A DSML for reversible transformations

Mickaël Kerboeuf, Jean-Philippe Babau

LISyC – University of Western Brittany, France

{kerboeuf,babau}@univ-brest.fr
Motivation

- Tools for DSML environments
- Reuse of legacy tools (transformations)
Motivation

- Tools for DSML environments
- Reuse of legacy tools (*transformations*)

![Diagram showing data in legacy tool, specific context, input metamodel, legacy tool, output metamodel, and specific tool]
Motivation

- Tools for DSML environments
- Reuse of legacy tools (transformations)

![Diagram]

- input metamodel
- legacy tool
- output metamodel
- specific context
- data

specific tool (by transitivity)
Motivation

- Tools for DSML environments
- Reuse of legacy tools (transformations)
Motivation

- Tools for DSML environments
- Reuse of legacy tools (transformations)

Modif: a DSL for transformations between structurally and semantically close subsets of metamodels
Motivation

- Rewriting tools (*endogenous* transformations)
- The tool’s outcome has to be put back into the original context
Outline

1. Motivation
2. Modif
   - Transformations targeted by Modif
   - Example: a simulator for *Finite State Machines* (FSM)
3. Reuse of rewriting tools
   - Addressed problem
   - Similar problems in other domains
   - Storage and recovery of deleted data
   - Join algorithm
   - Limitations
   - Example: a *flattener* for *Finite State Machines* (FSM)
4. Current and future works
5. Conclusion
Transformations targeted by Modif

Metamodel refactoring

*delete, rename, change, hide and flatten* inheritance

Process and tool support

1. generation of a *by default* specification
2. Modif update
3. generation of the target metamodel
4. verification of matching
5. generation of *injection*
Transformations targeted by Modif

**Metamodel refactoring**

*delete, rename, change, hide and flatten* inheritance

**Process and tool support**

1. generation of a *by default* specification
2. Modif update
3. generation of the target metamodel
4. verification of matching
5. generation of injection
Transformations targeted by Modif

Metamodel refactoring

*delete, rename, change, hide and flatten* inheritance

Process and tool support

1. generation of a *by default* specification
2. Modif update
3. generation of the target metamodel
4. verification of matching
5. generation of *injection*
Transformations targeted by Modif

**Metamodel refactoring**

*delete, rename, change, hide and flatten* inheritance

**Process and tool support**

1. generation of a *by default* specification
2. Modif update
3. generation of the target metamodel
4. verification of matching
5. generation of injection
Transformations targeted by Modif

Metamodel refactoring

*delete, rename, change, hide and flatten* inheritance

Process and tool support

1. generation of a *by default* specification
2. Modif update
3. generation of the target metamodel
4. verification of matching
5. generation of injection

![Diagram showing the process and tool support steps]

M. Kerboeuf & J.-P. Babau – *A DSML for reversible transformations*
Transformations targeted by Modif

Metamodel refactoring

delete, rename, change, hide and flatten inheritance

Process and tool support

1. generation of a by default specification
2. Modif update
3. generation of the target metamodel
4. verification of matching
5. generation of injection

Modif tool's input

tool's input

full matching

in

input

legacy tool

out

injection

specific context

specific tool (by transitivity)

Modif

data

M. Kerboeuf & J.-P. Babau – A DSML for reversible transformations
Example: a simulator for *Finite State Machines* (FSM)

![UML diagram]

**Legacy tool**

1. takes a FSM complying with the simple FSM metamodel
2. takes a set of incoming events
3. produces a sequence of reached states as an execution trace
Example: a simulator for **Finite State Machines (FSM)**

- **Motivation**
- **Modif**
- Reuse of rewriting tools
- Current and future works
- **Conclusion**

**Reuse of FSM simulator with UML thanks to Modif**

1. default Modif specification for UML: *deletes everything*
2. update: *meaningful concepts* for FSM are kept and renamed
3. target metamodel generation
4. target metamodel fully matches the simple FSM metamodel
5. generation of injection

**UML statecharts**

- much larger metamodel!
- come with tool support (e.g., editors)
Example: a simulator for *Finite State Machines (FSM)*

**Motivation**

**Modif**

**Reuse of rewriting tools**

**Current and future works**

**Conclusion**

**Example:** a simulator for *Finite State Machines (FSM)*

**UML statecharts**

- much larger metamodel!
- come with tool support (e.g. editors)

---

**Reuse of FSM simulator with UML thanks to Modif**

1. default Modif specification for UML: *deletes everything*
2. update: *meaningful concepts* for FSM are kept and renamed
3. target metamodel generation
4. target metamodel fully matches the simple FSM metamodel
5. generation of injection
Example: a simulator for *Finite State Machines* (FSM)

**Reuse of FSM simulator with UML thanks to Modif**

1. default Modif specification for UML: *deletes everything*
2. update: *meaningful concepts* for FSM are kept and renamed
3. target metamodel generation
4. target metamodel fully matches the simple FSM metamodel
5. generation of injection

**UML statecharts**
- much larger metamodel!
- come with tool support (e.g. editors)
Example: a simulator for *Finite State Machines (FSM)*

Motivation

Modif

Reuse of rewriting tools

Current and future works

Conclusion

**Example: a simulator for Finite State Machines (FSM)**

**UML statecharts**
- much larger metamodel!
- come with tool support (e.g. editors)

**Reuse of FSM simulator with UML thanks to Modif**

1. default Modif specification for UML: *deletes everything*
2. update: *meaningful concepts* for FSM are kept and renamed
3. target metamodel generation
4. target metamodel fully matches the simple FSM metamodel
5. generation of injection
Example: a simulator for *Finite State Machines (FSM)*

- **Motivation**
- **Modif**
- **Reuse of rewriting tools**
- **Current and future works**
- **Conclusion**

**Reuse of FSM simulator with UML thanks to Modif**

1. default Modif specification for UML: *deletes everything*
2. update: *meaningful concepts* for FSM are kept and renamed
3. target metamodel generation
4. target metamodel fully matches the simple FSM metamodel
5. generation of injection

**UML statecharts**

- much larger metamodel!
- come with tool support (e.g. editors)
Example: a simulator for *Finite State Machines* (FSM)

Reuse of FSM simulator with UML thanks to Modif

1. default Modif specification for UML: *deletes everything*
2. update: *meaningful concepts* for FSM are kept and renamed
3. target metamodel generation
4. target metamodel fully matches the simple FSM metamodel
5. generation of injection

**UML statecharts**
- much larger metamodel!
- come with tool support (e.g. editors)
**Reusability of tool’s outcome**

**Contextualization**

A Modif specification can be *inverted* to produce a transformation from the tool’s definition domain back into the specific context.

- Without deletion, the composition of *injection* and *contextualization* produces *identity*
- **Deleted data are lost**
Similar problems in other domains

The view-update problem in the context of databases

Under which conditions updates on views can be translated into updates on the underlying database?

- initial context: the underlying database
- tool's definition domain: view
- rewriting tool: set of updates applied to the view

- Modif is not restricted to functions to build views
- only the effects of the legacy tool can be analysed

Motivation | Modif | Reuse of rewriting tools | Current and future works | Conclusion
Similar problems in other domains

Bidirectional transformations (lenses)
Propagation of updates between connected structures (e.g. relevant for model versioning)

- injection and contextualization actually aim at implementing an asymmetrical lense...
- but Modif is not intended to maintain the consistency of two metamodels
Storage and recovery of deleted data

Data recovery
- data are backed up...
- and referenced by keys

- two target metamodels are generated
- the generated injection produces two sets of instances
- a specialized version of contextualization performs a joint
For each instance of \textbf{out} with key \( k \), the corresponding instance of \textbf{data+key} with key \( k \) is used to recover deleted attributes.

2. Each instance of \textbf{data+key} whose class had been deleted is recovered.

3. Each \textit{reference} of each instance of \textbf{out} which itself or its source or its target had been deleted is recovered.

4. Keys are removed and then the inverted Modif is applied.
Join algorithm

1. For each instance of \textbf{out} with key \textbf{k}, the corresponding instance of \textbf{data+key} with key \textbf{k} is used to recover deleted attributes.
2. Each instance of \textbf{data+key} whose class had been deleted is recovered.
3. Each \textit{reference} of each instance of \textbf{out} which itself or its source or its target had been deleted is recovered.
4. Keys are removed and then the inverted Modif is applied.
Join algorithm

1. For each instance of `out` with key `k`, the corresponding instance of `data+key` with key `k` is used to recover deleted attributes.

2. Each instance of `data+key` whose class had been deleted is recovered.

3. Each `reference` of each instance of `out` which itself or its source or its target had been deleted is recovered.

4. Keys are removed and then the inverted Modif is applied.
Join algorithm

1. For each instance of out with key \( k \), the corresponding instance of data+key with key \( k \) is used to recover deleted attributes.
2. Each instance of data+key whose class had been deleted is recovered.
3. Each reference of each instance of out which itself or its source or its target had been deleted is recovered.
4. Keys are removed and then the inverted Modif is applied.
Join algorithm

1. For each instance of `out` with key `k`, the corresponding instance of `data+key` with key `k` is used to recover deleted attributes.
2. Each instance of `data+key` whose class had been deleted is recovered.
3. Each reference of each instance of `out` which itself or its source or its target had been deleted is recovered.
4. Keys are removed and then the inverted Modif is applied.
The legacy tool is a *black box*

For now, only the *effects* of rewriting tools can be observed

- no traceability between new instances and the existing instances that have been used to compute them
- the way the legacy tool is designed impacts its *reusability*:
  - it is higher for rewriting tools that give priority to *update* instead of *recreation*
- the generated tools are *helpers*
Example: a flattener for **Finite State Machines (FSM)**

**Legacy rewriting tool: a flattener for simple FSM**

**Specific context: UML statecharts**

*including things that should be deleted (e.g. actions)*...
Example: a flattener for *Finite State Machines* (FSM)

Recovered and *lost* data

```
begin
idle
running
entry / start
exit / stop
nominal
entry / run1
degraded
ok
nok / runD
```

M. Kerboeuf & J.-P. Babau – A DSML for reversible transformations
Example: a flattener for **Finite State Machines (FSM)**

Recovered and *lost* data

![Diagram of FSM states and transitions](image)
Example: a flattener for Finite State Machines (FSM)

Recovered and lost data

M. Kerboeuf & J.-P. Babau – A DSML for reversible transformations
Example: a flattener for Finite State Machines (FSM)

Recovered and *lost* data
### Current and future works

**Graph semantics**

- Denotational semantics of Modif to formally:
  - prove the soundness of the join algorithm
  - identify a minimal set of core refactoring operations

**Ability to *add* data**

- customizable rules to create default new instances

**Levenshtein distance between two metamodels**

- to *generate* a modif specification between two given metamodels
- the smallest distance between two semantically close metamodels is expected to be relevant
Modif

- a transformation language dedicated to semantically close metamodels
- aims at promoting the reusability of legacy transformations

A relational mechanism upon Modif

- to enable the reuse of endogenous transformations...
- especially when they update instances

J.-P. Babau and M. Kerboeuf.
Domain Specific Language Modeling Facilities.
In proceedings of the 5th MoDELS workshop on Models and Evolution, 2011.