Generating Plausibly Real Synthetic Data using Dynamic Fuzzing

Tom Wallis and Tim Storer

Glasgow University

Abstract. Science requires data. Some work exists toward producing synthetic data sets for testing information systems in a controlled way, but never addresses whether these are representative of the real world. To produce data sets that can plausibly be described as "realistic", we note that it is necessary to account for unexpected behaviors which would produce noise in deployed business processes. We present a novel approach to simulation, where variance in behavior is captured as a component of a model, and used to introduce errors in the behavior represented by such a model. These possibly erroneous behaviors are simulated, introducing realistic noise to generated data sets. Implementation of models under this new paradigm is detailed. The new paradigm introduced lends a new perspective on modeling with impacts on the broader BPMDS community.

1 A Requirement for Data

Business process design is a field with high real-world impact. With a strong economic importance and the possibility of affecting the jobs and incomes of real people, the *design* of a business process has to be given serious attention: work which is undertaken without care, or with inadequate tooling, can have moral consequences. Equally, when undertaken properly, it can have a positive effect economically and on the people it concerns.

I feel like I don't make a very good case for the feedback loop here. Would like to go over this with Tim.

With this in mind, a standard experimental paradigm extends to the treatment of business processes. Models are constructed; initial design decisions are taken in this modelling process. Models then produce data, which is summarised into metrics of performance: the model can be analyzed via these metrics, and tweaks can be made until the design it represents is satisfactory.

insert a figure representing this.

This procedure is reliant on available data in a number of ways. First, the initial model must be constructed from a given understanding of what the desired process is composed of. In the case of modeling existing processes, the initial model could be derived from empirical data sets. The model also generates data, the quality of which are key to producing useful analyses. So long as the original model is sound, produces sound data, and produces data from which sound measurements might be performed, a feedback loop is constructed which can continually find improvements to the model's design.

This feedback loop is useful in both industry and in science. Such a practice is capable of testing tools produced in both contexts, by way of producing and analyzing data and asserting via measurement that the model's output is what is expected. It can also be used to verify new techniques for modeling and analysis: an example might be the assessment of experimental techniques such as log sanitization (Cheng and Kumar, 2015).

I feel this is quite waffley. Cut some stuff here.

Crucially, in industry, feedback loops such as these can be employed to prevent business process designs which function poorly from being deployed in a real-world environment, by simulating anticipated changes and making improvements to existing systems. Deploying new business processes is an expensive endeavor, and poorly-performing changes further increase costs, by way of both their performance and their need for rapid replacement with something workable.

Importantly, it is necessary to *generate* the data involved in this procedure. This stems from two observations: firstly, empirical data is difficult for one to collect without direct access to a model's subject process, and secondly, when this data *is* collected, it is typically privately held in the organization which collected it, as the collection takes resources and the data contained may not be suitable for public dissemination. Therefore, work in *synthesizing* realistic data may reduce the cost of data acquisition, and — where acquisition is not possible — may make data readily available, enabling a greater number of practitioners to make use of the above technique. The need for this has been established in similar work involving the synthesis of generated data (Accorsi and Stocker, 2013; Pourmasoumi et al., 2015; Mitsyuk et al., 2017).

2 Current Methods in Generating Data

Data can be generated via simulation from a model. Simulations must become as accurate as possible for the data generated to be reflective of the real world, yet this requires very realistic models. As business processes consist of complex systems, producing a realistic model requires capturing nuanced data about the subject process.

The concern of this article is the effect unexpected variation in behavior can have on the data generated by simulation, and the presentation of methods to better employ such phenomena as a component of the model so as to improve the quality of the data it produces.

To emulate the effect of behavioral variance on the output of a business process simulation, it is necessary to produce data which indicates that the behavior in a model deviates from what is expected. This ought to be evident in the an event log produced from a business process, which is the focus of existing work on business process variant generation.

2.1 Post-simulation insertion of variance

Accorsi and Stocker (2013) detail several modifications which can be made to event logs taken from simulated runs of business processes. Their modifications are high-level, focusing on changing certain properties of an event log which can impact security metrics of interest to a user of their tool. Examples of these might be the bypassing of authentication, or disregard for a requirement of separation of duty. SecSY, the tool produced by the work, emulates the affect of security policy breech by simulating a business process and applying these alterations post-hoc, to represent the effect of behavioral variance.

This approach satisfies the authors' goals of producing logs which represent the violation of a security policy, but the narrow focus of the work has consequences for its generality. The modifications developed for SecSY are largely security-oriented and do not serve a general-purpose use case. Behavioral variance as usually seen in business process execution, such as an overconfident or distracted employee, would be complex to represent in such a way.

More importantly, the approach — inserting the effect of variance in a log generated via a simulated run of a business process — cannot take into account the impact of knock-on effects from an initial lapse. For example, an under-confident employee might decide to repeat an action, which can appear in an event log as a duplicated step. The impact of this in the real world is dependent on the step duplicated: some steps in the business process might delegate work to other employees, for example. Should such a step be duplicated, a *realistic* event log ought to exhibit the traces of two delegated actions instead of one.

In other words, the choice to alter one part of a workflow can have consequences elsewhere; because post-simulation insertion of variance is concerned with only the initial workflow variance, and the relationship of affected steps with respect to other actors and the rest of the workflow is unclear from an event log, it is not feasible to make *realistic* adjustments to an event log after simulation.

2.2 Pre-simulation insertion of variance

If event logs cannot be altered to represent realistic variance after simulation, an alternative approach could be to make amendments to the model representing variation in behavior before executing the simulation.

Pourmasoumi et al. (2015) detail a tool which is capable of producing business process variants by applying modifications to block-structured models via process structure trees. Process structure trees are useful for representing changes to processes, because block-structured processes, which they were designed to represent, were devised with a mathematical rigor which aids in defining transformation functions over them (Li, 2010). These can be composed together to eventually represent significant changes to the original process.

This approach solves the problem of emulating knock-on effects in event logs, as the simulation of the business process producing an event log must take the any past effects of variance into account. Limiting the approach is a reliance on process structure trees: real-world variance can be complex to model, and representing changes to processes as low-level changes to mathematical structures make them tine-consuming to produce. Moreover, real-world systems are complex and their variance nuanced, making a low-level representation ill-suited.

Finally, a feature of real-world variance is that processes are not dynamic: they evolve over time, and part of the variance of a process is that actor-level and environmental aspects of the model can alter the structure of the business process from one moment to the next. While this can in theory be represented by a very large business process model with all possible paths at all points in time, at scale this will produce an intractably large model. More importantly, the variance may non-deterministically affect what the model looks like at a future point in time, and scenarios like these cannot be embedded in the model before it is run. Accurate simulation of behavioral variance include how the behaviors *change*; "realistic" models should therefore be dynamically self-modifying.

2.3 A new approach: just-in-time model variation

Rather than constructing a new model containing the required variance, or editing simulation output to represent the effect the variance might have had, we propose that an approach better-suited to represent the effects of real-world variance is to describe such a variance as a process change which is applied during the simulation of the original business process.

Depending on the implementation of such a technique, it can represent variance at a more high-level than something dependent on process structure trees, while retaining the benefits of pre-simulation insertion of variance: knock-on effects from simulation are retained, yet high-level descriptions of variance are possible, and models remain tractably small, as model variation can be inserted and removed as it is required. In addition, the approach offers dynamic restructuring of business processes, matching what a participant in a real-world business process observes and allowing for non-deterministic process definition.

3 A primer on Dynamic Fuzzing

To achieve just-in-time model variation, we suggest employing Dynamic Fuzzing. Dynamic Fuzzing is an experimental technique in software engineering where programs are rewritten during execution. The aim of this section is to give an overview of the technique; its strengths are explored in section 4.

Tim suggested a better phrase than this

Achieving mid-execution transformation of processes can be achieved in various ways. For example, actively developed tools (Wallis and Storer, 2018a) change the source code of a program while it executes. For the purposes of this explanation, a new approach is proposed, intended to reflect paradigms popular in the modern workflow modeling community. A reference tool for this is currently in development (Wallis, 2019).

3.1 Workflow representation

Before building a conceptual framework for mid-execution alteration of a process, we will first define our workflow meta-model.

I *think* this is a correct use of "meta-model"

Let us represent a workflow as a directed graph $G = \{V, E\}$, with entry point to the workflow defined as a starting node, $s \in V$, and a set of nodes to end a graph traversal on, $E \in G$, where each node on the graph represents an activity in the workflow of a business process. Edges on this graph represent possible future actions after the completion of the edge's origin.

Assuming some function which executes any associated business process for a given node, called `execute_process`, and another which selects the next action to be executed in the workflow, called `select_successor`, executing a process in the graph can be achieved using the following procedure:

```
curr_node = s
while curr_node $\notin E$:
    execute_process curr_node
    curr_node = ( select_successor curr_node )
```

The following log example can probably be deleted for space, or just explained succinctly

The simplicity of this approach assists in both easy implementation and modification to suit the needs of a given model. For example, a version of the procedure which produces an event log as output, assuming a function called `log` which adds a process to an event log, might be implemented as:

```
curr_node = s
while curr_node $\notin E$:
    execute_process curr_node
    log curr_node
    curr_node = ( select_successor curr_node )
```

3.2 Implementing Dynamic Fuzzing

With a representation of a process such as this, dynamic fuzzing can be implemented with “fuzzers”, which define the transformation on the workflow as a graph transformation. Informally, the type of a fuzzer is $fuzzer :: workflow\_graph \rightarrow workflow\_graph$. Let us assume all fuzzers have inverse functions which undo their effects, denoted f_i.

Assume mappings $v \in V, f(v) = F_v, i(v) = F_{v_i}$ where f associates some vertex on our graph v with its associated set of fuzzers F_v, and a similar mapping to collect the inverses of these fuzzers into a set of fuzzer inverses mapped to the current node F_{v_i}. The graph transform takes and returns a graph, meaning that the functions can be chained together, having equivalent input and output types. We can therefore define a point-free function $total\_fuzz\ v = \circ\{f(v)\}$ of type $total\_fuzz :: vertex \rightarrow workflow\_graph \rightarrow workflow\_graph$[1], and a similarly defined function to apply any inversions of previous fuzzings, $apply\_inverses$. This gives a trivial application of dynamic fuzzing:

[1] By $\circ\{f(v)\}$, we intend the composition of all functions in the set $f(v)$.

```
curr_node = s
while curr_node $\notin E$:
    G = ( total_fuzz curr_node G )
    execute_process
    curr_node G = ( apply_inverses curr_node G )
    curr_node = ( select_successor curr_node )
```

By applying inverses associated with the current node, previous fuzzings can be reset at the point at which they are no longer required, such as when the current node is no longer within the region of the graph the given fuzzer should be applied to. Before and after every execution of a part of a business process associated with a node, the graph representing the workflow G is replaced with an updated variant, with the appropriate modifications representing real-world variance applied or removed.

3.3 Existing implementations

Working implementations of Dynamic Fuzzing which can be immediately used for modeling purposes exist, using a different technique to that described in section 3.2. PyDySoFu(Wallis and Storer, 2018a) is a tool which dynamically alters Python source code via a tree representation[2]. Where models are implemented as simulations in Python, PyDySoFu provides a mechanism for just-in-time insertion of behavioral variance in a model. However, the authors doubt that this low-level implementation is immediately familiar to those in the workflow modeling community, hence the alternative approach described here. Use of PyDySoFu for dynamic fuzzing in workflow modeling is described in Wallis and Storer (2018b).

4 Dynamic Fuzzing as a Modeling Paradigm

The conceit of this section is that, properly employed, dynamically fuzzed programs can express realistic models of business pro-

[2] Specifically, PyDySoFu dynamically alters the abstract syntax tree associated with Python code, using aspect orientation to decouple behavioral variance — described as a graph transform — from behavior specification — described as a Python program simulating expected behavior

cesses with variance. Particularly, in this section we show that dynamic fuzzing can:

- Produce a realistic event log, complete with knock-on effects of behavioral variance
- Support the construction of models simple enough to design and support tractably
- Support the feedback loop mentioned in section 1

. . . the rationale for each point presented in their respective proceeding subsections.

4.1 Production of realistic event logs

Use of dynamic fuzzing permits just-in-time insertion of behavioral variance in a model, which provides the benefits of modeling knock-on effects and dynamically altering the existence and impact of variance on the model.

Inclusion of knock-on effects on the model, rather than inserting the immediate effects of variance, overcomes the weaknesses identified in post-simulation insertion of variance used in Accorsi and Stocker (2013). This is a result of knock-on effects from one stage of a simulation being taken into account in later stages of the simulation as they can alter the state of the model mid-execution.

Meanwhile, dynamically altering the nature, impact and existence of variance on a model by making its manifestation dependent on the state of a simulation mitigates the issues identified with the pre-simulation insertion of variance identified in Pourmasoumi et al. (2015).

As a result, the BPMDS paradigm dynamic fuzzing permits allows for models which can be made more realistic than its pre– and post– insertion alternatives.

4.2 Tractably design-able and support-able models

Models which can be designed and supported tractably should be small enough to maintain, without sacrificing the complexity they may capture.

Ideal behavior is specified using dynamic fuzzing by employing familiar workflow modeling techniques. This “blueprint” of expected behavior is then enhanced using descriptions of realistic variance separately to the original model. Model size should be expected to grow slowly with the added complexity of variance in this way: separation of concerns is a proven way to capture added complexity in concise, yet expressive ways (Kiczales et al., 1997).

This technique has been successfully demonstrated in Wallis and Storer (2018b), where software development methodologies were modeled and analyzed as an evaluation of the approach for the development of tractably sized models. Therefore, models can be expected to grow simply and maintain clarity by employing the same rationale as well-studied software engineering practices.

4.3 Feedback loop support

Change to reference to a figure?

The feedback loop described in section 1 relies on the availability of data for rapid prototyping of a business process, and the design and support of such a business process.

Modeling improvements such as Dynamic Fuzzing permit more realistic data which can be added to the aforementioned feedback loop, improving the overall efficacy of standard design and engineering practices in BPMDS. Furthermore, the paradigm dynamic fuzzing permits hints at an improved feedback loop for business process design and support, where models are influenced by the analysis of the data they produce *as well as specification of properties such as variance.*

diagram here

As this feedback loop iteratevely improves not only the business process model, but also the specification of its variance, this feature of the proposed paradigm can be expected to enhance the realism of the model further via the iterative improvement of both behavior and variance.

5 Next Steps

The proposed paradigm shows promise, but requires further experimental validation.

Some such validation can be found in Wallis and Storer (2018b). A model of software engineers working under different paradigms was developed so as to model the impact of distraction on the workers' productivity, using Dynamic Fuzzing via PyDySoFu's implementation via direct source code fuzzing. Further case studies such as these, ideally verified against empirical data, is necessary for a full validation of the technique via case study analysis.

An alternative angle for validating the paradigm would be to generate realistic event logs, and to evaluate them by identifying expected emergent properties in the synthetic data. This approach is similar to Accorsi and Stocker (2013) evaluation. Work toward employing this evaluation technique is underway, where the BPI 2013 challenge model is being rewritten using Dynamic Fuzzing, and emergent security vulnerabilities identified via STS-Tool (Salnitri et al., 2015).

cite?

Other approaches which could prove fruitful might involve the integration of the paradigm into more typical workflow modeling practices. For example, executable BPMN definitions (Mitsyuk et al., 2017) may benefit for such an approach, especially considering that data synthesis is a target use case for the work. Petri nets are swiftly becoming the de facto implementation underlying workflow modeling and process mining, and an implementation of Dynamic Fuzzing which operates over a Petri Net should be feasible, given that Petri Nets are also graph representations of workflow models.

Maybe cite van der aalst's PhD thesis? It's what's responsible for this situation, after all...

TODO: write a little about possible automata work designing safe and formal rewrites of the graph. Plenty of sexy maths-y stuff we haven't done that somebody could. Temporal graphs etc.

Such an approach could permit future work which tackles a formal verification of a fuzzer's "safety": careless transformations on a graph might cause semantic inconsistencies which could invalidate the model with respect to its subject. Semantic consistency and general fuzzer "safety" is an area of research which requires much exploration.

clumsy. Refactor this sentence? Maybe Tim knows a better way to phrase this!

The paradigm proposed represents a shift in what models are able to capture, and, as a result, the strength of general BPMDS practices. The capture of information within a model, as well as foundational improvements to design and support practices, make just-in-time insertion of behavioral variance via dynamic fuzzing a promising avenue of research for the BPMDS community, in particular its growing cohort with interest in data synthesis.

sp?

Acknowledgements

The authors would like to thank OBASHI Technology for helping to fund this work.

Bibliography

R. Accorsi and T. Stocker. Secsy: Synthesizing process event logs. *Enterprise Modelling and Information Systems Architectures (EMISA 2013)*, 2013.

H.-J. Cheng and A. Kumar. Process mining on noisy logs—can log sanitization help to improve performance? *Decision Support Systems*, 79:138–149, 2015.

G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *European conference on object-oriented programming*, pages 220–242. Springer, 1997.

C. Li. *Mining process model variants: Challenges, techniques, examples.* PhD thesis, University of Twente, The Netherlands, 2010.

A. A. Mitsyuk, I. S. Shugurov, A. A. Kalenkova, and W. M. van der Aalst. Generating event logs for high-level process models. *Simulation Modelling Practice and Theory*, 74:1–16, 2017.

A. Pourmasoumi, M. Kahani, E. Bagheri, and M. Asadi. On business process variants generation. In *CAiSE Forum*, pages 179–188, 2015.

M. Salnitri, E. Paja, M. Poggianella, and P. Giorgini. Sts-tool 3.0: Maintaining security in socio-technical systems. In *CAiSE Forum*, pages 205–212, 2015.

T. Wallis. Workflow_graphs. https://github.com/probablytom/workflow_graphs, 2019.

T. Wallis and T. Storer. Pydysofu. https://github.com/twsswt/pydysofu, 2018a.

T. Wallis and T. Storer. Modelling realistic user behaviour in information systems simulations as fuzzing aspects. In *International Conference on Advanced Information Systems Engineering*, pages 254–268. Springer, 2018b.