and iterative process, and any provisions for

preventing faults from causing failures may themselves be fallible. Attempting to enumerate a system's possible failures beforehand is normally impracticable. Instead, one can appeal to the notion of a 'judgemental system'.

The 'environment' of a system is the wider system that it affects (by its correct functioning, and by its failures), and is affected by. What constitutes correct (failure-free) functioning *might* be implied by a system specification — assuming that this exists, and is complete, accurate and agreed. (Often the specification is part of the problem!) However, in principle a third system, a *judgemental system*, is involved in determining whether any particular activity (or inactivity) of a system in a given environment constitutes or would constitute — *from its viewpoint* — a *failure*. Note that the judgemental system and the environmental system might be one and the same, and the judgement might be instant or delayed. The judgemental system might itself fail — as judged by some further system — and different judges, or the same judge at different times, might come to different judgements.

The term 'Judgemental System' is deliberately broad — it covers from on-line failure detector circuits, via someone equipped with a system specification, to the retrospective activities of a court of enquiry (just as the term 'system' is meant to range from simple hardware devices to complex computer-based systems, composed of hardware, software and people). Thus the judging activity may be clear-cut and automatic, or essentially subjective — though even in the latter case a degree of predictability is essential, otherwise the system designers' task would be impossible. The judgement is an action by a system, and so can in principle fail either positively or negatively. This possibility is allowed for in the legal system, hence the concept of a hierarchy of crown courts, appeal courts, supreme courts, etc.

In this paper we describe a means of modelling the activity of systems — operational computing systems, the systems of people and computers that created them or are adapting them, the systems that are passing judgements on them, etc. The formalism that we use in this paper is based on that of *occurrence nets* [3, 8, 22]. We introduce this formalism not just in order to clarify such concepts as fault-error-failure chains, and the role of judgemental systems, but also because the occurrence net formalism is well-supported by tools for system validation and synthesis [6, 10–12, 19], tools which we believe could be significantly enhanced by being extended so as to take advantage of the concept that we introduce in Sections 3-6 of this paper of 'structured occurrence nets'. (Section 7 sketches the ways in which we envisage exploiting such enhanced tools.)

As can be seen in Figure 1, occurrence nets are directed acyclic graphs that portray the (alleged) past and present state of affairs, in terms of places (i.e., conditions, represented by circles), transitions (i.e., events, represented by squares) and arrows (each from a place to a transition, or from a transition to a place, representing (alleged) causality). For simple nets, an actual graphical representation suffices — and will be used here using the notation shown in Figure 1. (In the case of complex nets, these are better represented in some linguistic or tabular form.) We will also take advantage of our belated realisation that the concepts of 'system' and 'state' are not separate, but just a question of abstraction, so that (different related) occurrence nets can represent both systems and their states using the same symbol — a 'place'. In fact in this paper we introduce and define, and discuss the utility of, several types of relationship, and term a set of related

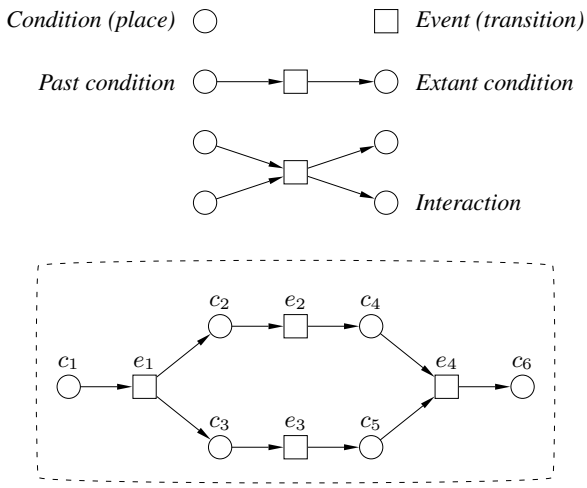

Fig. 1. Basic notation (top) and an occurrence net (bottom).

occurrence nets a *structured occurrence net*. These types of relationship differ depending on the specific means and objectives of a particular investigation. However, there are some fundamental constraints that any structured occurrence net ought to satisfy. Crucially, we will require that the structures we admit *avoid cycles* in systems' temporal behaviour as these contradict the accepted view on the way physical systems could possibly behave.

Note that it is easy to understand how occurrence nets could be 'generated' by executing Petri nets representing computing systems, but they could in fact be used to record the execution of any (potentially asynchronous) process, hardware or software, indeed human, no matter what notation or language might be used to define it. It is also worth noting that various other graphical notations similar to occurrence nets can be found in both the hardware and the software design worlds, e.g., strand spaces [23], signal diagrams [16] and message sequence charts [17].

2 Occurrence nets

In this section, we present the basic model of an occurrence net which is standard within Petri net theory [3, 8, 22]. Later on, we will extend it to express more intricate features of our approach to the modelling of complex behaviours. In a nutshell, an occurrence net is an abstract record of a single execution of some computing system (though they can be used to portray behaviours of quite general systems, e.g., ones that include people) in which only information about causality and concurrency between events and visited local states is represented. Together with a natural requirement that causal cycles do not occur in the physical world, this means that the underlying mathematical structure of an occurrence net is that of a partial order. This should be contrasted with

an 'interleaving' record of a computation which presupposes a sequential ordering of all the events involved, and has as an underlying structure a total order.

Definition 1 (occurrence net ON). *An* occurrence net *is a triple* ON $\stackrel{\text{df}}{=} (C, E, F)$ *where:* $C \neq \varnothing$ *and* E *are finite*[1] *disjoint sets of respectively* conditions *and* events *(collectively, conditions and events are the* nodes *of* ON*); and* $F \subseteq (C \times E) \cup (E \times C)$ *is a* flow relation *satisfying the following: (i) for every* $c \in C$ *there is at most one* e *such that* $(e, c) \in F$*, and at most one* f *such that* $(c, f) \in F$*; (ii) for every* $e \in E$ *there is* c *such that* $(c, e) \in F$*, and* d *such that* $(e, d) \in F$*; and (iii)* ON *forms an acyclic graph (in other words, the transitive closure of the relation* F*, denoted by* F^+*, is irreflexive).*

In the above definition — aimed at capturing the essence of a computation history — E represents the events which have actually been executed and C represents conditions (or holding of local states) enabling their executions. Here we will discuss computation histories as though they have actually occurred; however, the term will also be used of 'histories' that *might* have occurred, or that might occur in the future, given the existence of an appropriate system.

The flow relation records the causality relationship between events and conditions. Not all such relationships are meaningful, and so the first condition means that each non-initial condition is uniquely caused, and each of the non-final conditions caused a unique event.[2] The second condition states that each event has at least one cause and at least one effect, and the third one simply renders formal a common belief that causality is not circular. Now we introduce few useful notations:

- For each node x we use $pre(x)$ and $post(x)$ to denote the set of all nodes y such that $(y, x) \in F$ and $(x, y) \in F$, respectively. In other words, $pre()$ and $post()$ correspond to the incoming and outgoing arcs, respectively. For a set of nodes X, we denote

$$pre(X) \stackrel{\text{df}}{=} \bigcup_{x \in X} pre(x) \text{ and } post(X) \stackrel{\text{df}}{=} \bigcup_{x \in X} post(x) .$$

- Two nodes of ON, x and y, are *causally related* if $(x, y) \in F^+$ or $(y, x) \in F^+$; otherwise they are *concurrent*.
- During the execution captured by the occurrence net, the system has passed through a series of (global) states, and the concurrency relation in ON provides full information about all such potential states. A *cut* is a maximal (w.r.t. set inclusion) set of conditions $Cut \subseteq C$ which are mutually concurrent.
- Let $Init_{\text{ON}}$ and Fin_{ON} be the sets of all conditions c such that $pre(c) = \varnothing$ and $post(c) = \varnothing$, respectively. These two sets are cuts; the former corresponds to the initial state of the history represented by ON, and the latter to its final state.

[1] For simplicity, we only discuss finite behaviours and so all (structured) occurrence nets considered in this paper will be finite.

[2] Note that if an event is only *conditional* on the presence of a condition, but does not invalidate it, then the event can re-establish this condition by producing a fresh copy of the condition.

For the occurrence net depicted in Figure 1, we have $C = \{c_1, \ldots, c_6\}$ and $E = \{e_1, \ldots, e_4\}$. Moreover, $Init_{\mathrm{ON}} = \{c_1\}$ and $Fin_{\mathrm{ON}} = \{c_6\}$, and the other four cuts of this occurrence net are $\{c_2, c_3\}$, $\{c_2, c_5\}$, $\{c_4, c_3\}$ and $\{c_4, c_5\}$.

An occurrence net is usually derived from a single execution history of the system. However, since it only records essential (causal) orderings, it also conveys information about other potential executions. This calls for a precise notion of an execution of a given occurrence net.

Definition 2 (sequential execution). *A* sequential execution *of the occurrence net* ON *is $D_0\, e_1\, D_1 \ldots e_n\, D_n$, where each D_i is a set of conditions and each e_i is an event, such that $D_0 = Init_{\mathrm{ON}}$ and, for every $i \leq n$, $pre(e_i) \subseteq D_{i-1}$ and $D_i = (D_{i-1} \setminus pre(e_i)) \cup post(e_i)$. We will call $e_1 \ldots e_n$ a* firing sequence *of* ON.

For the occurrence net depicted in Figure 1, one of its possible sequential executions is $\{c_1\}\, e_1\, \{c_2, c_3\}\, e_2\, \{c_4, c_3\}\, e_3\, \{c_4, c_5\}\, e_4\, \{c_6\}$. Thus an execution starts in the initial global state, and each successive event transforms a current global state into another one according to the set of conditions in its vicinity. Basically, all conditions (local states) which made possible its execution cease to hold, and new conditions (local states) created by the event begin to hold. The above definition implies a couple of simple, yet important facts formulated next. Basically, they imply that ON is sound in the sense of obeying some natural temporal properties as well as testifying to the fact that ON does not contain redundant parts. We also have a complete characterisation of global states reachable from the default initial one — these are all the cuts of ON. In practical terms, the latter means that we can verify state properties of the computations captured by ON by running a model checker which inspects all the cuts. Such a model checker could be based on a SAT-solver or integer programming, e.g., as in [5, 10, 12].

Proposition 1 ([3]). *Given a sequential execution as in Definition 2, each D_i is a cut of* ON*, and no event occurs more than once. Moreover, $D_n = Fin_{\mathrm{ON}}$* iff *each event of E occurs in the execution.*

Proposition 2 ([3]). *Each cut of* ON *can be reached from the initial cut through some sequential execution, and each event of* ON *occurs in at least one sequential execution of* ON.

An alternative, more concurrent, notion of execution considers that in a single computational move, a set of events (called a *step*) rather than a single event is executed.

Definition 3 (step execution). *A* step execution *of an occurrence net* ON *is a sequence $\chi \stackrel{\mathrm{df}}{=} D_0\, G_1\, D_1 \ldots G_n\, D_n$, where each D_i is a set of conditions and each G_i is a set of events, such that $D_0 = Init_{\mathrm{ON}}$ and, for every $i \leq n$, we have $pre(G_i) \subseteq D_{i-1}$ and $D_i = (D_{i-1} \setminus pre(G_i)) \cup post(G_i)$. We also say that χ* leads *to D_n, and that D_n is* reachable.

For the net in Figure 1, $\{c_1\}\, \{e_1\}\, \{c_2, c_3\}\, \{e_2, e_3\}\, \{c_4, c_5\}\, \{e_4\}\, \{c_6\}$ is a possible step execution. For the basic model of occurrence nets, the sequential and step executions are broadly speaking equivalent; in particular, Propositions 1 and 2 hold also for step executions. However, for extended notions of occurrence nets, which we discussed in [13], sequential and step executions may, e.g., admit different sets of reachable global states.

The next result can be understood as a statement of a fundamental consistency between the causality captured by the flow relation and the temporal ordering of conditions and events involved in a step execution.

Proposition 3 ([3]). *Given a step execution as in Definition 3 and $i \leq n$,*

1. *each event f such that $(f, e) \in F^+$, for some $e \in G_i$, belongs to G_j with $j < i$;*
2. *if $c \in D_i \cap post(e)$ then e belongs to G_j with $j < i$;*
3. *if $e \in G_i$ and $c \in pre(e)$ then c does not belong to any D_j with $j \geq i$.*

A cut Cut of an occurrence net ON divides it into two subnets, $preon_{\mathrm{ON}}(Cut) \stackrel{\text{df}}{=} (C', E', F')$ and $poston_{\mathrm{ON}}(Cut) \stackrel{\text{df}}{=} (C'', E'', F'')$, given by:

$$\begin{array}{ll} C' \stackrel{\text{df}}{=} \{d \in C \mid \exists c \in Cut : (d, c) \in F^*\} & C'' \stackrel{\text{df}}{=} \{d \in C \mid \exists c \in Cut : (c, d) \in F^*\} \\ E' \stackrel{\text{df}}{=} \{e \in E \mid \exists c \in Cut : (e, c) \in F^*\} & E'' \stackrel{\text{df}}{=} \{e \in E \mid \exists c \in Cut : (c, e) \in F^*\} \\ F' \stackrel{\text{df}}{=} F|_{(C' \times E') \cup (E' \times C')} & F'' \stackrel{\text{df}}{=} F|_{(C'' \times E'') \cup (E'' \times C'')} \ . \end{array}$$

Intuitively, $preon_{\mathrm{ON}}(Cut)$ is the part of ON which has been executed to reach the cut Cut, and $poston_{\mathrm{ON}}(Cut)$ that which can still be executed after Cut.

Proposition 4 ([3]). *Let ON' and ON'' be respectively the subnets $preon_{\mathrm{ON}}(Cut)$ and $poston_{\mathrm{ON}}(Cut)$ defined above.*

1. *ON' and ON'' are occurrence nets such that: $C = C' \cup C''$, $C' \cap C'' = Cut$, $E = E' \uplus E''$ and $F = F' \uplus F''$.*
2. *$Init_{\mathrm{ON}'} = Init_{\mathrm{ON}}$, $Fin_{\mathrm{ON}'} = Cut = Init_{\mathrm{ON}''}$ and $Fin_{\mathrm{ON}''} = Fin_{\mathrm{ON}}$.*
3. *Given step executions, χ' and χ'', of respectively, ON' and ON',*
   - *χ' is a step execution of ON;*
   - *if χ' leads to Cut then $\chi' \varnothing \chi''$ is a step execution of ON.*

We end this section with (non-standard) definitions of two kinds of structures present in occurrence nets which will prove useful in the rest of this paper:

- A non-empty set of conditions D is a *phase* if there are two cuts, Cut and Cut', such that Cut' is a cut of $\mathrm{ON}' \stackrel{\text{df}}{=} poston_{\mathrm{ON}}(Cut)$ and D is the set of conditions of the occurrence net $preon_{\mathrm{ON}'}(Cut')$. We will denote $Min_D \stackrel{\text{df}}{=} Cut$ and $Max_D \stackrel{\text{df}}{=} Cut'$. Moreover, ON_D will denote the sub-occurrence net of ON induced by the conditions in D and all the events e such that $pre(e) \cup post(e) \subseteq D$. Phases will represent stages in the evolution of systems. Purely for notational convenience, we also admit the empty phase, $D = \varnothing$, for which $Min_D = Max_D = \varnothing$.
- A *block* is a non-empty B of nodes where both the maximal and minimal (w.r.t. F) elements are events, and for all nodes $x, y \in B$, $(x, z) \in F^+$ and $(z, y) \in F^+$ imply $z \in B$. Thus in a block there are no 'gaps' between the nodes it comprises.

In Figure 1, $\{c_2, c_4, c_3, c_5\}$ constitute a phase and $\{e_2, e_3, e_4, c_4, c_5\}$ a block.

In this section we introduced basic notions concerning occurrence nets and recalled some fundamental results about their behaviour which we will subsequently investigate in the extended model described in subsequent sections of this paper. These detail a number of different ways in which multiple occurrence nets (ONs) can be related together in order to construct *structured occurrence nets* (SONs).

3 Communication

We now outline the first of several ways of structuring occurrence nets, something that can be done either by defining one or more relations between a set of hitherto separate occurrence nets, or by adding structure to an existing occurrence net, i.e., by replacing it by an equivalent set of related smaller occurrence nets. In subsequent figures we follow a convention that conditions and events of different systems are identified by shading them differently - there are some obvious rules about legal such labellings (e.g., that they partition the nodes into disjoint sets, the members of each of which are connected). A further convention is that, in order to distinguish them from ordinary occurrence nets, structured occurrence nets which contain two or more component occurrence nets are shown surrounded by a solid line bounding box.

Our first method of structuring captures *communication*, i.e., a situation in which separate occurrence nets proceed concurrently and (occasionally) communicate with each other — see, for example, Figure 5, in which thick dashed arcs are used to represent communications so as to distinguish them from the interactions represented in conventional occurrence nets by causal arcs. (Note that another distinction is that interactions link conditions to events and events to conditions (as was shown in Figure 1), whereas communications link events — of separate occurrence nets — directly.)

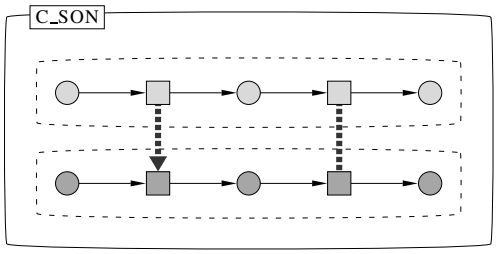

Fig. 2. A structured occurrence net composed out of two communicating occurrence nets.

In practice, when structuring a complex occurrence net into a set of simpler communicating occurrence nets, it is sometimes necessary to use synchronous communications. Hence, as shown in Figure 5, we allow for the use of two types of communication: thick dashed directed arcs indicate that an event in one occurrence net is a causal predecessor of an event in another occurrence net (i.e., information flow between occurrence nets was unidirectional), whereas undirected such arcs indicate that the two events have been executed synchronously (i.e., information flow was bidirectional). (In practice, interactions and communications of all the kinds described above can occur in the same overall structured occurrence net provided that a simple acyclicity constraint — similar to that used for ordinary occurrence nets — is satisfied.)

As an example of such structuring, Figure 3(a) shows a single occurrence net that is recording the interactions between two systems, the upper of which is itself exhibiting asynchronous behaviour, and Figure 3(b) shows a possible structuring of Figure 3(a)

into a structured occurrence net composed of two separate (communicating) occurrence nets, each portraying the activity of a single system. (Because of our rule that communications relate events directly, the upper occurrence net in Figure 3(b) has had to be augmented with additional events and implicit conditions.)

Note that the thick dashed arcs are abstractions of the details that correspond to such communications when one describes such a structured occurrence net by a single conventional occurrence net. The rules governing such abstractions, and the rationale for our introducing synchronous as well as causal communications, should become clearer later on when we discuss temporal abstraction (illustrated in Figure 9).

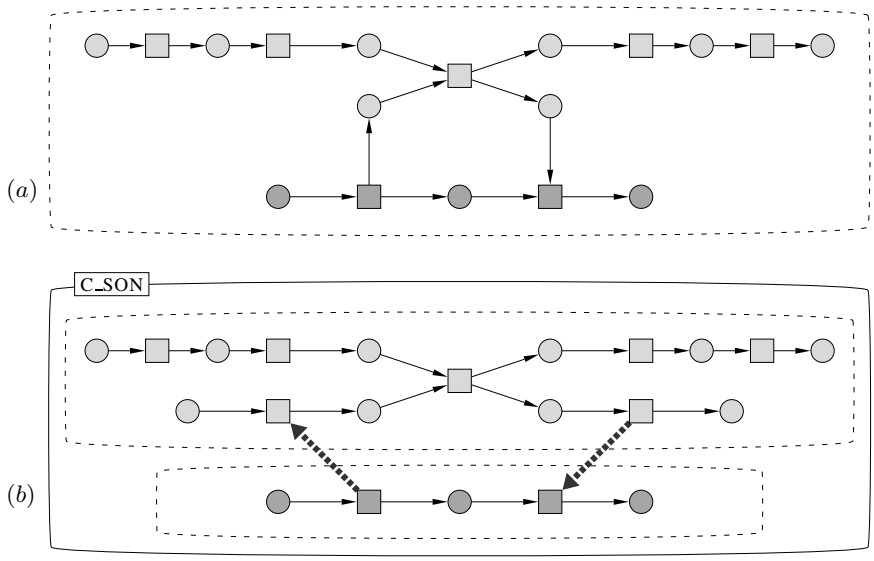

Fig. 3. (a) portrays the activity of two systems, one of which is exhibiting asynchronous behaviour, in a single occurrence net; (b) portrays an equivalent structured occurrence net, in which the activities of these two systems are shown in separate (communicating) occurrence nets.

Definition 4 (communication SON). *A* communication structured occurrence net *(or C_SON) is defined to be a tuple* $\text{C\_SON} \stackrel{\text{df}}{=} (\text{ON}_1, \ldots, \text{ON}_k, \kappa, \sigma)$*,* $k \geq 1$*, where each* $\text{ON}_i \stackrel{\text{df}}{=} (C_i, E_i, F_i)$ *is an occurrence net,*[3] *whereas* $\kappa, \sigma \subseteq \bigcup_{i \neq j} E_i \times E_j$ *are two relations (*σ *being symmetric) such that the following relation is acyclic:*

$$Prec_{\text{C\_SON}} \stackrel{\text{df}}{=} (\mathbf{F} \circ \mathbf{F})|_{\mathbf{C} \times \mathbf{C}} \cup (\mathbf{F} \circ (\kappa \cup \sigma)^+ \circ \mathbf{F}) \ .$$

In the above, as well as later on, we denote $\mathbf{C} \stackrel{\text{df}}{=} \bigcup_i C_i$*,* $\mathbf{F} \stackrel{\text{df}}{=} \bigcup_i F_i$ *and* $\mathbf{E} \stackrel{\text{df}}{=} \bigcup_i E_i$*. (These notations will also be used in indexed or primed form, e.g.,* $\mathbf{F}_j$ *and* $\mathbf{C}'$*.)*

[3] In this, and other similar definitions, different occurrence nets have *disjoint* sets of nodes.

Intuitively, if $(e, f) \in \kappa$ then e cannot happen after f, and if $(e, f) \in \sigma$ then e and f must happen synchronously. (Note that σ is included for convenience as it could be omitted after replacing κ by $\sigma \cup \kappa$.) For a communication structured occurrence net as in Definition 4, cuts and step executions need to be re-defined.

A *cut* of C_SON is a maximal (w.r.t. set inclusion) set of conditions $Cut \subseteq \mathbf{C}$ such that $(Cut \times Cut) \cap Prec^{+}_{\text{C\_SON}} = \varnothing$. The initial cut of C_SON, $Init_{\text{C\_SON}}$, is the union of the initial cuts of all the ON_i's, and the final cut, $Fin_{\text{C\_SON}}$, is the union of the final cuts of all the ON_i's. A useful characterisation of cuts can be obtained using the notion of a *causal chain* of C_SON which is any sequence of its nodes

$$\lambda \stackrel{\text{df}}{=} c_0 e^1_1 e^1_2 \ldots e^1_{k_1} c_1 e^2_1 e^2_2 \ldots e^2_{k_2} c_2 \ldots c_{m-1} e^m_1 e^m_2 \ldots e^m_{k_m} c_m \qquad (*)$$

where $m, k_1, \ldots, k_m \geq 1$, and all the c_i's are conditions and e^j_i's events of C_SON such that for all $i \leq m$ and $j < k_i$: $(c_{i-1}, e^i_1), (e^i_{k_i}, c_i,) \in \mathbf{F}$ and $(e^i_j, e^i_{j+1}) \in \sigma \cup \kappa$.

Proposition 5. *A set Cut of conditions is a cut of* C_SON iff *it is a maximal (w.r.t. set inclusion) subset of* $\mathbf{C}$ *such that there is no causal chain beginning and ending with a condition in Cut.*

Proof. Follows directly from the definitions. □

Proposition 6. *In a causal chain of* C_SON*: (i) no condition occurs more than once; and (ii) between two occurrences of an event there can only occur events and no conditions.*

Proof. (i) A repetition of a condition would contradict the acyclicity of $Prec_{\text{C\_SON}}$.
(ii) Suppose that λ is a causal chain with a sub-sequence $e\lambda' c\lambda'' e$ where e is an event and c condition. Then $c\lambda'' e\lambda' c$ is a causal chain, contradicting (i). □

The next result amounts to saying that a global state of a communication structured occurrence net is made up of local states of the component occurrence nets.

Proposition 7. *If Cut is a cut of* C_SON*, then $Cut \cap C_i$ is a cut of* ON_i*, for every $i \leq k$.*

Proof. Suppose that $C \stackrel{\text{df}}{=} Cut \cap C_i$ is not a cut. Then, since C is a set of concurrent conditions of ON_i as $(F_i \circ F_i)|_{C_i \times C_i} \subseteq Prec_{\text{C\_SON}}$, there is a condition $c \in C_i \setminus C = C_i \setminus Cut$ which is concurrent with all the conditions in C. Since $c \notin Cut$, we have that

$$(\{c\} \times Cut) \cap Prec^{+}_{\text{C\_SON}} \neq \varnothing \;\text{ or }\; (Cut \times \{c\}) \cap Prec^{+}_{\text{C\_SON}} \neq \varnothing \,.$$

Without loss of generality, we may assume that the former holds. Then the set Λ of all causal chains of C_SON starting with c and ending with a condition in Cut is non-empty.

Let $\lambda \in \Lambda$. We first observe that not all events in λ belong to the occurrence net ON_i as $(F_i \circ F_i)|_{C_i \times C_i} \subseteq Prec_{\text{ON}_i}$ and c is concurrent with all the conditions in C. Now denote by h_λ the number of initial events in λ which belong to ON_i. Since the first event in λ belongs to ON_i, $h_\lambda \geq 1$. Moreover, since the initial events in λ belonging to ON_i are separated by conditions in ON_i, by Proposition 6(i), $h_\lambda \leq |C_i|$. Hence there is a causal chain $\lambda \in \Lambda$ with the highest h_λ. We can partition λ as $\psi f g \psi' d$, where ψ comprises exactly $h_\lambda - 1$ events, $f \in E_i$ and $g \notin E_i$.

Take any condition $c' \in post(f)$. We now observe that $c' \in C_i \setminus Cut$ is not in the $Prec^+_{\text{C\_SON}}$ relation with any condition in Cut. Indeed, if $(c'', c') \in Prec^+_{\text{C\_SON}}$ and $c'' \in Cut$ then, by the fact that $|pre(c')| = 1$ and using the event f, we have $(c'', d) \in Prec^+_{\text{C\_SON}}$ a contradiction with the definition of a cut of C_SON. Moreover, if $(c', c'') \in Prec^+_{\text{C\_SON}}$ and $c'' \in Cut$ then, by $|post(c')| = 1$, we can find a new causal chain $\lambda' = \psi f c' f' \ldots \psi'' c''$ in Λ, where $f' \in E_i$, for which $h_{\lambda'}$ is greater than h_λ, contradicting the choice of λ.

Hence c' can be added to Cut, contradicting the latter's maximality. □

We now re-define the notion of a step execution.

Definition 5 (step execution of C_SON). *A* step execution *of the communication structured occurrence net* C_SON *is a sequence* $\chi \stackrel{\text{df}}{=} D_0\, G_1\, D_1 \ldots G_n\, D_n$, *where each* $D_i \subseteq \mathbf{C}$ *is a set of conditions and each* $G_i \subseteq \mathbf{E}$ *is a set of events, such that* $D_0 = Init_{\text{C\_SON}}$ *and, for every* $i \leq n$*:*

- $pre(G_i) \subseteq D_{i-1}$ *and* $D_i = (D_{i-1} \setminus pre(G_i)) \cup post(G_i)$*;*
- $(e, f) \in \kappa \cup \sigma$ *and* $f \in G_i$ *implies* $e \in \bigcup_{j \leq i} G_j$.

We also say that χ leads *to* D_n*, and that* D_n *is* reachable.

Note that if $(e, f) \in \sigma$ and $f \in G_i$ then also $e \in G_i$ as σ is symmetric and so $(f, e) \in \sigma$. We first show that a communication structured occurrence net cannot be completely blocked right at the beginning.

Proposition 8. *If* $\mathbf{E} \neq \varnothing$ *then there is at least one step execution involving a non-empty set of events.*

Proof. For a causal chain λ as in (*) and event e^i_j occurring in it, let $ind_\lambda(e^i_j) \stackrel{\text{df}}{=} i$. By Proposition 6(i,ii), this notion is well-defined as $e^i_j = e^{i'}_{j'}$ implies $i = i'$. Now, for any event e, let Λ_e be the set of all causal chains beginning with a condition in $Init_{\text{C\_SON}}$ and containing e. Clearly, $\Lambda_e \neq \varnothing$ as we can always find at least one suitable causal chain with all the elements in the component occurrence net to which e belongs. Finally, for every e, let $ind(e) \stackrel{\text{df}}{=} \max\{ind_\lambda(e) \mid \lambda \in \Lambda_e\}$. That $ind(e)$ is a well-defined integer follows from Proposition 6(i) which implies that $ind_\lambda(e) \leq |\mathbf{C}|$, for every $\lambda \in \Lambda_e$.

Let G be the (non-empty) set of all events e for which $ind(e)$ has the minimal value among all the events of C_SON. One can easily see that: (i) $e \in G$ implies that there is no event f such that $(f, e) \in \mathbf{F} \circ \mathbf{F}$; and (ii) if $(f, e) \in \kappa \cup \sigma$ then $f \in G$. Hence DGD', where $D = Init_{\text{C\_SON}}$ and $D' \stackrel{\text{df}}{=} (Init_{\text{C\_SON}} \setminus pre(G)) \cup post(G)$, is a step execution of C_SON involving a non-empty set of events G. □

Our aim now is to re-establish the basic behavioural characteristics of occurrence nets and, at the same time, capture a consistency between the individual and interactive views of computation.

Proposition 9. *Given a step execution as in Definition 5 and* $i \leq k$*,*

$$D'_0\, G'_1\, D'_1 \ldots G'_n\, D'_n\ ,$$

where $D'_i = D_i \cap C_i$ *and* $G'_i = G_i \cap E_i$*, is a step execution of* ON_i.

Proof. Follows directly from the definitions, $Init_{\mathrm{ON}_i} = Init_{\mathrm{C\_SON}} \cap C_i$ and the disjointness of the component occurrence nets. □

Note, however, that it may happen that a cut of an individual occurrence net ON_i may no longer be reachable through any step execution of C_SON.

Proposition 10. *Given a step execution as in Definition 5 and a causal chain as in (*), if $e^i_j \in G_l$ then:*

1. *each $e^i_{j'}$ with $j' < j$ belongs to some $G_{l'}$ with $l' \leq l$;*
2. *each $e^{i'}_{j'}$ with $i' < i$ belongs to some $G_{l'}$ with $l' < l$.*

Proof. The result follows from the following two observations. First, if $j = 1, i' = i-1$ and $j' = k_{i-1}$ then both e^i_j and $e^{i'}_{j'}$ belong to the same component occurrence net, and so by Propositions 3(1) and 9, $e^{i'}_{j'}$ belongs to some $G_{l'}$ with $l' < l$. Second, if $j' = j-1$ then, by Definition 5, $e^i_{j'}$ belongs to some $G_{l'}$ with $l' \leq l$. □

Theorem 1. *Given a step execution as in Definition 5, each D_i is a cut of* C_SON*, and no event occurs more than once.*

Proof. We observe that, by Propositions 1 and 9, for every $l \leq k$, $D_i \cap C_l$ is a cut of ON_l. Thus D_i cannot be extended by any new condition which is not in the $Prec^+_{\mathrm{C\_SON}}$ relation with the conditions in D_i. Hence, by Proposition 5, to show that D_i is a cut of C_SON it suffices to demonstrate that there is no causal chain $\lambda = ce \ldots fd$ such that $c, d \in D_i$. On the contrary, suppose that such a λ does exist. By $d \in D_i$ and $d \in post(f)$ and Propositions 3(2) and 9, we have $f \in G_h$ for some $h < i$. Hence, by Proposition 10, we have that $e \in G_z$ for some $z \leq h$ (note that we allow $e = f$). Thus, by $c \in pre(e)$ and Propositions 3(3) and 9, $c \notin D_i$, a contradiction. Hence D_i is a cut of C_SON.

The second part, i.e., that no event occurs more than once, follows directly from Propositions 1 and 9. □

As in the case of an occurrence net, a cut Cut of a communication structured occurrence net divides it into two parts. We define

$$\begin{aligned} preon_{\mathrm{C\_SON}}(Cut) &\stackrel{\mathrm{df}}{=} (\mathrm{ON}'_1, \ldots, \mathrm{ON}'_k, \kappa', \sigma') \\ poston_{\mathrm{C\_SON}}(Cut) &\stackrel{\mathrm{df}}{=} (\mathrm{ON}''_1, \ldots, \mathrm{ON}''_k, \kappa'', \sigma'') \ , \end{aligned}$$

where, for $i \leq n$, the component occurrence nets are defined by:

$$\mathrm{ON}'_i \stackrel{\mathrm{df}}{=} preon_{\mathrm{ON}_i}(Cut \cap C_i) \ \text{ and } \ \mathrm{ON}''_i \stackrel{\mathrm{df}}{=} poston_{\mathrm{ON}_i}(Cut \cap C_i) \ ,$$

and the relations capturing communication are given by:

$$\kappa' \stackrel{\mathrm{df}}{=} \kappa|_{\mathbf{E}' \times \mathbf{E}'} \quad \sigma' \stackrel{\mathrm{df}}{=} \sigma|_{\mathbf{E}' \times \mathbf{E}'} \quad \kappa'' \stackrel{\mathrm{df}}{=} \kappa|_{\mathbf{E}'' \times \mathbf{E}''} \quad \sigma'' \stackrel{\mathrm{df}}{=} \sigma|_{\mathbf{E}'' \times \mathbf{E}''} \ .$$

In the above, and later on, $\mathbf{E}'$ and $\mathbf{E}''$ are events in, respectively, all the ON'_i's and all the ON''_i's. Similar notation is used for sets of conditions and flow relations. Note that the ON'_i's and ON''_i are well-defined due to Proposition 7.

Proposition 11. *Let* C_SON′ *and* C_SON″ *be respectively the tuples* $preon_{\text{C\_SON}}(Cut)$ *and* $poston_{\text{C\_SON}}(Cut)$ *defined above.*

1. C_SON′ *and* C_SON″ *are communication structured occurrence nets such that:*

$$\mathbf{C} = \mathbf{C}' \cup \mathbf{C}'' \qquad \mathbf{C}' \cap \mathbf{C}'' = Cut \qquad \mathbf{E} = \mathbf{E}' \uplus \mathbf{E}'' \qquad \mathbf{F} = \mathbf{F}' \uplus \mathbf{F}'' \ .$$

2. $Init_{\text{C\_SON}'} = Init_{\text{C\_SON}}$, $Fin_{\text{C\_SON}'} = Cut = Init_{\text{C\_SON}''}$ *and* $Fin_{\text{C\_SON}''} = Fin_{\text{C\_SON}}$.
3. $\kappa' \cup \kappa'' = \kappa \setminus (\mathbf{E}' \times \mathbf{E}'')$ *and* $(\mathbf{E}'' \times \mathbf{E}') \cap \kappa = \varnothing$.
4. $\sigma' \cup \sigma'' = \sigma \setminus (\mathbf{E}' \times \mathbf{E}'')$ *and* $(\mathbf{E}'' \times \mathbf{E}') \cap \sigma = \varnothing$.
5. *Given step executions,* χ' *and* χ''*, of respectively,* C_SON′ *and* C_SON″*,*
   - χ' *is a step execution of* C_SON*;*
   - *if* χ' *leads to* Cut *then* $\chi' \varnothing \chi''$ *is a step execution of* C_SON.

Proof. (1,2) Follow from the definitions and Propositions 4 and 7.

(3,4) Follow from the fact that otherwise we would have had a causal chain in C_SON starting and ending at condition in Cut, contradicting Proposition 5.

(5) Follows from Proposition 4 and parts (1–4). □

Theorem 2. *One can always find a step execution involving all the events of a communication structured occurrence net.*

Proof. We proceed by induction on the number of events in a communication structured occurrence net. In the base case, when there are no events, there is nothing to show. In the induction step, from Proposition 8 it follows that we can find a step execution χ with a non-empty set of events. Let Cut be the resulting cut (see Theorem 1). Now, we can take $preon_{\text{C\_SON}}(Cut)$ and $poston_{\text{C\_SON}}(Cut)$, and find by induction a step execution χ' involving all the events of $poston_{\text{C\_SON}}(Cut)$. By Proposition 11(5), we then get that $\chi \varnothing \chi'$ is a step execution involving all the events of C_SON. □

Theorem 3. *Each cut of a communication structured occurrence net can be reached from the initial cut through some step execution.*

Proof. Let Cut be a cut of C_SON. We can take $preon_{\text{C\_SON}}(Cut)$. By Theorem 2, there is a step execution χ of $preon_{\text{C\_SON}}(Cut)$ leading to Cut (follows from Propositions 1 and 9). Thus, by Proposition 11(5), χ is also a step execution of C_SON leading to Cut. □

In this section we introduced a model of structured occurrence nets which captures communications between concurrently executed subsystems. We then demonstrated the soundness of this formalisation by showing that the model satisfies some key behavioural properties enjoyed by occurrence nets. We now go on to describe various other forms of relation that can be used to construct structured occurrence nets.

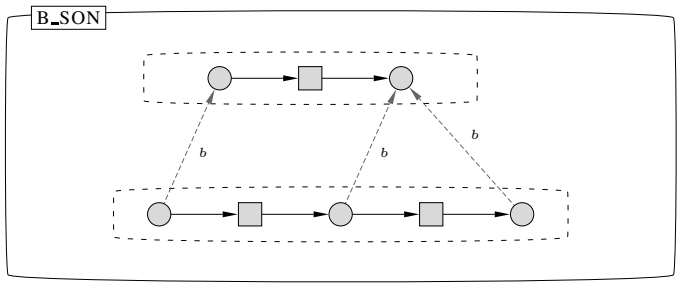

Fig. 4. Behavioural abstraction. The 'behaviour' relation is indicated by *b*-labelled edges.

4 Behavioural abstraction

Structures like that shown in Figure 2 and 3(b) capture communications between different systems but give no information about the evolution of individual systems. This orthogonal view is illustrated in Figure 4, where we have a two-level view of a system's history. (That it concerns a single system is indicated by the fact all conditions and events are similarly-shaded.)

A possible interpretation of Figure 4 is that the upper level provides a high-level view of a system which went through two successive versions which are represented by two conditions of the upper occurrence net (the event in the middle represents a version update). The lower occurrence net captures the behaviour of the system during the same period. Figure 4 also shows the 'behaviour' relation working across the two levels of description. The relation connects conditions in the lower part with those in the upper part which abstract them. We omit a formal definition of this two-level occurrence net as it is a special case of the construct introduced later in Definition 6.

In this section we aim at formalising the relationship which connects together different descriptions of the same system.

As already illustrated in Figure 4, any condition can be viewed either as a state (of some system), or as itself representing a system that presumably has its own states and events — just which is simply a matter of viewpoint. Moreover, as indicated in Figure 2 and 3(b), behaviours of different systems can interact with each other. In general, it is possible to have sets of related occurrence nets, some showing what has happened in terms of systems and their evolution, the other showing the behaviours of these systems. Thus the former can be viewed as the *behavioural abstraction* of the latter. What comes now is a combination of the structuring mechanisms that were illustrated in Figure 2, 3(b) and 4.

Figure 5 shows a simple example, involving the interacting activities of two systems (note that the same shading is used for the higher- and lower-level view of each system). This picture gives no information about the evolution of the two systems — some such additional information is portrayed in the following figures. Moreover, the upper part of the picture does not provide any information about the interactions between the two systems (basically, all it says is that 'there are two systems').

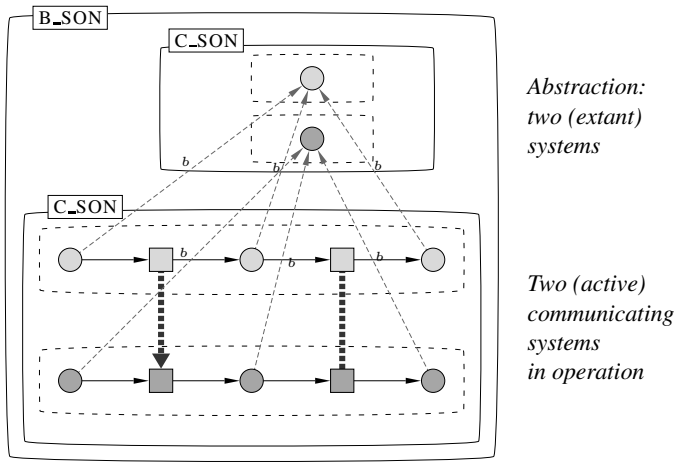

Fig. 5. Behavioural abstraction.

More interesting is Figure 6(a) which shows the history of an online modification of two systems, i.e., one in which the modified systems carry on from the states that had been reached by the original systems — a possibility that is easy to imagine, though often difficult to achieve dependably, especially with software systems. In this case, the 'abstracts' relation is non-trivial as it identifies those parts of the behaviours which are pre- and post-modification ones. (Strictly speaking, Figure 6(a) is not an exact reflection of the formal definition as different occurrence nets are assumed to be disjoint, and so the pair of overlapping occurrence nets is treated as a single occurrence net. However, one can portray a more abstract view of what is going on by showing two occurrence nets. Such a relationship can be deduced by looking at the nets of an upper level, and will be used below to identify the stages through which a system has passed during its execution.)

Another type of system modification is shown in Figure 6(b). It again shows that the two systems have each suffered some sort of modification, i.e., have evolved, once — the 'abstracts' relations between the two levels show which state sequences are associated with the systems before they were modified, and which with the modified systems. Note that in this case the behaviour of each system is represented by two disjoint occurrence nets. Thus the standard occurrence net theory does not work as desired as it would consider these two parts as concurrent whereas, in fact, one is meant to precede the other. In the proposed structured view the upper part provides the necessary information for the desired sequencing of the occurrence nets. Again, this is a feature which is due to the multi-level view of behaviours.

The last motivating example in this section, Figure 7, shows some of the earlier history of the two systems in Figure 5, i.e., that one system has spawned the other system, and after that both systems went through some independent further evolutions. Note that additional information could have been portrayed in the figures by showing relations, from the earlier versions of the two systems, to parts of the occurrence nets

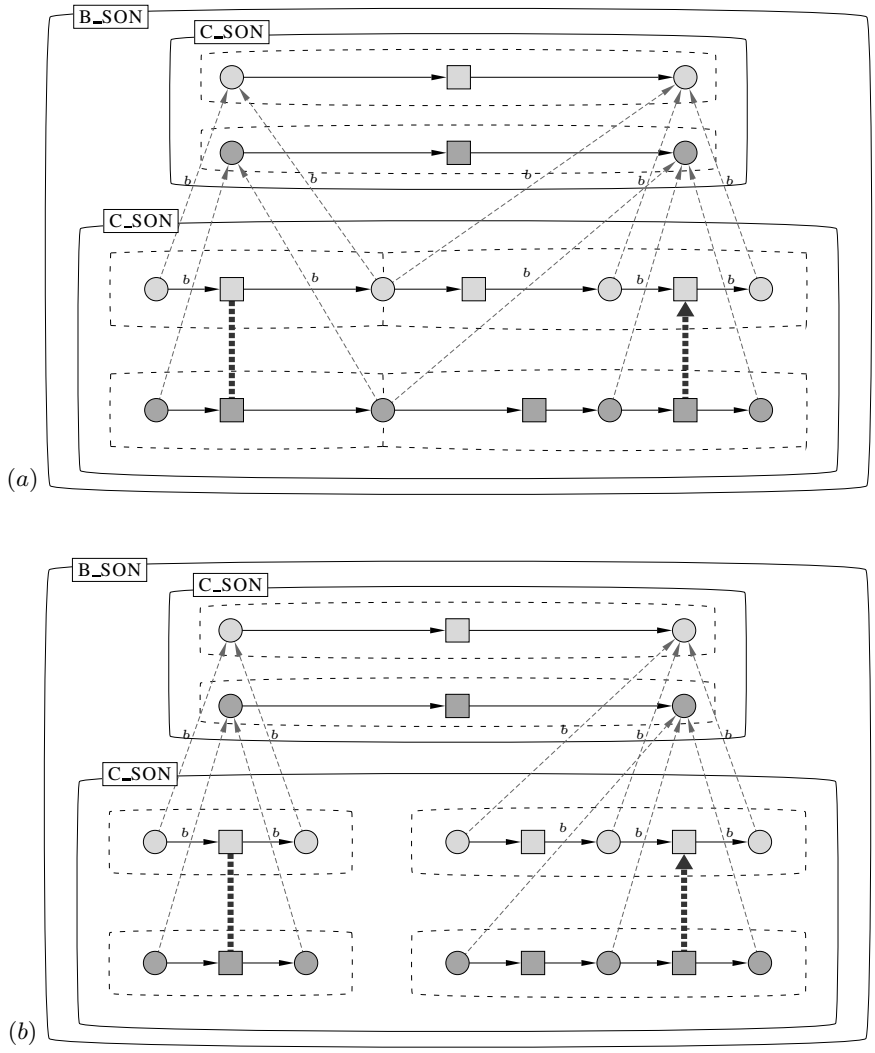

Fig. 6. System modifications: (a) online, with the modified systems carrying on from where they left off; (b) offline, with the modified systems restarting from a predefined initial state.

which recorded the behaviour that occurred when these earlier versions were active — but to avoid undue graphical complexity no attempt is made to show that here (it may happen that no records of the prior behaviour of the two systems are available).

We will now formalise the concept of 'behavioural abstraction' outlined above. In what follows, we call an occurrence net ON *linear* if $|Init_{\mathrm{ON}}| = 1$ and $|pre(e)| = |post(e)| = 1$, for every event e. Such an occurrence net represents a single chain of alternating conditions and events.

Definition 6 (behavioural SON). *A* behavioural structured occurrence net *(or B_SON) is a tuple* $\text{B\_SON} \stackrel{\text{df}}{=} (\text{C\_SON}, \text{C\_SON}', \beta)$*, where* C_SON *is a communication structured occurrence net as in Definition 4,* $\text{C\_SON} = (\text{ON}'_1, \ldots, \text{ON}'_l, \kappa', \sigma')$ *is a communication structured occurrence net with each* ON'_i *being linear, and* $\beta \subseteq \mathbf{C} \times \mathbf{C}'$ *is a relation such that:*

- $dom(\beta) = \mathbf{C}$*;*
- $\beta(pre(e)) \cap \beta(post(e)) \neq \varnothing$*, for every* $e \in \mathbf{E}$*;*
- *For every* $i \leq k$*, there is exactly one* $j \leq l$ *such that* $\beta(C_i) \subseteq C'_j$*;*
- *For every* $c' \in \mathbf{C}'$*, there is exactly one* $i \leq k$ *such that* $phse(c') \stackrel{\text{df}}{=} \beta^{-1}(c')$ *is a phase of* ON_i*;*
- *For all* $c', c'', c''' \in \mathbf{C}'$*, if* $(c', c''), (c'', c''') \in \mathbf{F}'^{+}$ *and both* $phse(c')$ *and* $phse(c''')$ *are non-empty then so is* $phse(c'')$*;*
- $Prec_{\text{B\_SON}} \stackrel{\text{df}}{=} Prec_{\text{C\_SON}} \cup Prec_{\text{C\_SON}'} \cup Prec$ *is an acyclic relation, where*

$$Prec \stackrel{\text{df}}{=} \bigcup_{(c,d) \in Prec^{+}_{\text{C\_SON}} | \mathbf{C} \times \mathbf{C}} \beta(c) \times \beta(d) \setminus id_{\mathbf{C}'} \;\cup \bigcup_{(c',d') \in Prec^{+}_{\text{C\_SON}'} | \mathbf{C}' \times \mathbf{C}'} Max_{phse(c')} \times Min_{phse(d')} \setminus id_{\mathbf{C}} \; .$$

Intuitively, $Prec_{\text{C\_SON}}$ and $Prec_{\text{C\_SON}'}$ capture causalities resulting from communications between behaviours, whereas $Prec$ reflects the succession of evolutions the system had undergone during the history captured by B_SON.

We can identify a 'continuation' relation between different occurrence nets ON_i as well as a between different phases within each occurrence net ON_i (note that each phase can be thought of as a sub-occurrence net if we include all events in-between its delimiting cuts). More precisely, we say that ON_i is a *continuation* of ON_j if there exist conditions c' and c'' such that $(c', c'') \in \mathbf{F}' \circ \mathbf{F}'$, $Fin_{\text{ON}_j} \subseteq \beta^{-1}(c')$ and $Init_{\text{ON}_i} \subseteq \beta^{-1}(c')$. Also, each ON_i can be represented as a union of occurrence nets, $\widetilde{\text{ON}}_1, \ldots, \widetilde{\text{ON}}_m$, where for each $1 < j \leq m$ there exist conditions c' and c'' such that $(c', c'') \in \mathbf{F}' \circ \mathbf{F}'$, $\widetilde{\text{ON}}_{j-1} = \text{ON}_{phse(c')}$ and $\widetilde{\text{ON}}_j = \text{ON}_{phse(c'')}$. We then say that $\widetilde{\text{ON}}_j$ is a *continuation* of $\widetilde{\text{ON}}_{j-1}$.

We now introduce cuts and step executions for the behavioural structured occurrence net in Definition 6. A *cut* of B_SON is a maximal (w.r.t. set inclusion) set of conditions $Cut \subseteq \mathbf{C} \cup \mathbf{C}'$ such that $(Cut \times Cut) \cap Prec^{+}_{\text{B\_SON}} = \varnothing$. The initial cut of B_SON is the union, $Init_{\text{B\_SON}}$, of the initial cut of $\text{C\_SON}'$ and the initial cuts of all the ON_i's such that $\beta(Init_{\text{ON}_i}) \subseteq Init_{\text{C\_SON}'}$.

Definition 7 (step execution of B_SON). *A* step execution *of the behavioural structured occurrence net* B_SON *is a sequence* $\chi \stackrel{\text{df}}{=} D_0 \, G_1 \, D_1 \ldots G_n \, D_n$*, where each* $D_i \subseteq \mathbf{C} \cup \mathbf{C}'$ *is a set of conditions and each* $G_i \subseteq \mathbf{E} \cup \mathbf{E}'$ *is a set of events, such that* $D_0 = Init_{\text{B\_SON}}$ *and, for every* $i \leq n$*:*

- $pre(G_i) \cup Max_i \subseteq D_{i-1}$*;*
- $(e, f) \in \kappa \cup \sigma \cup \kappa' \cup \sigma'$ *and* $f \in G_i$ *implies* $e \in \bigcup_{j \leq i} G_j$*;*
- $D_i = (D_{i-1} \setminus (pre(G_i) \cup Max_i)) \cup post(G_i) \cup Min_i$*;*

where $Min_i \stackrel{\text{df}}{=} \bigcup_{c' \in post(\mathbf{E}' \cap G_i)} Min_{phse(c')}$ *and* $Max_i \stackrel{\text{df}}{=} \bigcup_{c' \in pre(\mathbf{E}' \cap G_i)} Max_{phse(c')}$*. We also say that* χ leads *to* D_n*, and that* D_n *is* reachable.

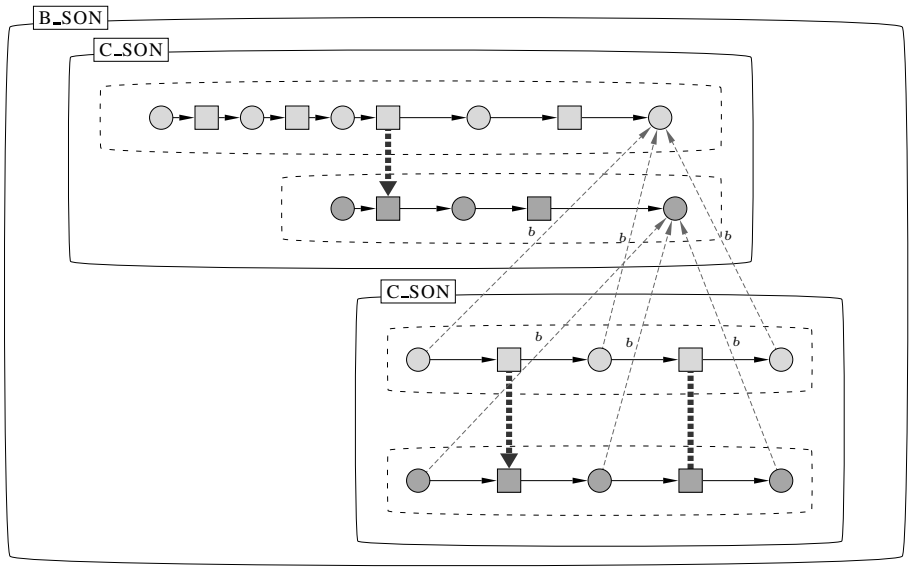

Fig. 7. System creation where the upper system has spawned the lower system.

We next establish a consistency between the individual and interactive views of computation, intertwined with the record of evolutions of the systems involved. The way we obtain the desired results is through a translation from a behavioural structured occurrence net to a communication structured occurrence net that closely simulates it. The basic idea is to chain together, using new events, the different phases representing stages of evolution of each sub-system, and then synchronise these events with the corresponding events in the linear occurrence nets capturing the succession of the stages.

Let B_SON be a behavioural structured occurrence net as in Definition 6. We construct a communication structured occurrence net

$$\widehat{\text{C\_SON}} \stackrel{\text{df}}{=} (\widehat{\text{ON}}_1, \ldots, \widehat{\text{ON}}_l, \text{ON}'_1, \ldots, \text{ON}'_l, \kappa \cup \kappa', \sigma \cup \sigma' \cup \widehat{\sigma}) \ ,$$

by creating, for each element of $i \leq l$, an occurrence net $\widehat{\text{ON}}_i$, as follows.

Suppose that $c_0, \ldots, c_m$ are all the conditions of ON'_i such that $phse(c_j)$ is non-empty listed according to the causality relation F'_i. Moreover, let $\{e_j\} = post(c_{j-1}) = pre(c_j)$ for $1 \leq j \leq m$. For every $j \leq m$, we define $\widetilde{\text{ON}}_j$ as $\text{ON}_{phse(c_j)}$ with all the conditions $c \in Max_{phse(c_j)}$ renamed to $\widehat{c}$. Then $\widehat{\text{ON}}_i$ is defined as the union of all the occurrence nets $\widetilde{\text{ON}}_j$ together with new events $\widetilde{e}_1, \ldots, \widetilde{e}_m$ satisfying $pre(e_j) = Max_{phse(c_{j-1})}$ and $post(e_j) = Min_{phse(c_j)}$, for $j \leq m$. Furthermore, we add $(\widehat{e}_j, e_j)$ and $(e_j, \widehat{e}_j)$ to $\widehat{\sigma}$.

It is easy to check that the translation satisfies the following.

Proposition 12. *$\widehat{\text{C\_SON}}$ is a communication structured occurrence net.*

Proposition 13. *No cut Cut of $\widehat{\text{C\_SON}}$ contains $\widehat{c}$, for any $c \in Cut$, and the cuts of* B_SON *are the cuts of $\widehat{\text{C\_SON}}$ after renaming each occurrence of $\widehat{c}$ to c.*

Proposition 14. *The step executions of* B_SON *are the step executions of* $\widehat{\text{C\_SON}}$ *after renaming each occurrence of* $\widehat{c}$ *to* c*.*

Our aim now is to re-establish the basic behavioural characteristics of occurrence nets, and at the same time establish the consistency between the individual and interactive views of computation. Given what we proved about the communication structured occurrence nets, and the above properties of $\widehat{\text{C\_SON}}$, we directly obtain the following.

Theorem 4. *Given a step execution as in Definition 7, for every* $j \leq k$*, the sequence*

$$D_0 \cap C_j \;\; G_1 \cap E_j \;\; D_1 \cap C_j \;\; \ldots \;\; G_n \cap E_j \;\; D_n \cap C_j$$

is either a sequence of empty steps, or a step execution of the occurrence net ON_j *possibly preceded and/or followed by a sequence of empty sets (in the former case, the first non-empty set is the initial cut of* ON_j*, and in the latter the final one). Moreover, for every* $j \leq l$*, the sequence*

$$D_0 \cap C'_j \;\; G_1 \cap E'_j \;\; D_1 \cap C'_j \;\; \ldots \;\; G_n \cap E'_j \;\; D_n \cap C'_j$$

is a step execution of ON'_j*.*

Theorem 5. *Given a step execution as in Definition 7, each* D_i *is a cut of* B_SON*, and no event occurs more than once.*

Theorem 6. *One can always find a step execution involving all the events of a behavioural structured occurrence net.*

In this section we introduced structured occurrence nets capturing an abstraction mechanism between different representations of the same system. We have also demonstrated its soundness through showing that the resulting structured representation retains the desirable properties of the basic occurrence net model.

5 Spatial and Temporal Abstractions

What we will call *spatial abstraction* is based on the relation 'contains/is component of'. Figure 8 shows the behaviour of a system and of its three systems of which it is composed, and how its behaviour is related to that of these components. (This figure does not represent the matter of *how*, or indeed whether, the component systems are enabled to communicate, i.e., what design is used, or what connectors are involved.) Having identified such a set of communicating systems, and hence the *containing* system which they make up, then each member of this set has the other members as its *environment*.

Definition 8 (spatial abstraction S_SON). *A* spatial abstraction structured occurrence net *(or S_SON) is a tuple* $\text{S\_SON} \stackrel{\text{df}}{=} (\text{C\_SON}, \text{C\_SON}', \varsigma)$*, where* C_SON *is a communication structured occurrence net as in Definition 4,* $\text{C\_SON}' = (\text{ON}_1, \ldots, \text{ON}_l, \text{ON}, \kappa', \sigma')$ *(*$l < k$*) is a communication structured occurrence net, and* $\varsigma \subseteq (\widehat{\mathbf{C}} \times C) \cup (\widehat{\mathbf{E}} \times E)$*, where* $\widehat{\mathbf{C}} \stackrel{\text{df}}{=} \bigcup_{i>l} C_i$ *and* $\widehat{\mathbf{E}} \stackrel{\text{df}}{=} \bigcup_{i>l} E_i$*, is a relation such that:*

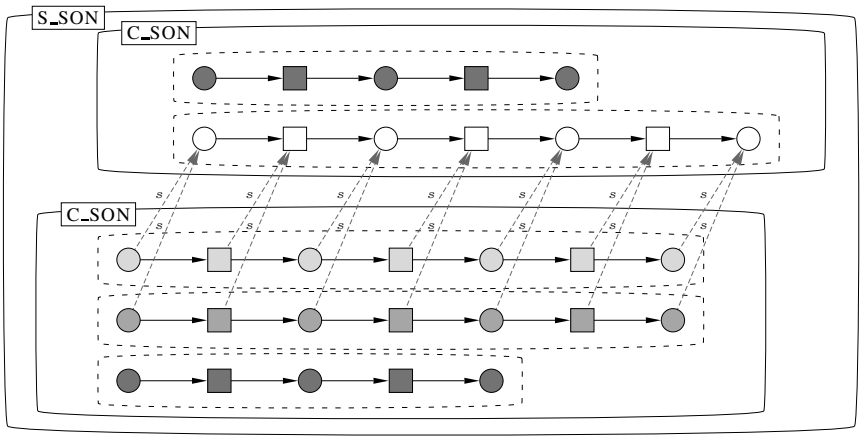

Fig. 8. System composition. The non-trivial part of the 'spatially-abstracts' relation is indicated by s-labelled edges.

- $dom(\varsigma) = \widehat{\mathbf{C}} \cup \widehat{\mathbf{E}}$ *and* $codom(\varsigma) = C \cup E$*;*
- $|\varsigma(x)| = 1$ *for all* $x \in \widehat{\mathbf{C}} \cup \widehat{\mathbf{E}}$*;*
- $(Cut \setminus \widehat{\mathbf{C}}) \cup \varsigma^{-1}(Cut \cap \widehat{\mathbf{C}})$ *is a cut of* C_SON*, for every cut* Cut *of* C_SON$'$*;*
- $pre(e) \subseteq \varsigma^{-1}(pre(\varsigma(e)))$ *and* $post(e) \subseteq \varsigma^{-1}(post(\varsigma(e)))$*, for every* $e \in \widehat{\mathbf{E}}$*;*
- $Prec_{\text{S\_SON}} \stackrel{\text{df}}{=} Prec_{\text{C\_SON}} \cup Prec'$ *is a acyclic, where* $Prec'$ *is the union of relations* $(\varsigma^{-1}(pre(e)) \setminus \varsigma^{-1}(post(e))) \times \varsigma^{-1}(e)$ *and* $\varsigma^{-1}(e) \times (\varsigma^{-1}(post(e)) \setminus \varsigma^{-1}(pre(e)))$*, for every* $e \in E'$*.*

One can define the cuts and step executions for S_SON similarly as has been done in Section 4 for B_SON, and then obtain results similar in essence and applicability to those formulated for B_SON.

The above is, as indicated, a *spatial* abstraction — one can also have a *temporal* abstraction, as shown in Figure 9(a). When one so 'abbreviates' parts of an occurrence net one is in effect defining atomic actions, i.e., actions that appear to be instantaneous to their environment. The rules that enable one to make such abbreviations are non-trivial when multiple concurrent activities are shown in the net. These rules are best illustrated by an alternative representation for an occurrence net together with its abbreviations, namely a structured occurrence net in which each abbreviated section (or 'atomic' activity) of the net is shown surrounded by an enclosing 'event box'. Figure 9(b) shows this alternative representation of Figure 9(a), the top part of which can readily be recreated by 'collapsing' Figure 9(b)'s occurrence net, i.e., by replacing the enclosed sections by simple event symbols, as shown in Figure 9(c). This net collapsing operation is much trickier with occurrence nets that represent asynchronous activity since there is a need to avoid introducing cycles into what is meant to be an acyclic directed graph. (Hence the need, on occasion, to use synchronous system interactions.) This is the main subject of [4] and is illustrated in Figure 10.

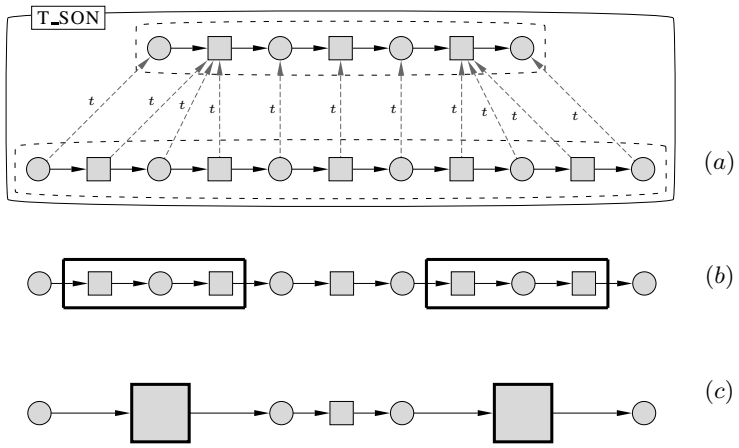

Fig. 9. System abbreviation. The 'temporally abstracts' relation is indicated by t-labelled edges. (a) depicts the temporal abstraction structured occurrence net view of what has happened, whereas (b) delineates the 'collapsed' parts of behaviour, and (c) the result of such a collapsing.

Definition 9 (temporal abstraction T_SON). *A* temporal abstraction structured occurrence net *(or T_SON) is a tuple* $\text{T\_SON} \stackrel{\text{df}}{=} (\text{C\_SON}, \text{C\_SON}', \tau)$ *where* C_SON *is a communication structured occurrence net as in Definition 4,* $\text{C\_SON}' = (\text{ON}'_1, \ldots, \text{ON}'_k, \kappa', \sigma')$ *is a communication structured occurrence net, and* $\tau : \mathbf{C}' \cup \mathbf{E}' \to \mathbf{C} \cup \mathbf{E}$ *is a mapping such that, for every* $i \leq k$*:*

- $\tau(C'_i \cup E'_i) = C_i \cup E_i$*,* $\tau^{-1}(C_i) \subseteq C'_i$ *and* $\tau(E'_i) = E_i$*;*
- $\tau^{-1}(e)$ *is a block of* ON'_i*, for every* $e \in E_i$*;*
- $|\tau^{-1}(c)| = 1$*, for every* $c \in C_i$*;*
- $F_i = \{(x, y) \in (\mathbf{C} \cup \mathbf{E}) \times (\mathbf{C} \cup \mathbf{E}) \mid (\tau^{-1}(x) \times \tau^{-1}(y)) \cap F'_i \neq \varnothing\}$*;*
- κ *is the set of all* $(e, f) \in \mathbf{E} \times \mathbf{E}$ *such that* $(\tau^{-1}(e) \times \tau^{-1}(f)) \cap \kappa'$ *is non-empty; and*
- σ *is the set of all* $(e, f) \in \mathbf{E} \times \mathbf{E}$ *such that* $(\tau^{-1}(e) \times \tau^{-1}(f)) \cap \sigma'$ *is non-empty, or both* $(\tau^{-1}(e) \times \tau^{-1}(f)) \cap \kappa'$ *and* $(\tau^{-1}(f) \times \tau^{-1}(e)) \cap \kappa'$ *are non-empty.*

A practical way in which temporal abstraction might be used is to analyse the behaviour at the higher level of abstraction, which can be done more efficiently, and after finding a problem mapping it to a corresponding behaviour at the lower level (and possibly continuing the analysis there). To give a flavour of the kind of result which would provide an underpinning for this approach, we have the following.

Theorem 7. *Let* T_SON *be a temporal abstraction structured occurrence net as in Definition 9, and* $D_0 \{e_1\} D_1 \ldots D_{n-1} \{e_n\} D_n$ *be a step execution of* C_SON*. Let* $i \leq k$ *and* $f_1 \ldots f_q$ *be the subsequence of* $e_1 \ldots e_n$ *comprising the events in* E_i*. For every* $j \leq q$*, let* $e_{j1} \ldots e_{jm_j}$ *be a listing of all the events of* $\tau^{-1}(f_j)$ *such that* $(e_{jr}, e_{js}) \notin F'_i \circ F'_i$*, for all* $r > s$*. Then* $e_{11} \ldots e_{1m_1} \ldots e_{n1} \ldots e_{qm_q}$ *is a firing sequence of* ON'_i*.*

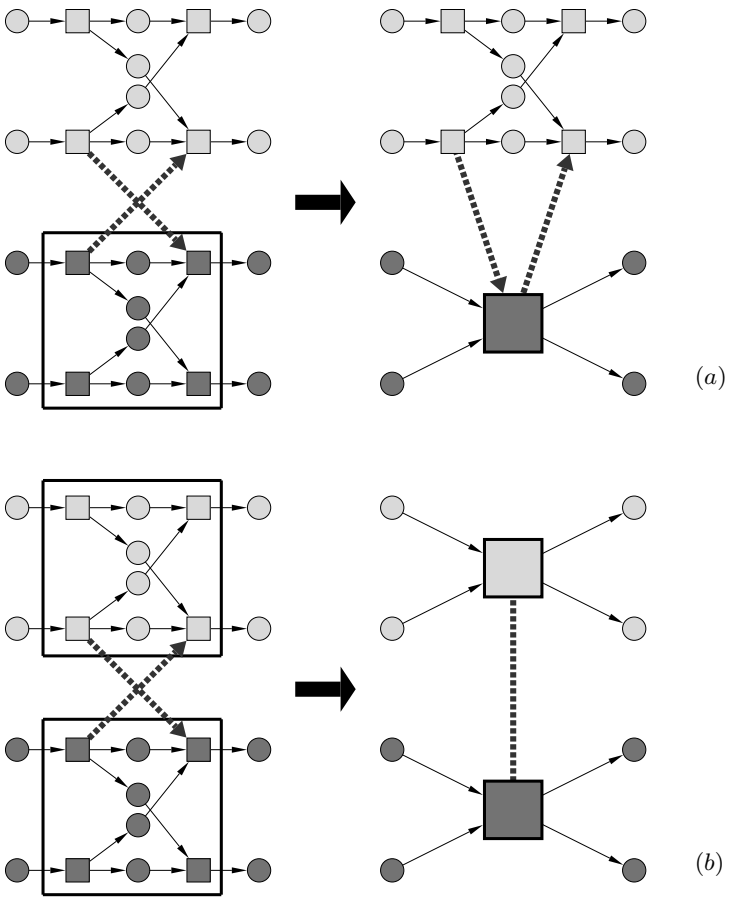

Fig. 10. Two valid collapsings which give rise to asynchronous (in (a)) and synchronous (in (b)) communication between abstract events.

Proof. Follows directly from the definitions. In particular, it is always possible to list the events of $\tau^{-1}(f_j)$ in a way which is consistent with the relation F_i' since F_i^+ is a partial order. Moreover, such a listing is a firing sequence from any cut C satisfying $pre(f_j) \subseteq C$ to the cut $(C \setminus pre(f_j)) \cup post(f_j)$. □

In this section we have presented composition and abbreviation, i.e., spatial and temporal abstraction, as though they are quite separate — in practice, it is likely that useful abstractions will result from successive applications of both spatial *and* temporal ones.

We envisage that abstraction mechanisms described in this section will be particularly useful in improving the efficiency of verification techniques based on structured occurrence nets. A possible step of achieving this would be to exploit the structuring of the execution histories allowing, e.g., to carry out separate checks on those fragments

which correspond to abbreviated behaviours, and then feeding the results to the final stage of verification.

6 Information retention and judgement

We now introduce the idea of one occurrence net *retaining* information about another occurrence net that is representing the activity of some given system. This could be, for example, in order to provide fault tolerance in the form of 'recovery points', enabling the given system to fall back and restart in order that failure might be averted, or to enable the post hoc assessment of the given system's activities by some separate 'judgmental' system, tasked with trying to determine whether and why a system failed.

A simple example of state retention aimed at supporting recovery points is shown in Figure 11(a). The upper system is shown as initially acquiring (through its first event) information about an initial fragment of the activity of the lower system, but then going on and retaining more information about this system. Intuitively, each event of the upper system is supposed to describe a recovery point given by the cut made out of the conditions of a cut in the lower system.

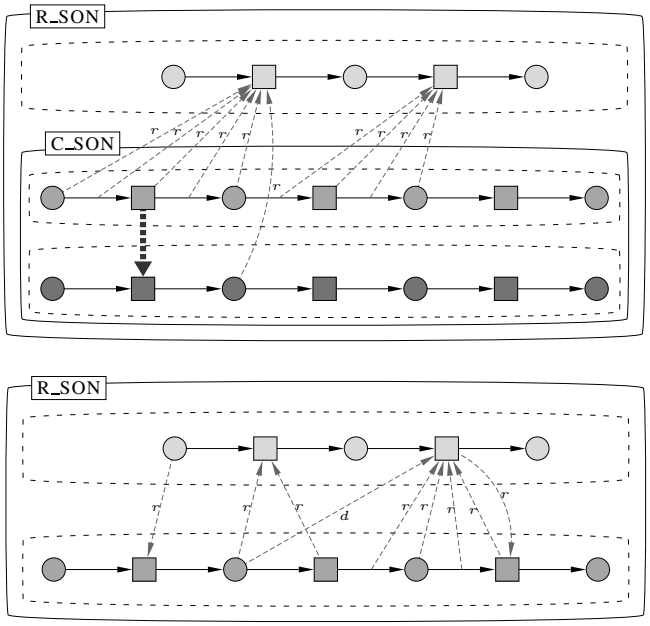

Fig. 11. State retention. The 'retains' and 'discards' relations are indicated by r-labelled and d-labelled edges, respectively. (a) shows how the upper systems accumulates retained information about the lower system, and (b) shows two systems which retain (and also discard) information about each other.

Definition 10 (state retention R_SON). *A* state retention structured occurrence net *(or R_SON) is a tuple* $\text{R\_SON} \stackrel{\text{df}}{=} (\text{C\_SON}_1, \ldots, \text{C\_SON}_m, \rho, \delta)$*, where each* C_SON_i *is a communication structured occurrence net and* ρ, δ *are relations such that:*

- $\rho, \delta \subseteq (\widehat{\mathbf{C}} \cup \widehat{\mathbf{E}} \cup \widehat{\mathbf{F}} \cup \widehat{\kappa} \cup \widehat{\sigma}) \times \widehat{\mathbf{E}}$ *where* $\widehat{X} \stackrel{\text{df}}{=} \bigcup_i X_i$*, for* $X = \mathbf{C}, \mathbf{E}, \mathbf{F}, \kappa, \sigma$.
- *if* $((x, y), e) \in \rho$ *then* $x, y \in \mathit{info}(e)$*, where for each* $e \in \widehat{\mathbf{E}}$*,* $\mathit{info}(e)$ *denotes the set of all* z *satisfying* $(z, e) \in (\rho \circ \widehat{\mathbf{F}}^*) \setminus (\delta \circ \widehat{\mathbf{F}}^*)$.
- *if* $(z, e) \in \delta$ *then* $(z, e) \in \mathit{info}(e) \setminus \rho$.
- *the relation* $\widehat{\mathbf{F}} \circ (\rho \cup \delta \cup \widehat{\mathbf{F}}) \circ \widehat{\mathbf{F}}|_{\widehat{\mathbf{C}} \times \widehat{\mathbf{C}}}$ *is acyclic.*

In the above definition, each C_SON_i represents a system's evolution for which some information is being retained during the evolution C_SON_j of other systems. The relation ρ formally captures this, and $(z, e) \in \rho$ means that the information about z has been acquired by executing e. This information does not need to be complete (indeed, there may even be 'gaps' in the retained information, as in the example in Figure 11(a)), and the only requirement is that information about a relationship (an arc or edge) implies that the elements it involves (the endpoints) are also known. This 'knowing' is assumed to be cumulative, i.e., if e' is a predecessor of e then all information acquired during the activation of e' is retained and available at e as well — see the definition of $\mathit{info}(e)$. Another relation, δ, is used to model the 'deletion' of previously acquired knowledge introduced, as illustrated with d-labelled edges in Figure 11.

Also, it is worth stressing that we do not assume that if z preceded z' in C_SON_i then the information about the former was necessarily acquired before z'. However we still have a general acyclicity requirement. For specific applications further assumptions related to acyclicity may be needed.

As already indicated, the notion of a 'failure' event involves, in principle, three systems — the given (possibly failing) system, its environment, and a judging system. This judging system may interact directly and immediately with the given system, in which case it is part of the system's environment, e.g., in VLSI an on-chip testing facility [14]; another example, in a very different world, is a football referee! Alternatively the judging system may be deployed after the fact using an occurrence net that represents how the failing event occurred. Figure 12 is an attempt to portray this. It deliberately represents a situation in which a judgement system has obtained and retained only incomplete evidence of the states and events, and even the causal relationships between conditions and events, of two interacting systems (each of which constitutes the other's environment).

The judgment system is shown as having gradually accumulated information about the two systems, and then used this information to help reach a judgement as to whether a particular event was erroneous, and a failure has occurred. Such 'retained' information might have been obtained directly by observation of an actual system and its environment, or may be little more than guesswork about the given system's presumed activity.

Definition 11 (judgement J_SON). *A* judgement structured occurrence net *(or J_SON) is a tuple* $\text{J\_SON} \stackrel{\text{df}}{=} (\text{C\_SON}_1, \ldots, \text{C\_SON}_m, \rho, \delta, \iota)$ *such that* $(\text{C\_SON}_1, \ldots, \text{C\_SON}_m, \rho, \delta)$ *is a state retention structured occurrence net as in Definition 10 and* ι *is a relation* $\iota \subseteq \widehat{\mathbf{E}} \times \widehat{\mathbf{E}}$ *satisfying* $e \in \mathit{info}(e')$*, for each* $(e, e') \in \iota$*, and the relation* $\widehat{\mathbf{F}} \circ (\rho \cup \delta \cup \iota \cup \widehat{\mathbf{F}}) \circ \widehat{\mathbf{F}}|_{\widehat{\mathbf{C}} \times \widehat{\mathbf{C}}}$ *is acyclic.*

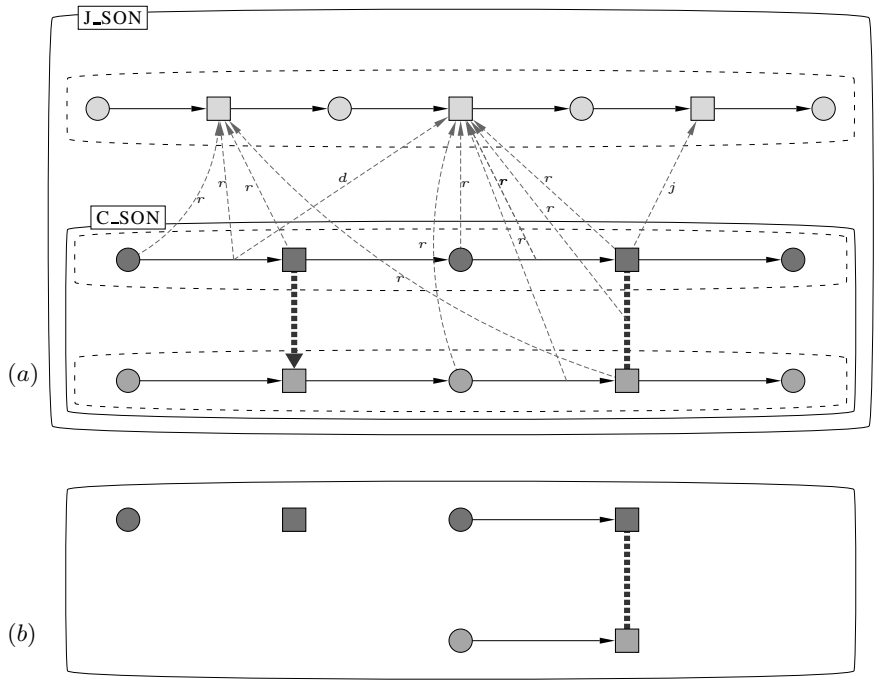

Fig. 12. (a) shows post-hoc judgement involving a judgemental system (upper part) and an active system together with its environment (lower part). The 'retains', 'deletes' and 'judges' relations are indicated by r-labelled, d-labelled and j-labelled edges, respectively. (b) shows the part of the active system behaviour on the basis of which judgement was made.

The relationship $(e, e') \in \iota$ represents an act of judging an event e through the event e' of the judging system. For this to be valid, e must be known (or retained) at e', and so we assume that $e \in \mathit{info}(e')$ using notation introduced in Definition 10.

Retracing the 'fault-error-failure' chain, after a judgment has been made that a particular event needs to be regarded as a failure involves following causal arrows *in either direction* within a given occurrence net, and following relations so as to move from one occurrence net to another. Thus one could retrace (i) the source and/or consequence of an interaction between systems, (ii) from a system to some guilty component(s), (iii) from a component to the system(s) built from it, or (iv) from a given system to the system(s) that created or modified it. All this tracing activity could be undertaken by some tracing system (perhaps a part of the judgement system) using whatever evidence is available (e.g., a retained occurrence net which is alleged to record what happened). This tracing system (just like a judgment system) can of course itself fail (in the eyes of some other judgment system)! The actual implementation of such tracing in situations of ongoing activity, and of potential further failures, e.g., such as interfering with witnesses and the jury (in a judicial context), involves problems such as those addressed by the *chase protocols* [20].

7 Applications of Structured Occurrence Nets

Structured occurrence nets have a number of different potential applications. For example, they could we believe be used to extend the capabilities of (i) existing occurrence net based model-checking approaches to system evaluation, and (ii) existing tools for the automated synthesis of systems (e.g. VLSI designs) from exemplar occurrence nets. Such applications involve fully-detailed structured occurrence nets (produced either from actual systems, or from models of intended systems). However, we first discuss the use of structured occurrence nets that were created after the fact from whatever evidence was available (and that as a consequence are likely to be far from complete) to assist the task of analyzing (accidental or malicious) failures in large complex evolving systems, possibly involving software, hardware and people.

One can envisage a given judge, having identified some system event as a failure, analysing a structured occurrence net, i.e., a set of related occurrence nets (dealing with the various abstractions of the various relevant systems), in an attempt to identify (i) the fault(s) that should be blamed for the failure, and/or (ii) the erroneous states that could and should be corrected or compensated for. Unless we assume that the occurrence nets are recorded correctly and completely as an automated by-product of system activity, in undertaking such a task it may well prove appropriate during such an analysis to correct or add to the occurrence nets, both individually and as a set, based on additional evidence or assumptions about what occurred.

Different judges (even different automated judgement systems) could of course, even if they identify the same failure event, come to different decisions regarding what actually happened and in determining the related faults and errors — possibly because they use different additional information (e.g., assumptions and information relating to system designs and specifications) to augment the information provided by the occurrence nets themselves. The result of such analyses could be thought of as involving the marking-up of the set of occurrence nets so as to indicate a four-way classification of all their places, namely as 'Erroneous', 'Correct', 'Undecided', and 'Not considered'.

As indicated earlier, the production of such a classification is likely to involve repeated partial traversals of the occurrence nets, following causal arrows backwards within a given occurrence net in a search for causes and forwards in a search for consequences. In addition it will involve following relations so as to move from one occurrence net to another. A simplistic example of this is the recognition that a given system's behaviour had, after a period of correct operation, started to exhibit a succession of errors might lead to investigating the related occurrence net representing the system's evolution to determine if it had suffered a modification at the relevant time.

This way of describing the failure analysis task using occurrence nets might be regarded as essentially metaphorical, i.e., essentially just as a way of describing (semi)-formally what is currently often done by expert investigators in the aftermath of a major system failure. However, at the other extreme one can imagine attempting to automate the recording and analysis of actual occurrence nets — indeed one could argue that this is likely to be a necessary function of any complex system that really merited the currently fashionable appellations 'self-healing' and 'autonomic'. The more likely, and practical, possibility — one that we plan to investigate — is the provision of computer assistance for the tasks of representing, checking the legality of, and performing analy-

ses of, structured occurrence nets. This is because the task of analysing and/or deriving the scenarios depicted by structured occurrence nets will, in real life, be too complex to be undertaken as a simple paper and pencil exercise. The main reason is that the systems we primarily aim at are (highly) concurrent and so automated analyses of their behaviour suffer from the so-called 'state explosion problem'. In a nutshell, even the most basic problems are then of non-polynomial complexity and so perhaps the only way to deal with them is to use highly optimised automated tools. This work could build on work in, e.g., [6, 10–12, 19], on the *unfoldings* of Petri nets introduced in [18].

Turning to the other types of application that we envisage, the utilisation of structured occurrence nets for system evaluation and synthesis is a more straightforward extension of existing research, and of existing proven tools. They could be used as a way of modelling complex system behaviour *prior* to system deployment, so as to facilitate the use of some form of automated model-checking in order to verify at least some aspects of the design of the system(s). Alternatively such automated model-checking might be used to assist analysis of the automatically-generated records of actual failures of complex systems. Such work could take good advantage of recent work on the model-checking of designs, originally expressed in the pi-calculus, work which involves the automated generation and analysis of occurrence nets [11].

System synthesis is yet another very different avenue that could be usefully explored. This would involve using structured occurrence nets which have been shown to exhibit desirable behaviour, including automated tolerance and/or diagnosis of faults, as an aid to designing systems that are guaranteed to exhibit such behaviour when deployed. We have in fact, with colleagues, already shown that it is possible to synthesise asynchronous VLSI sub-systems via the use of formal representations based on occurrence nets [12], but such designs are much less complex than those that we have had in mind while developing the concept of structured occurrence nets. The use of structured occurrence nets, in particular those consisting of a set of interacting occurrence nets, as input to an enhanced synthesiser is in effect a means of allowing the user to exercise a degree of control over the synthesis process. In effect the structuring is being used to constrain the synthesiser's search space. By such means the user could cooperate with the synthesiser so as to produce solutions to problems of a complexity that exceeds the practical limits of existing synthesis tools [12].

8 Concluding Remarks

A major aim of the present paper has been to introduce, and motivate the further study of, the concept that we term structured occurrence nets, a concept that we claim could serve as a basis for possible improved techniques of failure prevention and analysis of complex evolving systems. This is because the various types of abstractions that the concept of a structured occurrence nets make use of are all ones that we suggest could facilitate the task of understanding complex systems and their possible or actual failures, and that of the analysis of the cause(s) of such failures. These abstractions would in most cases be a natural consequence of the way the system(s) have been conceived and perceived, rather than abstractions that have to be generated after the fact, during analysis.

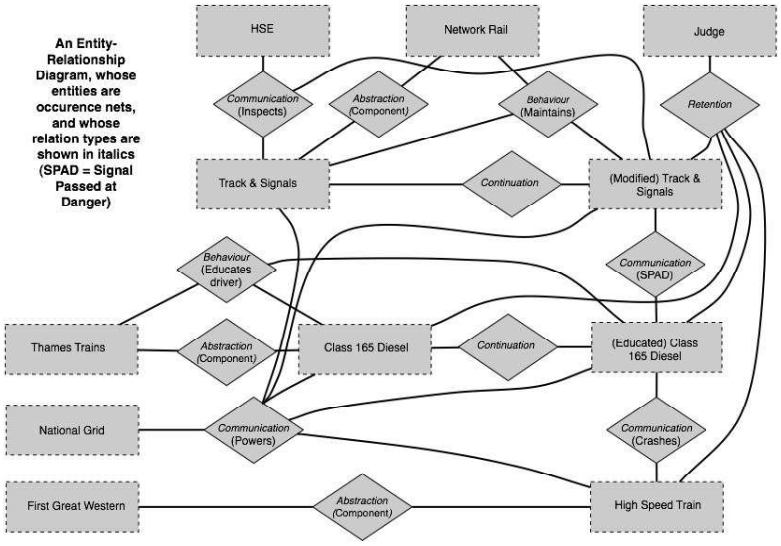

Fig. 13. Entity-Relationship diagram for the Ladbroke Grove Train crash.

In principle, a single huge conventional (i.e. unstructured) occurrence net could be used to represent the activity of a large evolving set of complex systems. However, by retaining structuring which matches the actual or perceived reality of a set of distinct systems and their largely distinct activities, the challenge of creating, understanding and validating the various systems' evolution and behaviour (and their failures) is greatly reduced. In particular, we believe it will prove possible to design automated SON analysis tools that take advantage of all the retained structure, and as a result are capable of dealing with much more complex system activities than could be handled by existing tools for analysing conventional occurrence nets. (Such tools could we hope be built as extensions of existing tools for supporting the analysis of conventional unstructured occurrence nets.)

Perhaps the most straightforward use to which structured occurrence nets could be put is for analysing fully detailed algorithmic models of a set of systems. This is because such models could be used to generate structured occurrence sets that would be known to be complete and accurate (as opposed to being in large part the fruit of speculation and guesswork, which may well be the case for forensic investigations and safety analyses). Such detailed structured nets can then be used for an enhanced version of conventional model-checking [5] in order to establish various formal properties of the set of systems, taking advantage of the structuring to enable more complex systems to be analysed than would be feasible with conventional techniques.

Of the various possible types of use of structured occurrence net that we have identified - post hoc analysis of partially-understood systems, a priori analysis of detailed models of fully specified systems, and synthesis of systems from exemplar occurrence nets - we have, ahead of the availability of any tool support, so far initiated exploration

of just the first. We have been working on a sketch of a structured representation of the various activities and mistakes which led up to the tragic Ladbroke Grove Train crash [24]. Figure 13 is an example result of this exploration - it uses the conventional Entity-Relationship graphical notation, the entities in fact being individual (un-detailed) occurrence nets, representing the existence of information about the activities of each of the trains that collided, the organisations responsible for train maintenance and inspection, the organisation that designed the signalling system, the organisation that was supposed to educate new drivers about the detailed location and operation of the signals, etc. In doing so, we made use of such SON relationships as communication, abstraction and behaviour. However, in our view tool support is needed in order for us to take such experiments on much further, and our main next priority is to explore the provision of such support.

Such a tool for recording, analysing and manipulating structured occurrence nets is best regarded as constituting a somewhat general purpose infrastructure, which would actually be used via a particular specialised application interface. The infrastructure tool would embody fairly general facilities for assessing and reporting on the completeness and consistency of a given structured occurrence net, using algorithms based on the various theorems and propositions given in earlier sections of this paper. The application interface could be the means by which for example (i) a structured occurrence net is constructed, and (ii) any additional information required to identify states and events that should be classified as erroneous is provided.

A possible example of a specialised application interface would be one supporting the performing of forensic analyses of extensive automatically-recorded audit trails of network traffic obtained from or about computers that are suspected of having been involved in cybercrime [7]. A related, but in practice very different, potential application concerns the evaluation of evidence from a major (conventional) crime. The aim of such evaluation is to formulate plausible scenarios worthy of further police investigation, an application for which a 'knowledge-based' modelling technique has been developed [9]. Somewhat similar in intent, though designed for aircraft accident analyses rather than criminal investigations, and very different technically, is the 'Why-Because' causal analysis scheme [15]. Our speculation is that at least in theory, and perhaps in practice, all these could benefit from the use of infrastructural support for structured occurrence nets.

Clearly, much remains to be done to explore how best to exploit the concept of structured occurrence nets, and to determine the adequacy of the set of relations that we have described in this paper. (We are considering further relations, in particular that of 'alternate', a relation which would be used when the need is to document and explore speculative alternative activities that might have led to some given situation, but have deferred discussion of this to a subsequent paper. Similarly, we have deferred discussion of how one might best extend the formalism developed here to deal with recursively defined structured occurrence nets, a challenging task for which the present paper provides the necessary groundwork.) However, we hope that the present account has demonstrated that our plans have a solid formal basis, a basis which can usefully be exploited through the provision of automated support and analysis tools, and has adequately explained why we believe such tool building activities are now justified.

References

1. A.Avizienis, J.-C.Laprie, B.Randell and C.Landwehr (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. on Dep. and Sec. Comp.* 1, 11–33.
2. P.Baldan, T.Chatain, S.Haar and B.König (2008). Unfolding-Based Diagnosis of Systems with an Evolving Topology. Proc. of *CONCUR'08*, LNCS 5201, 203–217.
3. E.Best and R.Devillers (1988). Sequential and Concurrent Behaviour in Petri Net Theory. *Theoretical Computer Science* 55, 87–136.
4. E.Best and B.Randell (1981). A Formal Model of Atomicity in Asynchronous Systems. *Acta Informatica* 16, 93–124.
5. J.Esparza and K.Heljanko (2008). *Unfoldings: A Partial-Order Approach to Model Checking*. Springer.
6. J.Esparza, S.Römer and W.Vogler (2002). An Improvement of McMillan's Unfolding Algorithm. *Formal Methods in System Design* 20, 285–310.
7. P.Gladyshev and A.Patel (2005). Formalising Event Time Bounding in Digital Investigations. *International Journal of Digital Evidence* 4.
8. A.W.Holt, R.M.Shapiro, H.Saint and S.Marshall (1968). Information System Theory Project. Report RADC-TR-68-305, US Air Force, Rome Air Development Center.
9. J.Keppens and B.Schafer (2006). Knowledge Based Crime Scenario Modelling. *Expert Syst. Appl.* 30, 203–222.
10. V.Khomenko and M.Koutny (2007). Verification of Bounded Petri Nets Using Integer Programming. *Formal Methods in System Design* 30, 143–176.
11. V.Khomenko, M.Koutny and A.Niaouris (2006). Applying Petri Net Unfoldings for Verification of Mobile Systems. Report CS-TR 953, Newcastle University.
12. V.Khomenko, M.Koutny and A.Yakovlev (2006). Logic Synthesis for Asynchronous Circuits Based on STG Unfoldings and Incremental SAT. *Fundamenta Informaticae* 70, 49–73.
13. H.C.M.Kleijn and M.Koutny (2004). Process Semantics of General Inhibitor Nets. *Information and Computation* 190, 18-69.
14. D.Koppad, D.Sokolov, A.Bystrov and A.Yakovlev (2006). Online Testing by Protocol Decomposition. Proc. of *IOLTS'06*, IEEE CS Press, 263–268.
15. P.B.Ladkin (2000). Causal Reasoning about Aircraft Accidents. Proc. of *SAFECOMP 2000*, LNCS 1943, 344–360.
16. S.Lenk (1994). Extended Timing Diagrams as a Specification Language. Proc. of *European Design Automation*, IEEE Computer Society Press, 28–33.
17. S.Mauw (1996). The Formalization of Message Sequence Charts. *Computer Networks and ISDN Systems* 28, 1643–1657.
18. K.L.McMillan (1995). A Technique of State Space Search Based on Unfoldings. *Formal Methods in System Design* 6, 45-65.
19. S.Melzer and S.Römer (1997). Deadlock Checking Using Net Unfoldings. Proc. of *CAV'97*, LNCS 1254, 352–363.
20. P.M.Merlin and B.Randell (1978). State Restoration in Distributed Systems. Proc. of *FTCS-8*, IEEE Computer Society Press, 129–134.
21. B.Randell and M.Koutny (2007). Failures: Their Definition, Modelling and Analysis. Proc. of *ICTAC'07*, LNCS 4711, 260–274.
22. G.Rozenberg and J.Engelfriet (1998). Elementary Net Systems. Proc. of *Advances in Petri Nets. Lectures on Petri Nets I: Basic Models*, LNCS 1491, 12–121.
23. F.J.Thayer, J.C.Herzog and J.D.Guttman (1999). Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security* 7, 191–230.
24. http://www.rail-reg.gov.uk/upload/pdf/incident-ladbrokegrove-ladbroke-optim.pdf