Software in Information Systems Research
Methods and Obstacles

including material by Niels Lohmann: “How to Implement a Theory of Correctness in the Area of Business Processes and Services.” BPM 2010
Conformance Checking using Cost-Based Fitness Analysis

A. Admazaysh, B.F. van Dongen, W.M.P. van der Aalst
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
Email: a.aadmazaysh, b.f.dongen, w.m.p.v.daalst@tue.nl

Abstract—The growing complexity of processes in many organizations highlights the need for process analysis techniques. Typically, such techniques are based on process models and assume that the operational processes in reality conform to these models. However, experience shows that reality often deviates from hand-made models. Therefore, the problem of checking to what extent the operational process conforms to the process model is important for process management, process improvement, and compliance.

Implements ideas (validation)

Use implementation (evaluation)
Repairing Process Models to Reflect Reality

Dirk Fahland and Wil M.P. van der Aalst
Eindhoven University of Technology, The Netherlands
d.fahland@tue.nl, w.m.p.vanderaalst@tue.nl

Abstract. Process mining techniques relate observed behavior (i.e., event logs) to modeled behaviors (i.e., a BPMN model or Petri net). Process models can be discovered from event logs and conformance checking techniques can be used to detect and diagnose differences between observed and modeled behavior. Existing process mining techniques can only uncover these differences, but the actual repair of the model is left to the user and is not supported. In this paper we present a novel approach to repair a process model with respect to a log such that the resulting repair process model is consistent to the input log (i.e., conforms to it) and is similar to possible model runs replay the log (i.e., conforms to it) and is similar to possible model runs replay the log (i.e., conforms to it) and is similar to possible model. First, we present a naïve approach based on the original model. First, we present a naïve approach based on the original model. First, we present a naïve approach based on the original model. First, we present a naïve approach based on the original model. First, we present a naïve approach based on the original model.

Keywords: process mining, model repair, Petri nets, alignments, conformance checking.
Repairing Process Models to Reflect Reality

Dirk Fahland and Wil M.P. van der Aalst
Eindhoven University of Technology, The Netherlands
d.fahland@tue.nl, w.m.p.d.aalst@tue.nl

Abstract. Process mining techniques relate observed behavior (i.e., event logs) to modeled behavior (e.g., a BPMN model or Bern rule). Process models can be discovered from event logs and conformance checking techniques can be used to detect and diagnose differences between observed and modeled behavior. Existing process mining techniques can only uncover these differences, but the actual repair of the model is left to the user and is not supported. In this paper we present a novel approach to repair a process model with respect to a log such that the resulting model can replay the log (i.e., conforms to it) and is as similar as possible to the original model. First, we present a naïve approach based on an optimal alignment of model and log. We use an existing conformance checker to align the edges of the given process model to the traces in the log. Although this approach produces a model that can replay the log, it is unsatisfactory because the produced model is not as similar as possible to the original model. Therefore, more intelligent approaches based on the alignment are proposed. Highlight maps are used to identify and repair process model, thereby incorporating as many original features of the original model as possible. The approach is implemented in the process mining toolkit ProM and has been validated on real-life process mining problems and has been validated on real-life process mining problems and has been validated on real-life process mining problems.
Circle of Insights

Impact-Driven Process Model Repair

ARTEM POLYVYANYI, Queensland University of Technology, Brisbane, Australia
WIL M.P. VAN DER AALST, Eindhoven University of Technology, Eindhoven, The Netherlands, and Queensland University of Technology, Brisbane, Australia
ARTHUR H.M. TER HOFSTEDE, Queensland University of Technology, Brisbane, Australia, and Eindhoven University of Technology, Eindhoven, The Netherlands
MOE T. WYNN, Queensland University of Technology, Brisbane, Australia

The abundance of event data in today's information systems makes it possible to "confront" process models with the actual observed behavior. Process mining techniques use event logs to discover process models that describe the observed behavior, and to check conformance of process models by detecting deviations between models and reality. In many cases, process models need to be repaired to make them more realistic. Traditionally, process model repair has been done manually by domain experts. However, manual repair of process models is time-consuming and error-prone.

This paper presents a novel approach to process model repair that is based on the Circle of Insights framework. The approach re-uses the insights gained from previous repair efforts to facilitate the repair of new process models. The re-use implementation for new ideas is part of a larger system for implementing ideas (validation).

...
What do we ask from software in research?

3 paradigms

2 successful paradigms (that I know of)

- service-technology.org
- ProM

Lessons learned

Disclaimer: I can only talk about model editors, execution engines, model verification, and model mining/synthesis tools
Role #1: validate ideas (paper & tools)

- tools often born as a means to support understanding and validation of ideas
- supports reflection on ideas and assumptions
Role #2: reproduce

- tools necessary to reproduce & compare to earlier results
- … own results and results by others
- the catch?
  … I’ll get to that …
Role #2: reproduce

Cheetah Experimental Platform: http://bpm.q-e.at/?page_id=56
Role #3: re-use & extension by others

- re-use of tools/libraries/modules facilitates incremental research
- … own research and research by others
- the catch?
  … I’ll get to that …
Role #4: demoing and teaching

- tools help making complex ideas tangible
- supports students picking up complex subjects in the classroom
Challenge #1: from feasible to working (efficiency, robust)

Is it possible?

How can it be done?

How does it really work?

we already struggle here

Is it actually useful?

decidability result/conceptual idea

algorithm

efficient algorithm

product

theory

practice

Verification

model

decide soundness in 5ms

based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Challenge #1: from feasible to working (efficiency, robust)

Is it possible?

How can it be done?

How does it really work? we already struggle here

Is it actually useful?

decidability result/ conceptual idea

algorithm

efficient algorithm

product

theory

practice

log -> discover -> model

fitting, precise, simple, on logs of >1.000.000 events

based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Challenge #1: from feasible to working (efficiency, robust)

Is it possible?

How can it be done?

How does it really work? we already struggle here

Is it actually useful?

decidability result /
conceptual idea

algorithm

efficient, stable
algorithms

product

theory

Efficient, scalable, transactional properties, can handle external events

model

engine

data

practice

based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Challenge #1: from feasible to working (efficiency, robust)

Is it possible?

decidability result/
conceptual idea

How can it be done?

algorithm

efficient, stable
algorithms & UI

How does it really work?

we already struggle here

Is it actually useful?

product

theory

practice

editor

model

Usable, does not crash,
gives diagnostic information
can handle all kinds of models not
made by the author

based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Universities ≠ professional software companies

- hardly any tenure programmers
- definitely no programming professionals
- **programming is never top priority**
- maintenance is not enforced
- programming is hardly enforced
- **frequently changing staff** (2-5 year frequency)
- hard to collect knowledge
- hard to keep consistency
Summary: Programming in research

- **What do we want to build?**
  - efficient verification programs
  - efficient construction programs
  - efficient engines
  - sophisticated editors
  - *at the cutting edge*

- **Why do we want to build it?**
  - demonstrate feasibility ideas
  - (allow to) reproduce results
  - (allow to) reuse programs/modules by others
  - education

- **Who do we have to build it?**
  - smart people with 120% other priorities and mediocre programming skills
  - too little time, no budget
  - *quick loss of knowledge/expertise*
What do we ask from software in research?

3 paradigms

2 successful paradigms (that I know of)
  • service-technology.org
  • ProM

Lessons learned
Three paradigms: monolithic, plugins, small tools
Large monolithic, typically editors

**CPN-Tools** (process modeling, simulation, verification, complex architecture, arcane programming language)

**Seda/Greta** (Eclipse-based editor for scenario-based process modeling & execution)
Don’t do Graphical UIs / Editors
(unless you are ok to abandon it after 1 year or have money for 5-10 years to spend on it)

- proper visualization & layout is hard
  - new language feature → revise visualization & layout

- users do things you did not intend
  - Enter strange values in fields
  - Don’t know the correct syntax
  - Want proper error message
  - Expect documented help

- impossible to test automatically
Three paradigms: monolithic, plugins, small tools
We provide a collection of approaches, tools, and research results which are related to behavioural aspects of service technology such as service composition, discovery, mediation, substitution, and others. This technology is the result of joint research efforts of the theory of programming groups at Humboldt-Universität zu Berlin, the University of Rostock, the Technische Universität Eindhoven, and several cooperation partners.
Partner(synthesis) for Services

Given: service P

Want to know: Does P have any partners at all?

- If not, then P is **uncontrollable**.
  (has an inherently bad design $\rightarrow$ Fix!)
- If yes, then P is **controllable** by some partner R.
  Can we **synthesize** such a partner R?
Monolithic vs small tool paradigm
Monolithic vs small tool paradigm
Re-design following the UNIX spirit

PN → LoLA → State space → Wendy

~50,000 loc

4000 loc

PN-API

~20,000 loc

Standardized plain-text file formats also stream via stdin/stdout
→ simple parsers/file writers
→ easy to debug
→ easy to create test-cases

Partner
Operating Guideline
Adapter Synthesis
Adapter
PLACE
  INTERNAL
    p0, p1, p2, p3, p4;
INPUT
  E, T, C;
OUTPUT
  B;

INITIALMARKING p0:1;
FINALCONDITION (p4 = 1 AND ALL_OTHER_PLACES_EMPTY);

TRANSITION t1 {?e}
CONSUME
  p0:1,
  E:1;
PRODUCE
  p1:1;

Exchange via clear, simple text-based files
Result? Many tools, many papers, many authors – all building on each other
leverage on strong small-tool development eco system available

GCC & autotools

- written in C/C++
- autoconf: specify dependencies on external libraries (standard and own) \(\rightarrow\) generates configuration script to check for dependencies
- automake: generate Makefile specific for target machine (build, run tests, document)
- numerous high-level configuration tools to manage testing, generating documentation etc.
Three paradigms: monolithic, plugins, small tools
Open source process mining platform
- 1200+ plugins
- maintained in Eindhoven, contributions from dozens of research groups
ProM: View Logs
ProM: Discover Models
ProM: Discover Models, Deviations, Performance
ProM: Discover Models, Deviations, Performance, Data
ProM 5.2: centralized plugin-architecture
ProM 6 Architecture

Workspace

ProM framework

Logs

Petri Nets

Log Filter

Alpha Algorithm

Inductive Miner

Log View

Dotted Chart

Petri net View

Inductive Visual Miner

Actions

Views
ProM: Packages & Dependencies

Code
+ package descriptor
+ package dependencies:
  previously: proprietary, now: Ivy&Maven

automatically retrieve required packages (IDE, runtime)
Example: Model repair

Invoked through generic ProM plugin invocation mechanism (Java annotation & reflection)

Invoked through method call

need to update whenever Alignments updated

invoked whenever Alignments updated

Invoked through generic ProM plugin invocation mechanism (Java annotation & reflection)
ProM: Managing Packages

Meta-Packages

Core Packages
Established
RunnerUp

Log
PetriNet
Alpha
Inductiver Miner

centralized package list (TU/e managed)

automatically retrieve required packages (at installation / startup)
It’s growing and it’s being used...
What do we ask from software in research?

3 paradigms

2 successful paradigms (that I know of)
  • service-technology.org
  • ProM

Lessons learned
Challenge #1

Is it possible?

How can it be done?

How does it really work? *we already struggle here*

Is it actually useful?

decidability result/conceptual idea

algorithm

efficient algorithm

product

theory

practice

Support continuous validation, evaluation, re-use of tools as a low priority activity

Based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Lesson #1: Prototypes

- Have them early!
- Have a lot of them!
- Pros:
  - detect bottlenecks early
  - avoid useless optimization
  - work on real data
  - propel technology transfer
- Cons:
  - a lot of resources required
  - programming/optimization skills required
  - rapid prototyping is hardly rewarded by performance evaluations
- Possible if you create the right environment!

based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Lesson #1b: Expose Prototypes

- prototypes
  - make research results **transparent**
  - make experiments **repeatable**
  - allow for community **benchmarks**
  - make **comparisons** simpler

Based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Catch: Managing versions (reproducing results)

**Monolithic/small tools**: store every released version (& release source code)

**Plugin-based**
- Developers own their packages (see next) → release new versions when they want
- Overall implementation is a continuously moving target

**How to fix?**
- Globally fix versions (e.g. yearly ProM release)
- Create local fork of all dependent plugins, make local release ❗️
Lesson #2: stay generic, separate concept and optimization

- complicated algorithms, formalisms, models, …

**Lesson**: keep meta-models/formalisms natural
  - Do *not* impose limitations to models.
  - Separate modeling and algorithm formalism.
  - Keep algorithm technology domain-unspecific.

**Side remark:**
  - Domain-unspecific tools can outperform specific tools
  - Communication between tools only relies on simple formalisms/meta-models

based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Challenge #2: Academic environment

Universities ≠ professional software companies

- hardly any tenure programmers
- definitely no coding pros
- **coding is never top priority**
- maintenance is not enforced
- programming is hardly enforced
- **frequently changing staff** (2-5 year frequency)
- hard to collect knowledge
- hard to keep consistency

based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Lesson #3: Single purpose tools

Goals:
- Limit exposed code base!
- Minimize dependencies!
- Reduce error horizons!

Realization: **single purpose tools**
- UNIX philosophy
  - small is beautiful
  - make each program do one thing well
  - everything is a stream
- complex tasks = tool interplay
- be minimalistic: no GUI
- simple integration via system call or service invocation

Based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Lesson #3: Single purpose tools - bonus

- single purpose tools:
  - faster release cycles
  - propels prototyping
  - easier maintenance

- other best practices applicable
  - pair programming
  - code reviews
  - test-driven development
  - continuous integration

Based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Lesson #4: you can do plugins if… isolation through stable interoperability

(1) properly defined core data objects (used by other plugins)

(2) Reliable data exchange/storage that’s easy to use

(3) Reliable plugin management, dependency management, easy enough to use

(4) high-quality example implementations where people can copy from → templates, programming style

Logs

Petri Nets

Actions

Views

Workspace

ProM framework
Lesson #5: Code ownership

- give code away to students
  - code ownership *motivates* improvement
  - direct acknowledgment
  - simplifies responsibilities
  - trust and freedom are important!

- propel development with *continuous integration*
  - reduces number of broken builds
  - visibility yields competition
  - implement rankings

---

Based on material by Niels Lohmann: How to Implement a Theory of Correctness in the Area of Business Processes and Services. BPM 2010
Catch: Testing in ProM - it’s tricky

- Integration/function tests on UI
  - to be done manually
  - using several test logs (covering several corner cases)
  - not by the plugin author
- Work intensive
- Done once/twice a year on a limited number of central packages

- Automated unit/integration testing
  - Requires authors to write plugins to be invoked without UI (not all do that)
  - Requires to write rather complex test code
    - Code that invokes various plugins through framework
    - External scripts (runtime compiled Java code, no IDE support, debugging very difficult)
Lesson #6/Catch: Community-building efforts

- development documentation & package/plugin templates

→ preserve knowledge of developers over time

- Forums/mailing lists
Don’t try to build a product: prototypes, iterate, many versions…

Make the implementations small
- “UNIX tools” paradigm
- Plugin-based paradigm (requires solid foundation = upfront investment by skilled programmers + maintenance effort)
- Works only if interchange between tools/plugins is solid
  - Standardized, simple data formats/meta-models
  - Optimization only within a small tool/plugin, never across
- Use continuous integration, testing, live-demos to motivate developers to product “sufficient” quality

Challenges: zero effort dependency version management, zero effort documentation, zero effort community building
Thank you!
Questions?