Failures: Their Definition, Modelling and Analysis

B. Randell and M. Koutny.

TECHNICAL REPORT SERIES

No. CS-TR-994 **December, 2006**

TECHNICAL REPORT SERIES

No. CS-TR-994 December, 2006

Failures: Their Definition, Modelling and Analysis

Brian Randell and Maciej Koutny

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 interact and evolve, or evidence that is being gathered and analyzed 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 interacting systems, (ii) between systems and their sub-systems, and (iii) involving systems that are controlling, supporting, 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.

Bibliographical details

RANDELL, B., KOUTNY, M..

Failures: Their Definition, Modelling and Analysis
[By] B. Randell, M. Koutny.

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2006.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-994)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-994

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 interact and evolve, or evidence that is being gathered and analyzed 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 interacting systems, (ii) between systems and their sub-systems, and (iii) involving systems that are controlling, supporting, 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

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.

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.

Suggested keywords

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

Failures: Their Definition, Modelling and Analysis

Brian Randell and Maciej Koutny
School of Computing Science
Newcastle University
Newcastle upon Tyne
NE1 7RU, United Kingdom

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 interact and evolve, or evidence that is being gathered and analyzed 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 interacting systems, (ii) between systems and their sub-systems, and (iii) involving systems that are controlling, supporting, 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:

- to improve our understanding — in part by formalising — of the concept of failure (and error and fault) as given by [1];
- 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
- to initiate an investigation of possible improved techniques of system failure prevention and analysis.

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 — a critical point that we will return to shortly.

A failure occurs 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':

$\ldots \rightarrow$ failure $\rightarrow$ fault $\rightarrow$ error $\rightarrow$ failure $\rightarrow$ fault $\rightarrow \ldots$

i.e.,

$\ldots \rightarrow$ event $\rightarrow$ cause $\rightarrow$ state $\rightarrow$ event $\rightarrow$ 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, in order to clarify, and exploit, such concepts as fault-error-failure chains, and the role of judgemental systems, is based on that of *occurrence nets* [2, 6, 17] and our extensions of such nets [13].

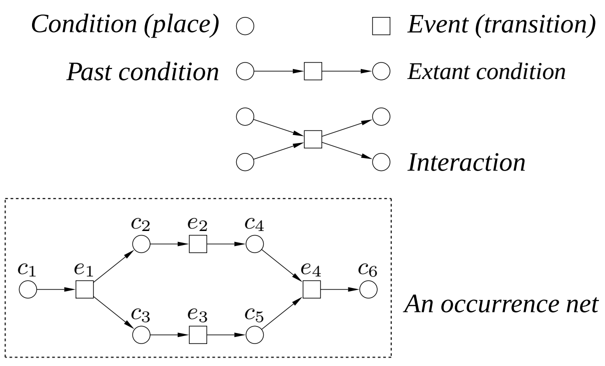


Figure 1. Basic notation.

As can be seen in Fig. 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 Fig. 1. (In the case of complex nets, these might be better represented in some linguistic or tabular form.)

We will also take advantage of our belated realization 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 occurrence nets a *structured occurrence net* (or *SON*).

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.

It 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 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 [18], signal diagrams [19] and message sequence charts [14].)

2 Occurrence nets

In this section, we present the basic model of an occurrence net, which is standard within Petri net theory [2, 6, 17]. 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[1] 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 (rather than partial) order.

Definition 2.1 (occurrence net) *An* occurrence net *is a triple $\mathcal{ON} = (C, E, F)$ where: $C \neq \varnothing$ and E are finite*[2] *disjoint sets of respectively* conditions *and* events *(collectively, conditions and events are the* nodes *of $\mathcal{ON}$); and $F \subseteq (C \times E) \cup (E \times C)$ is a* flow relation *satisfying the following:*

- *For every condition c there is at most one e such that $(e, c) \in F$, and at most one f such that $(c, f) \in F$.*
- *For every event e there is at least one c such that $(c, e) \in F$, and at least one d such that $(e, d) \in F$.*
- *$\mathcal{ON}$ 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. Clearly, not all such relationships are meaningful (both in a local and global sense) and the restrictions in the above definition are meant to exclude impossible scenarios. More precisely, the first condition means that each non-initial condition is uniquely caused, and each of the non-final conditions caused a unique event.[3] 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 condition or event x we use $pre(x)$ and $post(x)$ to denote the set of all elements 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 $X \subseteq C \cup E$, we denote $pre(X) = \bigcup_{x \in X} pre(x)$ and $post(X) = \bigcup_{x \in X} post(x)$.
- Two distinct nodes of $\mathcal{ON}$, 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 $\mathcal{ON}$ 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 $Cut^{init}_{\mathcal{ON}}$ and $Cut^{fin}_{\mathcal{ION}}$ be the sets of all conditions c such that $pre(c) = \varnothing$ and $post(c) = \varnothing$, respectively. These two sets are cuts and, intuitively, the former corresponds to the initial state of the history represented by $\mathcal{ON}$, and the latter to its final state.

For the occurrence net depicted in Fig. 1, we have $C = \{c_1, \ldots, c_6\}$ and $E = \{e_1, \ldots, e_4\}$. Moreover, $Cut^{init}_{\mathcal{ON}} = \{c_1\}$ and $Cut^{fin}_{\mathcal{ION}} = \{c_6\}$, and the other four cuts 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 2.2 (sequential execution) *A* sequential execution *of the occurrence net $\mathcal{ON}$ 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 $D_0 = Cut^{init}_{\mathcal{ON}}$ 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 $\mathcal{ON}$.*

[1]It is standard to describe an occurrence net as computation even though they can be used to portray behaviours of quite general systems, e.g., ones that include people.

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

[3]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 Fig. 1, we have that $\{c_1\}\, e_1\, \{c_2, c_3\}\, e_2\, \{c_4, c_3\}\, e_3\, \{c_4, c_5\}\, e_4\, \{c_6\}$ is one of its possible sequential executions.

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 $\mathcal{ON}$ is sound in the sense of obeying some natural temporal properties as well as testifying to the fact that $\mathcal{ON}$ 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 $\mathcal{ON}$. In practical terms, the latter means that we can verify state properties of the computations captured by $\mathcal{ON}$ by running a model checker which inspects all the cuts. Such a model checker could be based on a SAT-solver, e.g., as in [9], or integer programming, e.g., as in [7].

Theorem 2.3 ([2]) *Given a sequential execution as in Def. 2.2, each D_i is a cut of $\mathcal{ON}$, and no event occurs more than once.*

Theorem 2.4 ([2]) *Each cut of $\mathcal{ON}$ can be reached from the initial cut through some execution. Moreover, each event of $\mathcal{ON}$ is involved in at least one sequential execution.*

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

Definition 2.5 (step execution) *A* step execution *of the occurrence net $\mathcal{ON}$ is $D_0\, G_1\, D_1 \dots G_n\, D_n$, where each D_i is a set of conditions and each G_i is a set of events, such that $D_0 = Cut^{init}_{\mathcal{ON}}$ and, for every $i \leq n$, we have $pre(G_i) \subseteq D_{i-1}$ and $D_i = (D_{i-1} \setminus pre(G_i)) \cup post(G_i)$.*

For the occurrence net depicted in Fig. 1, we have that $\{c_1\}\, \{e_1\}\, \{c_2, c_3\}\, \{e_2, e_3\}\, \{c_4, c_5\}\, \{e_4\}\, \{c_6\}$ is one of its possible step executions.

For the basic model of occurrence nets, the sequential and step executions are broadly speaking equivalent; in particular, Theorems 2.3 and 2.4 hold also for step executions. However, for extended notions of occurrence nets, which we discussed in [10], sequential and step executions may, e.g., admit different sets of reachable global states.

3 Structuring occurrence nets

We now outline two simple ways of structuring occurrence nets which will be used in the subsequent sections of the paper.

The first one captures *system interaction*, i.e., a situation in which two or more systems (in other words, a *compound* system) proceed concurrently and (occasionally) interact with each other. As shown in Fig. 2, the systems — in this case two — or more precisely the actions of two systems, are represented by occurrence nets. We will follow a convention that conditions and events of different systems are recognised by shading them differently. There are some obvious rules about legal such colourings (e.g., that they partition the nodes into disjoint sets the members of each of which are connected; in this case, linearly but in general any occurrence net might be used).

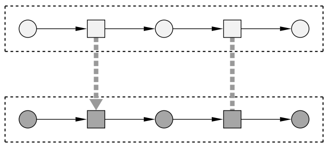


Figure 2. System interaction.

We have two types of interactions to relate events of separate systems, represented by thick dashed arcs (relation κ in the next definition) and edges (relation σ in the next definition). The former relation means that one event is a causal predecessor of another event (in other words, information flow was unidirectional), while the latter means that two events have been executed synchronously (in other words, information flow was bidirectional).

Definition 3.1 (interaction ON) *An* interaction occurrence net *is a tuple $\mathcal{ION} = (\mathcal{ON}_1, \dots, \mathcal{ON}_k, \kappa, \sigma)$, where each $\mathcal{ON}_i = (C_i, E_i, F_i)$ is an occurrence net,*[4] *and $\kappa, \sigma \subseteq \bigcup_{i \neq j} E_i \times E_j$ are two relations (σ being symmetric) such that the relation $Prec_{\mathcal{ION}} = \mathbf{F}|_{\mathbf{C}\times\mathbf{C}} \cup (\mathbf{F} \circ (\kappa \cup \sigma) \circ \mathbf{F})$ is acyclic.*
In the above, as well as later on, we denote $\mathbf{C} = \bigcup_i C_i$, $\mathbf{F} = \bigcup_i F_i$ and $\mathbf{E} = \bigcup_i E_i$.

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.

For an interaction occurrence net as in Def. 3.1, cuts and step executions need to be re-defined. A *cut* of $\mathcal{ION}$ is a maximal (w.r.t. set inclusion) set of conditions $Cut \subseteq \mathbf{C}$ such that $(Cut \times Cut) \cap Prec^{+}_{\mathcal{ION}} = \varnothing$. The initial cut of $\mathcal{ION}$, $Cut^{init}_{\mathcal{ION}}$, is the union of the initial cuts of all the $\mathcal{ON}_i$'s.

Definition 3.2 (step execution of ION) *A step execution of the interaction occurrence net $\mathcal{ION}$ is a sequence $D_0\, G_1\, D_1 \dots G_n\, D_n$, 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 $G_0 = Cut^{init}_{\mathcal{ION}}$ and, for every $i \leq n$, we have:*

[4]In this, and other similar definitions, we assume that different occurrence nets have *disjoint* sets of nodes.

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

We can re-establish the following basic behavioural characteristics of occurrence nets (proofs of these, and other new results formulated later on, are provided in [13]).

Theorem 3.3 *Given a step execution as in Def. 3.2, each* D_i *is a cut of* $\mathcal{ION}$*, and no event occurs more than once.*

Theorem 3.4 *Each cut of* $\mathcal{ION}$ *can be reached from the initial cut through some step execution. Moreover, each event of* $\mathcal{ION}$ *is involved in at least one step execution of* $\mathcal{ION}$*.*

We also have the following result which captures a consistency between the individual and interactive views of computation.

Theorem 3.5 *Given a step execution as in Def. 3.2,*

- $D_0 \cap C_m \;\; G_1 \cap E_m \;\; D_1 \cap C_m \; \ldots \; G_n \cap E_m \;\; D_n \cap C_m$ *is a step execution of* $\mathcal{ON}_m$*, for every* $m \leq k$*.*
- $e \in G_i$*,* $f \in G_j$ *and* $(e, f) \in \kappa$ *(or* $(e, f) \in \sigma$*) imply* $i \leq j$ *(resp.* $i = j$*), for all* $i, j \leq n$*.*

Note, however, that it may happen that a cut of an individual occurrence net $\mathcal{ON}_i$ may no longer be reachable through any step execution of the composite system $\mathcal{ION}$.

Structures like that shown in Fig. 2 capture interactions between different systems but give no information about the evolution of individual systems. This orthogonal view is illustrated in Fig. 3, where we have a two-level view of a system's history. The two levels are delineated by dashed boxes, whereas (as before) dotted boxes will delineate occurrence nets when there are multiple occurrence nets within a level.

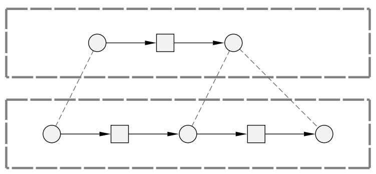


Figure 3. Simple abstraction.

A possible interpretation of Fig. 3 is that the upper level provides a high-level view of 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.

Fig. 3 also shows the 'abstracts' 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 the two-level occurrence net as it is a special case of the construct introduced later in Def. 4.3.

4 Evolutional Abstractions

As already indicated in Fig. 3, any condition can be viewed either as a state (of some system), or as a (sub)system itself that presumably has its own states and events — just which is simply a matter of viewpoint. Moreover, as indicated in Fig. 2, 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. In fact, 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 Fig. 2 and 3.

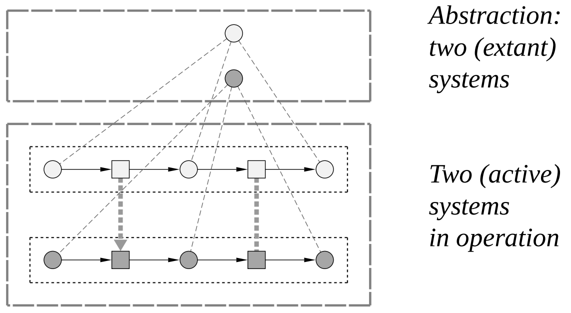
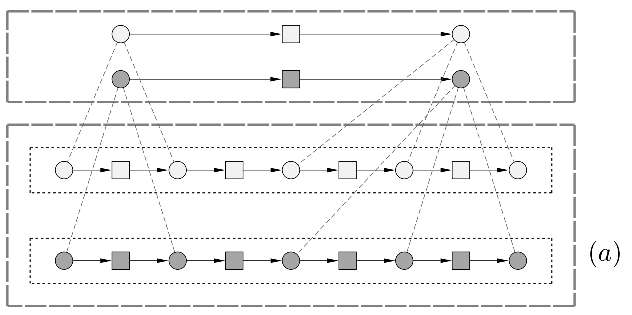


Figure 4. Behavioural abstraction.

Fig. 4 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').

More interesting is Fig. 5(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.



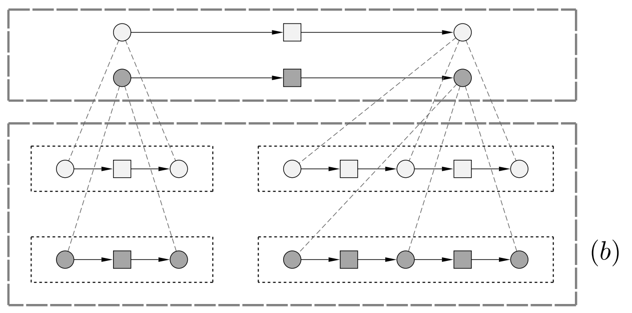


Figure 5. System modifications.

Another type of system modification is shown in Fig. 5(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 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.

The last motivating example in this section, Fig. 6, shows some of the earlier history of the two systems in Fig. 4, i.e., that one system has spawned the other system, and after that both systems went through some independent further evolutions.

We will now formalise the 'evolutional abstractions' outlined above. After an auxiliary definition, we introduce the notion of an occurrence net corresponding to a record of modification, creation, etc., of some compound system.

Definition 4.1 (interval) *Let $\mathcal{ON} = (C, E, F)$ be an occurrence net. Its* interval *is a non-empty set of conditions $int = \{c_1, \ldots, c_m\}$ such that there exist events $e_1, \ldots, e_{m-1}$ satisfying $(c_i, e_i) \in F$ and $(e_i, c_{i+1}) \in F$, for every $i \leq m - 1$.*

An interval satisfies two main properties: (i) the conditions it comprises are causally totally ordered, and (ii) there are no 'gaps' in the sequence of consecutive states it comprises. Both stem from the fact that int is meant to capture successive stages in the *evolution* of some system.

Definition 4.2 (evolutional ON) *An* evolutional *occurrence net is $\mathcal{EON} = (\mathcal{ON}, \ell)$, where $\mathcal{ON} = (C, E, F)$ is an occurrence net and $\ell : C \to \{1, \ldots, N\}$ ($N \geq 1$). Moreover, the inverse image $\ell^{-1}(i) = \{c \in C \mid \ell(c) = i\}$ is an interval, for every $i \in \mathcal{SYS}$.*

The next definition combines together the above ideas about structuring behaviours of interacting systems.

Definition 4.3 (evolutional SON) *An* evolutional structured occurrence net *is a tuple $\mathcal{ESON}{=}(\mathcal{EON}, \mathcal{ION}, \alpha)$, where $\mathcal{EON}$ and $\mathcal{ION}$ are as in Def. 4.2 and 3.1, respectively, and $\alpha : \mathbf{C} \to C$ is a mapping such that:*

- *$\ell(\alpha(\mathbf{C})) = \mathcal{SYS}$;*
- *$\alpha(C_i) \cap \alpha(C_j) \neq \varnothing$ implies $i = j$, for all $i, j \leq k$;*
- *$\alpha(C_i)$ is an interval and $|\ell(\alpha(C_i))|{=}1$, for all $i \leq k$;*
- *for every $i \leq k$ and every condition $c \in C$ with $\alpha^{-1}(c) \subseteq C_i$, the sets Min_c and Max_c of, respectively, all minimal and maximal elements of $\alpha^{-1}(c)$ w.r.t. the flow relation F_i are cuts of $\mathcal{ON}_i$;*
- *for every $i \leq N$ and all conditions $b, c, d \in \mathbf{C}$ such that $\ell(\alpha(b)){=}\ell(\alpha(d)){=}i$, if $(\alpha(b), \alpha(c)) \in F^+$ and $(\alpha(c), \alpha(d)) \in F^+$, then we have $\ell(\alpha(c)){=}i$;*
- *$Prec_{\mathcal{ESON}} = Prec_{\mathcal{ION}} \cup Prec$ is an acyclic relation, where $Prec$ is the union of sets $Max_c \times Min_d$, for all $e \in E$ and $(c, d) \in pre(e){\times}post(e)$.*

Intuitively, $Prec_{\mathcal{ION}}$ captures causalities resulting from intra-level interactions between behaviours, whereas $Prec$ reflects the succession of evolutions the system had undergone during the history captured by $\mathcal{ESON}$.

We now introduce cuts and step executions for the evolutional structured occurrence net in Def. 4.3.

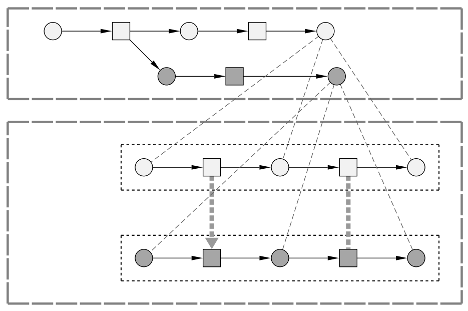


Figure 6. System creation.

A *cut* of $\mathcal{ESON}$ is a maximal (w.r.t. set inclusion) set of conditions $Cut \subseteq C \cup \mathbf{C}$ such that $(Cut \times Cut) \cap (Prec^{+}_{\mathcal{ESON}} \cup F^{+}) = \varnothing$ and, moreover, $\alpha(Cut \cap \mathbf{C}) = Cut \cap C$. (Taking into account F^{+} means that only a single version of a system can be active at any time.)
The initial cut of $\mathcal{ESON}$ is the union, $Cut^{init}_{\mathcal{ESON}}$, of the initial cut of $\mathcal{ON}$ and the initial cuts of all the $\mathcal{ON}_i$'s such that $\alpha(Cut^{init}_{\mathcal{ON}_i}) \cap Cut^{init}_{\mathcal{ON}} \neq \varnothing$.

Definition 4.4 (step execution of SON) *A step execution of the evolutional structured occurrence net $\mathcal{ESON}$ is a sequence $D_0\, G_1\, D_1 \ldots G_n\, D_n$, where each $D_i \subseteq C \cup \mathbf{C}$ is a set of conditions and each $G_i \subseteq E \cup \mathbf{E}$ is a set of events, such that $G_0 = Cut^{init}_{\mathcal{ESON}}$ and, for every $i \leq n$, we have the following (below $Min = \bigcup_{c \in post(E \cap G_i)} Min_c$ and $Max = \bigcup_{c \in pre(E \cap G_i)} Max_c$):*

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

As before, the notion of a structured occurrence net has behavioural properties similar to those satisfied by standard occurrence nets.

Theorem 4.5 *Given a step execution as in Def. 4.4, each D_i is a cut of $\mathcal{ESON}$, and no event occurs more than once.*

Theorem 4.6 *Each cut of $\mathcal{ESON}$ can be reached from the initial cut through some step execution. Moreover, each event of $\mathcal{ESON}$ is involved in at least one step execution of $\mathcal{ESON}$.*

We next establish a consistency between the individual and interactive views of computation, intertwined with the record of evolutions of the systems involved.

Theorem 4.7 *Given a step execution as in Def. 4.4, for every $m \leq k$, we have:*

$$D_0 \cap C_m \;\; G_1 \cap E_m \;\; D_1 \cap C_m \; \ldots \; G_n \cap E_m \;\; D_n \cap C_m$$

is either a sequence of empty steps, or a step execution of the occurrence net $\mathcal{ON}_m$ 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 $\mathcal{ON}_m$, and in the latter the final one). Moreover, $D_0 \cap C \;\; G_1 \cap E \;\; D_1 \cap C \; \ldots \; G_n \cap E \;\; D_n \cap C$ is a step execution of the occurrence net $\mathcal{ON}$.

A result similar to the second part of Theorem 3.5 also holds. The results obtained here form a starting point for a systematic development of model checking techniques based on $\mathcal{ESON}$. For example, one can attempt to verify state based properties of evolving systems by inspecting all cuts of $\mathcal{ESON}$.

5 Spatial and Temporal Abstractions

Another type of abstraction, that we will call *composition abstraction*, is based on the relation 'contains / is component of'. Fig. 7 shows the behaviour of a system and of its three component systems, and how its behaviour is related to that of its components. (This figure does not represent the matter of *how*, or indeed whether, the component systems are enabled to interact, i.e., what design is used, or what connectors are involved.) Having identified such a set of interacting systems, and hence the *containing* system which they make up, then each member of this set has the other members as its *environment*.

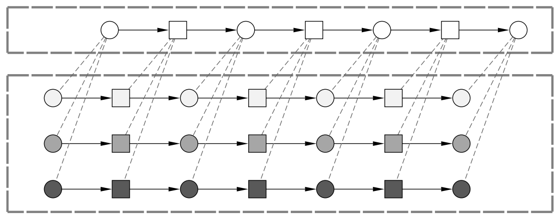


Figure 7. System composition.

Definition 5.1 (spatial abstraction SON) *A* spatial abstraction structured occurrence net *is a tuple $\mathcal{SASON} = (\mathcal{ON}, \mathcal{ION}, \vartheta, \epsilon)$, where $\mathcal{ON}$ and $\mathcal{ION}$ are as in Def. 2.1 and 3.1, $\vartheta : C \to 2^{\mathbf{C}}$ and $\epsilon : \mathbf{E} \to E$, and, moreover, the following hold (below $\vartheta(H) = \bigcup_{c \in H} \vartheta(c)$, for every $H \subseteq C$):*

- *$\vartheta(C) = \mathbf{C}$ and $\epsilon(\mathbf{E}) = E$;*
- *if Cut is a cut of $\mathcal{ON}$ and $c, d \in Cut$, then $\vartheta(Cut)$ is a cut of $\mathcal{ION}$ and $\vartheta(c) \cap \vartheta(d) = \varnothing$;*
- *for every event $e \in \mathbf{E}$, $pre(e) \subseteq \vartheta(pre(\epsilon(e)))$ and $post(e) \subseteq \vartheta(post(\epsilon(e)))$;*
- *$Prec_{\mathcal{SASON}} = Prec_{\mathcal{ION}} \cup Prec'$ is an acyclic relation, where $Prec'$ is the union of relations $(\vartheta(pre(e)) \setminus \vartheta(post(e))) \times \epsilon^{-1}(e)$ and $\epsilon^{-1}(e) \times (\vartheta(post(e)) \setminus \vartheta(pre(e)))$, for all $e \in E$.*

One can define the cuts and step executions for $\mathcal{SASON}$ similarly as it has been done in for $\mathcal{ESON}$, and then obtain results similar in essence and applicability to those formulated for $\mathcal{ESON}$.

The above is in effect a *spatial* abstraction — one can also have a *temporal* abstraction, through the 'abbreviation' relation, i.e., an *abbreviation abstraction*, as shown in Fig. 8(a).

When one '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

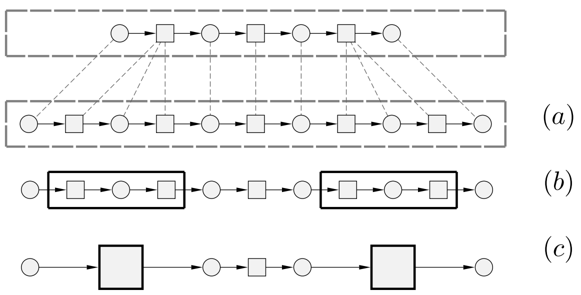


Figure 8. System abbreviation.

one to make such abbreviations are non-trivial when multiple concurrent activities are shown in the net. These 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'.

Fig. 8(b) shows this alternative representation of Fig. 8(a), the top part of which can readily be recreated by 'collapsing' Fig. 8(b)'s occurrence net, i.e., by replacing the enclosed sections by simple event symbols, as shown in Fig. 8(b).

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 [3] and is illustrated in Fig. 9.

After an auxiliary definition, we introduce the notion of a structured occurrence net corresponding to a temporal abstraction of some compound system.

Definition 5.2 (block) *A* block *of an occurrence net $\mathcal{ON} = (C, E, F)$ is a non-empty set $Bl \subset C \cup E$ of nodes where both the maximal and minimal (w.r.t. the flow relation F) elements are events, and for all nodes $x, y \in Bl$, $(x, z) \in F^+$ and $(z, y) \in F^+$ imply $z \in Bl$.*

Thus in a block there are no 'gaps' between the nodes it comprises, and it starts and ends in a set of events. (Note that a single event is a block.)

Definition 5.3 (temporal abstraction SON) *A* temporal abstraction structured occurrence net *is $\mathcal{TASON} = (\mathcal{ION}, \mathcal{ION}', \xi)$ where $\mathcal{ION}$ is as in Def. 3.1, $\mathcal{ION}' = (\mathcal{ON}'_1, \ldots, \mathcal{ON}'_k, \kappa', \sigma')$ is an interaction occurrence net with $\mathcal{ON}'_i = (C'_i, E'_i, F'_i)$ (for $i \leq k$), and $\xi : \mathbf{C}' \cup \mathbf{E}' \to \mathbf{C} \cup \mathbf{E}$; and, moreover, the following are satisfied, for every $i \leq k$ (below $\mathbf{C}' = \bigcup_i C'_i$, $\mathbf{F}' = \bigcup_i F'_i$ and $\mathbf{E}' = \bigcup_i E'_i$):*

- *$\xi(C'_i \cup E'_i) = C_i \cup E_i$, $\xi^{-1}(C_i) \subseteq C'_i$ and $\xi(E'_i) = E_i$;*

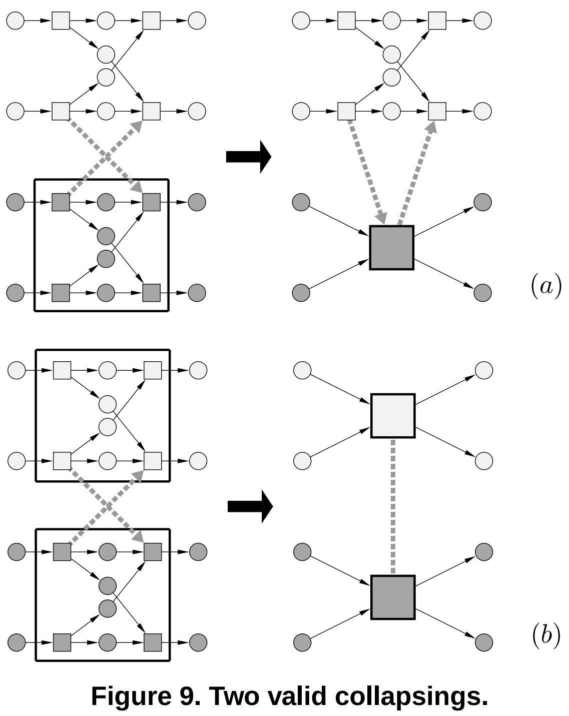


Figure 9. Two valid collapsings.

- *$\xi^{-1}(e)$ is a block of $\mathcal{ON}'_i$, for every $e \in E_i$;*
- *$|\xi^{-1}(c)| = 1$, for every $c \in C_i$;*
- *$F_i = \{(x, y) \mid (\xi^{-1}(x) \times \xi^{-1}(y)) \cap F'_i \neq \varnothing\}$;*
- *$\kappa = \{(e, f) \mid (\xi^{-1}(e) \times \xi^{-1}(f)) \cap \kappa' \neq \varnothing\}$; and*
- *$\sigma = \{(e, f) \mid (\xi^{-1}(e) \times \xi^{-1}(f)) \cap \sigma' \neq \varnothing\} \cup \{(e, f) \mid (((\xi^{-1}(e) \times \xi^{-1}(f)) \cap \kappa' \neq \varnothing) \wedge ((\xi^{-1}(f) \times \xi^{-1}(e)) \cap \kappa' \neq \varnothing))\}$.*

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 5.4 *Let $\mathcal{TASON}$ be a temporal abstraction structured occurrence net as in Def. 5.3, and $D_0 \{e_1\} D_1 \ldots \{e_n\} D_n$ be a step execution of $\mathcal{ION}$. 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 firing sequence of $\mathcal{ON}'_i$ involving exactly the events of $\xi^{-1}(f_j)$ starting from $pre(f_j)$ (which is possible). Then $e_{11} \ldots e_{1m_1} \ldots e_{n1} \ldots e_{qm_q}$ is a firing sequence of $\mathcal{ON}'_i$.*

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 often be both spatial *and* temporal.

6 Dependability

To allow for the possibility of failure a system might, e.g., make use of 'recovery points'. Such recovery points can be recorded in states that take no further (direct) part in the system's ongoing (normal) behavior, as shown below.

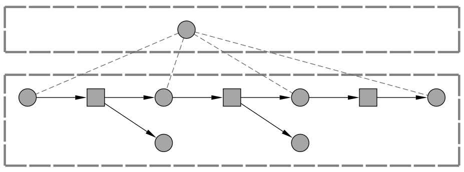


Figure 10. State retention.

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 facility [12]; another example, in a very different world, is a football referee! Alternatively the judging system may also be deployed after the fact using an occurrence net that represents how the failing event occurred. Such an occurrence net is also something that can for example be recorded in a retained state, e.g., that of the judgment system. Fig. 11 is an attempt to portray this. It deliberately portrays a situation in which a judgement system has obtained only incomplete evidence of the systems' states and events and even the causal relationships between conditions and events.

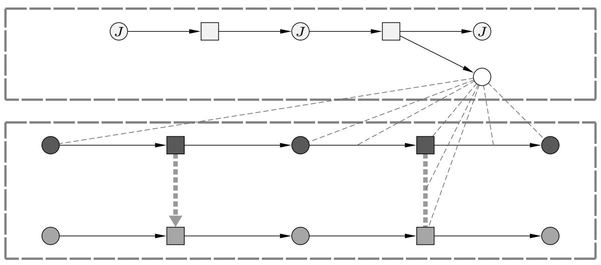


Figure 11. Post-hoc judgement involving a judgemental system (top) and an active system (bottom).

In practice, judgement is likely to involve consideration not just of what (allegedly) happened but also what could have and should have happened, perhaps based on a system design or specification. An extended occurrence net notation that is used to represent such matters is the 'barb' (introduced in [11]), namely an event that could have occurred, given the condition(s) that existed, but which did not — see Fig. 12 below, where the barbs are represented by a distinctive kind of boxes.

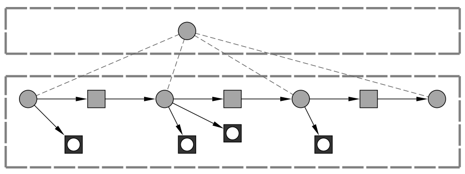


Figure 12. What did not occur.

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) that has been built from it, or (iv) from a given system to the system(s) that created or modified it, or to the system(s) that should have allowed it to continue to exist. All this tracing activity can 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* [16]. (The name reflects the fact that error diagnosis and reporting messages have to chase after the ever-spreading errors.)

7 Utilising Structured Occurrence Nets

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. Two simplistic examples of this are: (i) the recognition that a given system's behaviour had, after a period of correct operation, started to exhibit a succession of faults, 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, and (ii) evidence of the non-occurrence of an expected event might be found to be due to a failure of an infrastructural system, such as a power supply. (Due to page limit we omitted here a discussion, included in [13], of an abstraction relation which can be used to model the relationship between hardware and the software processes that are running on it, and indeed between the electricity source and the hardware that it powers.)

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 often currently 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 analyses 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 pencil exercise. The main reason is that the systems we primarily aim at are (highly) concurrent and so their behaviour suffers 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 earlier work at Newcastle [7, 9] on the *unfoldings* of Petri nets introduced in [15], and also benefit, e.g., from recent work at Rennes [4] on the diagnosis of executions of concurrent systems.

A quite different use of such sets of related occurrence nets might in fact prove feasible. This would be to use them 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 records of actual failures of complex systems. Such work could take good advantage of recent work at Newcastle on the model-checking of designs, originally expressed in the pi-calculus, work which involves the automated generation and analysis of occurrence nets [8]. For the integration of different formalisms, solvers and quantitative tools one could follow an approach adopted in modelling tools like Möbius [5].

There is in principle yet another avenue that could be explored, namely that of 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 synthesize asynchronous VLSI sub-systems via the use of formal representations based on occurrence nets [9], but such designs are much less complex than those that we have had in mind while developing the concept of structured occurrence nets.

8 Concluding Remarks

A major aim of the present paper has been to introduce, and motivate the 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 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, or rather systems, have been conceived and perceived, rather than abstractions that have to be generated after the fact, during analysis. As such they can in fact be regarded as providing a somewhat sophisticated means of naturally structuring what would otherwise be an impossibly large and complex occurrence net. Alternatively, they can be regarded as a way of reducing the combinatorial complexity of the information that is accumulated and the analyses that are performed on this information in following fault-error-failure chains after the fact, in complex systems containing not just hardware and software, but perhaps also human 'components'. In either case, however, we envisage the necessity of providing computer assistance, something we plan to do by building on existing relevant work at Newcastle and elsewhere.

A second aim of the paper has therefore been that of providing the formalizations of the various types of abstraction used in structured occurrence nets that are needed as a starting point for the task of implementing, or enhancing existing, analysis tools. The formal definitions and theorems that form a major part of the present paper provide a necessary basis for the development of effective software for such purposes. To this end an extended version of the present paper that completes this formalization and includes our proofs of relevant results, and goes into details of algorithms that might be used, is in preparation [13].

Acknowledgements We would like to thank Cliff Jones and Alex Yakovlev for their helpful comments. This research was supported by the EC IST grant 511599 (RODIN).

References

[1] A.Avizienis, J.-C.Laprie, B.Randell and C.Landwehr (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing* 1, 11–33.

[2] E.Best and R.Devillers (1988). Sequential and Concurrent Behaviour in Petri Net Theory. *Theoretical Computer Science* 55, 87–136.

[3] E.Best and B.Randell (1981). A Formal Model of Atomicity in Asynchronous Systems. *Acta Informatica* 16, 93–124.

[4] T.Chatain and C.Jard (2004). Symbolic Diagnosis of Partially Observable Concurrent Systems. Proc. of *FORTE'04*, Springer-Verlag, Lecture Notes in Computer Science 3235, 326–342.

[5] G.Clark, T.Courtney, D.Daly, D.Deavours, S.Derisavi, J.M.Doyle, W.H.Sanders and P.Webster (2001). The Möbius Modeling Tool. Proc. of *PNPM'01*, IEEE Computer Society, 241–250.

[6] 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.

[7] V.Khomenko and M.Koutny (2007) Verification of Bounded Petri Nets Using Integer Programming. Formal Methods in System Design. To appear.

[8] V.Khomenko, M.Koutny and A.Niaouris (2006). Applying Petri Net Unfoldings for Verification of Mobile Systems. Report CS-TR 953, Newcastle University.

[9] 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.

[10] H.C.M.Kleijn and M.Koutny (2004). Process Semantics of General Inhibitor Nets. *Information and Computation* 190, 18-69.

[11] J.Kleijn, M.Koutny and G.Rozenberg (2006). Towards a Petri Net Semantics for Membrane Systems. Proc. of *WMC'05*, Springer-Verlag, Lecture Notes in Computer Science 3850, 292–309.

[12] D.Koppad, D.Sokolov, A.Bystrov and A.Yakovlev (2006). Online Testing by Protocol Decomposition. Proc. of *IOLTS'06*, IEEE CS Press, 263–268.

[13] M.Koutny and B.Randell (2006). Understanding Failures. In preparation.

[14] S.Mauw (1996). The Formalization of Message Sequence Charts. *Computer Networks and ISDN Systems* 28, 1643–1657.

[15] K.L.McMillan (1995). A Technique of State Space Search Based on Unfoldings. *Formal Methods in System Design* 6, 45-65.

[16] P.M.Merlin and B.Randell (1978). State Restoration in Distributed Systems. Proc. of *FTCS-8*, IEEE Computer Society Press, 129–134.

[17] G.Rozenberg and J.Engelfriet (1998). Elementary Net Systems. In: *Advances in Petri Nets. Lectures on Petri Nets I: Basic Models*, W.Reisig and G.Rozenberg (Eds.). Springer-Verlag, Lecture Notes in Computer Science 1491. 12–121.

[18] F.J.Thayer, J.C.Herzog and J.D.Guttman (1999). Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security* 7, 191–230.

[19] S.Lenk (1994). Extended Timing Diagrams as a Specification Language. Proc. of *European Design Automation*, IEEE Computer Society Press, 28–33.