Towards automated provenance collection for experimental runs of agent-based models

No Author Given

No Institute Given

Abstract. We demonstrate a working framework for the automatic recording of provenance and metadata for primarily agent-based models that could easily be adapted to the other modelling environments. We discuss the need for such a framework, the philosophy behind the design we adopted, the implementation, discuss the results and demonstrate a simple tool for for tracing bad data through a provenance graph.

Keywords: Provenance · metadata · modelling · automation · replication.

1 Introduction

Replication of social simulation results has been highlighted as a significant issue for the agent-based modelling (ABM) community for a number of years (e.g. [6]). The paper that forms the basis of this work ([16]) shows that the analysis of the outputs of the model can potentially be just as complex a process as developing and running the model itself. Analysis of outputs is no less difficult to replicate unless adequate records are kept. The TRACE protocol [21, 2] provides some guidance highlighting the need to keep a notebook of the analysis done and a standardised approach to making that notebook. There are also many scientific workflow tools, such as Snakemake [11], NextFlow [5], Kepler [12] and Taverna [9]. However, since these are workflow tools, the focus is on automation and repeatability rather than provenance, which, if it is included, is as an afterthought.

One of the lessons learned from the replication exercise in [16] was that, for the purposes of replication, more detailed guidance on the information that should be recorded is needed. Since recording such metadata is tedious (and error-prone) for humans, any such guidance should be accompanied by specification of tools that could be used to support the process. In the ideal world, the process of recording provenance metadata would be completely automated, essentially providing a complete graph from data (including their sources) through applications (model, scripts and analysis tools, including versions thereof) to result.

The output analysis replication in this paper concerns earlier work with FEARLUS-SPOMM, which is a coupled ABM of agricultural decision- making and species stochastic patch occupancy metacommunity model that has been used to explore incentivisation strategies to improve biodiversity [18, 8]. Belonging to the 'typification' class of social simulations [3], this work involved the analysis of the outputs of approximately 20,000 runs of the model using a number of techniques aimed at demonstrating nonlinearities in the relationship between incentivisation and biodiversity outcome. Recording provenance metadata on the process used to analyse and visualise the outputs is challenging, and currently there are no codified standards as to how this should be done for ABMs. For FEARLUS-SPOMM, the analysis and visualisation methods used drew heavily on statistical techniques available as R packages. Although R allows transcripts of interactive terminal sessions to be saved, the work involved great deal of exploration of alternative analyses and visualisations, not all of which were reported in the manuscript as finally accepted. Such transcripts are therefore not the best way to record the means by which a model's outputs were analysed, and hence the strategy used was to save each analysis or visualisation in a(n R) script. Since the output from the (Swarm) model software used a mixture of text formats, some Perl scripts were also written to process that output into CSV format for easy import into R. When the MIRACLE project [15] provided a context in which the replication of that analysis was necessary, an opportunity was created to test the viability of the strategy of relying on scripts to record provenance.

'Multifinality' (the same initial conditions and parameter settings having qualitatively different results) in ABMs means we need to ensure that reported results are not just down to a matter of chance. This is one of many reasons (e.g. in empirical contexts especially, calibration, validation and sensitivity analysis) why experiments with ABMs involve large numbers of runs. The tedium (and in larger-scale cases, infeasibility) of conducting each run by hand means most ABM experiments resort to some kind of automation, including of the kind provided by workflow tools mentioned above, but also using built-in features of ABM software (e.g. in the case of NetLogo, BehaviorSpace). This is fine if all we want to do is repeat the same process, but what if we are interested in why a particular instance of that process led to an unexpected outcome? Re-executing the workflow will not necessarily generate *exactly* the same result unless we have a record of everything needed to do that (including seeds for pseudo-random number generators). We refer to this as the 'automation and replication' problem.

The automation of experiments is already solved by *scripting*. A script – in a scripting language such as Python, Julia, or Perl, but more typically an operating system 'shell' language such as Bash – records the sequence of programs that need to be executed to reproduce the experiment. This at least repeats the experiment, rather than *precisely* replicating a specific set of results. We believe that anybody who does experiments with their agent-based models should at the very least be scripting most of their experiments. In our experience the preparation and the execution of the experiment is relatively straightforward to script. Post-processing to the final data tables, diagrams, etc., that appear in manuscripts and other documents should be included in these scripts. These scripts can be conceived as *workflow* metadata – the means by which a *class* of outputs is achieved, rather than a specific instance.[1]

Provenance metadata is used to record the means by which a specific output is achieved, which is more suitably stored in a database. That database is updated using a series of scripts written in Python, allowing us to develop on laptops and run the resultant code in high-performance computing environments with little or no reconfiguration. Bash scripts automating the experiment workflow are then modified to call the Python scripts to update the provenance database. We implemented the database in Sqlite3 for local development and PostgreSQL in high-performance infrastructure. In this way, we have created a provenance tool that can record provenance during workflow execution.

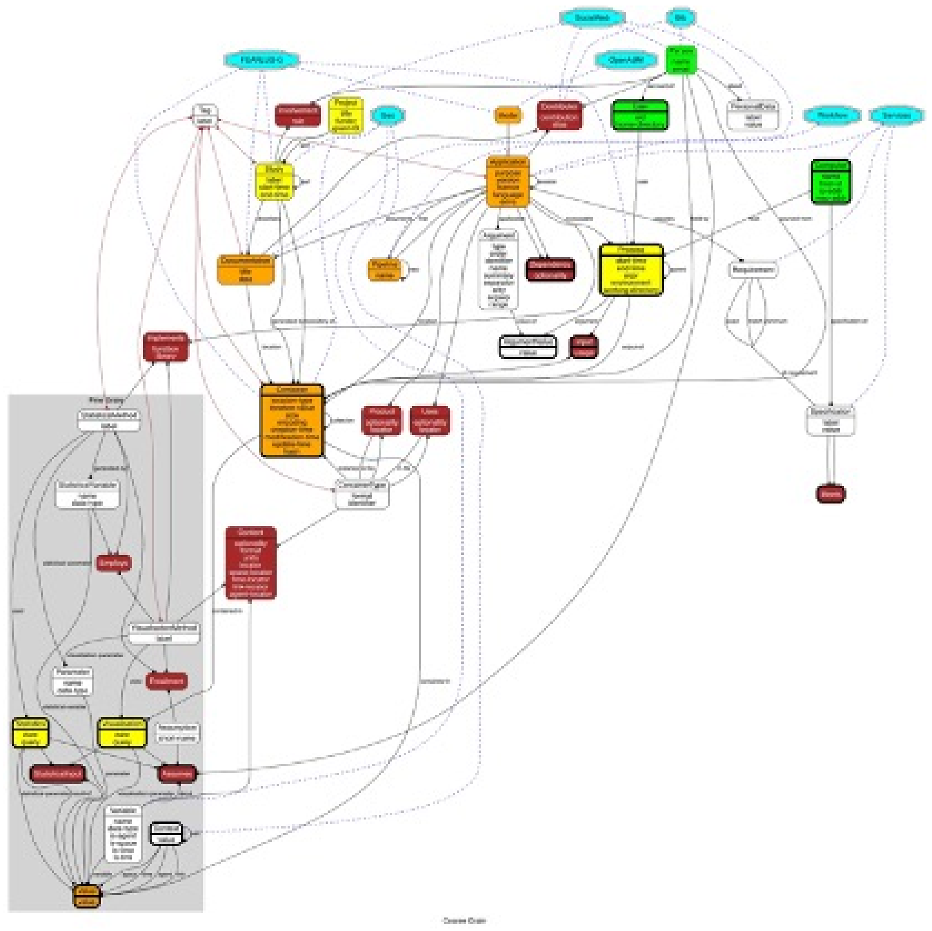

Fig. 1. SSREPI Schema

[1] In the case that all of the programs executed are deterministic, the class is a singleton.

Another project adopting a similar approach is the `fair` data pipeline [14], in which the provenance is automatically recorded by embedded model code in R, C++, Java, Julia and Python. However, one of our requirements was not to modify the model code if this could be avoided.[2] In the example above, the FEARLUS-SPOMM model is written in Objective-C [10]; with additional supporting Perl and R scripts to prepare the data and conduct the post analysis. We would also prefer not to modify these scripts. Instead, we wrote wrapper scripts in Bash [19]. A further observation is that our framework generalises to more source metadata (papers, data, other experiments), and is written with the requirements of agent-based modelling in mind first and foremost.

In the rest of this paper, we describe the scripting tool we have developed for automatically recording metadata, which can be incorporated into the analysis replication process, and how this was used to regenerate some of the figures in [18]. In addition we show a simple tool we have already developed to trace data through the provenance graph, and suggest additional work we would like to pursue.

2 The `SSREPI` provenance and workflow tool

One of the artefacts of the MIRACLE project [15] was the Social Simulation REplication Interface or `SSREPI`. This is the schema shown in figure 1. This schema has evolved since it original conception; [17] is the latest version. The schema is derived from the Dublin Core [24] and PROV-O ontologies [13] among others, and is designed with agnosticism towards the underlying database technology, having been implemented in PostgresSQL [22] and Sqlite3 [4].

Besides the distinction between workflow and provenance described above, we can discriminate fine- versus coarse- grained metadata. Coarse-grained metadata describes how particular files come (or came) into being, or were (or could be) used to bring other files into being. Fine-grained metadata describes specific values recorded in social simulation outputs. To make the distinction concrete, suppose a simulation produces a CSV file. The data within the CSV file are covered by fine-grained metadata, whilst the fact that the simulation produces the CSV file is coarse-grained.

To build `SSREPI`, each table in schema shown in fig. 1 was coded as an object type in Python. Each table row was represented as an instance of such an object. This design approach was adopted to enforce a consistent coding methodology across all tables. To these were added a few simple coordinating commands that could be called from Bash:

- `create_database.py` - creates a database idempotently.
- `exists.py` - checks if a particular row in a table exists.
- `get_value.py` - gets any specified single value from a table given the primary key.
- `get_values.py` - gets one or more rows given the search values.

[2] If the model code did not output the seed it used, then it would need to be modified.

– `next_study.py` - gets the next available and unique study number.
– `update.py` - idempotently updates a particular row in a table.

'Idempotent' indicates that multiple operations on the same entity will leave it unchanged after the initial operation. For example multiplying something by 1 is idempotent. This allows for less rigorous exception criteria, but implies that initialisation must be performed with care, as older data will not necessarily be destroyed when overwritten with newer values, but retain any values that already exist.[3]

Each of the Bash commands listed in table 1 composes over the above Python commands to populate the `SSREPI` database in a consistent and logical manner. Broadly speaking, `SSREPI_application`, `SSREPI_run`, `SSREPI_batch`, `SSREPI_input`, `SSREPI_output` and `SSREPI_argument` are responsible for recording coarse grain provenance. `SSREPI_value`, `SSREPI_visualisation_variable_value`, `SSREPI_statistical_variable_value`, `SSREPI_run` and `SSREPI_batch` record fine-grain provenance. The remaining primitives are largely about recording other metadata. Further information may be found in the tool's public repository.

Using `SSREPI` for existing workflow scripts is a somewhat laborious task entailing adding calls to the provenance metadata scripts in Table 1. Improvements to the 'interface' is the subject of future work.

3 Demonstration

Here, we demonstrate the use of `SSREPI` by repeating the experiment in the paper [16], showing some of the workflow and provenance graphs, and then using fictitious examples to apply the workflow and provenance metadata to addressing issues in large-scale experiments. From a workflow perspective, the main result is that the principal diagrams in that paper were successfully recreated, with the same, albeit not identical, results.

After a run had completed successfully the database was run through several supplementary programs to produce the following visualisations of the provenance metadata recorded:

– **Analysis** - fine grain provenance pertaining to statistical and visualisation outputs.
– **Finegrain** - a provenance diagram down to the level of variables.
– **Folksonomomy** - a diagram showing annotations against the database, produced and categorised at the discretion of the user doing the annotation
– **Project** - management metadata. Largest granularity of metadata supported
– **Provenance** - provenance diagram at the level of file and parameter
– **Services** - service provided and requirement description
– **Workflow** - the actual workflow

[3] This is a design decision that may need revisiting.

Primitive	Type	Purpose
SSREPI_require_minimum	M	Lower bound on software hardware required
SSREPI_require_exact	M	Exact bound on software hardware required
SSREPI_application	P & M	specifies some executable
SSREPI_me	P & M	Determines executable being run or returns a proper reference to the executable being run.
SSREPI_argument	P	An argument type to an executable
SSREPI_output	P	An output type from an executable
SSREPI_input	P	An input type for an executable
SSREPI_person	M	Provide metadata for a particular actor within this system
SSREPI_project	M	Specifies a project which contains all studies
SSREPI_study	M	A set of experiments makes up a single study
SSREPI_set	M	Sets the default licence and other metadata
SSREPI_involvement	M	Links personnel to a study
SSREPI_paper	M	A paper associated with this study
SSREPI_make_tag	M	Used for building a folksonomy
SSREPI_tag	M	Used to tag any entity with a folksonomy tag
SSREPI_contributor	M	A person with some kind of relation to an executable or script.
SSREPI_statistical_method	M	Record a statistical method
SSREPI_visualisation	M	Record a method to create an image to depict one or more values.
SSREPI_statistics	M	Record activities that compute and populate the values of statistical variables.
SSREPI_visualisation_method	M	Methods for generating visualisations.
SSREPI_implements	M	Links a statistical or visualisation method to an application
SSREPI_parameter	M	Record the name of a parameter taken by a statistical or visualisation method.
SSREPI_statistical_variable	M	A name for (one of) the result(s) of a statistical method.
SSREPI_visualisation_variable	P & M	Declares a named variable of interest
SSREPI_variable	M	Names a variable of interest
SSREPI_statistical_variable_value	P & M	Sets an actual value for a named statistical variable
SSREPI_value	P	Sets a value.
SSREPI_content	M	Links a kind of output/input/argument to a variable
SSREPI_person_makes_assumption	M	Links a person to an assumption

Table 1. SSREPI's Bash commands for provenance (Type column entry 'P') and other metadata (Type 'M') recording

The supplementary programs use the Dot language for input to Graphviz[7], producing the diagrams such as those in figures 3, 2, 4 and 5. The workflow graph is in fig. 2. The resultant provenance graph is massive, since there were 20,000 runs (so 20,000 sets of outputs to record), so we only show a *very small* section of the provenance graph in fig. 3.

So what use is this provenance metadata? Imagine for the purposes of illustration that we have found a bug in a script. The entity `Applications.application_3831436655` is an experiment setup script `SSS-StopC2-Cluster-create.pl`. We can use the workflow graph to see what else in the workflow might be affected. As might be expected for an experiment setup script, there are serious cascading consequences, which we can visualise in Figure 4. The Dot language used by Graphviz constitutes a primitive graph database. Indeed there are programs that can transform Dot files into TinkerPop GraphSON format [23, 1]. To generate Figure 4, we used the workflow visualisation Dot file rather than the relational database.

Similarly, the provenance metadata can be used to check the damage caused to a large-scale experiment by a single bad dataset. In the real experiment, the input files are generated by running a script, but for the purposes of the example, suppose the data in `Containers.container_42949672955` is wrong. The data in this file forms a input file to a handful of runs, and we want an idea of how it has affected our results. We can visualise the propagation of the error in a *very small* section of of the provenance graph in fig. 5, but we can use the database to list the entities affected:

```
Applications.application_3450918915 Applications.application_648609270
Computers.asterix.local             Containers.container_1814970370
Containers.container_1982026419     Containers.container_2050039078
Containers.container_2056384913     Containers.container_2060874102
Containers.container_2387213333     Containers.container_2486610989
Containers.container_2582525701     Containers.container_2759060318
Containers.container_2865400753     Containers.container_3025688835
Containers.container_3307537171     Containers.container_3470971297
Containers.container_354343442      Containers.container_4235735972
Containers.container_4294967295     Containers.container_441913555
Containers.container_505627104      Containers.container_800277554
Containers.container_878886043      Containers.container_900718909
Persons.doug                        Persons.doug_salt
Processes.process_232221298886493...Processes.process_326475499597022...
Specifications.R                    Specifications.bash
Specifications.cpus                 Specifications.disk_space
Specifications.memory               Specifications.os
Specifications.perl                 Specifications.python
Studies.1                           Users.doug
```

Having done so, we are then in a position to fix the data file, and rerun only that part of the workflow needed to regenerate the affected containers.

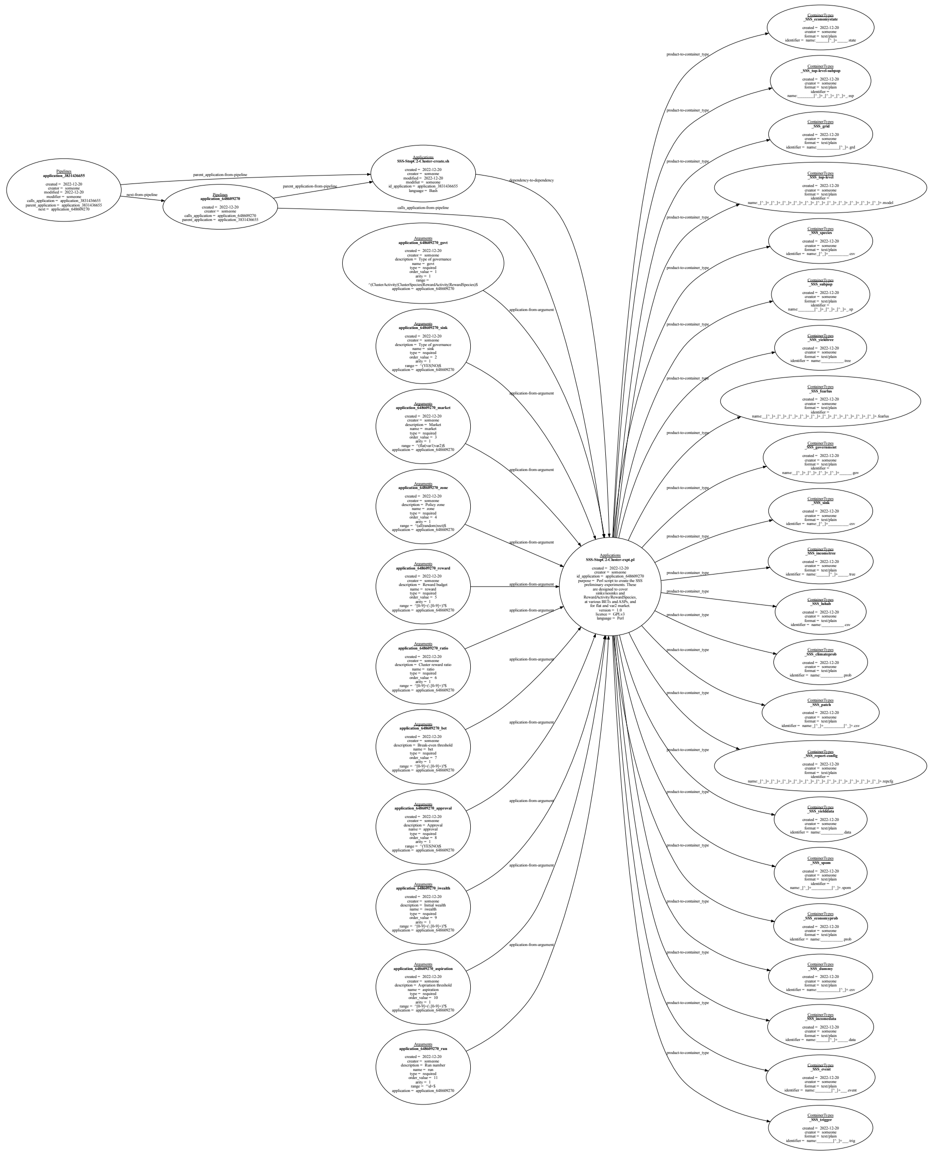
Pipelines
Applications
Arguments
ContainerTypes
parent_application-from-pipeline
next-from-pipeline
calls_application-from-pipeline
dependency-to-dependency
application-from-argument
product-to-container_type

Fig. 2. The workflow sub-graph

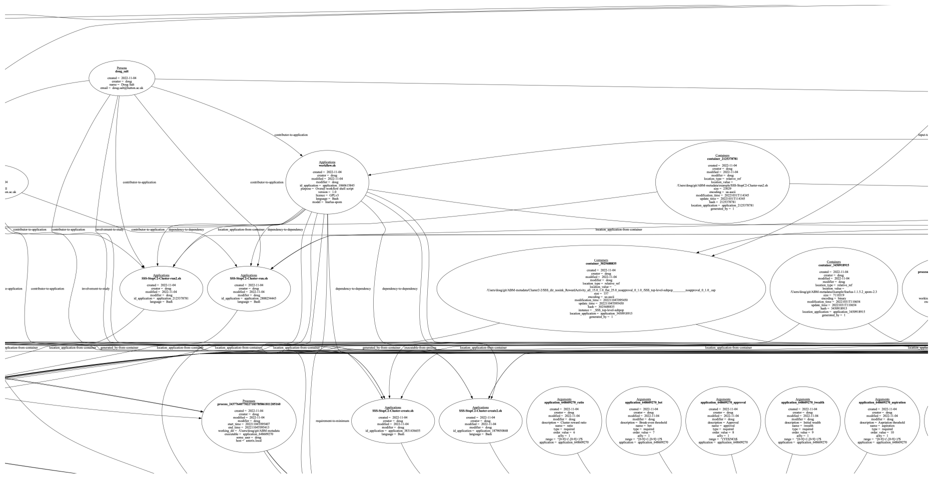

Fig. 3. A small sub-section of the provenance graph

4 Future work

Provenance should be a directed acyclic graph. As the schema stands, it does not guarantee that the graph is acyclic. In future work, we can normalise the schema to remove redundancy and the conflicts arising therefrom; and once this complete, we can formally prove the schema to be acyclic. Until the schema is formally analysed, then we can not say with confidence that any recorded provenance is not inherently contradictory. Since such a schema is likely to be iteratively revised if the underlying provenance model changes or additional functionality is required, then it would be advisable to automate such proving of consistency (assuming correct normalisation).

We should probably be using a graphing language such as Gremlin [20]. The advantages of using graphing databases over relational databases is that such languages are inherently designed to store, traverse and query the graphs that are the main product of this framework. This makes querying in such languages trivial and fast. In a relational databases the SQL statement to do this are cumbersome, awkward and probably (human) error-prone given their size and complexity to construct. Moreover on huge datasets they are reportedly slow. Relational databases do have the advantage of being an extremely mature technology and the availability of utilities that implies. A further advantage of using relational databases is that Structured Query Language is standard for all such databases, and therefore queries written for any relational database should be the same. A tool we already use, Graphviz, already does act like a graphing database language to a certain extent. Indeed we used the Graphviz Dot files produced by the visualisation supplementary programs, rather than the relational schema, in our simple examples to trace bad data through our provenance graphs. However

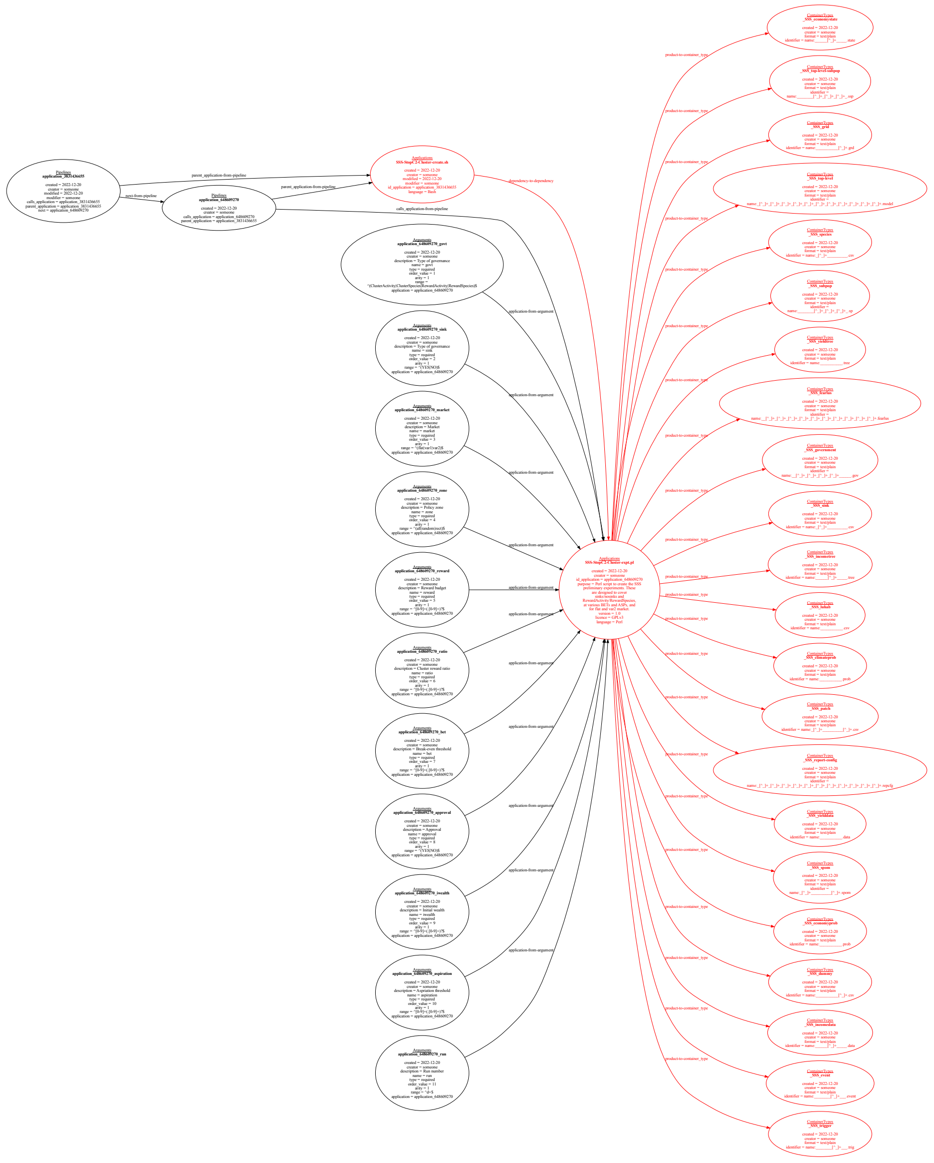

Fig. 4. The workflow sub-graph show the propagation of a broken application (in red) to anything transitively making use of its outputs

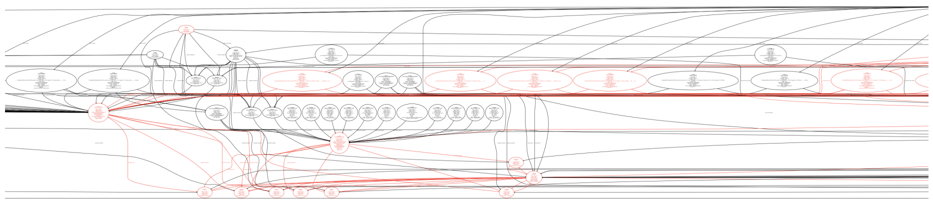
Fig. 5. A provenance sub-graph showing (in red) the propagation of bad data through the system

the Dot language of Graphviz is primarily purposed as a diagramming language and lacks the sophistication, such as a built-in query language of say, Gremlin.

We would eventually like to take many provenance databases run some machine learning across them to see if there are any commonalities. We hope to uncover similarities in setup, execution and post-processing that could form core, reusable and proven components for primarily agent-based modelling but also for other modelling frameworks. These can then be used to *suggest* workflow – e.g. given files of these kinds, what have others done to visualise them?

The plan is to adapt this provenance framework to other model running language frameworks, in particular R, Python, Julia, Java and thence NetLogo. This would take the approach of the `fair` data pipeline [14], but unlike the example we have presented here such provenance would be embedded in the experiments as a matter of course. In the meantime, our demonstration with FEARLUS-SPOMM shows that the `SSREPI` provenance framework may be 'retrofitted' to existing experiments (although possibly at the cost of the programmer's sanity with the present interface), and we can successfully repeat the published experiment. In the short-term, we have tentative potential adopters using R and Python as their primary modelling language.

References

1. Ayllón, D., Railsback, S.F., Gallagher, C., Augusiak, J., Baveco, H., Berger, U., Charles, S., Martin, R., Focks, A., Galic, N., et al.: Keeping modelling notebooks with trace: Good for you and good for environmental research and management support. Environmental Modelling & Software **136**, 104932 (2021)
2. Boero, R., Squazzoni, F.: Does empirical embeddedness matter? methodological issues on agent-based models for analytical social science. Journal of artificial societies and social simulation **8**(4) (2005)
3. Consortium, T.S.: Sqlite syntax. https://www.sqlite.org/lang.html (2023), accessed: 2023-01-16
4. Di Tommaso, P., Chatzou, M., Floden, E.W., Barja, P.P., Palumbo, E., Notredame, C.: Nextflow enables reproducible computational workflows. Nature biotechnology **35**(4), 316–319 (2017)

5. Edmonds, B., Hales, D.: Replication, replication and replication: Some hard lessons from model alignment. Journal of Artificial Societies and Social Simulation **6**(4) (2003)
6. Ellson, J., Gansner, E., Koutsofios, L., North, S.C., Woodhull, G.: Graphviz—open source graph drawing tools. In: Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9. pp. 483–484. Springer (2002)
7. Gimona, A., Polhill, J.G.: Exploring robustness of biodiversity policy with a coupled metacommunity and agent-based model. Journal of Land Use Science **6**(2-3), 175–193 (2011)
8. Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M.R., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. Nucleic acids research **34**(suppl_2), W729–W732 (2006)
9. Inc, A.: Objective-c programming language. https://developer.apple.com/library/archive/documentation/Cocoa/Co accessed: 2023-04-11
10. Köster, J., Rahmann, S.: Snakemake—a scalable bioinformatics workflow engine. Bioinformatics **28**(19), 2520–2522 (2012)
11. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the kepler system. Concurrency and computation: Practice and experience **18**(10), 1039–1065 (2006)
12. Missier, P., Belhajjame, K., Cheney, J.: The w3c prov family of specifications for modelling provenance metadata. In: Proceedings of the 16th International Conference on Extending Database Technology. pp. 773–776 (2013)
13. Mitchell, S.N., Lahiff, A., Cummings, N., Hollocombe, J., Boskamp, B., Field, R., Reddyhoff, D., Zarebski, K., Wilson, A., Viola, B., et al.: Fair data pipeline: provenance-driven data management for traceable scientific workflows. Philosophical Transactions of the Royal Society A **380**(2233), 20210300 (2022)
14. Parker, D.C., Barton, M.J., Filatova, T., Polhill, J.G., Jin, X., Lee, A., Lee, J.S., (Wright), K.R., , Pritchard, C.: Final white paper: MIRACLE project (MIning Relationships Among variables in large datasets from CompLEx systems). Tech. rep., Arizona State University and University of Twente and University of Waterloo and The James Hutton Institute (01 2019), accessed: 2023-03-11
15. Polhill, G., Milazzo, L., Dawson, T., Gimona, A., Parker, D.: Lessons learned replicating the analysis of outputs from a social simulation of biodiversity incentivisation. In: Advances in Social Simulation 2015, pp. 355–365. Springer (2017)
16. Polhill, G., Milazzo, L., Parker, D., Jin, X., Pritchard, C., Lee, J.S., Salt, D.: Miracle simulation outputs metadata specification version 2.0.0. Tech. rep., The James Hutton Institute (2022), accessed: 2023-04-11
17. Polhill, J.G., Gimona, A., Gotts, N.M.: Nonlinearities in biodiversity incentive schemes: A study using an integrated agent-based and metacommunity model. Environmental Modelling & Software **45**, 74–91 (jul 2013). https://doi.org/10.1016/j.envsoft.2012.11.011, http://linkinghub.elsevier.com/retrieve/pii/S1364815212002824
18. Project, G.: Bash 4.4.20. https://tiswww.case.edu/php/chet/bash/bashtop.html, accessed: 2023-04-11, Version = 4.4.20
19. Rodriguez, M.A.: The gremlin graph traversal machine and language (invited talk). In: Proceedings of the 15th Symposium on Database Programming Languages. pp. 1–10 (2015)
20. Schmolke, A., Thorbek, P., DeAngelis, D.L., Grimm, V.: Ecological models supporting environmental decision making: a strategy for the future. Trends in ecology & evolution **25**(8), 479–486 (2010)

21. Stonebraker, M., Kemnitz, G.: The postgres next generation database management system. Communications of the ACM **34**(10), 78–92 (1991)
22. Tinkerpop, A.: Graphson data format (2015)
23. Uslontsev, A.: gv2graphson. https://github.com/sanychsamara/gv2graphson, accessed: 2023-04-13
24. Weibel, S.L., Koch, T.: The dublin core metadata initiative. D-lib magazine **6**(12), 1082–9873 (2000)