Simulated Domain-specific Provenance

Pinar Alper[1], Elliot Fairweather[2], and Vasa Curcin[2]

[1] University of Luxembourg
[2] King's College London

Abstract. The main driver for provenance adoption is the need to collect and understand knowledge about the processes and data that occur in some environment. Before analytical and storage tools can be designed to address this challenge, exemplar data is required both to prototype the analytical techniques and to design infrastructure solutions. Previous attempts to address this requirement have tried to use existing applications as a source; either by collecting data from provenance-enabled applications or by building tools that can extract provenance from the logs of other applications. However, provenance sourced this way can be one-sided, exhibiting only certain patterns, or exhibit correlations or trends present only at the time of collection, and so may be of limited use in other contexts. A better approach is to use a simulator that conforms to explicitly specified domain constraints, and generate provenance data synthetically, replicating the patterns, rules and trends present within the target domain; we describe such a constraint-based simulator here. At the heart of our approach are templates - abstract, reusable provenance patterns within a domain that may be instantiated by concrete substitutions. Domain constraints are configurable and solved using a Constraint Satisfaction Problem solver to produce viable substitutions. Workflows are represented by sequences of templates using probabilistic automata. The simulator is fully integrated within our template-based provenance server architecture, and we illustrate its use in the context of a clinical trials software infrastructure.

1 Motivations and approach

A key requirement for the progression and adoption of provenance research is the availability of realistic provenance datasets. Such data is necessary to support the prototyping of new techniques or tools for provenance capture, analysis and visualisation. Thus far the community has tackled this requirement by using existing applications as data sources. One such effort is the ProvBench series of challenges [3, 2] that built up a corpus from the output of a diverse selection of provenance-enabled applications, such as those used in scientific workflows and file systems.

Provenance sourced from a particular application is inherently tied to that domain, which can be limiting. Firstly, it may be one-sided and exhibit only certain patterns. For example, the Wikipedia corpus of the Reconstruction Challenge [4] focuses only on document revisions and not on delegation of author

responsibilities. In reality data is often produced through processes involving multiple applications, such as security layers, content and collaboration management systems, local tools, and remote services. Focusing on a single element makes it difficult to obtain a full description of the provenance of data.

Secondly, the provenance data may exhibit trends and correlations that exist in the domain environment only at the time of collection. For example, the file system corpus [9] contains data traces taken from applications that are run only with a fixed default workload configuration. This only captures reality partially, as the applications experience changing workloads over time.

An alternative approach is to use a simulator and generate provenance synthetically. Synthetic data generation has been investigated in the context of relational databases [11, 10] and graphs [17], but not as thoroughly in the context of provenance. In order for synthetic data to be useful for a particular domain, it needs to be *valid* (observe the allowed structure of the domain), and *realistic* (observe the data correlations present with in the domain). However it is important to note that true realism is only achievable by sampling distributions derived from real-world data, which is sometimes not possible.

In this paper we describe a simulator that is configurable to a particular domain and generates such data. Validity is achieved using *provenance templates*, an emerging approach for provenance recording and management, which represent abstract, reusable provenance fragments that may be instantiated using concrete data. Realism is approximated by providing the simulator with a set of constraints that represent the data correlations and trends of the target domain. Value sets for certain constraints are generated by sampling statistical distributions and we delegate the task of constraint solving to a Constraint Satisfaction Problem (CSP) solver.

Single templates are not however sufficient to generate meaningful traces. We therefore introduce the concept of processes to represent possible workflows. A process is a defined as a probabilistic finite automaton, in which each state is associated with a template. The simulator generates a path through the automaton that is used to produce a sequence of templates to be instantiated. The simulator is fully incorporated within our template-based provenance server.

2 Provenance templates

A provenance template [5] is a abstract fragment of a provenance document, that may be instantiated using concrete substitutions for variables contained with the template. Variables are of two kinds; identifier variables inhabiting the `var` namespace which are placeholders for node or relation identifiers, and value variables under `vvar`, which can be used in the place of an attribute value. A provenance template is itself a valid provenance document and as such allows nodes to be semantically annotated, allowing the inclusion of domain-specific information. Concrete provenance fragments are generated by an algorithm that accepts as input a template and a substitution comprised of a set of variable-value *bindings*, and replaces variables for values in a copy of the template.

The template approach does not prescribe how one produces bindings, as this might differ across applications. This insulation from the application layer makes templates a suitable mechanism to represent the combined provenance of multiple applications.

Figure 1 gives an example of a template from the domain of health informatics. It outlines the provenance trace that is to be collected from the use of diagnostic support tool for the management of potential secondary stroke. An initial assessment is performed by a clinician at a GP practice. The assessment uses the available clinical guidelines and the patient's health record and produces a stroke prevention plan for that patient. The patient's progress with the plan is checked in a follow-up assessment and a revised plan is produced. The template illustrates the use of both identifier variables such as that for the entity `var:record`, and value variables, such as `vvar:riskLevel` given as the value of the attribute `ex:priority`.

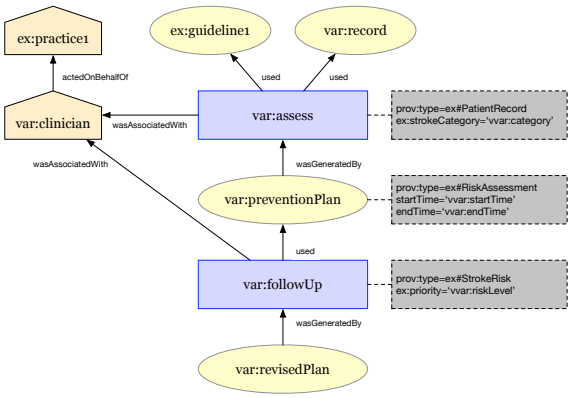

Fig. 1. Provenance template for stroke risk assessment

We require one extension to the published template model [5]. Nodes or relations using value variables must also be annotated with an attribute `pgt:vvarTypes` that contains a map from the names of the value variables present to their intended value type. The template model also provides the ability to define iterable sub-graphs within templates; we do not consider this functionality in this paper.

2.1 Variable domains

Our first contribution is to introduce the concept of domains for template variables. Let V_T denote the set of variables occurring in the template T. For each

variable $x \in V_T$, we may specify a set of values D_x specifying the values that may be used in bindings for x. If x is an identifier variable each element of the domain must be of type `prov:QUALIFIED_NAME`, and if it is a value variable the domain of each value must be of the type specified in the variable type map for that node or relation.

3 Domain-specific constraints for templates

We now focus on constraints that ensure our simulated traces are not only valid, but also realistic. Application domains are often associated with restrictions or trends beyond those encoded in the data schema of an application. For instance, clinical guidelines contain rules restricting the ordering of events within a process, e.g. a patient follow-up assessment should occur between 30 and 90 days following the initial assessment. As another example, consider the extensive use of medical ontologies describing medical conditions and interventions - when multiple entities from ontologies are brought together in a particular context, their co-occurrence may require co-ordination (e.g. no pregnancy events in male patients).

We model such restrictions with constraints over the variables occurring within a template. We currently support the following kinds of constraints.

3.1 Constraint types

We now formalise the types of constraints supported in our simulations. Note that, for simplicity, we avoid the use of formal medical terminologies and ontologies.

Relation constraints Relation constraints are formed from Boolean comparisons between binary arithmetic expressions involving the variables occurring in a template and numeric constants. They can be used to model domain-specific event ordering requirements. An example for the stroke assessment template would be as follows:

$$(preventionPlan.endTime - preventionPlan.startTime) > 30$$

Value-dependency Constraints Value-dependency constraints are conditional expressions that enforce a dependency between the possible values that two different variables may assume. A constraint of this type is constructed from set membership tests between the value $val(z)$ of a variable z and a subset of the values in the domain of that variable, $V_z \subset D_z$, such that:

$$\mathsf{if}\ val(x) \in V_x\ \mathsf{then}\ val(y) \in V_y$$

Relations between domain-specific semantic attributes can be represented using value-dependency constraints. For example, stating that in our simulations diagnosis of diabetes should be followed by either a diet or insulin treatment or both, would be expressed as:

$$\begin{aligned}&\textsf{if } val(diagnosisKind) \in \{Type1Diabetes,\ Type2Diabetes\}\\&\textsf{then } val(treatmentKind) \in \{CardioProtectiveDiet,\ InsulinTreatment\}\end{aligned}$$

Distribution Constraints Distribution constraints specify how domain values are picked for certain variables. They are configurations represented as triples of the form $\langle x, k, F_x \rangle$ where $x \in V_T$, and $k \in \{$ *uniform, exponential, pie* $\}$ denotes a distribution kind, and the set F_x represents the frequency of the occurrence of each possible domain value for the variable x. Each $f \in F_x$ is a pair $\langle d, p \rangle$, such that $d \in D_x$, where D_x denotes the domain of x, and a probability $p \in \mathbb{R}$.

We use frequency sets derived from discrete probability distributions and currently support three types of distribution: uniform distributions in which each domain value has equal probability, pie distributions in which each domain value has an associated probability, and the Zipf power law distribution.

Note that, in our implementation, any variables that have an associated relation or value-dependency constraint cannot also have a distribution constraint.

3.2 Solving constraints

We make use of a Constraint Satisfaction Problem (CSP) solver [16] to solve the constraints given for a template. CSP solvers operate on problems of the form $\langle \mathbb{V}, D, C \rangle$, where $\mathbb{V}$ is a set of variables, D is a set of domains for variables and C is a set of constraints. They use optimised search algorithms to find solutions to a given problem. A solution is a set of variable-value assignments, which ideally should be *consistent* and *complete*. A solution is consistent if it does not violate any of the constraints, and it is complete if it contains assignments for all variables.

We use the CSP solver in its most basic configuration; that is, where all constraints and variables are mandatory and a consistent and complete solution is sought. We create a problem with only integer variables. Each constraint type identified in the section 3.1 can be mapped to a CSP constraint types as follows.

Binary CSP constraints are those involving two variables. Relation constraints with complex arithmetic expressions can be mapped to Binary CSP constraints through use of intermediary variables. Reified CSP constraints are those that involve constraints combined with logical operators. We use this mechanism to implement value-dependency constraints. Global CSP constraints are a portfolio of constraints that capture commonly encountered constraint patterns, and are defined over an arbitrary number of variables. Specifically, we use the Global Cardinality Constraint (gcc), which allows us to set the (min-max) number of times a value can be assigned to a set of variables.

4 Processes and Simulation

In order to produce simulated provenance traces, we first need to specify the processes involved, and map them onto templates.

4.1 Processes

Processes represent simple workflows constructed from the instantiation of sequences of templates, modelled as probabilistic finite automata, with each automata state associated with a template. When generating a trace the simulator first takes a possible path through the automata and outputs the respective sequence of transitions. This information is then used to determine a sequence of templates to be instantiated for that trace. The initial and terminal states of an automaton are not associated with a state and the initial state is chosen from a probability vector. Transition probabilities for each state sum to unity. An example is shown in Figure 2.

When template instances are *merged* into the provenance document being constructed, any values given in bindings for variable identifiers that already exist in the document are reused and, if not present, are freshly created. This process by which nodes within template instances are *grafted* upon existing nodes is what enables larger documents to be constructed from the fragment documents created from the instantiation of templates. For more information on the document construction process and how it is carried out within the provenance server, see [7].

Whilst merging allows the natural building of complex documents under normal operation the simulator requires that graft points between the templates of a process be explicitly marked in order to control the way in which template instances are joined. This is achieved in the following way. Each transition of a process may be annotated with pairs of identifier variables called *anchors*. The lefthand-side of each pair is a variable from the preceding template and the righthand-side a variable from the subsequent template. Under simulation the righthand-side must be instantiated with the same value as the left, thus ensuring that a graft is created.

We also introduce the practice that a *partial substitution* may be associated with a given process state. This substitution will be pre-applied to the template of that state before the remaining bindings are generated. This allows more generic templates to be defined which may be reused in similar but distinct contexts.

Figure 2 shows an example automaton describing the data workflow for a randomised clinical trial application. The eligibility check, consent gathering and randomisation states are each associated with a distinct template, whilst the Patient Reported Outcome Measure (PROM) and Clinician Reported Outcome Measure (CROM) states (representing the completion of form-based assessments) make use of the same single template, pre-applied with a partial substitution specifying whether the form is to be completed by the patient or clinician.

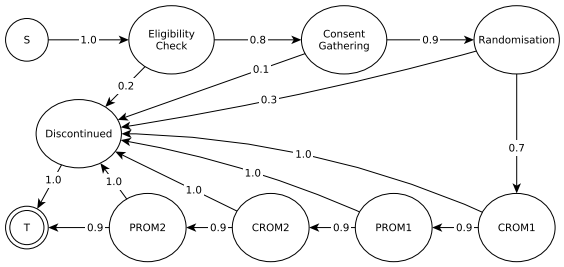

Fig. 2. Automaton for randomised clinical trial workflow

4.2 Simulation

To produce a simulated provenance trace, the simulator operates in conjunction with the provenance server. After reading the configuration, the given variable domains are mapped to integer values and stored for use by the CSP solver. The process configuration is then used to construct the described automaton, and the template used in each state is read from the provenance server database and stored. The variables for each template and any associated constraints are then mapped to their counterparts in the CSP solver, and any required distribution constraint value domains generated.

For each requested trace, the automaton is first used to generate a path, giving a sequence of templates to be instantiated. Then for each template in order, bindings for its variables are generated. If a distribution constraint exists for a variable, a sample is taken and together with any relation or value-dependency constraints upon the same variable, submitted to the CSP solver to be solved. If no constraint is present, a value is selected at random from the domain of the variable, or in the case that no domain is specified, a value of the correct type is generated at random.

These bindings are then submitted to the provenance server as a substitution, which constructs a new instance of the respective template and stores it in the database. Following the first instantiated template of a process the most recent bindings are stored and any values generated for anchored variables in the following transition reused.

5 Implementation and architecture

The architecture of the simulator component and the template server are given in Figure 3. The simulated process is represented by an XML configuration comprising the templates required, associated value domains for template variables, domain constraints, and transitions between process states. The configuration is passed to the simulator via the provenance server and template definitions are

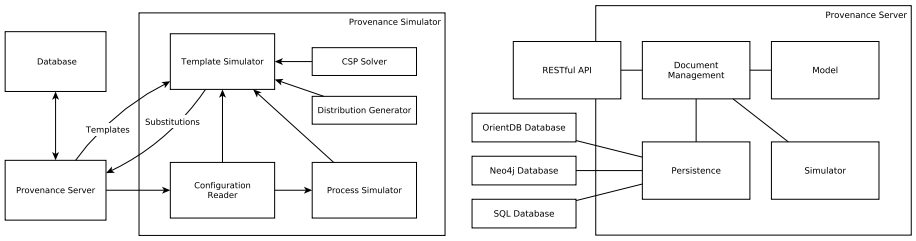

Fig. 3. Architecture of the simulator (left) and server (right)

read from the server database. We use the Choco 4.0 solver [15] for CSP solving and the Apache Commons Mathematics library for distribution sampling.

Provenance documents are modelled as graphs in a formalism-agnostic way by the server's Model component. Interoperability with PROV is provided using ProvToolbox. The Model component provides templates, substitutions and the instantiation algorithm by which new provenance fragments are generated. The Document Management component controls and executes the operations outlined in the document building workflow such as the creation of new target documents, namespace management, the registering of templates, and the generation and merging of new fragment documents. Storage of data in the system is abstracted by a persistence layer component to enable the use of different database technologies - at present Neo4J and OrientDB graph databases are supported, as is a relational SQL format. The management API may be accessed directly via a RESTful web interface. Analysis of target documents may be performed either at the database level, or otherwise by exporting target documents or fragments and using existing PROV tools.

The clients invoke the simulator via the management API by providing a configuration file, a target document and a number of iterations to be executed.

6 Evaluation

Following our initial work on decision support systems, we are now focusing on simulating clinical trial traces. Clinical trials have become increasingly reliant on contextual data sources that complement the data obtained directly from the patients, e.g. Electronic Health Record (EHR) systems are used to identify eligible patients and feed part of the required trial data into Electronic Case Report Forms (eCRFs). Tractability of such system is of essence in order to understand, evaluate and potentially improve the trial design. This requires the minute study details, e.g. eligibility criteria encodings, how they were applied to individual patients who presented to the clinician, data extracted from the EHR systems, data collected through eCRFs and the analysis performed on the collected data. Assembling the trace of the entire process requires provenance to

be captured from the Clinical Trial Management System, EHR system and the patient/clinician data collection tools.

The overall goal is to use that provenance to demonstrate compliance of the software tasks executed during clinical trials with regulations such as US's 21 CFR Part 11 and Good Clinical Practice (GCP) standards. Techniques to validate the clinical trial provenance data against such standards need to be developed initially on simulated data and validated themselves before being attempted on real trial data, making it essential that synthetic data is structurally identical to real provenance traces.

The trials we are observing are based on the TRANSFoRm clinical trial infrastructure [6]. The key steps involve: 1) flagging up the patients eligible for the trial to the clinician by checking their EHR in the background; 2) obtaining patient consent; 3) randomising them into one arm of the trial; and 4) fill in a series of forms. Figure 4 shows the four templates used in this context. Our initial set of constraints states that:

- Forms must be completed between the start and end date of the trial: `vvar:formCompletionDate` >= `var:formStartDate`, `vvar:formCompletionDate` <= `var:formEndDate`
- We want an equal proportion of male and female participants: `vvar:patientSex` is a uniform distribution
- Ages should be uniform across age band values '18-30', '31-50', '51-70', '70+': `vvar:patientAge` is a pie distribution over the four categories defined.

With these constraints and templates in place, we run the simulation. Figure 5 shows an image from the Neo4j visualisation tool of a document built from five iterations of the RCT process, together with the original templates (yellow nodes) and the document metadata and indexing (green). The main subgraph shows the simulated trace under construction, with entities depicted in blue, activities in pink and agents in red. The trace can now be used to define various checks as required by the trial and have them validated and ready for the real data, once the trial starts.

7 Related work

Synthetic data generation has been studied in the context of databases and graphs. Commercial database systems such as those developed by Microsoft and IBM have associated generators, however these rely on random generation and only cater minimally for realism. One line of research has focused on generating test queries, data and verifying oracles for a given database schema [11], in order to create data with appropriate test coverage. Another approach is to generate data to be used against known query benchmarks [10] or applications [18]. The focus here is to have data with different (realistic or experimental) distributions. These generators cater for dependencies such as foreign keys, but do not address constraints among other data elements.

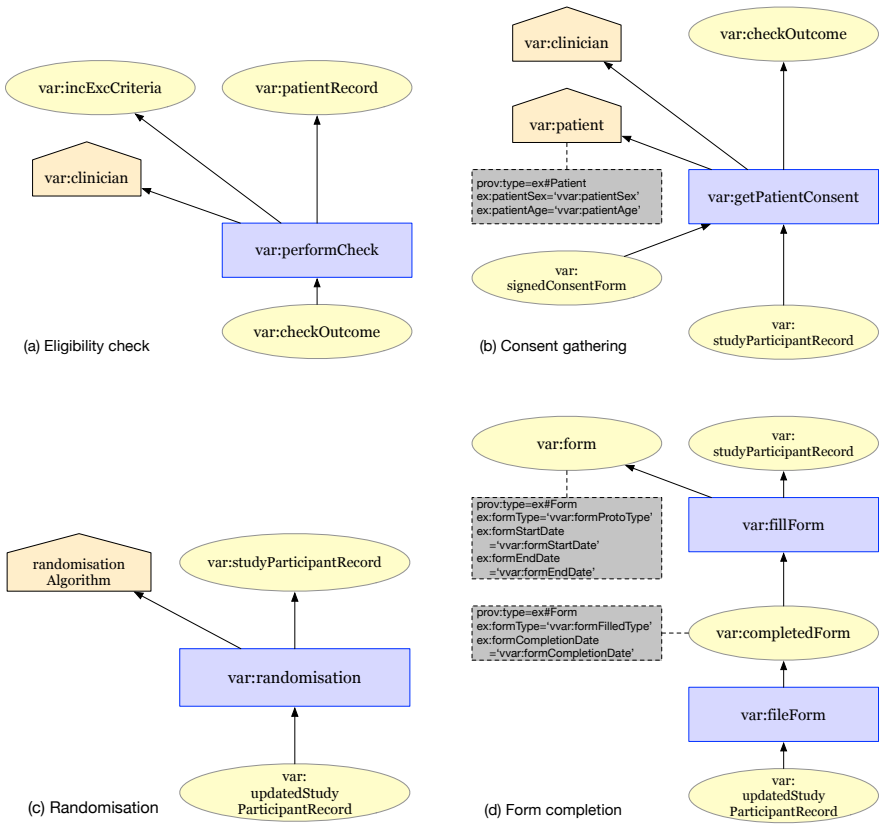

Fig. 4. Templates used in RCT scenario

Graph-based data is encountered in social networks, the internet, or the power grid. In order to assist the development of graph analysis and visualisation techniques, research has been undertaken on synthetic graph generation [13], focusing on statistical properties, with one recent work addressing the use of predefined patterns when generating graphs [17]. These patterns can somewhat be likened to templates, in that they constrain the structure of the generated graph. However, this work does not cater for constraints and allows the manipulations of the generated patterns to achieve statistical properties.

Two research groups are actively studying templates with similar approaches [14] and [5]. A detailed comparison of these is beyond the scope of this paper; but their difference mainly lies in the instantiation procedure. In [14] instantiation uses the Cartesian product of available bindings, whereas in [5] bindings can be given either simultaneously or incrementally using well-defined iteration zones.

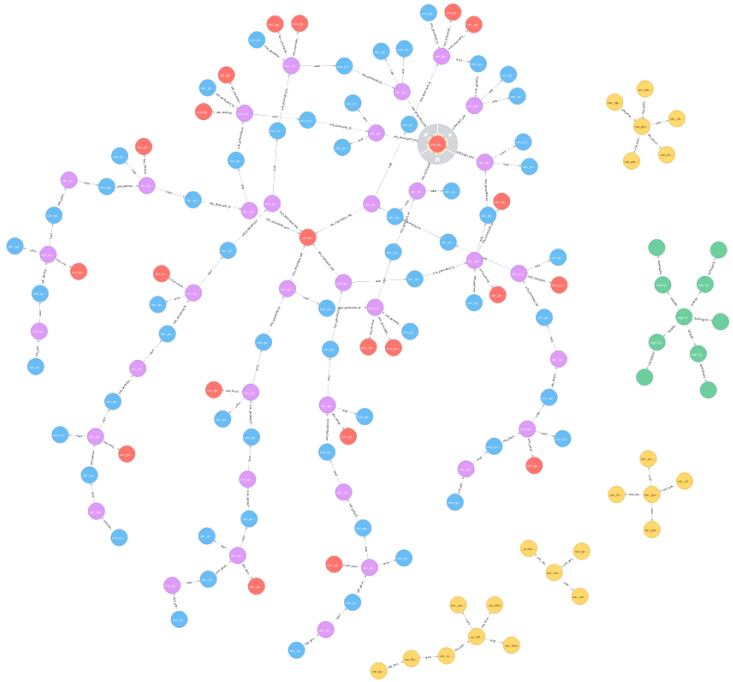

Fig. 5. Neo4j visualisation of five traces of the RCT process

To the best of our knowledge ProvGen [8] and DAGaholic [1] are the only approaches that have studied the generation of synthetic provenance data. These systems accept a seed graph as input, which identifies the PROV statements allowed to be emitted by the generator, in the context of a set of additional constraints. Constraints are restrictions that help enforce patterns or control statistical graph properties (such as vertex degree), but do not support the control of the domain-specific attributes of provenance elements. This highlights an important distinction to our approach, where provenance is an information model first, and its graph-based nature only of secondary importance. Hence we focus on restricting domain-specific attributes to indirectly control statistical graph properties.

Shapes Constraint Language (SHACL) [12] is a recent W3C specification for validating RDF data. SHACL has a comprehensive set of built-in constraints over RDF literals, such as regular expressions or integer ranges, and allows constraints involving complex graph patterns surrounding the focus node to be defined as SPARQL ASK and SELECT queries, the results of which are used to determine whether or not a constraint has been violated. We see SHACL as a complementary foundational technology for encoding templates. Our tem-

plates can be encoded as shape graphs, and our relation constraints as SPARQL constraints. However, SHACL has no means to represent value-dependency and distribution constraints and, furthermore does not support variables, which, for us are crucial abstractions for collecting and incrementally building provenance.

8 Conclusions and future work

In this paper we described a simulator for generating synthetic provenance in a way that observes domain-specific constraints. Our approach uses templates [5] to control and output provenance of a valid structure and three categories of constraints mapped to and solved by a CSP solver to help model domain-specific patterns in a realistic way. We described how workflows are modelled as probabilistic automata, traversals of which represent sequences of templates to be instantiated, and explained how the simulator fits within our provenance server architecture. Finally we illustrated our system with a case study based upon randomised clinical trials.

The implementation and testing of our simulator is ongoing. CSPs have high computational complexity, so an important next step for us is to determine the performance of our system for large output sizes in terms of number of variables, and number and types of constraints.

The clinical trial use case will be fully evaluated within the REST clinical trial (Runny Ear STudy: Immediate oral, immediate topical or delayed oral antibiotics for acute otitis media with discharge) later in 2018, before which provenance data will be synthesised in accordance with the design presented here, to be compared against the data collected.

References

[1] M. Allen, A. Chapman, and B. Blaustein. Engineering choices for open world provenance. In *Provenance and Annotation of Data and Processes*. Springer International Publishing, 2015.

[2] K. Belhajjame and A. Chapman. 2nd ProvBench: Benchmarking Provenance Management Systems, 2014. https://sites.google.com/site/provbench/home/provbench-provenance-week-2014.

[3] K. Belhajjame and J. Zhao. 1st ProvBench: Benchmarking Provenance Management Systems. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, 2013.

[4] K. Belhajjame, J. Zhao, D. Garijo, et al. A Workflow PROV-corpus Based on Taverna and Wings. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops, ProvBench: Provenance Benchmark Challenge*, 2013.

[5] V. Curcin, E. Fairweather, R. Danger, et al. Templates as a method for implementing data provenance in decision support systems. *Journal of Biomedical Informatics*, 65, 2017.

[6] B. Delaney, V. Curcin, A. Andreasson, et al. Translational Medicine and Patient Safety in Europe: TRANSFoRm - Architecture for the Learning Health System in Europe. *Biomed Research Int, special edition on Improving Performance of Clinical Research: Development and Interest of Electronic Health Records*, 2015.

[7] E. Fairweather, P. Alper, T. Porat, et al. Architecture for Building Provenance Documents using Templates. https://elliot.fairweather.eu/resources/ArchProvTemp.pdf, 2017.

[8] H. Firth and P. Missier. ProvGen: generating synthetic PROV graphs with predictable structure. In *5th International Provenance and Annotation Workshop (IPAW)*, 2014.

[9] A. Gehani and D. Tariq. Cross-platform provenance. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, 2013.

[10] K. Houkjær, K. Torp, and R. Wind. Simple and realistic data generation. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, 2006.

[11] S. A. Khalek, B. Elkarablieh, Y. O. Laleye, et al. Query-aware test generation using a relational constraint solver. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, 2008.

[12] H. Knublauch and D. Kontokostas. Shapes constraint language (SHACL). Technical report, W3C, 2017.

[13] J. Leskovec, D. Chakrabarti, J. Kleinberg, et al. Realistic, Mathematically Tractable Graph Generation and Evolution, Using Kronecker Multiplication. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2005.

[14] D. Michaelides, T. D. Huynh, and L. Moreau. PROV-TEMPLATE: A template system for PROV documents, 2014. https://provenance.ecs.soton.ac.uk/prov-template/.

[15] C. Prud'homme, J.-G. Fages, and X. Lorca. *Choco Documentation*, 2016.

[16] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., 2006.

[17] H.-H. Shuai, D.-N. Yang, P. S. Yu, et al. On Pattern Preserving Graph Generation. In *2013 IEEE 13th International Conference on Data Mining*, 2013.

[18] G. Soltana, N. Sannier, M. Sabetzadeh, et al. A model-based framework for probabilistic simulation of legal policies. In *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015)*, 2015.