Modelling Provenance Using Structured Occurrence Networks

Paolo Missier, Brian Randell, and Maciej Koutny

Newcastle University, School of Computing Science,
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 the 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 and 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]. In fact ONs are suitable for representing the activities of asynchronous systems whose design is expressed in various different notations, not just Petri Nets; indeed they have, since their invention in the 1970s, been re-invented, and re-named, by many different research communities, e.g. as "strand spaces" by security researchers [KR09], and as "message sequence charts" [HT04] by networking researchers. Moreover, they can be used for modelling the observed or envisaged behaviour of systems whose design is not available, eg. the undocumented process of papers selection and review associated with some publications.

P. Groth and J. Frew (Eds.): IPAW 2012, LNCS 7525, pp. 183–197, 2012.
© Springer-Verlag Berlin Heidelberg 2012

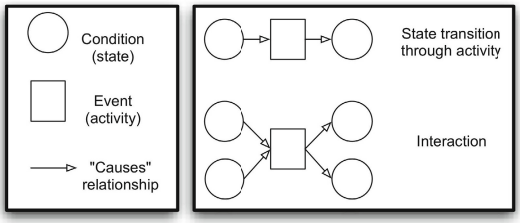


Fig. 1. Basic graphical ON notation

In this paper we show how SONs provide a suitable formal grounding to express the provenance of data that 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 completely avoid formal definitions, which can be found in [KR09]. As shown in Fig. 1, the basic ON formalism is very simple. Circles represent conditions (i.e. the 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 rules, portrayed in the right part of the figure (see Def. 1 in [KR09]): events have at least one incoming arc and one outgoing arc (top in the figure) and states have at most one input and one output arc (bottom).

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. This process may have been pre-defined, but it could also have been merely observed. 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".

An ON is thus simply a means of recording what is observed or believed to have 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

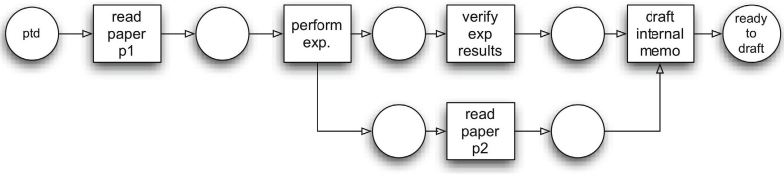


Fig. 2. Simple ON example

given ON represents the (possibly asynchronous) activity of a single un-identified "system" (whose design may or may not have been known). 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) *communication* relationships to specify interactions amongst systems; (ii) *behaviour abstraction* relationships, which provide 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) *temporal abstraction* relationships by which events that appear instantaneous at one level of abstraction, unfold into complex state-event nets at another level. 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.

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.

1.1 Benefits and Limitations

Some of the benefits expected from this work include 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, including checking whether a temporal logic formula is satisfied, or whether a specific state (or set of states) can ever be reached. This, however, is beyond 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[1]. 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, ONs, gate-level circuits, Static Data Flow Structures and

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

Conditional Partial Order Graphs. Support for SONs that make use of communications relations has recently been added.

1.2 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, mentioned in Sec. 3.2. While all of these have been developed with particular 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[2]. PROV follows in the steps of the Open Provenance Model [MCF+11].

The PROV approach to modelling provenance is based on the general concept of *entity*, an abstraction for anything that may have a provenance record associated to it, and their relationships to *activities*, which are capable of generating and using entities, and *agents* (including both humans and computer programs) which are responsible for carrying out the activities. A PROV statement is a fact that relates entities to other entities or to activities and agents. For example, one may state that entity e_1 *was derived from* another entity, e_2. A collection of such facts forms a graph of relations that represents an observer's *account* of past interactions amongst the elements mentioned in those facts.

PROV and SON are only superficially similar in the way they represent provenance, differing in at least three main aspects. Firstly, in PROV the notion of causality is deliberately avoided (facts are asserted based on observations or on any form of background knowledge, which is not made explicit), while it is central to the ON and thus the SON models. Secondly, SONs are naturally suited for *modelling the evolution of agents*, because those are simply modelled as systems, a point that is made more concretely in Sec. 3.3. In PROV, modelling agents' evolution is possible in principle, as agents can be viewed as entities themselves, however this involves overloading, or perhaps specializing, the meaning of the `generation` relation (i.e., one could assert that a new version of an agent was "generated by" an activity carried out by the previous version of the same). Finally, and perhaps more importantly, as we will see in the next section SONs extend ONs by introducing a set of relations that provide different forms of abstraction (communication, temporal, behavioural, and spatial). This in turn makes it possible for different abstractions over the same provenance facts to co-exist in the same SONs. In contrast, PROV only defines a single flat space of facts, and completely lacks any mechanism for abstraction[3].

A more detailed comparison between PROV and SON-based provenance regarding these three aspects can be found in Sec. 4. A formal account of the differences between the two approaches is, however, beyond the scope of this paper.

[2] PROV will become a W3C recommendation by the end of 2012. The current working draft can be found here: `http://www.w3.org/TR/prov-dm/`

[3] One can, however, arbitrarily group facts into *bundles*, which can be nested.

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. 5 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 one or more of a number of different types of relations [KR09]. Here we will make use of just three types of relation, namely behaviour relations, (asynchronous and synchronous) communication relations, and temporal abstraction relations. 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 the use of 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[4].

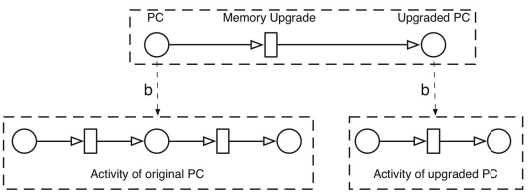
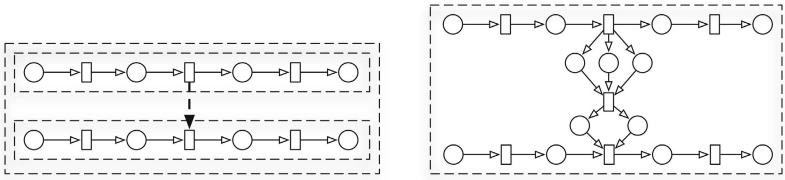


Fig. 3. Duality of systems and states, shown using behavioural abstraction

[4] The example refers to a hardware evolution, but it could have equally been a software upgrade. Also, the above example shows offline system evolution, in that there is no direct connection between the final state of the computer's activity pre-evolution and the initial state post-evolution. In contrast, one can use online system evolution, where the final state of an activity pre-evolution is taken as the initial state post-evolution.



(a) Abstract view

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

Fig. 4. Asynchronous communication relation

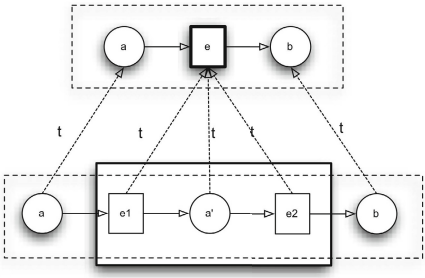


Fig. 5. SON pattern for temporal abstraction

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[5]. 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).

Asynchronous or synchronous relations[6] 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.

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

[5] 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.

[6] 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 used later in the paper to model activities with a finite duration (Sec. 3.4).

unfold into a possibly lengthy and complex asynchronous activity. 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 events, 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).

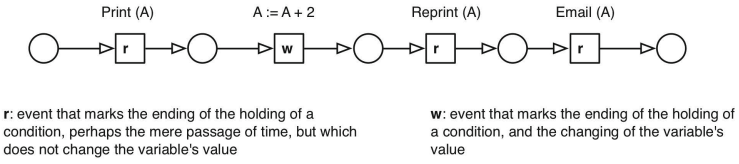
3 SONs Modelling Patterns for Provenance

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.

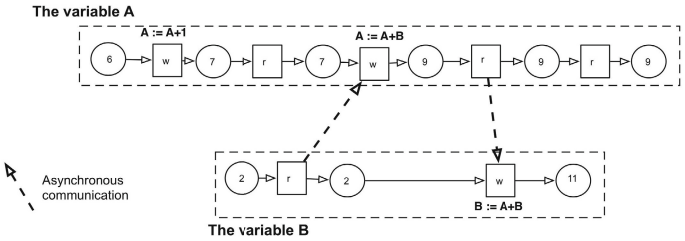
3.1 Simple Values Manipulation and Variable Assignment

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 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 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. Event A:=A+B in particular depends on the current state of B as well as the state of A. 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)[7]).

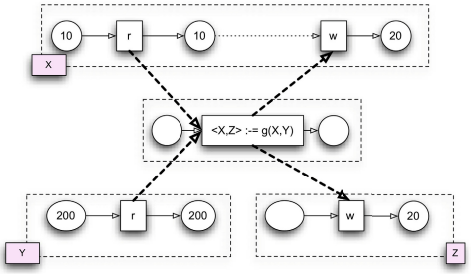
[7] A printing activity would involve communication with a separate printing system, however there is no obligation to represent such interaction, either because it is not of interest for tracing provenance, or because such level of detail is simply not available.



(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

Expanding on this second example, consider a function application of the form: $\langle X, Z\rangle := g(X, Y)$, where g doubles the value of its first argument, 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

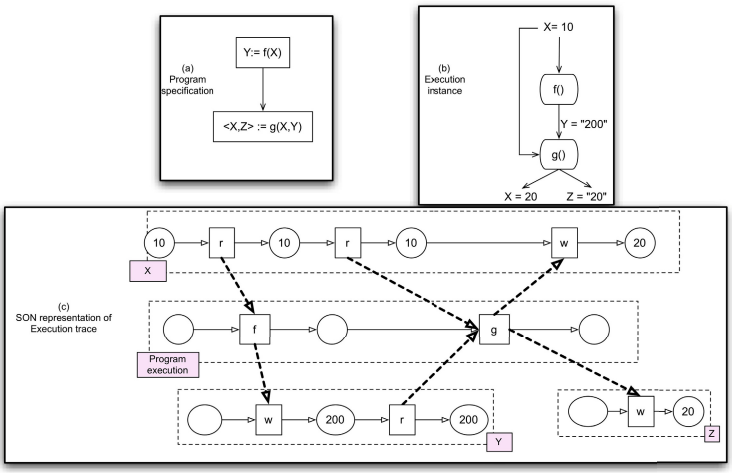


Fig. 7. Dataflow fragment, one execution, and SON portraying the execution trace

applications [DGST09]. A dataflow is a system design (a program) in the form of a graph whose nodes represent executable tasks, and directed arcs denote data dependencies between a source node (producer) and a sink node (consumer). A basic example is given in Fig. 7(a)[8]. Part (b) depicts one execution of this system. 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 [ABJF06, MMW11, MPB10, KSM+11]. Provenance is recorded by instrumenting the workflow enactment engine with suitable monitoring
capability.

A possible SON representation of an 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 could associate one SON *to each task*, a modelling choice that makes it possible to capture *the evolution, eg. represented by software updates, 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]) and [LLCF10] support the modelling of the dataflow itself, in addition to its execution traces. The VisTrails provenance model [SVKF08] is, to the best of our knowledge, the only model that can describe the evolution of the dataflow, as a tree of its versions.

[8] This is a simplified flowchart-like visual depiction. A variety of visual languages are employed in actual systems.

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, such as a computer or a person, 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.

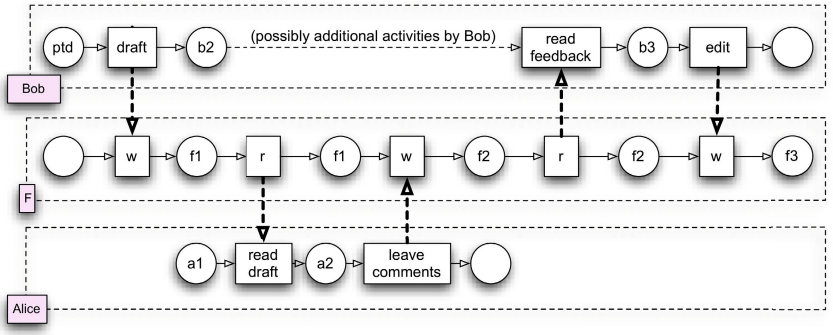


Fig. 8. Alice and Bob collaborate on document editing

Fig. 8 shows an example in which an actor Bob collaborates with Alice in editing a document. The systems modelling follows our familiar pattern: the F ON captures the evolution of the file itself, according to activities that occur in two other ONs ("Bob" and "Alice"). The overall 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 the new version f3."* This model makes it 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.

An additional advantage of modelling agents within the SON framework (as opposed to other approaches, including PROV) is that behavioural abstraction can be used to expand on the activities that correspond to an agent's state,

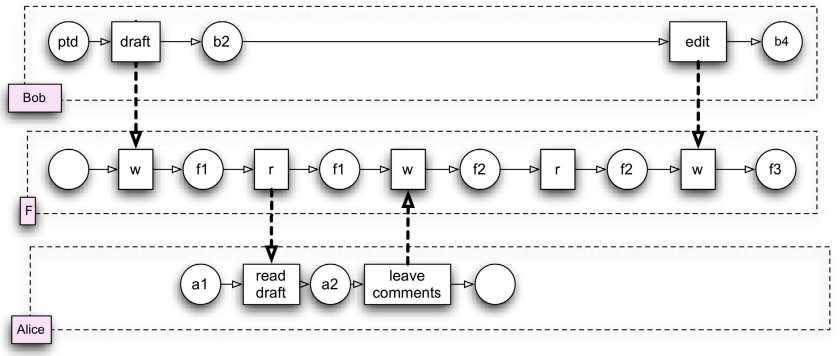


Fig. 9. Bob ignores Alice's comments

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 (assuming it is available) by explicitly modelling the internal memo itself as a system, and then adding appropriate communication relations amongst the SONs, using our familiar pattern.

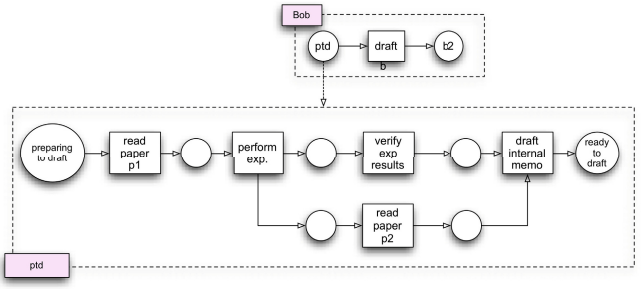


Fig. 10. Bob prepares to draft a manuscript

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.

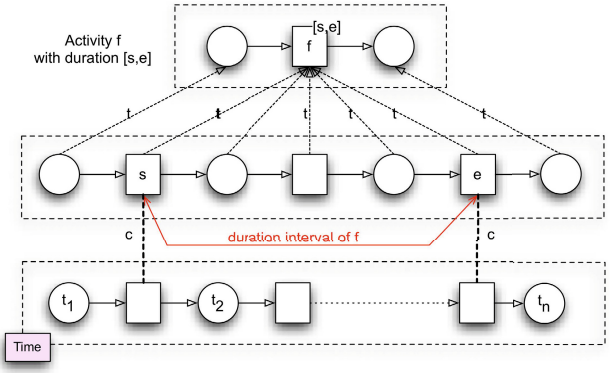


Fig. 11. Representing activities with explicit start and end events, and time

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 relations, 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 to associate a time to events s and e. This type of communication relation appears in [KR09]. It indicates that two events in separate ONs in fact are perceived as occurring simultaneously. Note that this convention leaves the freedom to introduce multiple time lines to account for events seen by different observers, possibly using differing clocks.

4 SON and PROV

We conclude our overview of SON-based provenance modelling with an informal comparison with the PROV provenance model, from the W3C Provenance Working Group[9]. We have already remarked earlier (Sec. 1.2) that PROV is not designed to support multiple abstractions over provenance. In contrast, SONs do this by supporting an explicit dual view of states and systems, which we have described in the paper. We also remarked, in the same section, that it is possible in PROV to model at the same time the evolution of data and of the agents who are responsible for it, a feature we have argued for, but that this involves overloading some of the PROV relations. Indeed, one can assert that agent `ag` was responsible for activity `a`: `wasAssociatedWith(a,ag)`, that entity `e` was generated by `a`: `wasGeneratedBy(e,a)`, and because agents can be viewed as entities, which are therefore are entitled to their own provenance, `wasGeneratedBy(ag,a)` is a valid assertion, too. This makes it possible to encode (a simplified version of) the SON fragment of Fig. 9 (reproduced in Fig. 12(a)), as shown in Fig. 12(b).

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

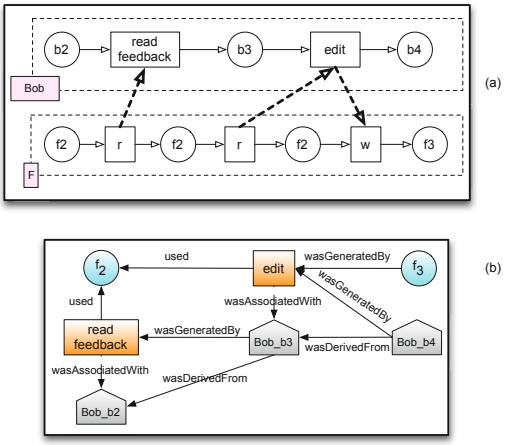


Fig. 12. SON and PROV model fragments for the document editing example

In this encoding, ON states are mapped to PROV entities or agents. In particular, states for agent `Bob` become agents `Bob_b2`, `Bob_b3`, and `Bob_b4`. Data evolution through an activity is modelled by `used(edit,f2)` and `wasGeneratedBy(f3, edit)`. Because modelling system evolution is not part of PROV, agent evolution must be encoded using relations such as `wasGeneratedBy(Bob_b4, edit)`, `wasDerivedFrom(Bob_b4, Bob_b3)`. However, `f3` and `Bob_b4` are now both "outputs" of `edit`, despite their different nature and role, a confusion that can only be resolved by adding properties to the `wasGeneratedBy` relations themselves[10].

Other differences concern the use of system communication along with state changes, a SON feature that is missing in PROV and that makes it possible, for example, to expand the communication links into complex system interactions.

5 Conclusions

In this paper we have presented an initial exploration into the use of Structured Occurrence Nets as a model for describing the execution traces of interacting asynchronous systems, and thus as a manifestation of the provenance of data produced and consumed by those systems. Provenance analysis informs the formulation of judgments regarding the quality and reliability of data products. SON-based modelling of provenance makes it possible to view agents, in addition to the data, as evolving interacting systems. This is a distinctive feature of this modelling approach, which leads to potentially more accurate judgments

[10] Additonal subtleties make this pattern less than natural: in PROV, from `wasDerivedFrom(e2,e1)` one can infer the existence of an entity `e` that used `e1` and generated `e2`. This would be `edit`, which would therefore both use `Bob_b3` and be associated with it.

as the state of agents (programs, or humans) are taken seamlessly into account. In addition, the formal grounding of Occurrence Nets provides a foundation for provenance validation and analysis.

We have presented a number of modelling patterns as an informal demonstration of the capabilities of the model, and Sec. 4 shows an example of how it compares with the W3C PROV modelling language for provenance. A more formal account of the provenance model, as well as a rigorous comparison with PROV, are left for further work.

References

[ABJF06] Altintas, I., Barney, O., Jaeger-Frank, E.: Provenance Collection Support in the Kepler Scientific Workflow System. In: Moreau, L., Foster, I. (eds.) IPAW 2006. LNCS, vol. 4145, pp. 118–132. Springer, Heidelberg (2006)

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

[DGST09] Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-Science: An overview of workflow system features and capabilities. Future Generation Computer Systems 25(5), 528–540 (2009)

[GG11] Garijo, D., Gil, Y.: 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 2011), held in conjunction with SC 2011, Seattle, Washington (2011)

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

[HT04] Harel, D., Thiagarajan, P.: Message Sequence Charts. In: Lavagno, L., Martin, G., Selic, B. (eds.) UML for Real, pp. 77–105. Springer US (2004)

[KK11] Kleijn, J., Koutny, M.: Causality in Structured Occurrence Nets. In: Jones, C.B., Lloyd, J.L. (eds.) Dependable and Historic Computing. LNCS, vol. 6875, pp. 283–297. Springer, Heidelberg (2011)

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

[KSM+11] Koop, D., Santos, E., Mates, P., Vo, H.T., Bonnet, P., Bauer, B., Surer, B., Troyer, M., Williams, D.N., Tohline, J.E., Freire, J., Silva, C.T.: A Provenance-Based Infrastructure to Support the Life Cycle of Executable Papers. Procedia CS 4, 648–657 (2011)

[LLCF10] Lim, C., Lu, S., Chebotko, A., Fotouhi, F.: Prospective and Retrospective Provenance Collection in Scientific Workflow Environments. In: 2010 IEEE International Conference on Services Computing (SCC), pp. 449–456 (July 2010)

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

[MCF+11] Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E., Van Den Bussche, J.: The Open Provenance Model — Core Specification (v1.1). Future Generation Computer Systems 7(21), 743–756 (2011)

[MMW11] Marinho, A., Murta, L., Werner, C.: ProvManager: a provenance management system for scientific workflows. Concurrency and Computation: Practice and Experience, n/a–n/a (2011)

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

[MSZ+10] Missier, P., Sahoo, S.S., Zhao, J., Goble, C., Sheth, A.: *Janus*: From Workflows to Semantic Provenance and Linked Open Data. In: McGuinness, D.L., Michaelis, J.R., Moreau, L. (eds.) IPAW 2010. LNCS, vol. 6378, pp. 129–141. Springer, Heidelberg (2010)

[Ran11] Randell, B.: Occurrence Nets Then and Now: The Path to Structured Occurrence Nets. In: Kristensen, L.M., Petrucci, L. (eds.) PETRI NETS 2011. LNCS, vol. 6709, pp. 1–16. Springer, Heidelberg (2011)

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

[SVKF08] Scheidegger, C.E., Vo, H.T., Koop, D., Freire, J.: Querying and Re-Using Workflows with VisTrails. In: Procs. SIGMOD, pp. 1251–1254 (2008)