# Memorable Occurrences

Brian Randell

School of Computing, Newcastle University, Newcastle upon Tyne, UK
Brian.Randell@ncl.ac.uk

My first interaction with Maciej Koutny was rather one-sided, and indeed unknown to me at the time! This occurrence arose from a project that I initiated at Newcastle University in 1977, on the construction and exploitation of an experimental highly parallel real time control system. The project involved a recently-purchased large collection of hardware that incorporated a half-a-dozen mobile actuators which were to be controlled by a minicomputer and several microcomputers. The movements of the actuators were strictly constrained, and the whereabouts of the actuators relative to their constraints could be monitored with the aid of fifty or so fixed sensors. I should mention that uninformed spectators tended to view, and refer to, this experimental highly parallel real time digital control system as a model railway or train set, in which the movement constraints (which they termed "track sections", which were thirty-two in number) were also the means of delivering power to, and hence causing movement of, selected actuators (or "trains"), between sensors ("stations"). It was as far as I know the first such computer-controlled model railway, at least in any computer science department, anywhere. (The idea for it came from a computer-controlled model railway I'd seen at Toronto University while on sabbatical there – but this was a much simpler system, that merely demonstrated the speed control of just a single engine.)

This was long before the availability of model railways embodying a microprocessor in each engine that could be controlled by signals sent along the rails. But we realised that just a single computer selectively controlling the power supply to each of the set of electrically-isolated track sections, so that a number of "dumb" trains (we acquired six) could be directed to travel independently, would provide a challenging concurrent real time programming environment. In this environment each train (actually just an engine – we didn't bother with passenger carriages or goods wagons) was in effect an independent process, and one of the most obvious interesting problems was how to control these "processes" so as to prevent them from crashing into each other! In fact, potential collisions between the trains were fairly easily prevented, at the cost of occasional partial or complete system deadlocks.

The problem of how to avoid such deadlocks, while facilitating multiple concurrent train movements along pre-defined journeys, was one that Phil Merlin, a visiting colleague, and I investigated and solved, fully and fairly quickly (albeit somewhat informally), using the concepts of occurrence nets, the acyclic directed graph formalism, allied to Petri Nets, with which he was already familiar [HOLT1968]. (Phil was a brilliant young researcher from IBM Research and the Technion, Haifa, who was to die tragically young, just two years later. His memory is honoured to this day at the Technion by the Annual Dr. Philip M. Merlin Memorial Lecture and Prize Award.) We documented our solution in a brief technical note, and left it at that.

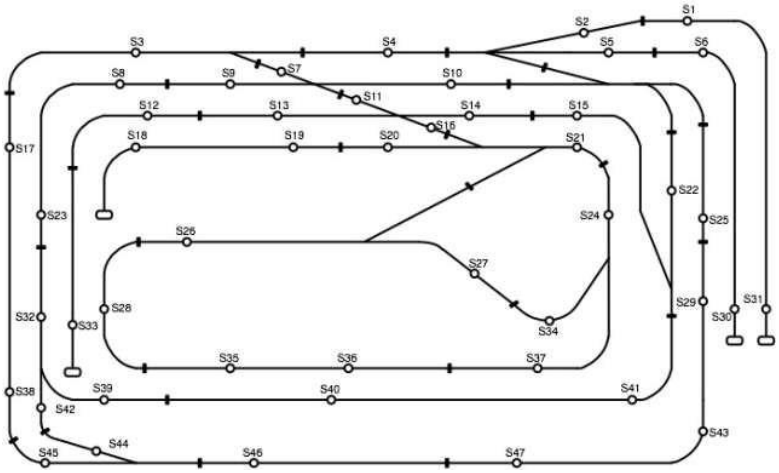
A number of other people elsewhere took up the study of what came to be called "The Merlin-Randell train control problem". It was only years later that I learnt that Maciej Koutny had taken this problem as the subject of his PhD research, and that I saw – I'm afraid I cannot say I read – his impressively lengthy thesis *O Problemie Pociagów Merlina-Randella* [KOUT1984A].

A year later, Maciej became a Research Associate at Newcastle, where his research career – subsequently first as a Lecturer, then Reader, and since 2000 as a Professor of Computing Science and head of Newcastle's theoretical computing group – has flourished ever since.

For a while he continued his study of the Merlin-Randell problem — in [KOUT1985] he provided perhaps the best informal statement of the problem in the following terms:

"*There is a finite set of trains and a layout. The layout is represented by an undirected graph, the nodes of which represent places where trains can reside (stations), the arcs of which represent possible moves. Each station can hold only one train. Each train has a program to follow consisting of a directed path through the graph. The train can leave a station when the station it is immediately to travel to is empty. The problem is to find a synchronisation among train movements which allows parallel movements where possible and enables each journey to be completed*."

This problem of finding such sets of synchronised train movements, which was in essence that of calculating a suitable occurrence net, remained in vogue for quite a while, much like the "Dining Philosophers Problem" before it; I took no part in this research, but I did spend quite a bit of time on various issues to do with controlling the train set, though my main concern remained that of system dependability.




Newcastle's First Train Set

Newcastle's research on dependability started in 1970, and initially concerned the problem of tolerating residual accidental design faults in simple sequential programs. (I had formed the then unfashionable view that current work on proving programs correct would not suffice for large complex programs, and could perhaps be usefully complemented by work on software fault tolerance.) From this, we had soon moved on to considering the problems of hardware and malicious faults, not just software faults, in concurrent programs, and then in distributed computing systems.

Phil Merlin enthusiastically joined in on this research while he was with us, and worked with me on a particular (backward) error recovery problem, i.e. the task of restoring a distributed system to a previous state which it is believed or hoped preceded the occurrence of any existing errors. Our work was based on the use of what we called "Occurrence Graphs",

which we described as similar to occurrence nets, differing mainly in that we viewed an occurrence graph as a dynamic structure that was "generated" as the system that it was modelling was executed, and which contained certain additional information indicating which prior states have been archived and so were restorable.

We tackled the problems arising from concurrency in their full generality, so as to deal with the possibility of there being multiple concurrent faults, some even occurring during error recovery. The solution we produced was a decentralized recovery mechanism that we entitled the "chase protocol". We assumed that each node of a distributed system would hold a record of the part of the occurrence graph that related to its contribution to the overall system behaviour. Then each node would execute a protocol that had the effect of causing error recovery commands to "chase" through the dynamically-growing occurrence graph so as to overtake ongoing error propagations, and the nodes to co-operate in identifying a consistent set of restorable states [MERL1978].

My third involvement with occurrence nets was in collaboration with Eike Best, whilst he was one of Peter Lauer's PhD students, in Maciej's research group at Newcastle. This resulted in the development of a formal model of atomicity [BEST1981]. One of our starting points was David Lomet's work on "Atomic Actions" [LOME1977]. This work was undertaken during David's sabbatical at Newcastle from the IBM Research Center at Yorktown Heights, and involved extending the recovery block concept [HORN1974]. (This involved the use of a "recovery cache" to provide programs with a means of undoing all of the effects of a recovery block after an error had been detected, in order that an alternative program strategy could be attempted.)

"Atomic actions are similar to procedures . . . And in an isolated setting, they behave just like procedures. When concurrent activity is present, the body of an atomic action continues to behave as it did in isolation, while an ordinary procedure, if it accesses shared state, may behave very differently" [LOME1977].

Thus atomic actions allowed the recovery block concept to work in the presence of concurrently executing activities. (They can be viewed as the programming language equivalent to the transaction concept, an idea that the database community was also developing at this time [LOME2011].)

Eike Best and I pursued this idea of an atomic action as a programming concept, rather than a "hardware feature" or a "synchronisation method", and inspired by [MERL1978] were led to propose a formal model for atomic actions based on what we termed structured occurrence graphs. Using an occurrence net to represent a program's activities, the dynamic structure arising from any single programmer-defined atomic action can be collapsed to a single event. Multiple co-existing such collapsings can be performed, as long as the graph remains cycle-free. The concept of structured occurrence graphs involves the imposition of a nested structure of possible collapsings, i.e. of atomic actions, on a conventional occurrence graph. Being nested, one can be sure that multiple co-existing collapsings of any the so-identified sub-graphs would not introduce any cycles into the graph.

My most recent interactions with Maciej again have centred on occurrence nets. A very fruitful, indeed memorable, collaboration ensued when Maciej saw how I was using little occurrence net diagrams to explore whether and how such nets could be used to model the activity of an evolving system. I was actually reconsidering, yet again, the fundamental dependability concepts and definitions that I and colleagues have developed over many years, and whose most complete description is in [AVIZ2004].

I had belatedly realised that our notions about 'fault/error/failure chains' did not cope well with situations in which the set of components (i.e. sub-systems) in a system of systems was

for whatever reason changing. Examples include a large hardware system which suffers component break-downs, reconfigurations and replacements, a large distributed system whose software is continually updated (or patched), a multi-organisational computer system whose human operators undergo regular re-training, or a typical large bureaucracy. However, the little example I often used in my initial thought experiments was that of the confusion that could be caused to someone struggling to create a WORD document when unaware of the fact that the WORD system being used was itself suffering a succession of faults and updates!)

I was trying to use occurrence nets to represent both the activities of such a system, and of whatever was causing it to evolve. I was sketching diagrams that involved multiple interrelated occurrence nets, the relationship I initially tried to formulate being that of "is a behaviour of". The idea was that one occurrence net would portray the activity of the system that was in fact evolving, and another net would portray the activity of the entity (system) that was controlling this evolution. States portrayed in this second (in fact more abstract) net corresponded to the different versions of the evolving system, and were related to the relevant sections of the evolving system's occurrence net. Maciej saw my diagrams, and said "That looks interesting – can I join in?" Needless to say, I warmly welcomed this suggestion!

This led to a much-needed clarification, and a belated formalization, of the concept of behaviour relations, and to our defining a number of other relations between occurrence nets in order to produce what we called "Structured Occurrence Nets" (SONs), as a means of describing the activities of an evolving system and aiding the analysis of its failures. The other types of relations that we defined enabled the activities of separate sub-systems to be distinguished from each other, and provided various means of abstraction, both temporal and spatial, that aided the representation and analysis of complex systems. In total what we had developed was a very general formal notation for representing and investigating complex causalities, a notation that significantly extended the expressive power and practicality of occurrence nets.

These ideas have been documented in a number of papers, e.g. [RAND2007], [KOUT2009], [RAND2011], and Maciej has led an EPSRC-sponsored research project with the rather contrived title and acronym "UNderstanding Complex eVolution through structured behaviours (UNCOVER)". This project extended the initial definitions of SONs with new features related to alternate behaviours and timing information [BHAT2016], and implemented all the basic functionalities in a protoype tool called SONCraft [LIRA2018]. This tool, which was designed and implemented by Bowen Li, provides a user-friendly graphical interface which facilitates model entry, supports interactive visual simulation, and enables the use of a set of analytical tools, e.g. for reachability analysis. (SONCraft is based on Workcraft, a framework for interpreted graph models, a product of Newcastle's joint EEE/CS Asynchronous Systems Laboratory – see http://workcraft.org/, which provides download links for SONCraft.)

The original motivation for SONs and for building SONCraft was the problem of analyzing the causes of failures in complex *computer* systems. But in fact the SON ideas, and the SONCraft tool are as indicated above very general in nature, and essentially just about the representation of causality. Indeed one of the application areas we found ourselves concentrating on was in fact criminal investigation. This involved our viewing criminals or criminal gangs as 'systems', and their crimes as 'failures'! Our view is that portrayals of the criminal activities that result in crimes, i.e. of the observed or inferred causality links between criminals' various actions, are potentially useful both for identifying the criminals, and for portraying and explaining the evidence-based reasoning that will later be needed to convict them.

The UNCOVER project has thus had numerous very useful interactions with a number of police and other investigative agencies, and tool suppliers to these agencies. As a result, rather pleasingly, support has recently been obtained from Innovate UK for a two-year "Knowledge Transfer Project" that will enable the collaborative incorporation by the Bristol-based company Clue Computing Co. of SONcraft-like facilities into their CLUE investigation support system.

In fact our work together on SONs has been the most memorable of the various occurrences I have described here of my very fruitful interactions with Maciej over the years. What is particularly pleasing is that it is still continuing, as we explore the possible ramifications of what we have come to call "structured causality".

## References

[AVIZ2004] A. Avizienis A, J.-C. Laprie, B. Randell, C. Landwehr: *Basic concepts and taxonomy of dependable and secure computing*. IEEE Transactions on Dependable and Secure Computing 2004, 1(1), pp.11-33.

[BHAT2016] A. Bhattacharyya, B. Li, B. Randell: *Time in Structured Occurrence Nets*. In: International Workshop on Petri Nets and Software Engineering (PNSE'16), pp.35-55.

[BEST1981] E. Best, B. Randell: *A Formal Model of Atomicity in Asynchronous Systems*. Acta Informatica 16 (1981) pp. 93-124.

[HOLT1968] A.W.Holt, R.M.Shapiro, H.Saint, S.Marshall. *Information System Theory Project*. Report RADC-TR-68-305, US Air Force, Rome Air Development Center (1968).

[HORN1974] J.J. Horning, H.C. Lauer, P.M. Melliar-Smith, B. Randell: A Program Structure for Error Detection and Recovery, in *International Symposium on Operating Systems: Theoretical and Practical Aspects.* (Springer Verlag, 1974), pp. 171-187.

[KOUT1984A] M. Koutny: *O Problemie Pociagoẃ Merlina-Randella (in Polish).* PhD Thesis, Department of Mathematics, Warsaw University of Technology, Warsaw, Poland (1984)

[KOUT1984] M. Koutny: *On the Merlin-Randell Problem of Train Journeys*. In: Proc. of 6th International Symposium on Programming, Springer Verlag, Lecture Notes in Computer Science 167 (1984) 179–190.

[KOUT2009] M. Koutny, B. Randell: *Structured Occurrence Nets: A formalism for aiding system failure prevention and analysis techniques*. Fundamenta Informaticae 2009, 97(1-2), 41-91.

[LIRA2018] B. Li, B. Randell, A. Bhattacharyya, T. Alharbi, M. Koutny: *SONCraft – A Tool for Construction, Simulation and Analysis of Structured Occurrence Nets*. In: Proc. ACSD (2018).

[LOME1977] D.B. Lomet: *Process Structuring, Synchronization and Recovery Using Atomic Actions*. In: Language Design for Reliable Software, pp. 128-137.

[LOME2011] D.B. Lomet: Transactions: From Local Atomicity to Atomicity in the Cloud. In Dependable and Historic Computing. Springer (2011) pp.38-51.

[MERL1978] P. Merlin, B. Randell: *State Restoration in Distributed Systems*. In: FTCS-8, IEEE Toulouse (1978) pp. 129-137.

[RAND2007] B. Randell, M. Koutny: *Failures – Their Definition, Modelling and Analysis*. In: Theoretical Aspects of Computing: 4th International Colloquium (ICTAC). 2007, Macao, China: Springer-Verlag.

[RAND2011] B. Randell: *Occurrence Nets Then and Now – The Path to Structured Occurrence Nets*. In: Application and Theory of Petri Nets. 2011, Springer-Verlag.