Modelling provenance using Structured Occurrence Networks

Paolo Missier, Brian Randell, and Maciej Koutny

Newcastle University, School of Computing,
Newcastle upon Tyne, UK
{firstname.lastname}@cs.ncl.ac.uk

Abstract. Occurrence Nets (ON) are directed acyclic graphs that represent causality and concurrency information concerning a single execution of a system. Structured Occurrence Nets (SONs) extend ONs by adding new relationships, which provide a means of recording the activities of multiple interacting, and evolving, systems. Although the initial motivations for their development focused on the analysis of system failures, their structure makes them a natural candidate as a model for expressing the execution traces of interacting systems. These traces can then be exhibited as the provenance of the data produced by the systems under observation. In this paper we present a number of patterns that make use of SONs to provide principled modelling of provenance. We discuss some of the benefits of this modelling approach, and briefly compare it with others that have been proposed recently. SON-based modelling of provenance combines simplicity with expressiveness, leading to provenance graphs that capture multiple levels of abstraction in the description of a process execution, are easy to understand and can be analysed using partial order techniques underpinning their behavioural semantics.

1 Introduction

Structured Occurrence Nets (SONs) [KR09,Ran11] are a formalism that provides a means of recording the activities of a set of interacting, and evolving, systems. They were initially developed to address problems of validating, synthesizing, and analyzing failures of complex, evolving computer-based systems. SONs are an extension of Occurrence Nets (ON) [BD87], which are "acyclic Petri nets that can be used to record execution histories of concurrent systems, in particular, the concurrency and causality relations between events." [KK11]. ONs are suitable for representing causality and concurrency information concerning a *single execution of a (possibly asynchronous) system*. In this paper we show how SONs provide a suitable formal grounding to express the provenance of data that is involved in (i.e., is produced or consumed by) multiple interacting systems.

Although SONs can be expressed set-theoretically, in this paper we choose to use a simpler and more immediate graph representation and will completely avoid formal definitions. Those can be found in [KR09]. As shown in Fig. 1, the basic ON formalism is very simple. Circles represent conditions (i.e. the

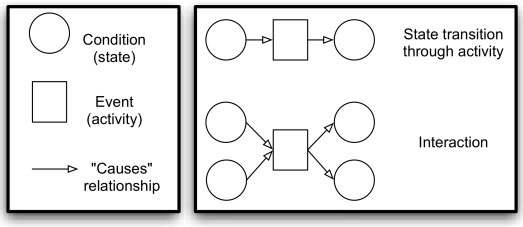


Fig. 1. Basic graphical ON notation

holding of a state); an event, represented by a box, can be caused by one or more conditions, and can result in one or more new conditions. Since the arcs are intended to represent causality, ONs must be *acyclic* directed graphs. In addition, well-formed ONs are defined by two conditions (see Def. 1 in [KR09]): (i) states have at most one input and one output arc, and (ii) events have at least one incoming arc and one outgoing arc.

Fig. 2 shows a simple ON portraying the execution trace of a process, during which information needed to draft a document about some experiment was acquired. It includes several activities, two of which ("verify experimental results" and "read paper p2") were concurrent. Labels may be associated to states, but they have no formal meaning in the model. In this example, `ptd`, for "preparing to draft", indicates an initial state for a sequence of actions that lead to a new state, "ready to draft".

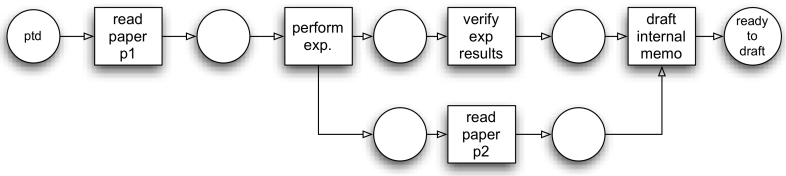


Fig. 2. Simple ON example

An ON is thus simply a means of recording what happened, indicating "what caused what". It does not in itself indicate "who" caused a particular event. Rather, the basic formalism implies that the whole of any given ON represents the (possibly asynchronous) activity of a single un-identified "system". The issue of identifying the various separate systems that together give rise to some given complex activity is one that is addressed by SONs, described in more detail in the next section. Briefly, SONs extend ONs with relationships for describing: (i) a *communication* relationships to specify interactions amongst systems; (ii) a *behaviour abstraction* relationships, which provides a *dual view between state and*

system, whereby a state that appears in one ON unfolds into a whole system, in which internal activities that pertain to that state can be made explicit; and (iii) a *temporal abstraction* relationship by which events that appear instantaneous at one level of abstraction, unfold into complex state-event nets at another level.

In this paper we show how SONs provide a convenient and intuitive formalism for representing data provenance, by introducing modelling patterns that make use of these relationships. A particularly interesting feature exhibited by these patterns is the uniformity of representation of the evolution of data, and *the evolution of the agents that were responsible for performing the activities*. The ability to represent agents as evolving systems has benefits for decision support applications based on provenance. For example, one's provenance-informed judgment on the quality of a document may be affected by the knowledge that the author was aware of certain papers at the time the document was prepared. This knowledge is easily encoded by modelling the author as a system characterized by evolving states, with activities such as "read paper X" that determine state transitions. We give a simple example of this encoding in Sec. 3.3. It is worth noting that the formal rules that govern these SON relations take into account the subtle complications that can arise from asynchrony, complications that are not evident in the relatively simple examples shown in the rest of the paper.

1.1 Related work

The modelling approach proposed in this paper is alternative to others that have been proposed recently, including the Karma model [Sim08], Janus [MSZ+10], PASS [HSBMR08], as well as a few that are typically tied to workflow systems (see Sec. 3.2). While all of these have been developed with partuclar applications in mind (typically in the area of e-science), the PROV generic model of provenance stands out, as it is (at the time of writing) in the process of crystallizing as a W3C recommendation[1]. PROV follows in the steps of the Open Provenance Model [MCF+11] and, at first glance, is sufficiently expressive to model the provenance of agents in addition to the causal relationships mentioned in this paper. It does not, however, make it easy to model the specification of processes that While a complete and formal comparison between the two formalisms is not yet available, a brief discussion on PROV encoding of our "Alice and Bob" example can be found in Appendix 3.

1.2 Benefits and limitations

Some of the benefits expected from this work includes seamless modelling of the provenance of data, activities, and agents, all at multiple levels of abstraction. In addition, SONs provide a formal syntax and semantics that will make it possible to carry out formal validation of provenance graphs. This, however, is beyond

[1] PROV will become a W3C recommendation by the end of 2012. The current working drafts can be found here: `http://www.w3.org/2011/prov/wiki/Main_Page`

the scope of this exploratory paper and is left for future work, as is the analysis of the types of queries supported by the model.

Implementation issues, including the encoding of SON graphs in machine-processable form, are being addressed using the WorkCraft platform, developed by the Asynchronous Systems Laboratory at Newcastle[2]. Workcraft provides a flexible, general framework for the visual editing, (co-)simulation and analysis of a variety of Interpreted Graph Models with a common graph structure, including Petri Nets, gate-level circuits, Static Data Flow Structures and Conditional Partial Order Graphs.

1.3 Paper organization.

The rest of the paper is organized as follows. An overview on SONs is provided in the next section, followed in Sec. 3 by the description of SON patterns for modelling provenance. Sec. 4 concludes the paper with a brief discussion on ongoing work.

2 Structured Occurrence Networks

A SON is a set of ONs that are formally related to each other using a number of different types of relations [KR09]. Here we will make use of just three types of relation, namely the behaviour relation, the asynchronous communication relation, and the temporal abstraction relation. These provide a direct means of recording which systems give rise to which parts of some overall activity, how these systems interacted during this overall activity, and how these systems have themselves perhaps evolved.

Behaviour relation. The behaviour relation is the means by which some portion of a complex overall activity is associated with a particular system. It embodies the system-state duality alluded to earlier, by allowing to use the same symbol (a circle representing a condition) at two different levels of (behavioural) abstraction to represent both a system and a state of an activity of that system. Given this, it is then possible to represent an evolving system, and to link appropriate activities to the appropriate versions of this evolving system. This is illustrated in Fig. 3, which uses dashed rectangles to delineate ONs, and portrays the pre- and post-upgrade history of an evolving computer system. The relation is portrayed by a link to the rectangular box enclosing, and hence identifying this set of states and events[3].

[2] `http://www.workcraft.org`.

[3] The above example portrays offline system evolution, in that there is no direct connection between the final state of the computers activity pre-evolution and the initial state post-evolution. (With online system evolution, the final state of an activity pre-evolution is taken as the initial state post-evolution).

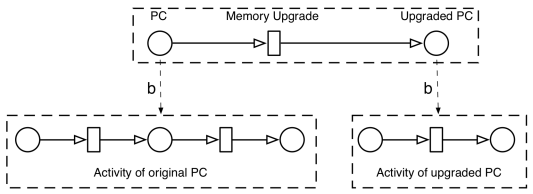
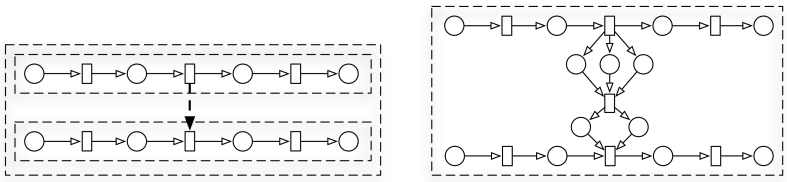


Fig. 3. Duality of systems and states, shown using behavioural abstraction.

Asynchronous communication relation. This relation states a temporal ordering between two events. An example of asynchronous communication between otherwise separate ONs is shown in Fig. 4(a), using a bold dashed arrow[4]. This communication might be very simple, or might in reality be much more complicated, involving sophisticated buffering or networked communication, as in Fig. 4(b).



(a) Abstract view

(b) A possible structured, more detailed, view of (a)

Fig. 4. Asynchronous communication relation

It is important to note that, while the behavioural relation adds expressive power in terms of reachable states w.r.t. plain ONs, the same is not true for communication relations (either asynchronous or synchronous [5]). These relations enable one to abstract away the details of interactions, should these not be regarded as relevant, and to use a set of relatively simple separate ONs in a conveniently structured representation of what would otherwise have to be shown as an unstructured and hence much more complex single ON.

[4] Note that such an arrow connects two events, whereas the directed arcs in a conventional ON connect an event to a condition or a condition to an event.

[5] Synchronous communication [KR09] is used to indicate that two events in separate ONs are perceived as occurring simultaneously. The fact that such a relation is undirected allows one to relax the rule that any ON (and any SON) must be an acyclic directed graph, without however violating conventional notions of causality. This relation is not used in this paper as so far it has not been used in provenance modelling.

Temporal abstraction relation. Temporal abstraction enables the abbreviation of part of an occurrence net, in such a way that some of its actions appear instantaneous to their environment, and yet at a different level of abstraction, they unfold into a sequence of condition-event pairs. One particular pattern involving temporal abstraction is shown in Fig. 5. In this pattern, event e appears instantaneous in the top view of the system, while it expands into multiple activities, namely e_1 and e_2, at the more detailed level at the bottom (the latter represents the temporary existence of an intermediate value a, for example). This pattern is useful when using events, which are instantaneous in ON, to model provenance traces that involve activities with a finite duration (see Section 3.4).

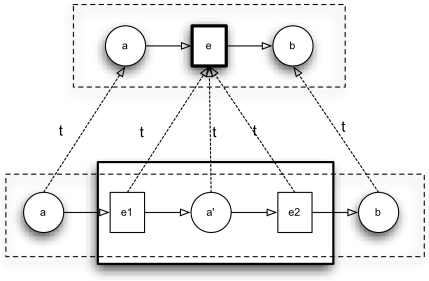


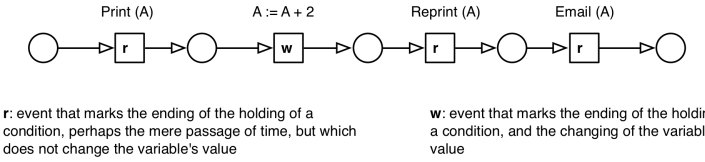
Fig. 5. SON pattern for temporal abstraction.

3 SONs modelling patterns for provenance.

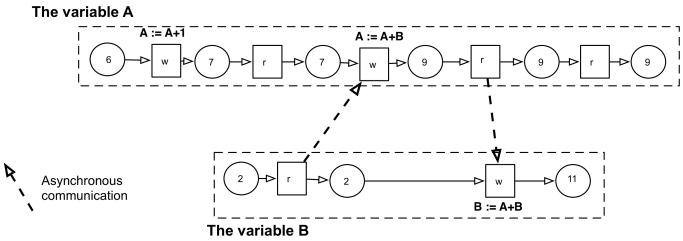
Here we propose, by means of examples, a set of modelling patterns that make use of SONs for representing the provenance of data associated to processes that are at least partially observable, possibly at multiple levels of abstraction.

3.1 Simple values manipulation and variable assignment.

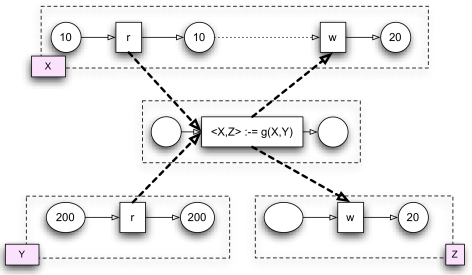
To focus the ideas, we begin with the simplest case of a sequence of operations that act upon data held in a single variable, shown in the ON of Fig. 6(a). As mentioned earlier, the labels associated to the events, i.e., 'r' for read, 'w' for write, are conventional and optional and have no formal meaning. In this example, they are used to clarify whether the events modify the state of the variable. Here the variable name is left implicit. For the more common case where multiple variables are involved, we propose the pattern of Fig. 6(b), consisting on multiple ONs, one for each variable, each labelled with the variable name and linked together by communication relations. For example, the graph in the figure captures the effect of the composed activity "A:=A+1; A:=A+B; B:=A+B" as a SON consisting of a pair of communicating ONs. This SON records how



(a) Single variable



(b) Two variables as interacting systems



(c) Function application changing the values of multiple variables

Fig. 6. Capturing the provenance of multiple variables

the various data read and write operations occurred, as well as their partial ordering, making it possible to trace the provenance of any particular recorded data value. (A more complex example could show actual use being made of the data obtained by all the various read operations). In each system included in this SON, the activities that occur during the system's lifetime are exposed, including interactions (asynchronous, in this case) with other systems. In this example, the two systems, for variables A and B, interact using read and write operations, denoted with r and w, respectively. Event A:=A+B in particular depends on the current state of B. This is represented by the asynchronous communication relation connecting the r event in B's activity, to the w event in A's activity. Similarly, the event B:=B+A receives the current value of A from A's SON to compute the new value for B. Note that conveying the state of the system to

another system is one of many possible read events that do not modify the state of the system (printing the value is another, shown in Fig. 6(a)[6]).

Expanding on this second example, consider a function application of the form: $\langle X, Z \rangle := g(X, Y)$, where g changes the value of one its arguments, as well as of a new variable Z. To capture its execution, we include an additional SON to represent the function g itself. The resulting pattern is shown in Fig. 6(c). One advantage of representing g as a system is that its own evolution can be captured as part of provenance, using behavioural abstraction. We show this feature in action later (Sec. 3.3).

3.2 Workflow fragments.

The pattern just illustrated in Fig. 6(c) is a stepping stone for modelling the provenance of data produced by dataflows [LP95], which provide the formal underpinning for a number of workflow systems used across e-science domains and applications [DGST09]. As a general definition, a dataflow is a graph whose nodes represent executable tasks, and directed arcs denote data flow dependencies between a source node (data producer) and a sink node (data consumer). A basic example is given in Fig. 7(a)[7]. Part (b) of the Fig. depicts one execution of this fragment.

The scientific workflow community has been amongst the earliest and most eager to support provenance recording of workflow outputs, motivated by the need to associate an evidence trail to valuable datasets which are destined for publication [Nek10,ABJF06,MMW11,MPB10,KSM+11]. Provenance is recorded by instrumenting the workflow enactment engine with suitable monitoring capability. A possible SON encoding of the execution of Fig. 7(b) appears in Fig. 7(c). Note that a choice has been made to model both workflow tasks (the invocation of functions f and g) as part of the same system, which represents the entire workflow execution. As an alternative, one can associate one SON *to each task*, a modelling choice that makes it possible to capture *the evolution of the tasks themselves*. This means that SONs can be used to seamlessly model both workflow execution traces, workflow tasks, and their evolution over time. Only a few other documented provenance data models, including Janus [MSZ+10] and OPMW [GG11] (both of which extend the Open Provenance Model [MCF+11]) support the modelling of the dataflow itself, in addition to its execution traces.

[6] A printing activity would involve communication with a separate printing system, however there is no obligation to represent such interaction, either because it is not of interest for tracing provenance, or because such level of detail is simply not available.

[7] This generic flowchart-like visual depiction is sufficient to illustrate the point of provenance, as it makes it clear which values are produced and consumed by which task block. A variety of visual languages are employed in actual systems.

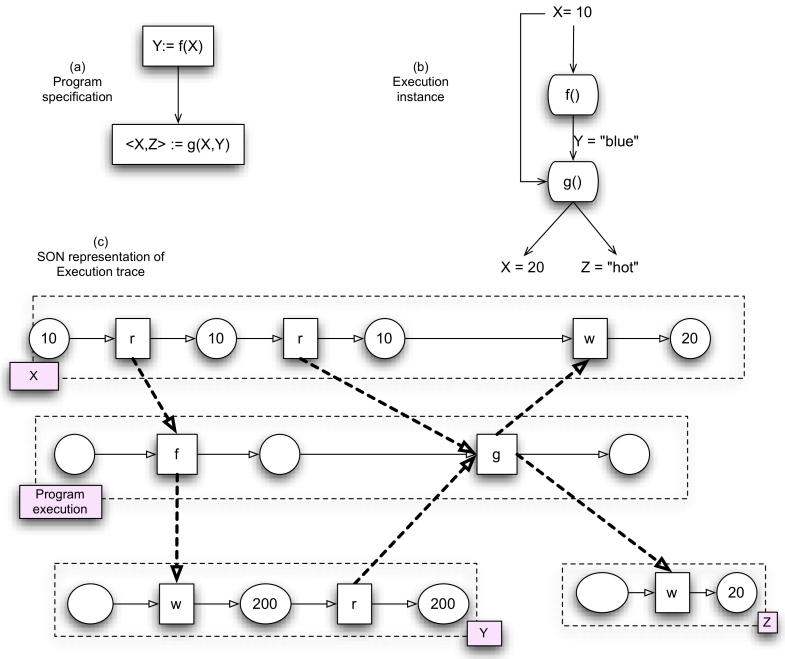


Fig. 7. Dataflow fragment, one execution, and SON portraying the execution trace.

3.3 Agents and their provenance

As mentioned in the introduction, one of the considerations that make SONs appealing for encoding execution traces, is the uniform representation of the evolution of the data and of the agents that are responsible for its manipulation, namely both as systems (in this setting, we use the term *agent* to refer, informally, to a system that performs the activities that account for changes in the state of the data). We have already made the point that knowledge of the state of agents, and of how that state evolves in response to interactions with other systems (including other agents), contributes to formulating sensible judgments regarding the reliabIlity of the data products under the agents' control.

Fig. 8 shows an example in which our actor Bob, who already featured in our earlier scenario (see Fig.2), collaborates with Alice in editing a document. The systems modelling follows our familiar pattern: the F SON captures the evolution of the file itself, according to activities that occur in two other SONs ("Bob" and "Alice"). The SON unambiguously models the following situation: *"Bob drafts version f1 of file F (he then goes on to perform other activities that are of no interest here). At some later point in time, Alice reads the draft f1 and leaves some comments as part of the same file. This results in a new version f2 of F. Later, Bob reads the comments (this leaves the file unchanged), then performs additional edits which result in new version f3."* This model makes it

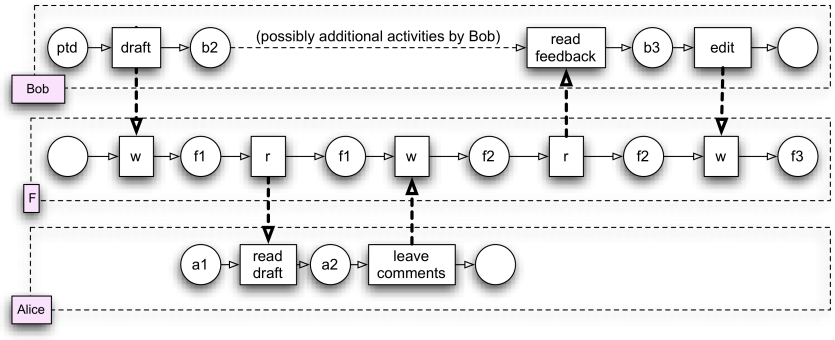


Fig. 8. Alice and Bob collaborate on document editing.

explicit that Bob does the edits after reading Alice's feedback, i.e., while he is in state b_3. Contrast this with an alternative model, shown in Fig. 9, in which Bob is unaware of Alice's comments when he performs the editing activity. Arguably, these two models may lead to different conclusions as to the quality of the final document.

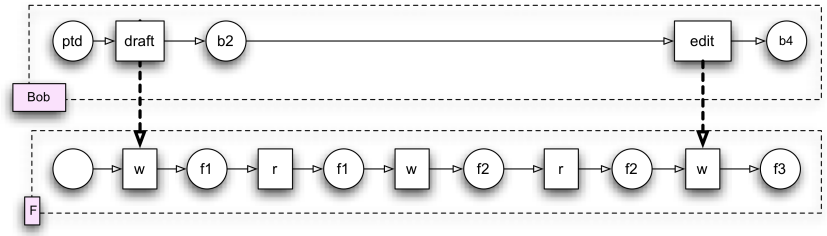


Fig. 9. Bob ignores Alice's comments.

An additional advantage of modelling agents within the SON framework, is that behavioural abstraction can be used to expand on the activities that correspond to an agent's state, thus revealing further details that may be relevant for judgment. This is shown in Fig. 10, where Bob's state `ptd` (preparing-to-draft) expands into a set of activities that describe the preparation phase (shown in Fig. 1 as our initial example). Note that we still do not have a complete picture of how the draft manuscript was produced: for example, we do not know whether the memo was actually used during the drafting of the manuscript. We can, however, easily add this additional information (if it is available at all) by explicitly modelling the internal memo itself as a system, and then adding appropriate communication edges amongst the SONs, using our familiar pattern.

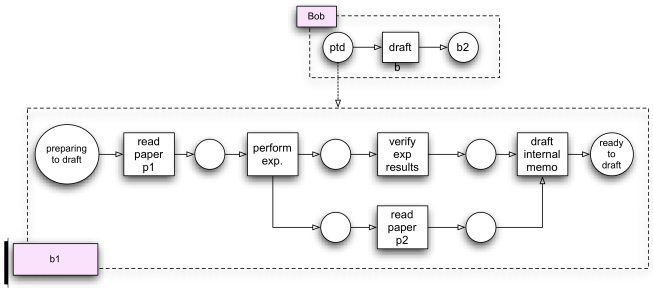


Fig. 10. Bob prepares to draft a manuscript.

3.4 Modelling activities with a finite duration

So far we have used ON events, which are by definition instantaneous, to model activities, ignoring that the latter generally span some finite, non-zero time duration. To reconcile this contrast, we introduce a further pattern which makes use of the temporal abstraction relation (see Sec. 2) as shown in Fig. 11.

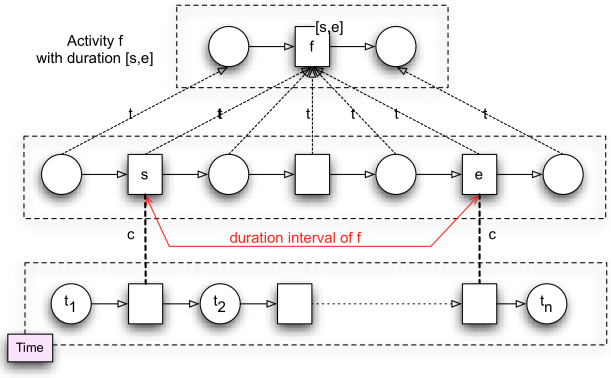


Fig. 11. Representing activities with explicit start and end events, and time.

The top ON in the figure includes a new shorthand notation to indicate that activity f is demarcated by start and end events s and e, respectively. This ON is mapped to the one in the middle by way of temporal abstraction arcs, following the (graphical) rules set out in Sec. 2. In turn, one can optionally introduce a new ON to represent a time line, and use *synchronous* communication relations[8] to

[8] This type of communication relation appears in [KR09] but has not been introduced here. It indicates that two events in separate ONs in fact are perceived as occurring simultaneously. Note that the fact that such a relation is undirected allows one to

associate a time to events s and e. Note that this convention leaves the freedom to introduce multiple time lines to account for events seen by different observers.

4 Conclusions

TBD

References

[ABJF06] I Altintas, O Barney, and E Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. In *IPAW*, pages 118–132, 2006.

[BD87] Eike Best and Raymond Devillers. Sequential and concurrent behaviour in Petri net theory. *Theoretical Computer Science*, 55(1):87–136, 1987.

[DGST09] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[GG11] Daniel Garijo and Yolanda Gil. A New Approach for Publishing Workflows: Abstractions, Standards, and Linked Data. In *Proceedings of the Sixth Workshop on Workflows in Support of Large-Scale Science (WORKS'11), held in conjunction with SC 2011*, Seattle, Washington, 2011.

[HSBMR08] D A Holland, M I Seltzer, U Braun, and K.-K. Muniswamy-Reddy. PASSing the provenance challenge. *Concurrency and Computation: Practice and Experience*, 20:531–540, 2008.

[KK11] Jetty Kleijn and Maciej Koutny. Causality in Structured Occurrence Nets. In Cliff Jones and John Lloyd, editors, *Dependable and Historic Computing*, volume 6875 of *Lecture Notes in Computer Science*, pages 283–297. Springer Berlin / Heidelberg, 2011.

[KR09] M. Koutny and B Randell. Structured Occurrence Nets: A Formalism for Aiding System Failure Prevention and Analysis Techniques. *Fundamenta Informaticae*, 97, 2009.

[KSM+11] David Koop, Emanuele Santos, Phillip Mates, Huy T Vo, Philippe Bonnet, Bela Bauer, Brigitte Surer, Matthias Troyer, Dean N Williams, Joel E Tohline, Juliana Freire, and Cláudio T Silva. A Provenance-Based Infrastructure to Support the Life Cycle of Executable Papers. *Procedia CS*, 4:648–657, 2011.

[LP95] E A Lee and T. M. Parks. Dataflow Process Networks. Memorandum 5, UC Berkeley EECS Dept, 1995.

[MCF+11] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van Den Bussche. The Open Provenance Model — Core Specification (v1.1). *Future Generation Computer Systems*, 7(21):743–756, 2011.

[MMW11] Anderson Marinho, Leonardo Murta, and Claudia Werner. ProvManager: a provenance management system for scientific workflows. *Concurrency and Computation: Practice and Experience*, pages n/a—-n/a, 2011.

relax the rule that any ON (and any SON) must be an acyclic directed graph, without however violating conventional notions of causality.

[MPB10] P. Missier, N. Paton, and K. Belhajjame. Fine-grained and efficient lineage querying of collection-based workflow provenance. In *Procs. EDBT*, Lausanne, Switzerland, 2010.

[MSZ+10] Paolo Missier, Satya S Sahoo, Jun Zhao, Amit Sheth, and Carole Goble. Janus: from Workflows to Semantic Provenance and Linked Open Data. In *Procs. IPAW 2010*, Troy, NY, 2010.

[Nek10] Anton Nekrutenko. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.

[Ran11] Brian Randell. Occurrence Nets Then and Now: The Path to Structured Occurrence Nets. In Lars Kristensen and Laure Petrucci, editors, *Applications and Theory of Petri Nets*, volume 6709 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin / Heidelberg, 2011.

[Sim08] Y L Simmhan Plale, B., Gannon, D. Karma2: Provenance management for data driven workflows. *International Journal of Web Services Research*, 5(1), 2008.

A A SON provenance example encoded using PROV

The PROV provenance model from the W3C Provenance Working Group[9] is centered on a few basic modelling elements: entities, activities, and agents. One can then assert that entity e was generated by activity a: `wasGeneratedBy(e,a)`, that agent ag was responsible for a: `wasAssociatedWith(a,ag)`, that a used entity e: `used(a,e)`, that an element el_1 (either an agent or an entity) was derived from another element el_2:`wasDerivedFrom(el1, el2)` (a few more relationships are available). These relationships are sufficient to model the agents' interaction pattern as well as a simplified notion of agents' evolution. Fig. 12(b) portrays one possible PROV encoding of a simplified version, in Fig. 12(a), of the interaction shown in Fig. 9. In this encoding, individual ON states are generally mapped to sets of PROV entities or agents. For example, states from the `F` system, f_2, f_3 become PROV entities, while individual states for agent `Bob` become agents `Bob_b2` and `Bob_b3`. Agent evolution is captured by associations such as `wasDerivedFrom(Bob_b3, Bob_b2)`, while data evolution through an activity is modelled using `used(edit,f2)` and `wasGeneratedBy(f3, edit)`. Responsibility of agent ag for activity a, which in the SON model appears as an activity within the agent's ON, becomes relation `wasAssociatedWith(ag, a)`.

A more complete discussion on mappings between SON and PROV modelling patterns is beyond the scope of this paper, and will appear in a forthcoming technical report.

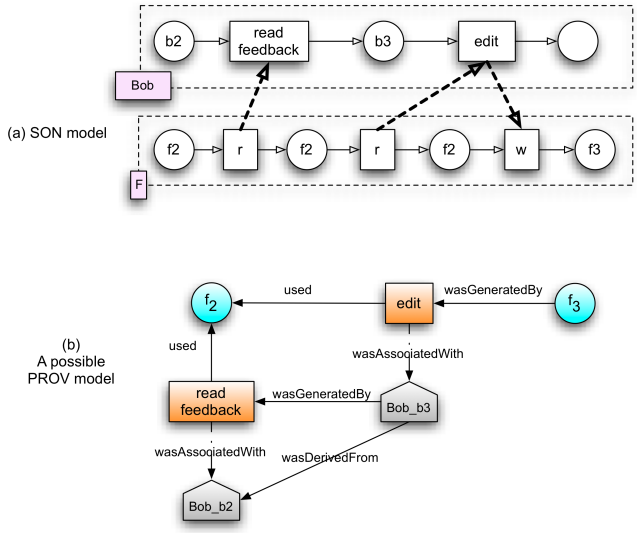


Fig. 12. SON and PROV model fragments for the document editing example.

[9] `http://www.w3.org/2011/prov/wiki/Main_Page`