Incremental Construction of Structured Occurrence Nets

Brian Randell
Incremental Construction of Structured Occurrence Nets

B. Randell

Abstract

This note discusses strategies, based on the use of Occurrence Nets (ONs) and Structured Occurrence Nets (SONs), for constructing representations of the actual or presumed behaviour (the "activity") of a complex evolving system (of systems) in such a way that these representations remain manageable and understandable, despite the complexity of the activities they portray. This note is couched in terms of making incremental additions to ON and SON diagrams, and hence would seem to be aimed directly at issues of the design and use of a graphical interface to a SON workbench. However, it is in fact intended to be of at least equal relevance to non-graphical interfaces, in fact perhaps ones serving as APIs for use by specialist software applications that make little or no use of a graphical interface.

For convenience, and because of the plans of the upcoming UNCOVER Research Project, many of the illustrative examples relate to the use of a criminal investigation system such as might be employed by the police to record and manage a growing collection of information and evidence regarding some large scale criminal activity. However, the issues discussed are intended to be of relevance to a variety of systems and application domains.
Abstract

This note discusses strategies, based on the use of Occurrence Nets (ONs) and Structured Occurrence Nets (SONs), for constructing representations of the actual or presumed behaviour (the "activity") of a complex evolving system (of systems) in such a way that these representations remain manageable and understandable, despite the complexity of the activities they portray. This note is couched in terms of making incremental additions to ON and SON diagrams, and hence would seem to be aimed directly at issues of the design and use of a graphical interface to a SON workbench. However, it is in fact intended to be of at least equal relevance to non-graphical interfaces, in fact perhaps ones serving as APIs for use by specialist software applications that make little or no use of a graphical interface.

For convenience, and because of the plans of the upcoming UNCOVER Research Project, many of the illustrative examples relate to the use of a criminal investigation system such as might be employed by the police to record and manage a growing collection of information and evidence regarding some large scale criminal activity. However, the issues discussed are intended to be of relevance to a variety of systems and application domains.

About the authors

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

OCCURRENCE NETS
ABSTRACTION
CONSTRUCTION
GRAPHICAL INTERFACE
API
CRIME INVESTIGATION SUPPORT
Incremental Construction of Structured Occurrence Nets

Brian Randell

Abstract: This note discusses strategies, based on the use of Occurrence Nets (ONs) and Structured Occurrence Nets (SONs), for constructing representations of the actual or presumed behaviour (the “activity”) of a complex evolving system (of systems) in such a way that these representations remain manageable and understandable, despite the complexity of the activities they portray. This note is couched in terms of making incremental additions to ON and SON diagrams, and hence would seem to be aimed directly at issues of the design and use of a graphical interface to a SON workbench. However, it is in fact intended to be of at least equal relevance to non-graphical interfaces, in fact perhaps ones serving as APIs for use by specialist software applications that make little or no use of a graphical interface.

For convenience, and because of the plans of the upcoming UNCOVER Research Project, many of the illustrative examples relate to the use of a criminal investigation system such as might be employed by the police to record and manage a growing collection of information and evidence regarding some large scale criminal activity. However, the issues discussed are intended to be of relevance to a variety of systems and application domains.

Keywords: Occurrence Nets, Abstraction, Construction, Graphical interface, A.P.I., Crime investigation support

1. Introduction

The concept of a ‘structured occurrence net’ (SON), which as its name implies is based on that of an ‘occurrence net’, was introduced in [1] and described in more detail in [2]. Occurrence Nets (ONs) are directed acyclic graphs that represent causality and concurrency information concerning a single execution of a system (of systems). SONs consist of multiple ONs associated together by means of various types of formal relationship, and provide a means of representing the behaviour of a complex evolving system. Some uses of SONs, e.g. for system verification and system synthesis, involve recording and utilising information about the intended behaviour of a system. In other cases, the aim is to analyze the past behaviour of an actual system. This note concerns how to use SONs to support information collection for such analyses.

The tasks of analysing a computing system failure, investigating a crime, or establishing the provenance of the information in some document of interest [6], could all involve the incremental construction of Occurrence Nets (ONs) and Structured Occurrence Nets (SONs), as information is obtained or assumptions are made, about the behaviours that occurred, so that these nets can then be “completed” and analysed. (In contrast, with some of the other envisaged uses of SONs, e.g. for system verification, there may be little or no need for such incremental development.)
One can imagine a number of different scenarios involving quite different ways of approaching the incremental construction of a SON, depending on what sort of information and infrastructure are available.

1. When there is prior availability of a sophisticated investigation support system, e.g. based on a relational database management system, ONs and SONs are probably best regarded as additional forms of linking of the entries in such a database, i.e., they are best implemented by adding further relations to the database system.

2. In other situations, lacking such an investigation support system or the like, the starting point might be the creation of an ON or SON implemented on an infrastructure based on Workcraft’s interpreted graph model, with other information being added to the causality information expressed in the ONs and SONs by annotating the graph’s conditions, events and arcs. (The UNCOVER Project plans to produce such a prototype infrastructure.)

3. Other issues concern timing and identity information. One might have information about lots of timed events, involving identified systems. Or there may be very little timing information, and/or considerable uncertainty as to the identity of the systems involved.

4. A further difference is that the individual systems whose behaviours are to be represented may themselves be highly asynchronous (e.g. electronic sub-assemblies), or may just have simple linear behaviour (e.g. individual criminals).

These various different scenarios are likely to require markedly different sets of SON construction facilities in practice, but they should all have a common theoretical basis, the (formal) basis of which is the subject of this (informal) note.

For convenience this note is couched in terms of drawing ON and SON diagrams, and hence would seem to be aimed directly at issues of the design of a graphical interface to a SON workbench. However, it is in fact intended to be of at least equal relevance to non-graphical interfaces, in fact perhaps ones serving as APIs for use by specialist software applications that make little or no use of a graphical interface. (With a graphical interface particular elements could be identified simply by pointing, though an element labelling scheme – which would be needed for an API – could be used instead of or additionally to pointing.)

2. Occurrence Nets

First some definitions:

- **A fragment** is a connected acyclic graph whose nodes are *conditions* and *events*, and whose directed *arcs* express causality. There must be at least one condition. Ignoring, for the moment, the possibility of alternative behaviours, each condition can be followed or preceded by at most one event; each event must be preceded and followed by one or more conditions. Thus the end
points (termed sources and sinks in graph theory) of an ON fragment must be conditions; they cannot be arcs or events.

- Conditions, events and arcs (and also links – see Section 3 – and relations – see Section 4 et seq) are collectively termed elements.

- An ON is a delineated set of one or more fragments. Graphically a delineation is denoted using an enclosing “dotted line” box. In terms of an API, delineation involves recording the set of graph elements concerned and providing a means of identifying and referring to this set. The directed graph making up an ON is not necessarily connected, but all its end points must be conditions.

- A set of related ONs will constitute a SON – such a SON is itself delineated using an enclosing box (labelled with the type of relation: communication, representational, or spatial, temporal, or behavioural, abstraction) and then can be regarded and used as an ON. In other words our intention is that SONs should provide a recursive form of structuring of the behaviours of a complex evolving system (CES).

![Diagram of basic notation](image)

**Fig. 1: Basic Notation**

### 3. Syntax-driven ON construction

Ideally, syntax-directed ON construction would involve a series of additions to and deletions from an ON, after each of which it was guaranteed that the result was again syntactically valid. (For convenience various higher-level “modify” operations could also be provided, involving multiple addition and deletion operations – a point that will not be pursued further here.)

Checking the validity of an ON involves confirming its acyclicity. Although in practice such checks might be deferred, this note will assume that they are carried out immediately after every operation that could create a cycle.
In this note the first types of incremental construction concern ONs. It then goes on to discuss Communication SONS – leaving the various forms of abstraction, and hence other types of SON, till afterwards.

The most basic ON is a single condition, with no incoming or outgoing arcs. (Such a condition is classed as both an initial and a final condition.) Since each condition can have at most one incoming and one outgoing arc, nothing can be added to an internal condition (i.e. a condition which is not an endpoint), because such conditions must already have both an incoming and an outgoing arc. And nothing can ever be added to an arc, because all arcs each join a single condition to a single event.

These various rules are illustrated and summarised in Figure 2. The three addition rules also imply the set of validity-preserving deletions from an ON. (Note that, in line with the fact that Link operations can only add a further arc to an event’s one or more pre-existing arcs, undoing a “Link” operation is valid only if it does not remove a solitary arc between a condition and an event, and thus leave the event as an endpoint.) The “Merge” rule is for use when it is belatedly realised that a final condition of one fragment is in fact also the initial condition of another fragment – it turns these fragments into a single fragment without introducing any additional events. The reverse operation, “Separate”, could in fact be applied to any condition, not just conditions that are the result of previous “Merge” operations. (These various operations all involve identification of the elements concerned – this can be done either by pointing or by using element labels.)

On grounds of practicality, it is presumed that the system that supports the use of these operations, and which maintains the resulting ON, is appropriately robust. Thus, it should provide means of protecting the ON information from loss or corruption, e.g. resulting from hardware failures, or from multiple clashing updates.

---

**Fig. 2: Operations that preserve the validity of an ON, with their “reverses”**.
Apart from using the possibilities listed in this figure, detail can be added to an ON only by replacing a Condition or an Event by a fragment or a block, respectively. (Blocks are defined in Section 4.) However rather than actually add the detail one can instead leave the Event or Condition in place and treat it as an abstraction whose details are provided elsewhere. Thus this is in effect an example of part of a top down approach to the construction of SONs.

Note that operations such as Join, Link, and Merge, which involve, or might involve, two fragments can apply to fragments that are in separate ONs, and so involve creating a single ON out of two separate ONs.

Things are fairly simple when we limit ourselves to considering ON fragments for systems that are not perceived as carrying out any concurrent behaviours, e.g. an ON corresponding to a single criminal suspect, which represents the sequence of occasions on which this suspect was sighted. This would give rise to a simple linear ON, i.e. one in which each event is preceded or followed by a single condition.

4. Communication SONs

One can envisage the process of incrementally constructing a complex overall SON as being likely to start with the definition of a set of ONs representing the behaviours of a set of systems. When it is determined or presumed that there have been interactions between systems, asynchronous or synchronous communication links can be made between events in the different ONs so as to portray these interactions. (Such links are in fact a special form of relation.) A delineated set of ONs so linked would constitute a Communication SON. The critical rule that has to be followed, and which can be enforced by a graphical interface or an API, is that cycles must not be introduced by this linking, something that could only result from added asynchronous communications links. (Separate ONs have no arcs linking them, and synchronous links pose no problems, because of being undirected, though any synchronous link is in effect equivalent to a pair of asynchronous ones.)

If however it is belatedly realised that a given ON is in fact an attempted portrayal not just of a single system but of the behaviours of several systems that need to be distinguished from each other, then it will be necessary to split the ON into a set of distinct ONs. (If the ON is not already a set of distinct fragments, then one or more “split” operations will first be needed.) Then any necessary communication links can be created between these separated ONs so that the set constitutes a Communication SON, portraying the combined behaviour of the several systems, e.g. of the individual members of a criminal gang.

5. Temporal and Spatial Abstractions

Over and above the operations listed in Figure 2 (and their reverse equivalents, for removing bits of the graph), and the addition/deletion of communication links, there is a need for operations to do with the creation, removal, and exploitation of the various kinds of abstractions. First let us summarize the concepts of temporal abstraction and of spatial abstraction.
**Temporal abstraction** allows a **block** (a part of an ON that starts and ends in events, in which all ending events are causally linked to all starting events) to be "replaced" by, i.e. abstracted to, a single abstract event that inherits all the incoming and outgoing arcs of the block.

**Spatial abstraction** allows two or more ONs whose nodes (conditions and events, with their arcs) are in 1:many correspondence to be "replaced" by, i.e. abstracted to, a **single** abstract ON with a corresponding set of nodes.

Figure 3 shows an ON portraying the “behaviour” of a single variable A. In this figure events are labelled with the names of the operations they perform on the value held in variable A, and conditions with the current value of the variable – so that the abstract event in the upper ON is labelled with the composite effect that it has on A.

![Figure 3: Temporal abstraction on data operations and values](image)

Figure 4 shows an ON, within which a block involving some of the ON’s asynchronous behaviour is to be replaced by its temporal abstraction, the result of which is shown in the upper ON.
The use of temporal abstraction within a single ON could result in an explicit causal cycle, which could be resolved by splitting the ON into two separate ONs and replacing the arcs that would cause such a cycle by a synchronous communication between the separate ONs – as shown in Figure 5.

Figure 6 shows what happens when spatial abstraction is applied to a set of three ONs, each of which is recording the behaviour of a different data variable as it undergoes various changes, the current value of the variable being shown in the relevant conditions. In the resulting spatially-abstracted ON, each abstracted event has been labelled to show the three operations that occur at each abstract event, and each abstract condition shows the current state of the set of three variables.
There are two ways in such temporal and spatial abstractions can be used. The comparatively simple use is “top-down”, which has the effect of adding detail to an existing SON. The more complicated use is “bottom-up”, the “belated” introduction of temporal and abstract abstractions so as to “structure”, i.e. to reduce the complexity of, an existing SON. This involves delineation of the block to be temporally abstracted (i.e. of the block’s entry and exit events), or the set of matching ONs to be spatially abstracted (i.e. of the set’s entry and exit conditions).

6. Behavioural Abstractions

The systems that have been identified as being “behind” the various behaviours of interest (e.g. of various alleged criminals) portrayed by particular ONs need to be related to the respective behaviours. This involves establishing Behavioural SONs, each relating a condition in one ON to the ON that provides details of its behaviour.

Figure 7 shows a very simple example of behavioural abstraction in a data provenance SON. A slightly more complicated example, portraying the behaviour of two distinct criminals (a burglar and the fence that he contacts about his ill-gotten
goods) and their interaction is shown in Figure 8. The isolated condition representing the burglar, say, could of course be part of some further occurrence net. For example, it could be part of one recording his evolving status, e.g. from junior offender, to convicted criminal, to prisoner, to being on probation.

![Diagram of criminal behavior](image)

**Figure 8: The behaviour of a pair of criminals**

Figures 7 and 8 in fact illustrate that we have, in behavioural abstraction, a very simple way of expanding from or contracting back to a single condition, in effect by choosing to portray, or to conceal, a system’s activity. This matches the means of expanding from or contracting back to a single event, portrayed in Figure 3, illustrating temporal abstraction.

### 7. Representational Structuring

Another form of structuring, which is here termed **Representational Structuring**, is available for portraying the relations between (i) activities that are involved in constructing and using representations of other activities, and (ii) these represented activities.

Retention/deletion relations are two of a number of relations that create such Representational Structurings. The retention relation was originally introduced [1] as the means by which information about a given activity (specifically about one or more of the “elements”, i.e. events, conditions, arcs and relations of the SON representing this activity) can be acquired by an event and held within a condition, or more likely a whole sequence of conditions, of some other SON. (The discard relation is the means by which such retained information can be removed.)

The motivation was to enable modelling of system activities that provide means of fault tolerance, such as back-up and recovery, to another system. An example is portrayed in Figure 9, in which ‘r’ and ‘d’ labels identify recovery point retention and discard relations, respectively. The conditions are, for identification purposes, numbered sequentially in the ON representing the process that is being protected through the taking of recovery points; the Checkpointing ON shows which of these
recovery points are being held on record, at each stage, and how these recovery points relate to each other.

Figure 9: The Retention/Deletion Activity

Figure 10 is an alternative portrayal, indicating what representations of the checkpointed process are held at each stage of the activity of the checkpointing process. The “h” (holds) relation shown links conditions in one ON to representations of other ONs, and exemplifies the concept of representational structuring. Note that this form of structuring has more in common with the use of communication relations than that of the various forms of abstraction, in that it is not hierarchic in nature. (One can readily imagine a pair of systems, each retaining back up information for the other.)

Figure 10: Equivalently, conditions holding representations of other ONs

Moreover it has become evident that somewhat similar relations, i.e. between an element in one SON and a set of elements in another SON, were also appropriate for modelling both the information being held for purposes of post hoc failure analysis, e.g. within a crime investigation support system, and the actual processes involved in acquiring this information.

For example one might wish to record in, so-to-speak, their own SON, the activities of the investigators who are creating, in an investigation support system, a representation of some complex crime in the form of a SON consisting of a large set of ON fragments and relations. By this means it would be possible to show how the investigators' activities relate to the various parts of their representation of what they have discovered about this criminal activity, e.g. the sequence in which they added information into the representation, and who added which information. (The problem of ensuring the safety of simultaneous updates to a database by separate
users is of course of relevance here, but is well-understood and will not be pursued further.)

Figure 11: Construction

Figure 11 records how a pair of investigators created a Crime Investigation SON, made up of three simple ONs, using three creation (“c”) relations, and showing also how the judge makes a post-hoc judgement (“Guilty!”) on the basis of information from this SON, using a judgement relation (“J”). In practice the investigators will have had to do more than just collect a set of separate fragments; they will also have to use “merge” operations to join fragments together, for example, when it becomes evident that the car that jumped the light is the car that subsequently crashed (and that the same driver was involved). Thus in general the “c” relation links an event in one ON to a set of elements in another ON. Similarly, the judge’s decision is likely to be based on more than a record of a single event.

Of necessity the picture assembled by the investigators will be partial – what really happened is perhaps as shown in Figure 12, but such a “complete” portrayal is most unlikely ever to be available.
Note that, as is the case with the checkpointing and the checkpointed process, there is no necessary hierarchical relation between the investigators’ activities, and the activities they are investigating. One can, after all, imagine two separate investigation agencies, each investigating and trying to model the other’s activities!

A closely-related form of representational structuring uses relations (marked by “e” for “evidence”) that record the evidential basis for particular elements (event, condition, arc, etc.) represented in a given SON. Using as an example the different crime portrayed earlier (in Figure 8), this usage is shown in Figure 13. This illustrates just when information was obtained by the police that provides evidence for particular elements in the SON that they are constructing of a crook’s activity, namely events of the crook being photographed in the burgled house, and being overheard having a synchronous interaction with the fence.

The definition of retention in [2] incorporates the idea that retained information is held in all subsequent conditions, until a discard operation. Using this idea for evidential relations, after the phone record is obtained, both this record and the previously acquired photograph will be held in subsequent conditions in the Investigator’s SON, as shown in Figure 13. (This is a particularly simple example, since in fact multiple events in one SON can be involved in providing evidence for a single element in another SON, just as one such event can provide evidence for multiple elements.)

Such evidence relations allow one to model, in as much detail as required, the activity, for example, of the police in making use of (and perhaps mistakes with) their crime investigation support system – and to deal with such complications as further evidence substantiating (or throwing doubt on) existing evidence, separate evidence being collected by separate police forces, etc.
A yet further, albeit somewhat similar, use of such relations is for documenting the evidence underlying some claimed or actual data provenance record. (One can readily imagine a relabelled version of Figure 13, in which the lower SON could be an actual recorded (rather than merely conjectured) SON, documenting some data provenance, and the upper SON could be the record of the curation (e.g. acquisition and documentation) of the evidence behind this data provenance record.) However, in some data provenance scenarios, there may be little interest in modelling the activities involved in the acquisition, and perhaps the (mis)use and discarding, of the evidence that was used to justify the creation of a SON. In such circumstances evidence might simply be held as part of the information associated with the various SON elements, and evidence relations might be conventional RDBMS-type relations involving such information. This would be very easily implemented if (i) the whole SON structure was implemented on a relational database infrastructure, or (ii) a SON structure was added into a crime investigation support system that already had facilities for recording such relationships.

8. Abstraction Hierarchies

We have yet to provide a formal treatment of the idea of abstraction hierarchies. However the assumption is that we can have behavioural, temporal and/or spatial abstractions of behavioural, temporal and/or spatial abstractions of ONs and Communication SONs, provided that these are in general properly “nested”, i.e. that
the result, shown graphically, will not have any overlapping boxes. (On the other hand, it is perhaps appropriate to allow such overlaps to occur when multiple alternative abstractions are defined – see below.)

With respect to a GUI there will also need to be operations for exploring and controlling the viewing of complex SONs, but these will presumably be fairly standard. Perhaps – unlike the case with some of the diagrams in our explanatory publications about SONs and their abstractions – the GUI should avoid (in general) showing both an abstract and a detailed view of what is being abstracted within the same screen. (This is in line with a strategy introduced many years ago for SADT diagrams [4].)

9. Alternatives

In the extended formalism [5] that permits the portrayal within a single ON of multiple possible alternative segments of behaviour (describing happenings in what are in effect different “worlds”), conditions can have multiple outgoing/incoming arcs, provided these are to/from alternative events.

Perhaps the simplest form of inclusion of alternative behaviours involves taking several named ON fragments, each with a single initial and final condition, labelling each of their internal conditions and events with the appropriate fragment name, and then combining the fragments’ initial conditions, and their final conditions. Use of the Add and Link operations to link together events bearing different names (i.e. from different worlds), would then not be allowed. Figure 14 shows an ON incorporating two examples of the use of alternate behaviours (one using fragments named ‘A’ and ‘B’, the other ‘C’ and ‘D’), and portrays an example of a disallowed link (between ‘A’ and ‘B’).

![Figure 14: Example of an invalid interaction among alternatives](image)

Adding some alternative behaviour, especially some asynchronous behaviour with multiple initial and final conditions, into the middle of a large asynchronous ON will be a much more complicated and error-prone affair. The relevant basic concept is that of a cut – a **cut** being a maximal set of mutually concurrent conditions [2]. Any non-intersecting pair of such cuts identifies a segment of behaviour that could have associated with it an alternative ON.

Figure 15(a) shows an example of a pair of cuts (portrayed by dotted lines) defining a segment of an ON, and Figure 15(b) an ON which could be used as an alternative to this segment; numbers are used to indicate how the input and output conditions
identified by the two chosen cuts are to be matched with those in the ON that is to serve as an alternative to the behaviour delineated by the cuts.

Figure 15(a): A pair of cuts in an original ON

Figure 15(b): An alternative segment

Figure 15(c): The ON showing the alternative behaviour in situ

Figure 15(c) shows the alternative behavior in place, so that the pair of views portrayed in Figures 15(a) and 15(c) are needed to show the two alternatives, as opposed to the scheme used in Figure 14 of showing alternative behaviours within a single view, identified by the use of labels.

Note that a GUI which only allowed one of a set of alternative views, such as Figures 15(a) and 15(c), to be shown at any one time would make it literally impossible to draw the sort of disallowed link shown in Figure 14. This would help to avoid introducing errors, though would make comparison of the alternatives difficult. If one can quickly and easily switch to and fro among the alternative views this might provide adequate means of comparison. However such issues will best be explored using realistic case studies, when we have adequate software support for SONs.

In theory it would be possible to modify and extend independently each of a set of alternative views, though this would make their comparison even more difficult. However, the aim would presumably be to end up eventually without any alternatives. Ideally this will be done simply by discarding each view when it is determined that it is no longer needed – the notion of trying to “rescue” part of an about-to-be discarded view, for incorporation in a continuing view, does not sound
very attractive. (One can draw some parallels to the problem of bringing a previously “forked” software design project back together.)

It is not evident at this point what levels of complication need to be considered, and how these can best catered for. For example, is it worth allowing alternatives within alternatives? (Showing alternatives via separate views would facilitate the representation and analysis of such possibilities.) Other possibilities include the provision of tools for identifying possible cuts, and for checking that the number of input and output conditions in a potential alternative ON matches the numbers of conditions associated with the two chosen cuts.

Alternative behaviour relations are likely to be needed to cater for situations in which it is unclear which system gave rise to which behaviour, e.g. which of two likely burglars committed a particular burglary, or which of two simultaneous burglaries in separate locations was committed by a given burglar. Similarly, there seems to be good reason for supporting alternative communication relations – did A talk to B or to C? However the case for allowing alternative temporal and spatial abstractions within a single large composite SON is less clear cut.

The incremental construction of a SON can include associating probabilities with alternative possible behaviours (and relations) [5]. Later, when more definite information is obtained alternatives can be discarded. Ideally, if at the end of this process any alternatives remain, they all will have had probabilities associated with them.

10. Time

If the times of various events were known, these times could be recorded in the form of information associated with the respective events – or by using synchronous communication links between these events and a linear ON representing a clock. (Multiple such ONs could of course be used if several independent clocks are involved.)

Timing information can be used to invalidate causality assumptions, and can provide suggestions as to possible causalities. An interesting issue for the future concerns what could be provided in the way of automated tools for proposing possible causalities, and for identifying incorrect causality arcs, based on whatever timing (and other) information is available.

11. Where Will This All End?

Ideally, the construction process will not end until the entire set of SONs and ONs is fully connected via appropriate relations and constitute a single overall SON that makes good use of appropriate abstractions. This iterative process might be characterised as aiming towards syntactic completeness, while retaining syntactic correctness. One can imagine some simple, or rather simplistic, tool support that might help guide the process of approaching syntactic completeness, e.g. by supporting the above types of incremental addition to the graph, together with drawing attention to remaining gaps between fragments, missing relations, and
alternatives whose probabilities have not yet been estimated. This is reminiscent of the old idea of a syntax-directed editor [3].

Semantic completeness will require associating additional (application-specific) information with the various SONs, by – to use programming language terminology – specifying an adequate set of “invariants” and/or “pre- and post-conditions” (i.e. logical conditions). These will be couched in terms of whatever annotations are held with the various ON’s conditions, events and arcs, so that semantic consistency checks can be carried out. (The “labels” attached to conditions and events in Figures 3 and 6 are in fact examples of such semantic annotations.) There will of course remain the question of how fully and accurately a given SON, even one that is semantically complete, reflects the actual behaviour that it is intended to model, be it a computing system failure, a crime or a document trail. (However these semantics issues will presumably be largely application-specific, rather than part of the basic SON theory and tool infrastructure.)

11 References


