Newcastle University

Structured Occurrence Nets:
Failure Analysis of Complex Evolving Systems

Brian Randell
School of Computing Science
Newcastle University, UK

Martinique, Jan. 2012

Complex Evolving Systems

- A ***complex evolving system*** (**CES**) is the term I am using for a system that is composed of a large number of concurrently-acting systems interacting, in general asynchronously, with each other and with the system's environment, with each system being possibly subject to modification by other systems. Examples include:
  - a large hardware system which suffers component break-downs, reconfigurations and replacements
  - a distributed system whose software is continually updated (or patched)
  - a multi-organisational computer system whose human operators undergo regular re-training
  - a typical large bureaucracy
- Such very diverse 'event-based' systems all suffer from a very high complexity of both design and behaviour.

Structure

- The importance of *structure* in helping designers to cope with *design complexity* is well-accepted, especially in software engineering (hence procedures, threads, classes, types, etc.) and VLSI design (higher order logics, graph based models, etc.)
- The effective use of structuring greatly reduces the cognitive complexity of designs, and the resources, both storage and computational, involved in their representation and manipulation.
  - But very few design structuring techniques support *system evolution*.
- Notations for recording, and especially structuring, representations of actual or potential *system behaviour* are much less developed, even for non-evolving systems.
  - (This is probably because detailed records of the behaviour of complex systems are mainly used *within* tools, e.g. for system verification and failure analysis, rather than in documents and user interfaces.)
- In fact I started working on the topic of behaviour structuring when I revisited our community's beloved "fault-error-failure" chain concept a few years ago.

Faults, Errors and Failures

- The dependability community has been interested for many years in the dependability of both hardware and software-controlled systems, and more recently also of complex computer-based systems (e.g., composed of computers, users, and even attackers).
- We have long accepted the utility of distinguishing between three different concepts – fault, error and failure – and that these follow a "fundamental chain":

  . . . → failure → fault → error → failure → fault →. . .

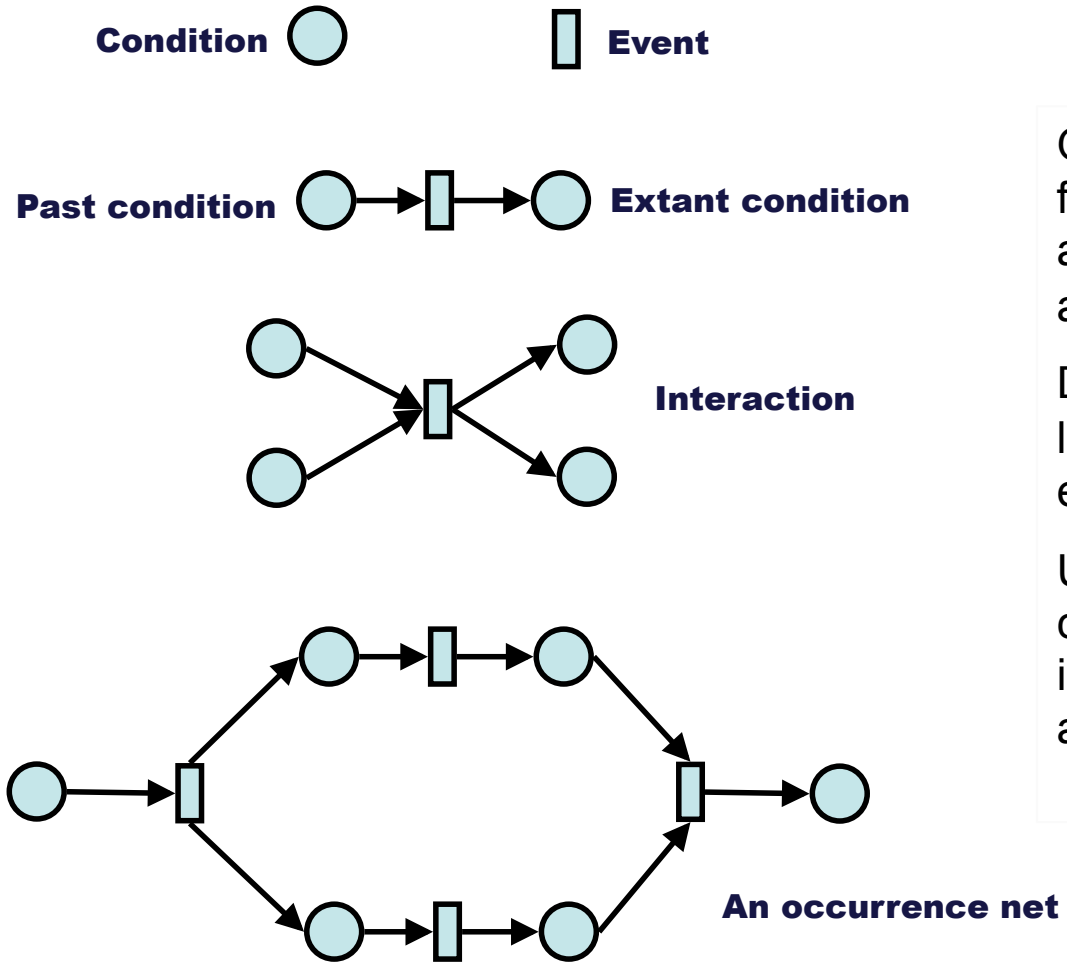
  i.e.

  . . . → event → cause → state → event → cause → . . .
- A few years ago I found myself trying to gain greater understanding of how fault-error-failure chains can progress within complex systems, i.e. from a system to:
  - the system of which it is *part,*
  - a separate system with which it is *interacting,* or
  - a system that it *creates, sustains* and/or *modifies*.
- Because of some past familiarity with it I used the (graphical but formally-based) Occurrence Net notation.

Occurrence Nets

- ***Occurrence Nets (ONs)*** are a well-established mathematical formalism (now with extensive software tool support) for representing causality and concurrency information concerning a *single execution of a system*.
- ONs are directed acyclic graphs that portray the (alleged) past and present, or the predicted, state of affairs in a system, in terms of *conditions* (i.e. *states*), *transitions* (i.e. *events*) and *directed arcs* (representing known or alleged *causality*).
- They are one early outcome of the work, started in 1962, on the conceptual foundations of a theory of communication, by the late Carl Adam Petri – whose Petri Net notation is extensively exploited for understanding synchronisation, and for designing and validating asynchronous systems.
  - "*Information System Theory Project*" by Anatol Holt et al (USAF, RADC, 1968).
  - "*Events, Causality and Symmetry*" by Glyn Winskel (BCS Int. Academic Conf. – Visions of Computer Science, 2008).

Occurrence Nets –
A Deceptively Simple Notation



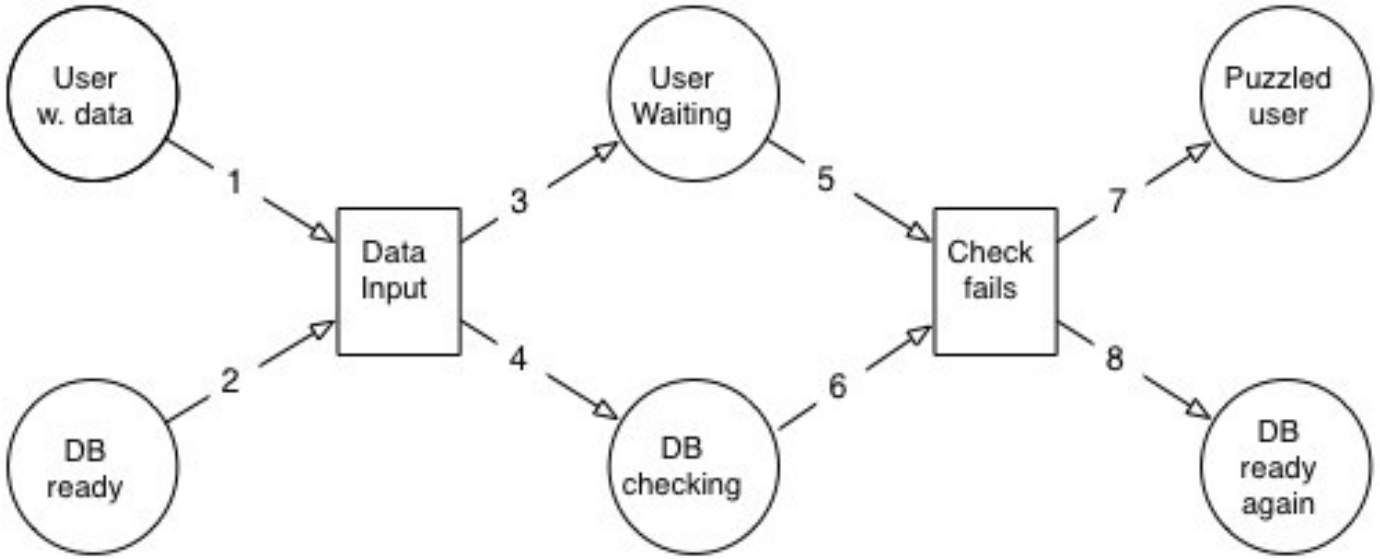
Occurrence nets must be fully-connected and acyclic – and must start and end with conditions.

Directed arcs (causal links) join conditions and events.

Unlike events, conditions can only have a single incoming and outgoing arc.

An Example ON –

two representations



Link	From	To	Info
1	User w. data	Data input	
2	DB ready	Data input	
3	Data input	User waiting	
4	Data input	DB checking	
5	User waiting	Check fails	
6	DB checking	Check fails	
7	Check fails	Puzzled user	
8	Check fails	DB ready again	

Event	Caused by	Causes	Info
Data input	1, 2	3, 4	
Check fails	5, 6	7, 8	

Condition	From	To	Info
User w. data		1	
User waiting	3	5	
DB ready		2	
DB checking	4	6	
Puzzled user	7		
DB ready again	8		

Using Occurrence Nets

- Simple examples of occurrence nets can be portrayed and studied as stylized diagrams, but in practice large ONs, to be useful, need to be stored (and analyzed) inside a computer.
- An ON looks like an *unfolded* (i.e. an asynchronous “trace” of a) Petri Net, but they have no necessary link to Petri Nets.
- ONs have in fact been re-invented, and re-named, by many different research communities (e.g. as “strand spaces” by security researchers, and as “message sequence charts” by networking researchers).
- ONs can be used for recording the actual or planned activities of any kind of complex system – hardware, software or organizational.
- ONs are extensively used in the computer industry, e.g. inside model-checking tools for validating system designs.
- With various colleagues, I have over the years employed them in research on deadlock avoidance, on error recovery in distributed systems (the “chase protocols”), and on atomicity.

Occurrence Nets at Newcastle

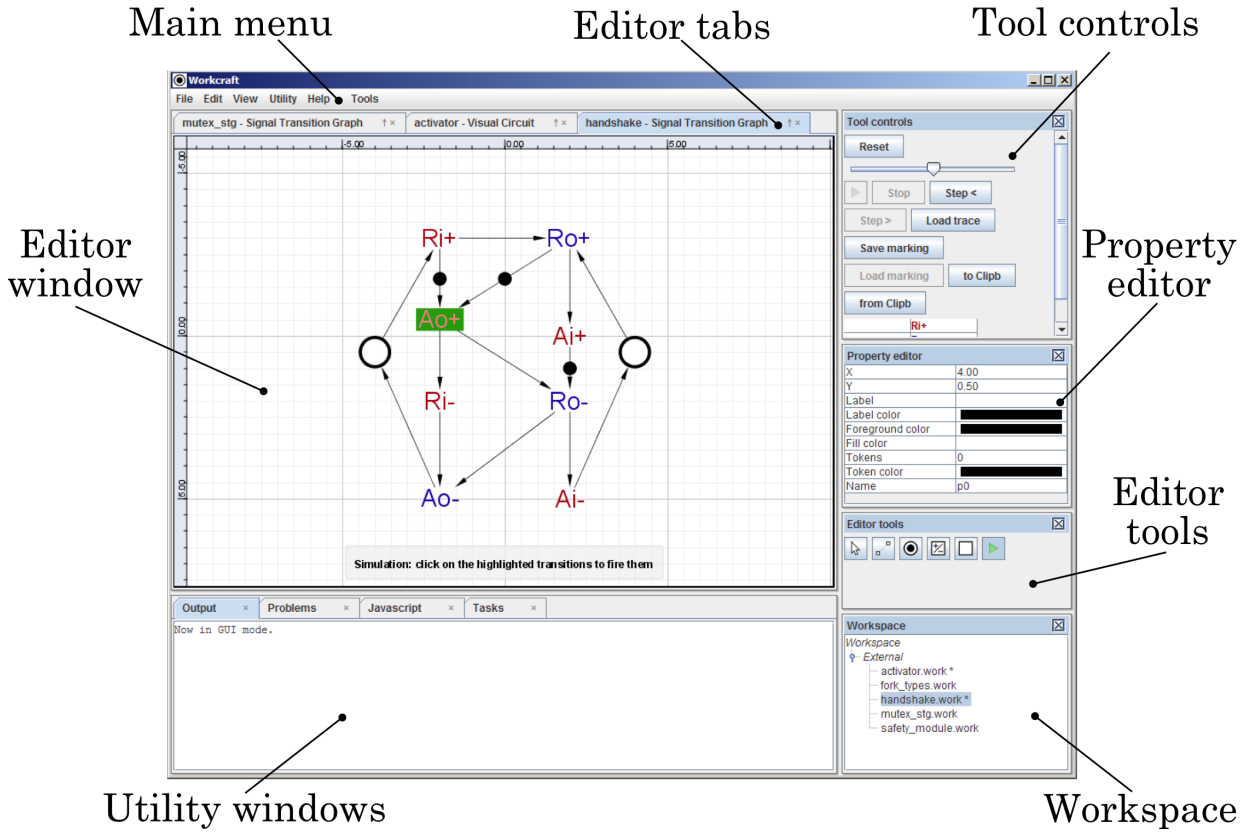
Much of the theoretical research, and tool development, over many years in ASL (Newcastle's joint EE/CS Asynchronous Systems Laboratory), on system design, system validation and system synthesis, has made use of ONs.

ASL's most recent interactive tool is WORKCRAFT (an infrastructure for interpreted graph models).

WORKCRAFT supports ON-based verification, synthesis and visualisation.

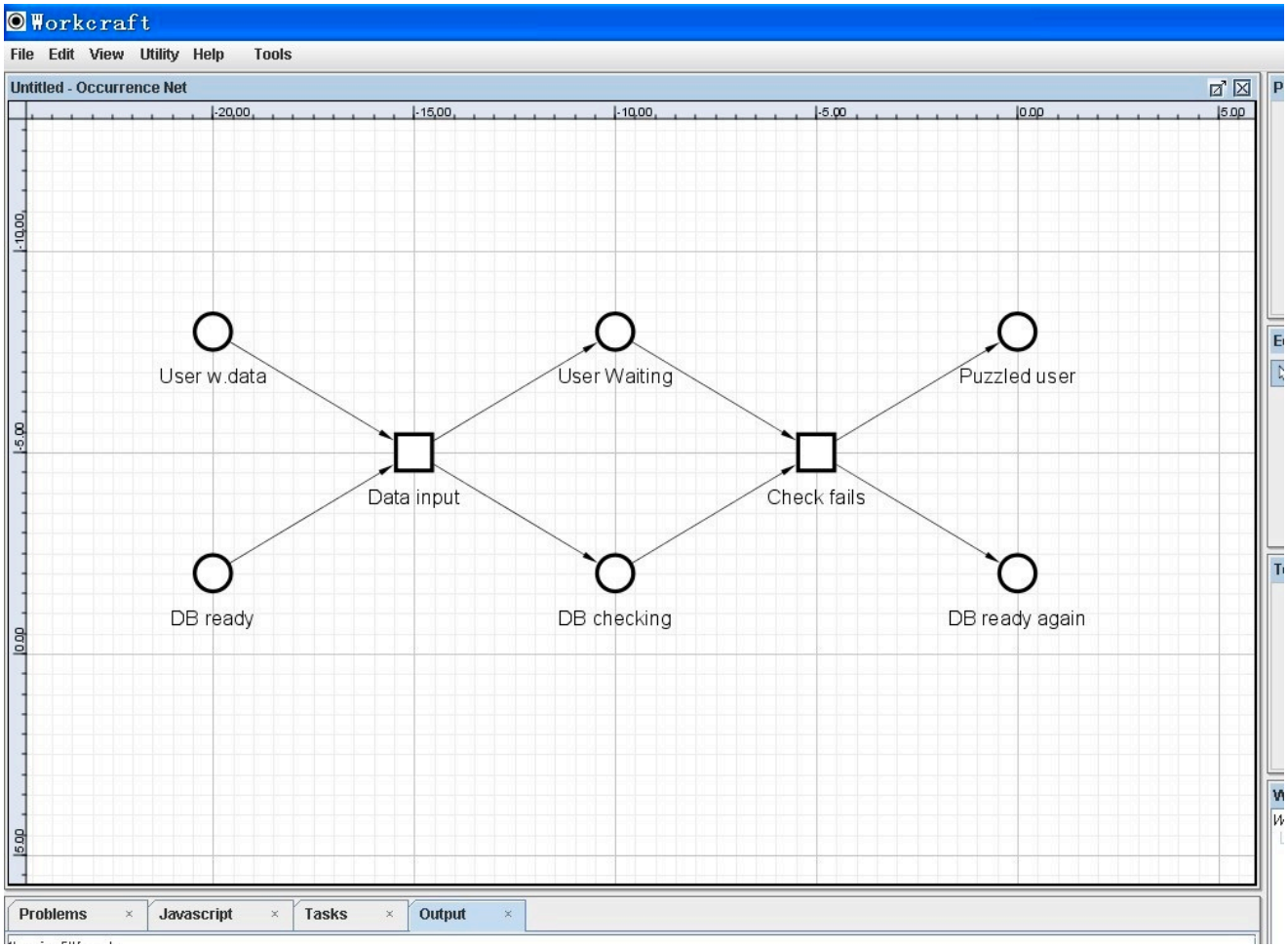
Existing plug-ins include BDD packages and SAT solvers.

Workcraft



http://workcraft.org/

The earlier ON example in Workcraft

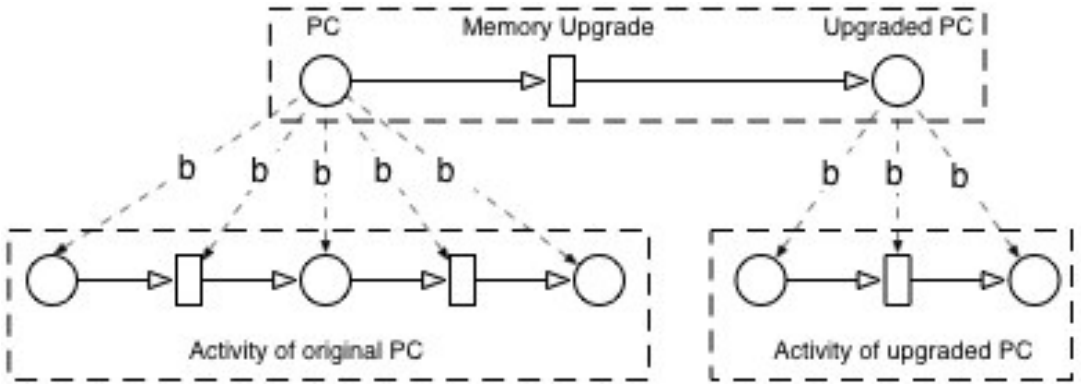


States and Systems

- I started trying to gain greater understanding of fault-error-failure propagation chains amongst interacting systems, and between such systems and any other systems that they created or modified, by drawing lots of ON diagrams.
- As in my previous work, I found them to be an excellent thinking aid, even when used very informally.
- I realised that one ON could be used to show the different stages of evolution of a system, and be associated with further ONs showing the activities that each successive version of this system was involved in.
- Thus I found myself using the same symbol – a "condition" (O) – to represent both systems and their states (in different related ONs).
- As a result I belatedly came to view '*system*' and '*state*' not as separate concepts, but just a result of a rather special form of abstraction that I termed "behavioural abstraction".
- Maciej Koutny joined in and clarified, corrected and formalised, and greatly developed, my initial vague and confused ideas on what we came to call *Structured Occurrence Nets.*

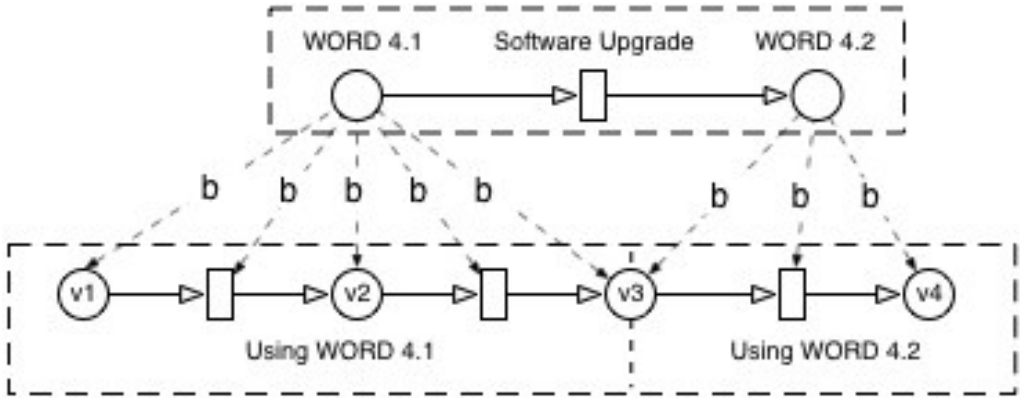
Examples of Behavioural Abstraction

Offline Evolution



Behaviour Relation b

Online Evolution



Note – Such behavioural abstractions *cannot* be represented in ordinary occurrence nets

Abstraction, and Structured Occurrence Nets

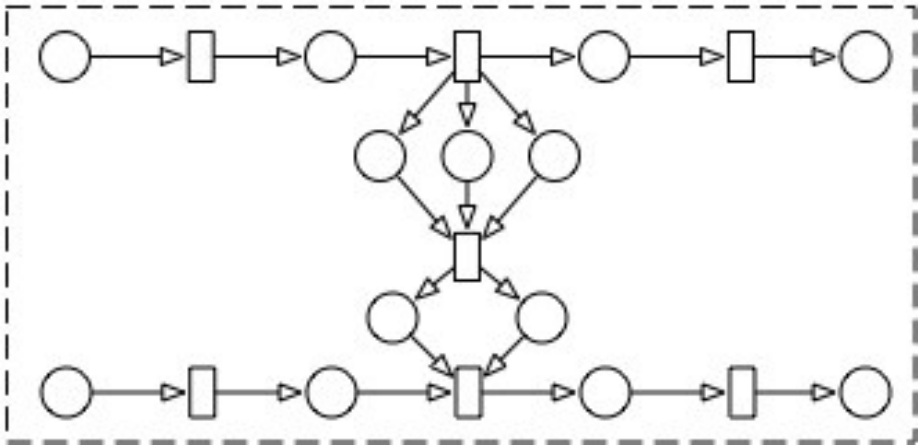
- Behavioural abstraction is just one of a number of forms of abstraction that we defined for occurrence nets. Two others are *spatial* and *temporal* abstraction. These provide a means of structuring an ON analogous to, say, the program structuring techniques listed earlier.
- What we term a **Structured Occurrence Net** (SON) is a set of related Occurrence Nets (using several specific forms of abstraction and other forms of relation).
- The various relations we have defined are all such that SON's, like ONs, are acyclic – and so respect the causality rules.
- The significance of SONs is that (i) they provide (through behavioural abstraction) a direct means of modelling evolving systems, and (ii) their structuring (using temporal and spatial abstraction) reduces their cognitive complexity, compared to that of an equivalent ON.
- These advantages can we believe facilitate such tasks as system validation (via model-checking), system synthesis, and system failure analysis. (The first two are major long-term interests of ASL – this presentation concentrates on system failure analysis, and takes a rather general view of the concept of a "failure").

Multiple Systems

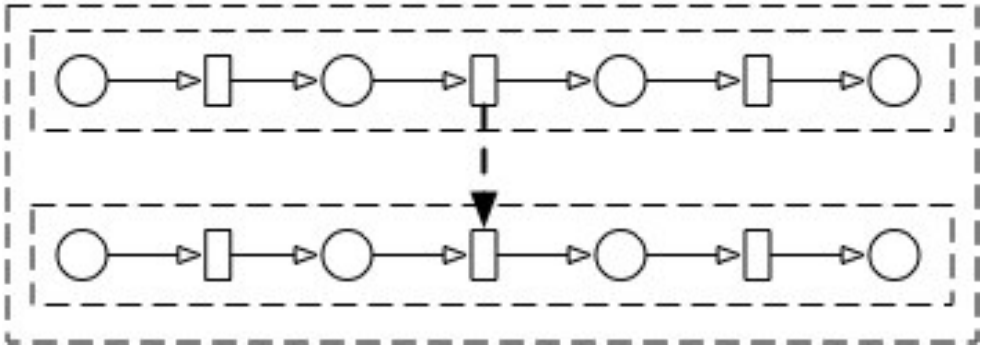
- Basic ONs are appropriate for single (non-evolving, typically asynchronous) systems.
- We delineate (using enclosing rectangular dashed boxes) the ONs that represent the behaviour of different systems.
- And define explicit ***communication relations*** representing any interactions that occur *between* these systems, so constructing a **Communication SON**.
- One advantage of this form of structuring is that we can hide away the details of possibly quite complicated interactions. (In doing so we can make use of *temporal abstraction*.)
- Note: unlike behaviour relations, neither communications relations, nor spatial or temporal abstraction relations, actually increase the logical, as opposed to practical, power of ONs.

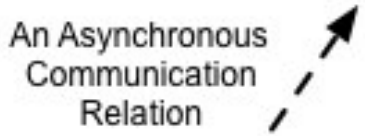
Communication Relations

An (unstructured) ON



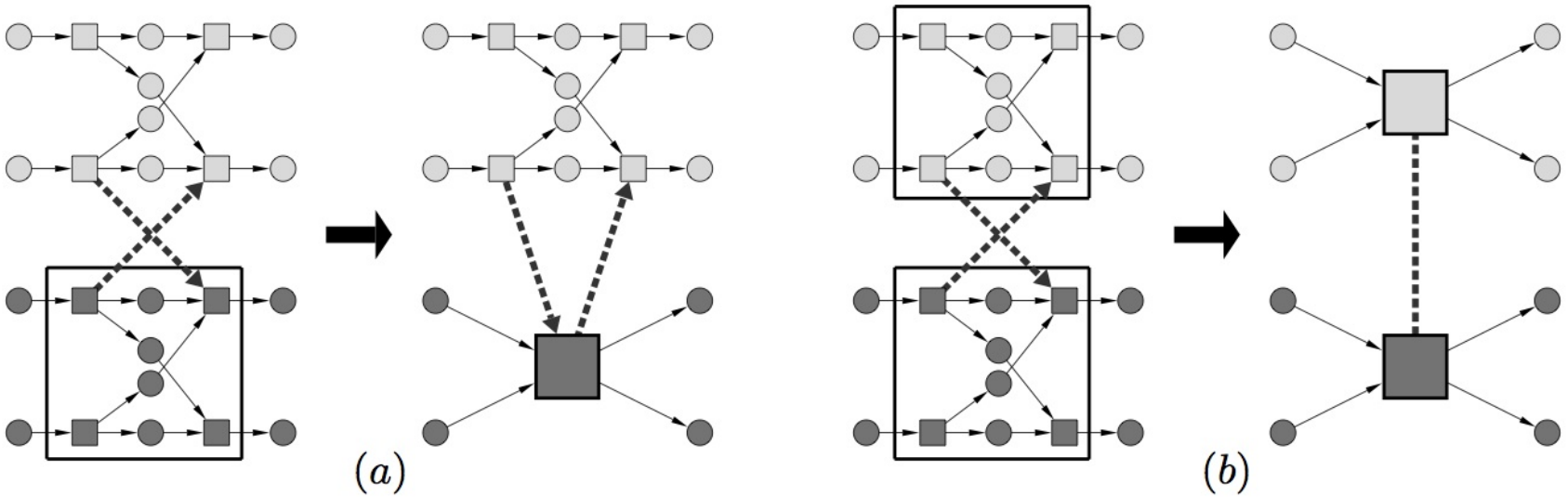
The equivalent Communication SON





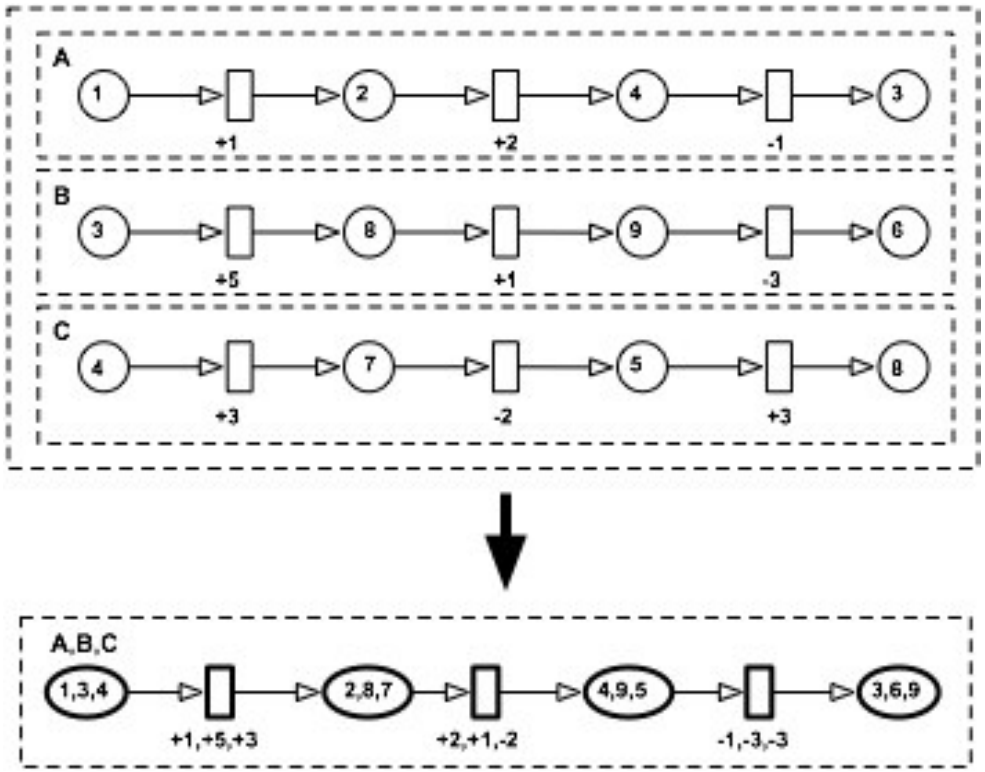
As above, communication relations can hide many details of the actual communications – using “temporal abstraction”

Temporal Abstraction and its Perils



- Temporal abstraction replaces segments of an ON that start and end with events by single (abstract) events
- But this risks introducing a cycle
- Hence the use of a *synchronous* communication relation (an undirected arc) in (b) above to avoid causing a cycle to exist

An Example of Spatial Abstraction



Typically spatial abstraction is used together with temporal abstraction, and perhaps repeatedly, when seeking to structure a large ON

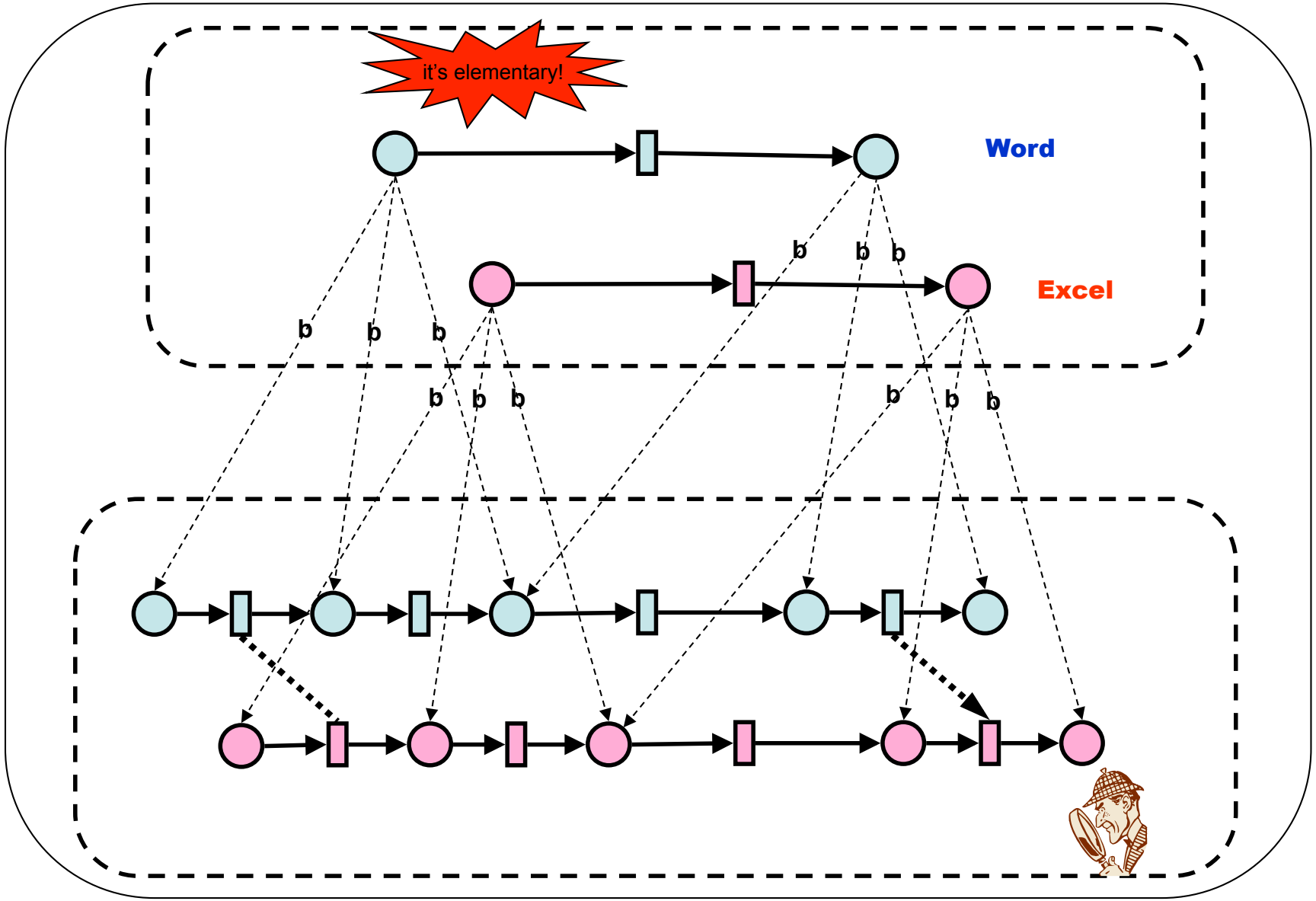
Other SON Abstractions

- So far just behavioural, communication, temporal and spatial abstractions have been shown.
- All have been defined rigorously, and basic theorems formulated and proved about them, mainly concerning the preservation of causality – these theorems provide the theoretical basis for our planned tool building efforts. (So far, just Communication SONs have been implemented in WORKCRAFT.)
- The other abstractions we have investigated support the following further relations:
  - Information retention (for recoverability and for post hoc failure analysis)
  - judgement (inline – for built-in error detection, and offline – in support of post hoc analysis)
- And we have investigated techniques for representing incomplete, contradictory and uncertain evidence regarding past system activity, e.g. that available to a police investigation, or an accident enquiry.

Failure Analysis
(of Complex Evolving Systems)

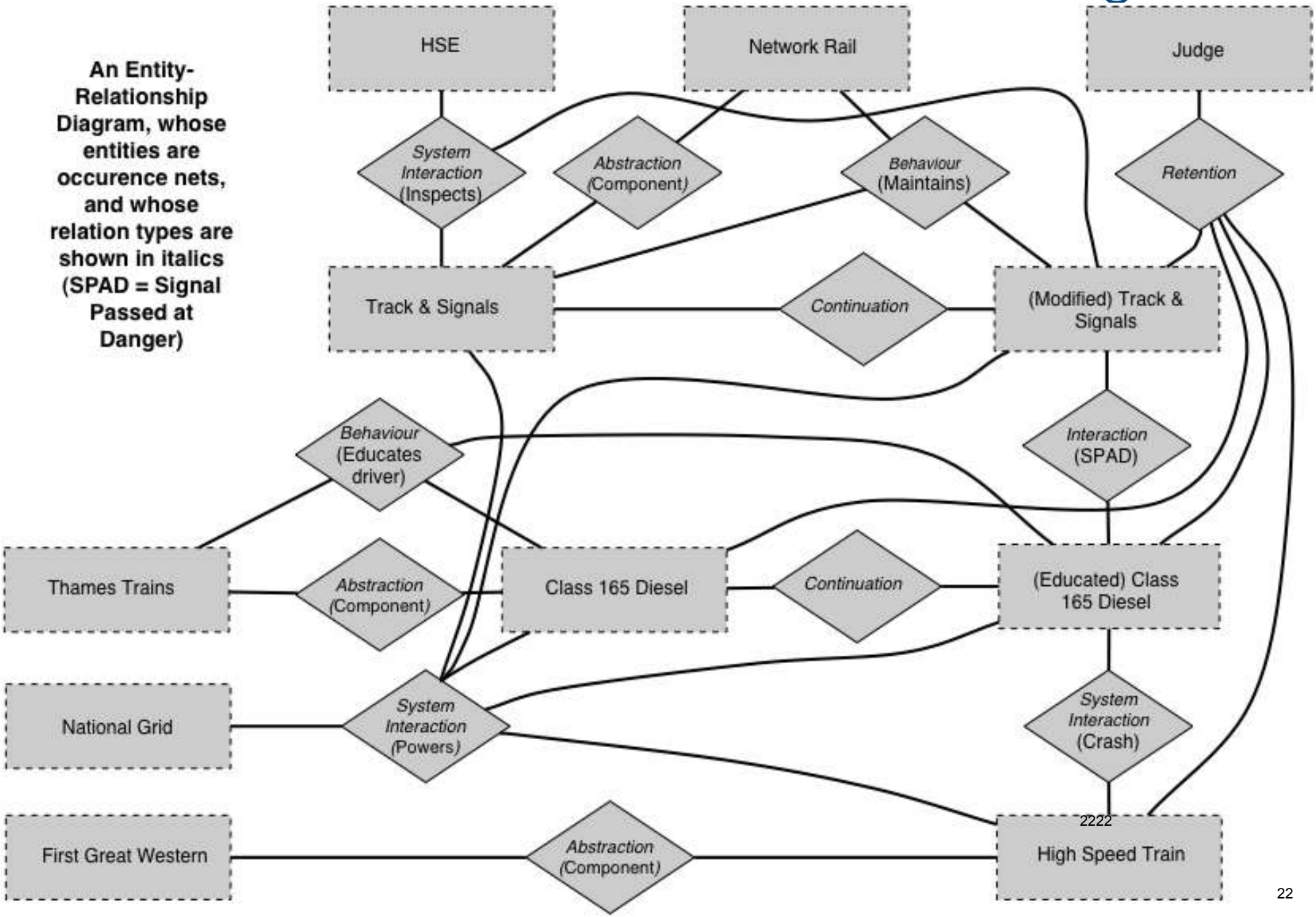
- We plan to use SONs not just for computer systems, but also for criminal investigation support systems (i.e. treating crimes as system “failures”).
- Failure analysis can involve following links in ONs *backwards* from a failure in order to identify causes (faults), and then *forwards* to identify further errors and potential failures – a strategy that was the basis of the “chase protocols”.
- Behaviour relations between ONs in a SON can similarly be followed in each direction, to trace fault-error-failure chains between a system and the systems it created or controls.
- Other types of relations between ONs can also be involved in such analysis.
- However, the actual identification of failures, errors and faults as such requires additional information, e.g. obtained from system specifications, or users.

Failure Analysis of Multiple Systems



A SON (thought) experiment

- Ladbrooke Grove was the scene of a bad railway accident in October 1999, when a three-car Class 165 diesel train operated by Thames Trains collided with a First Great Western High Speed Train
- The immediate cause of the disaster – the diesel train passed a particular signal when red.
- A lengthy enquiry identified many more issues, and many systems (rail companies, government organizations, drivers, trains, signalling mechanisms, etc.) were implicated.
- As a (thought) experiment we have considered how the huge mass of evidence considered by the enquiry could be represented and analyzed.
- We have used the conventional Entity-Relationship graphical notation, the entities in fact being individual (un-detailed) occurrence nets, representing information about the activities of each of the systems involved, the whole being a *very* large SON.
- Our belief is that, with the right tool support, the use of a SON could greatly aid the documentation and analysis of such a complex failure situation.


An Entity-Relationship Diagram, whose entities are occurence nets, and whose relation types are shown in italics (SPAD = Signal Passed at Danger)
HSE
Network Rail
Judge
System Interaction (Inspects)
Abstraction (Component)
Behaviour (Maintains)
Retention
Track & Signals
Continuation
(Modified) Track & Signals
Behaviour (Educates driver)
Interaction (SPAD)
Thames Trains
Abstraction (Component)
Class 165 Diesel
Continuation
(Educated) Class 165 Diesel
National Grid
System Interaction (Powers)
System Interaction (Crash)
First Great Western
Abstraction (Component)
High Speed Train

Concluding Remarks

- We believe that SONs, and their ability to help reduce the complexity of complicated activity records, and to deal simply with evolving systems, are quite novel (and very promising).
- Our planned future work involves some further theory development, but mainly concerns the implementation of means of representing and analyzing fully-general SONs in the WORKCRAFT platform.
- We plan to re-implement the platform's existing system validation and system synthesis tools so as to make use of SONs' support for communications, temporal, and spatial abstraction – the aim being to demonstrate the ability to handle much larger and more complicated problems than these tools can currently cope with.
- Using also behavioural, data retention, and judgemental abstractions, we will investigate the utility of SONs as an infrastructure for a crime and accident investigation support system.
- This investigation is to be carried out in co-operation with a leading commercial developer of such systems, who is interested in enhancing their systems' ability to perform analyses for very large and complex investigations.

Selected References

- V. Khomenko, M. Koutny, A. Yakovlev: Logic Synthesis for Asynchronous Circuits Based on Petri Net Unfoldings and Incremental SAT. Fundamenta Informaticae 70, 2006.
  - http://www.cs.ncl.ac.uk/publications/inproceedings/papers/749.pdf
- M. Koutny, B. Randell: Structured Occurrence Nets: A Formalism for Aiding System Failure Prevention and Analysis Techniques. Fundamenta Informaticae 97, 2009.
  - http://www.cs.ncl.ac.uk/publications/trs/papers/1162.pdf
- B. Li, M. Koutny: Verification and Simulation Tool for Communication Structured Occurrence Nets. CS-TR, Newcastle University, (to appear).
- P.M. Merlin and B. Randell: State Restoration in Distributed Systems. Proc. FTCS-8, 1978.
  - http://www.cs.ncl.ac.uk/publications/inproceedings/papers/347.pdf
- I. Poliakov, V. Khomenko, A. Yakovlev: Workcraft - A Framework for Interpreted Graph Models. LNCS 5606, 2009.
- B. Randell, M. Koutny: Failures: Their Definition, Modelling and Analysis. LNCS 4711, 2007.
  - http://www.cs.ncl.ac.uk/publications/trs/papers/994.pdf
- B. Randell, M. Koutny: Structured Occurrence Nets: Incomplete, Contradictory and Uncertain Failure Evidence. CS-TR 1170, Newcastle University, 2009.
  - http://www.cs.ncl.ac.uk/publications/trs/papers/1170.pdf