LotTraveler: Workflow Management in Failure Analysis[1]

Kučko, Matija
mkucko@edu.aau.at

Alpen-Adria-Universität Klagenfurt

[1] Thanking Konstantin Schekotihin for mentoring and guidance.

Outline

Context I

In *typical* production environments, the ordered deliverable is either:

- ready-made and provided according to a static process description
- chosen from a predefined set of product variants
- further adjustable through configuration options
- custom-made according to a specific sequence of activities;
  by stating not *what* the desired outcome is, but *how* it is to be achieved

Context II

Chemistry lab example:

- offers services for the chemical analysis of substances provided by the client
- non-invasive vs non-reversible activities (optical inspection vs bio-degradation)
- prerequisite tasks (chemical synthesis of substance used afterwards)

Sequence of activities needs to make sense and be feasible!

1. client must ensure that the activities form a meaningful workflow that leads to the desired result
2. domain experts must review incoming requests and reject any invalid ones

Problem: repetitive and error-prone!

Figure: Custom-made deliverable obtained by completing a sequence of activities supplied by the client, checked manually by an expert

Solution: automate!

Figure: Custom-made deliverable obtained by completing a sequence of activities supplied by the client, checked by an automatic procedure

Use case: Failure Analysis (FA) in Infineon I

Real-world application area for improving day-to-day operations in a *typical* FA lab:

- diagnosis of failures in integrated circuits delivered by manufacturing dept. or clients
- job = box of samples + activities to be performed on them
- requirements may be fulfilled by following a specific workflow template;
- or more specific needs covered by custom-tailored workflow of activities
- job is then carried out by qualified workers on appropriate tools

Use case: Failure Analysis (FA) in Infineon II

Custom-tailored workflows need to follow some ground rules!

- e.g. the order of visual inspection and sonography is not important, whereas microscope inspections can only be done after dissolving
- e.g. an internal visual inspection task cannot be performed unless the integrated circuit has been decapsulated and its "internals" have been exposed
- e.g. some visual inspections have to be performed just after chemical dissolving, since the sample may deteriorate further over time

Goal: Improve job management by automatically checking requested workflows!

Terminology

Meta-model

Amalgamation of these ground rules $\equiv$ meta-model of FA jobs

Model

Actual (template or custom-tailored) workflow following these rules $\equiv$ model of FA jobs

Instance

Each executed FA job $\equiv$ instance of the respective (workflow) model

Problem statement I

In the scope of this work, a workflow (model) is a bounded linear sequence of tasks. In FA, these linear sequences *typically* consists of no more than 15 tasks.

The automated checker needs to provide the following functionalities:

- *F1* Design-time representation of valid workflows through task ordering and additional restrictions
- *F2* Run-time detection of potentially invalid workflows, as well as automated repair of such inconsistent workflows

Problem statement II

The checker is then used as follows:

F1 Domain experts create a workflow meta-model which captures all ground rules, which lists all possible tasks, which tasks may follow other tasks, as well as mandatory tasks that must appear alongside other ones.

F2 An automated procedure checks a workflow against those ground rules, whether it contains unknown tasks, whether the tasks adhere to the specified order and whether mandatory dependencies are included in the workflow.
If the workflow is invalid (an invalid instance of the meta-model), the "best" valid alternative is recommended instead.

Correspondence

Reduce the subject matter by one abstraction level:

- (model) instances need to adhere to the syntax and semantics facilitated by their models
- similarly, models need to adhere to their respective meta-model framework

- solutions for modeling workflows and checking whether specific workflow executions adhere to such model
- therefore, these are also solutions for creating workflow meta-models and determining whether workflow models are valid instances of those meta-models

Idea

Workflow modeling (*F1*):

- leverage a suitable process modeling language for presenting ground rules (meta-model) for all workflow (models)

Workflow validation (*F2a*):

- using conformance checking, determine the fitness of how well a sequence of tasks matches the behavior allowed by the process model
- valid workflow (models) are only those that fit the process (meta-)model perfectly with a fitness value of 1

Workflow repair (*F2b*):

- use a declarative process model for workflow (meta-) modeling, with which we can validate whether a workflow (model)'s sequence of tasks passes all the rules in such (meta-)model
- generate valid workflow (models) and use an edit distance similarity metric to find the best possible alternative to a given invalid workflow (model)

Idea: References

Workflow modeling (*F1*):

- an overview of the state-of-the-art in process modeling languages is provided in Abbad Andaloussi et al.'s work [2]

Workflow validation (*F2a*):

- an overview of process mining techniques is provided in Van der Aalst's work [1]

Workflow repair (*F2b*):

- declarative modeling elements inspired by Grambow, Oberhauser and Reichert's work [3]
- generation of valid workflows and use of similarity metric inspired by Van der Aalst's work [1]

Evaluation of candidates

Workflow (meta-)modeling:

- Business Process Model and Notation (BPMN)
- Case Management Model and Notation (CMMN)
- Workflow nets
- Ontology
- Answer set programming (ASP)
- **Domain-specific modeling language** (DSML)

Workflow reasoning:

- Ontology
- **Answer set programming** (ASP)
- Other approaches

BPMN

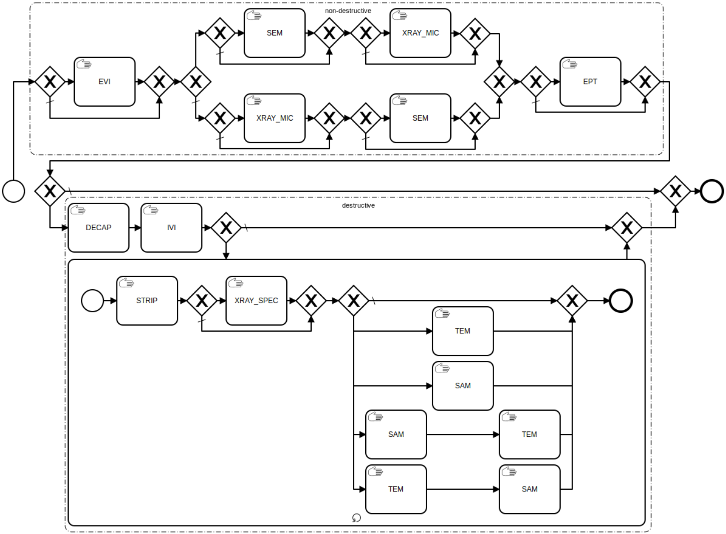

Figure: Example FA process modeled using BPMN

Domain concepts I

Tasks:

- represent an activity type in the workflow
- *non-destructive* tasks must be performed before any *destructive* tasks; or may belong to *both* groups
- can optionally be *repeated* as many times as desired

Connections:

- represent directed associations between tasks
- establish partial order and dependencies between tasks

Domain concepts II

Connection types:

- $A \xrightarrow{has\_successor} B$
  task A may not be performed after task B, given that they are in the same sub-process iteration
- $B \xrightarrow{has\_predecessor} A$ $(\equiv A \xrightarrow{has\_successor} B)$
  task B may not be executed before task A, given that they are in the same sub-process iteration
- $A \xrightarrow{has\_mandatory\_successor} B$ $(\Rightarrow A \xrightarrow{has\_successor} B)$
  task B must be done after A, in the same sub-process iteration
- $B \xrightarrow{has\_mandatory\_predecessor} A$ $(\not\equiv A \xrightarrow{has\_mandatory\_successor} B)$
  task A must be carried out before B, in the same sub-process iteration

Specification (excerpt)

$$
\begin{array}{l}
PartialOrderConstraint \triangleq \\
\quad \forall\, s, d \in TaskNames : \\
\qquad \land\ s \neq d \land Connected[s, d] \\
\qquad \land\ Contains(Workflow, s) \land Contains(Workflow, d) \\
\qquad \land\ \neg Task(s).repeatable \lor \neg Task(d).repeatable \\
\qquad \Rightarrow LastIndex(Workflow, s) < FirstIndex(Workflow, d)
\end{array}
$$

Figure: Excerpt showing the partial order constraint from the TLA+ specification

DSML: Meta-model

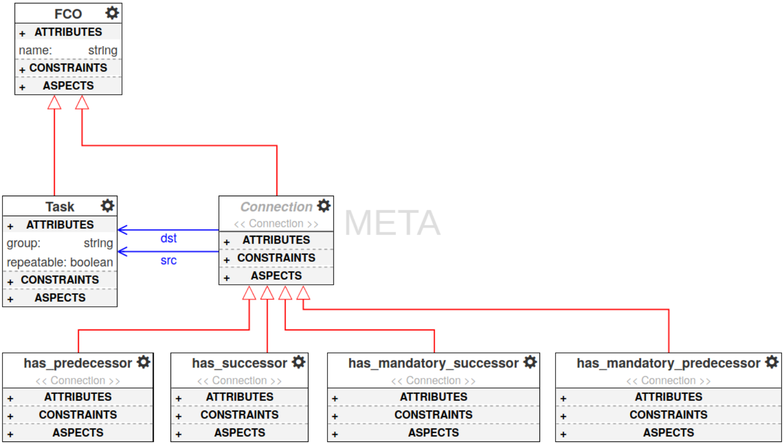

Figure: Meta-model of the DSML used to model the example FA process

DSML: Model

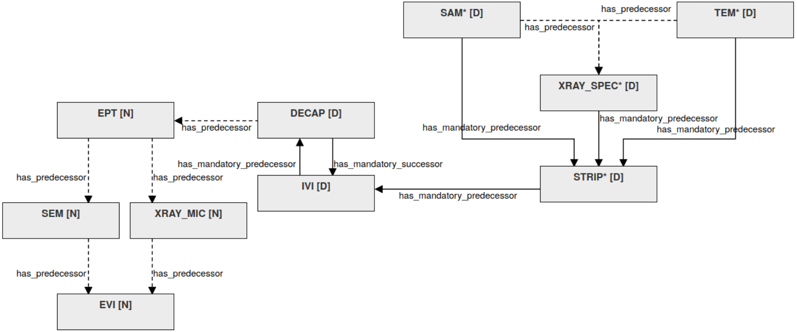

Figure: Domain-specific model of the example FA process

ASP: Tasks & connections (excerpt)

Listing 1: Excerpt from ASP that models example FA process

```
task("EPT").

group_non_destructive(task("EPT")).

% atom just omitted for default negation, instead of explicit classical negation
%-repeatable(task("EPT")).

has_predecessor(task("EPT"), task("SEM")).
has_predecessor(task("EPT"), task("XRAY_MIC")).
```

ASP: Workflow input

Listing 2: Invalid workflow input for example FA process in ASP

```
workflow("Input").

orderNumber(0..1).

% example invalid workflow
workflowTaskAssignment(workflow("Input"), task("EPT"), orderNumber(0)).
workflowTaskAssignment(workflow("Input"), task("EVI"), orderNumber(1)).
```

ASP: Workflow validation I (excerpt)

Listing 3: Excerpt showing part of the partial order constraint ASP implementation

```
% check that task order for non-repeatable predecessor tasks
%    and non-repeatable successor tasks is satisfied
error(workflow(W), reason(partial_order_violation), task(TA), task(TB)) :-
    not repeatable(task(TA)),
    workflowTaskAssignment(workflow(W), task(TA), orderNumber(OA)),
    orderNumber(OA),
    connected(task(TA), task(TB)), TA != TB,
    not repeatable(task(TB)),
    workflowTaskAssignment(workflow(W), task(TB), orderNumber(OB)),
    orderNumber(OB),
    OA > OB.
```

Listing 4: Excerpt showing part of the partial order constraint ASP implementation

```
% check that task order for repeatable predecessor tasks
%    and non-repeatable successor tasks is satisfied
error(workflow(W), reason(partial_order_violation), task(TA), task(TB)) :-
    repeatable(task(TA)),
    latestWorkflowTaskAssignment(workflow(W), task(TA), orderNumber(OAMax)),
    orderNumber(OAMax),
    connected(task(TA), task(TB)), TA != TB,
    not repeatable(task(TB)),
    workflowTaskAssignment(workflow(W), task(TB), orderNumber(OB)),
    orderNumber(OB),
    OAMax > OB.

% check that task order for non-repeatable predecessor tasks
%    and repeatable successor tasks is satisfied
error(workflow(W), reason(partial_order_violation), task(TA), task(TB)) :-
    not repeatable(task(TA)),
    workflowTaskAssignment(workflow(W), task(TA), orderNumber(OA)),
    orderNumber(OA),
    connected(task(TA), task(TB)), TA != TB,
    repeatable(task(TB)),
    firstWorkflowTaskAssignment(workflow(W), task(TB), orderNumber(OBMin)),
    orderNumber(OBMin),
    OA > OBMin.
```

ASP: Workflow validation output

Listing 5: ASP workflow validation output for the given input workflow of the example FA process

```
workflowTaskAssignment(workflow("Input"),task("EPT"),orderNumber(0)).
workflowTaskAssignment(workflow("Input"),task("EVI"),orderNumber(1)).

error(workflow("Input"),reason(partial_order_violation),task("EVI"),task("EPT")).
```

ASP: Workflow generation

Listing 6: Valid workflow generation for example FA process using generate & test in ASP

```
workflow("Output").
orderNumber(0..maxDepth).

% Generate for each orderNumber potential task assignments to the workflow
{ workflowTaskAssignment(workflow(W), task(T), orderNumber(O)) : task(T) } :-
    workflow(W),
    orderNumber(O),
   W = "Output".

% Ordernumbers must start at 0 and must be continuous
:- workflowOrderNumber(workflow(W), orderNumber(O)),
    not workflowOrderNumber(workflow(W), orderNumber(O2)),
    orderNumber(O2), O2 < O.

% Workflow breadth = 1
:- workflow(W),
    orderNumber(O),
    workflowOrderNumber(workflow(W), orderNumber(O)),
   #count
        {
            task(T) :
            workflowTaskAssignment(workflow(W), task(T), orderNumber(O))
        } != 1.

% No errors must occur
:- error(workflow("Output"), _, _).
:- error(workflow("Output"), _, _, _).
```

ASP: Workflow repair

Listing 7: Excerpt showing task order difference optimization statement in ASP

```
% determine tasks that are differently ordered in both workflows
diffTaskOrder(workflow("Output"), workflow("Input"), task(T), |OOutput - OInput|) :-
    workflowTaskAssignment(workflow("Output"), task(T), orderNumber(OOutput)),
    workflowTaskAssignment(workflow("Input"), task(T), orderNumber(OInput)).


% @3: minimize difference of different task order
%         in instance and generated workflow
#minimize {
    ODiff@3,T :
        diffTaskOrder(workflow("Output"), workflow("Input"), task(T), ODiff)
}.
```

ASP: Workflow repair output

Listing 8: ASP workflow repair output for the given input workflow of the example FA process

```
workflowTaskAssignment(workflow("Input"),task("EPT"),orderNumber(0)).
workflowTaskAssignment(workflow("Input"),task("EVI"),orderNumber(1)).

error(workflow("Input"),reason(partial_order_violation),task("EVI"),task("EPT")).

workflowTaskAssignment(workflow("Output"),task("EVI"),orderNumber(0)).
workflowTaskAssignment(workflow("Output"),task("EPT"),orderNumber(1)).
```

Combining modeling & reasoning

There is no one-solution-fits-all approach that suits both:

1. design-time experts which need an easy-to-use and easy-to-understand workflow meta-model for expressing their domain knowledge about valid workflow models
2. service providers that prefer to implement workflow validation & repair based on a declarative approach stating only "*what* is to be computed, but not necessarily *how* it is to be computed" [4]

Combine the best of both worlds by handling each part of the problem with a different technology:

- use a DSML for encoding the design-time requirements of the domain
- use ASP for run-time reasoning on this domain
- since both use equal domain concepts, the meta-model can directly be mapped between them

High-level implementation overview

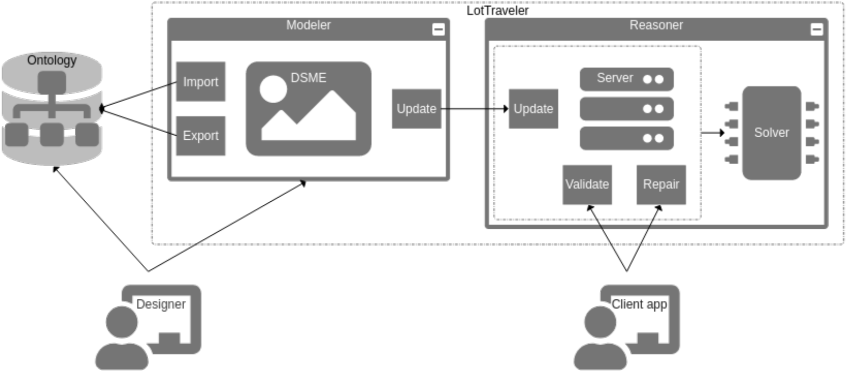

Figure: Overview of LotTraveler's implementation, its modules, users & external components, and the available interactions between them

Modeler

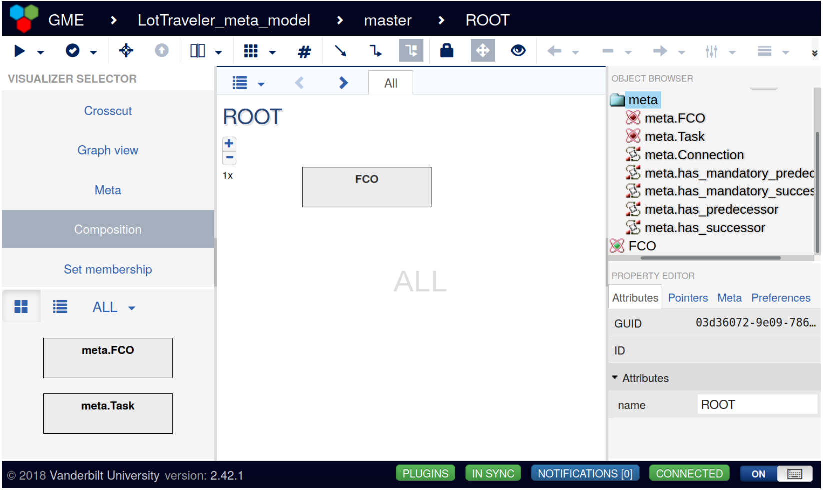

Figure: GUI of the WebGME modeler

Modeler: Create task

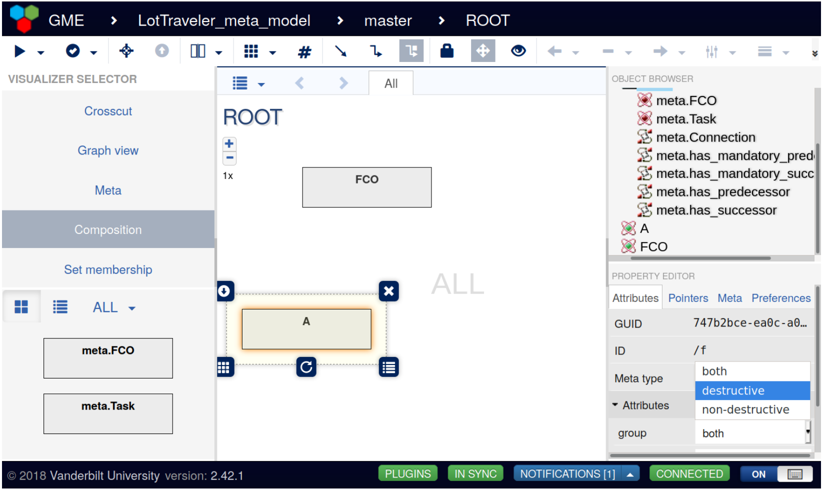

Figure: State of the workflow meta-model after adding a task

Modeler: Create connection

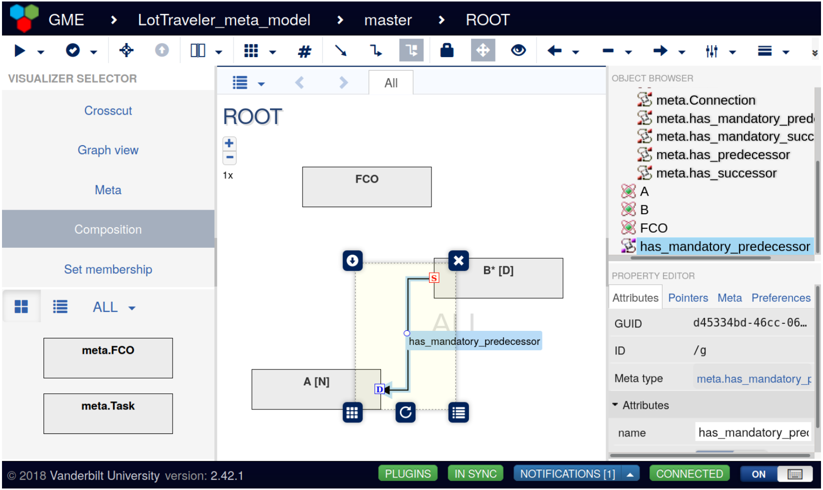

Figure: State of the workflow meta-model after adding a connection

Modeler: Error

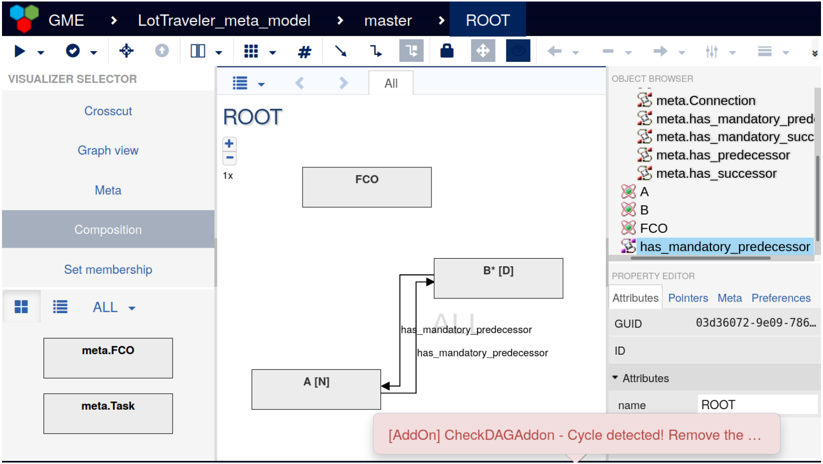

Figure: State of the workflow meta-model after creating a cycle

Modeler: Export

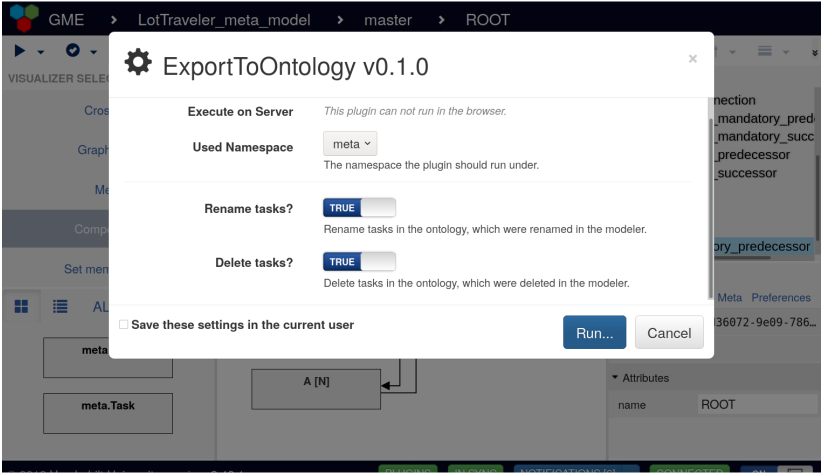

Figure: State of the workflow meta-model during export

Modeler: Import

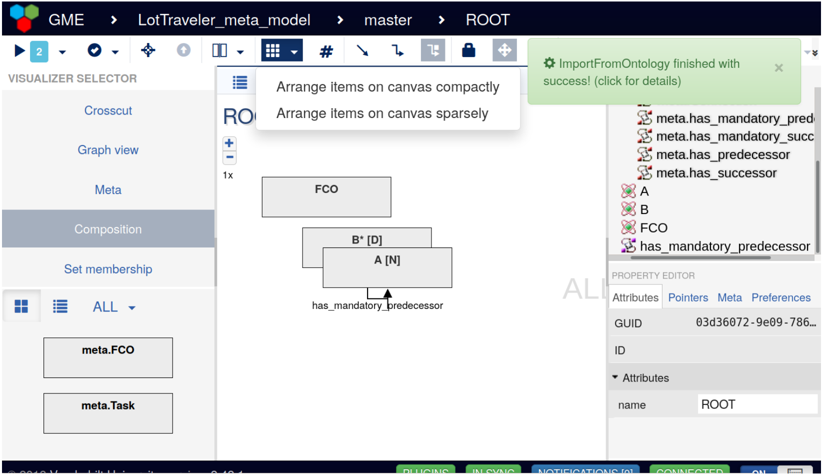

Figure: State of the workflow meta-model after import

Modeler: Update reasoner

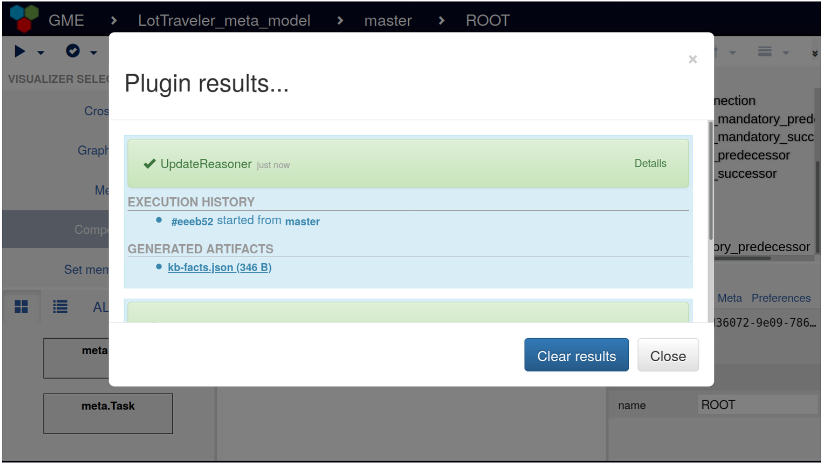

Figure: Result dialog after updating the reasoner component

Reasoner

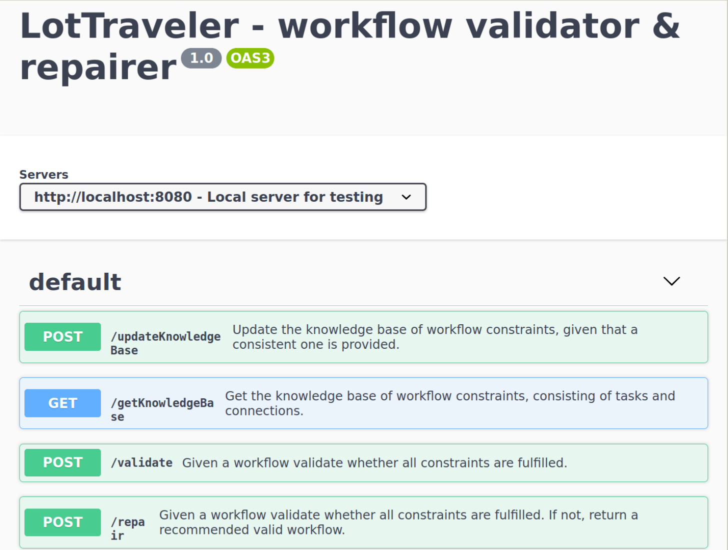

Figure: GUI of the reasoner's REST playground

Reasoner: Get KB

Request URL

```
http://localhost:8080/getKnowledgeBase
```

Server response

Code	Details
200	**Response body**


```
{
  "tasks": [
    {
      "name": "A",
      "repeatable": false,
      "group": "non-destructive"
    },
    {
      "name": "B",
      "repeatable": true,
      "group": "destructive"
    }
  ],
  "connections": [
    {
      "name": "has_mandatory_predecessor",
      "transitive": true,
      "srcName": "B",
      "dstName": "A"
    }
  ]
}
```


Download

Figure: Response of retrieving the current knowledge base

Reasoner: Update KB (success)

Request body required

application/json

The potential new knowledge base of workflow constraints

Examples: [Modified value]


```json
{
  "tasks": [
    {
      "name": "A",
      "repeatable": false,
      "group": "non-destructive"
    },
    {
      "name": "C",
      "repeatable": true,
      "group": "destructive"
    }
  ],
  "connections": [
    {
      "name": "has_mandatory_predecessor",
      "transitive": true,
      "srcName": "C",
      "dstName": "A"
    }
  ]
}
```


Execute | Clear

Figure: Request for updating the current knowledge base with a valid one

Reasoner: Validate ["A", "C"]

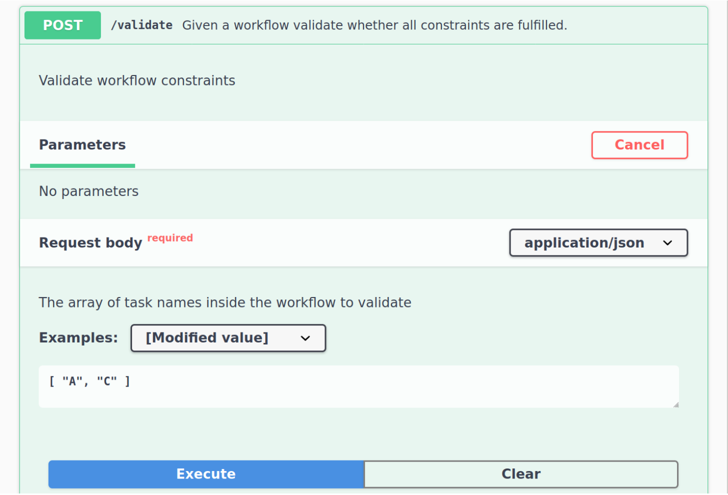

Figure: Request for checking a valid workflow

Reasoner: Repair ["A", "C", "A", "C"]

Request URL

```
http://localhost:8080/repair
```

Server response

Code	Details
200	**Response body**


```
{
  "errors": [
    {
      "errorCode": "reason(non_repeatable)",
      "description": "reason(non_repeatable)",
      "errorArg0": "task(\"A\")"
    },
    {
      "errorCode": "reason(destructive_before_non_destructive_task)",
      "description": "reason(destructive_before_non_destructive_task)",
      "errorArg0": "task(\"C\")"
    },
    {
      "errorCode": "reason(partial_order_violation)",
      "description": "reason(partial_order_violation)",
      "errorArg0": "task(\"A\")",
      "errorArg1": "task(\"C\")"
    }
  ],
  "recommendation": [
    "A",
    "C",
    "C"
  ]
}
```


Download

Figure: Response of repairing an invalid workflow

Reasoner: Update KB (error)

Figure: Response of updating the current knowledge base with an invalid one

Conclusion

The following contributions are made, in order of significance:

- A customized approach that addresses the particular requirements of workflow management in *typical* production environments, alongside a formal specification thereof.
- A working and tested system implementing the proposed approach, with auxiliary materials, such as setup and run instructions, as well as a comprehensive user guide.
- Usage examples of various technologies for addressing not only the problem statement but also for further application areas that may be of interest in the pursuit of Industry 4.0.
- An overview of theoretical foundations of the underlying concepts behind this work so that interested readers may gain a deeper understanding of such concepts.
- Bug reports for two defects in the reference implementation of the W3C Shapes Constraint Language, which were fixed by the authors in the meantime.

Summary

In summary, this work implements a solution which improves the day-to-day operations involving custom-tailored workflows in a *typical* production environment.

A combination of various technologies was used in the life-cycle of this work, from the design phase to the actual system implementation & tests, in order to overcome respective challenges with appropriate tools.

References

Wil MP Van der Aalst. *Process mining: data science in action*. Springer, 2016.

Amine Abbad Andaloussi et al. "On the declarative paradigm in hybrid business process representations: A conceptual framework and a systematic literature study". In: *Information Systems* 91 (2020), p. 101505. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2020.101505. URL: https://www.sciencedirect.com/science/article/pii/S0306437920300168.

G. Grambow, R. Oberhauser, and M. Reichert. "Semantically-Driven Workflow Generation Using Declarative Modeling for Processes in Software Engineering". In: *2011 IEEE 15th International Enterprise Distributed Object Computing Conference Workshops*. 2011, pp. 164–173. DOI: 10.1109/EDOCW.2011.54.

John W. Lloyd. "Practical Advtanages of Declarative Programming". In: *1994 Joint Conference on Declarative Programming, GULP-PRODE'94 Peñiscola, Spain, September 19-22, 1994, Volume*

Outro

Thank you for your attention!
Questions?