Model Evolution
Towards Live Domain-Specific Languages

IPA Fall Days – Garderen, Friday November 2nd 2018

Riemer van Rozen\textsuperscript{1,2,3} @rvrozen

joint work with Tijs van der Storm\textsuperscript{2,4} @tvdstorm

\textsuperscript{1} Amsterdam University of Applied Sciences (HvA)
\textsuperscript{2} Centrum Wiskunde & Informatica (CWI)
\textsuperscript{3} University of Amsterdam – Master SE (UvA)
\textsuperscript{4} University of Groningen (RUG)
Textual Models

- Models encoded as text
  - Textual DSLs
  - Programming Languages

- DSL for the Game Domain: Micro-Machinations is a language and library that enables game designers to modify a game’s rules at run-time.

- Example: Johnny Jetstream

Source:

```
kill
income: kill -10-> gold
pool gold is "$" at 20
cost: gold -10-> buyHp
user converter buyHp
benefit: buyHp -20-> hp
pool hp is "+" at 100
damage: hp -10-> hit
drain hit
```
Textual Models

• Evolution perspective
  – Changes between different versions of a program
  – Live DSLs modify running programs

• How to (1) determine the difference between two textual models and (2) evolve running programs?

Step 1: Play Test v1

Step 2: Re-design

Step 3: Play Test v2
Live Modeling aims to bridge the “gulf of evaluation” (D. Norman 1988)

**Goal:** Support gradually Improving insight “voortschrijdend inzicht” and mental models of how source code changes affect systems

**Problem:** cognitive gap between user action and feedback on that action + long edit-compile cycles
Live Programming

• Live programming aims to bridge the gulf of evaluation by shortening the feedback loop between editing a program’s textual source code and observing its behavior.

• In a live programming environment, the running program is updated instantly after every change in the code.
  – see the behavioral effects of actions immediately
  – learn predicting how the program adapts to targeted improvements to the code

• Question: how to bridge the gap between running programs and textual DSLs?
Suggestion: Not State Machines

• Games Research
  – "Applications to games other than Super Mario Bros are especially welcome" – Call for papers of the Procedural Content Generation in Games Workshop.

• Language Research
  – "Applications to languages other than State Machines are especially welcome" – future call for papers

• Suggested Alternatives
  – Behavior Trees
  – PuzzleScript
    [https://www.puzzlescript.net](https://www.puzzlescript.net)
  – Machinations

Pros: state machines are simple, explainable, research can be compared
Cons: state machines may not be representative, tedious repetition
Problem Statement and Objectives

• **Challenge:** How to build DSLs for live programming?

• **Objective:** provide generic language technology for constructing DSLs for live programming

• **Question:** How can a textual difference between successive source code versions and origin tracking be leveraged for obtaining a run-time difference in behavior?

```
foo.lang  execute  Behavior(foo)
     ↓   ↓         ↓
        ?           ?
       “diff”          
       ↓    execute   
foo’.lang  →  Behavior(foo’)
```
Approach

- **Approach:** Apply Textual Model Differencing (TMDiff) to obtain model-based deltas and Run-time Model Patching (RMPatch) to migrate models at run time.
  - Program migrations as part of the language semantics
  - One correct result of a state migration is assumed

```
foo.lang --> MM --> MM+
|        | TMDiff
|        | “diff”
|        | Δ(MM) --> ⌦[δ]
|        | RMPatch
foo’.lang --> MM+
```

### Programming Environment

- **Edit**
- **Textual Model**
- **TMDIFF**

### Running Program

- **Events**
- **Model + State**
- **Delta**
- **RMPATCH**
Background:
Difference and Union of Models

UML, 2003

Marcus Alanen and Ivan Porres
Model Differencing

• Difference and Union of Models
  – **Context:** version control
  – **Motivation:** Two designers make separate changes to a model. How to merge the two models?

**Source:** Marcus Alanen and Ivan Porres. Difference and union of models. UML 2003.

**Source:** Ivan Porres, Difference and Union of Models, 10 years later (invited presentation). MODELS 2013

![Diagram](image)
Model Differencing

- **Difference and Union of Models**
  - **Difference.** calculate the difference between two models. \( M_2 - M_1 = \Delta \)
  - **Union.** merging two models by applying the difference. \( M_1 + \Delta = M_2 \)

**Source:** Marcus Alanen and Ivan Porres. Difference and union of models. UML 2003.

**Source:** Ivan Porres, Difference and Union of Models, 10 years later (invited presentation). MODELS 2013
Edit Script Operations

• Edit script operations
  – Differences or deltas are expressed as a sequence of operations, the definition of Δ.

• Element creation and deletion
  – **new**(*e, t*) : Create a new element of type *t* with UUID *e*. By default, a new element has all its features set to their default values.
  
  – **del**(*e, t*) : Delete an element of type *t* with UUID *e*. An element may only be deleted if all its features are set to their default values.

Edit Script Operations

- Modification of a feature of type $f$ of an element with UUID $e$. Where necessary, $e_t$ refers to another element.
  - $\text{set}(e, f, v_o, v_n)$: Set the value of $e.f$ from $v_o$ to $v_n$ for an attribute of primitive type.
  - $\text{insert}(e, f, e_t)$: Add a link from $e.f$ to $e_t$, for an unordered feature.
  - $\text{insertAt}(e, f, e_t, i)$: Add a link from $e.f$ to $e_t$, at index $i$, for an ordered feature.
  - $\text{removeAt}(e, f, e_t, i)$: Remove a link from $e.f$ to $e_t$, which is at index $i$, for an ordered feature.

<table>
<thead>
<tr>
<th>Operation $O$</th>
<th>Dual operation $\tilde{O}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{new}(e, t)$</td>
<td>$\text{del}(e, t)$</td>
</tr>
<tr>
<td>$\text{del}(e, t)$</td>
<td>$\text{new}(e, t)$</td>
</tr>
<tr>
<td>$\text{set}(e, f, v_o, v_n)$</td>
<td>$\text{set}(e, f, v_n, v_o)$</td>
</tr>
<tr>
<td>$\text{insert}(e, f, e_t)$</td>
<td>$\text{remove}(e, f, e_t)$</td>
</tr>
<tr>
<td>$\text{remove}(e, f, e_t)$</td>
<td>$\text{insert}(e, f, e_t)$</td>
</tr>
<tr>
<td>$\text{insertAt}(e, f, e_t, i)$</td>
<td>$\text{removeAt}(e, f, e_t, i)$</td>
</tr>
<tr>
<td>$\text{removeAt}(e, f, e_t, i)$</td>
<td>$\text{insertAt}(e, f, e_t, i)$</td>
</tr>
</tbody>
</table>

Table 1: The Map Between Operations and Dual Operations.

Edit Script Example

$\Delta = [[\text{new}(\text{Class}, u_2),
\text{new}(\text{Generalization}, u_3)],
[\text{insert}(u_3, \text{namespace}, u_0),
\text{insert}(u_3, \text{parent}, u_1),
\text{insert}(u_3, \text{child}, u_2),
\text{insert}(u_1, \text{specialization}, u_3),
\text{insert}(u_0, \text{ownedElement}, u_2),
\text{insert}(u_0, \text{ownedElement}, u_3),
\text{insert}(u_2, \text{namespace}, u_0),
\text{insert}(u_2, \text{generalization}, u_3),
\text{set}(u_2, \text{name}, "\text{"}, \text{"Sub")}],
[ ]
]

Figure 4: Difference Between Two Simple Models.

Implications, Benefits and Limitations

• Differences can be
  – Programmed manually
  – Leveraged for algorithms and modeling tools
  – Generated from DSLs
  – Recorded, played back
  – Applied on systems and rolled back
  – Analyzed formally for predicting results
  – Used for understanding the evolution of models

• Main limitations of A&P approach.
  – Requires unique, stable, universal model element identifiers across model revisions.
  – Metamodel is assumed to be static.

• In addition: Encode history, NOT scripts! (operations go stale)

Source: Ivan Porres, Difference and Union of Models, 10 years later (invited presentation). MODELS 2013
Origin Tracking + Text Differencing = Textual Model Differencing

Theory and Practice of Model Transformations, 2015

Riemer van Rozen\textsuperscript{1,2,3} and Tijs van der Storm\textsuperscript{2,4}

\texttt{@rvrozen} \hspace{1cm} \texttt{@tvdstorm}

\textsuperscript{1} Amsterdam University of Applied Sciences (HvA)
\textsuperscript{2} Centrum Wiskunde & Informatica (CWI)
\textsuperscript{3} University of Amsterdam – Master SE (UvA)
\textsuperscript{4} Rijksuniversiteit Groningen (RUG)
Problem: Differencing with identity

Doors Model (v1):

- d1: Machine
- d2: State
  - : Trans
    - event: “close”
- d3: State
  - : Trans
    - event: “open”

Doors Model (v2):

- d1: Machine
- d2: State
- d3: State
- d4: State
  - : Trans
    - event: “lock”
  - : Trans
    - event: “unlock”

Problem

- We cannot simply apply model differencing to models encoded as text.
Problem: Textual Model Differencing

• What are the entities?
  – First parse to obtain a tree
  – Referential structure is determined by scoping rules
    • Definitions: machine, state
    • Uses: transition

• Problem
  – Textual model elements have no stable identity across source versions.

Doors.sml (v1):

1 machine doors d1
2 state closed d2
3 open => opened u1
4 state opened d3
5 close => closed u2
6 end
Doors.sml (v1):
1  machine doors
2    state closed
3      open => opened
4
5    state opened
6      close => closed
7    end

Doors.sml (v2):
1  machine doors
2    state closed
3      open => opened
4      lock => locked
5
6    state opened
7      close => closed
8
9    state locked
10      unlock => closed
11    end

Textual model elements have no stable identity across source versions
Objectives: Computing Deltas

- **Question**
  - How to apply model differencing to models encoded as text?

- **What are the differences?**
  - Imperative edit scripts encode deltas
  - Multiple deltas can express the difference between two models → ambiguity
  - Deltas can capture user intent

```python
//create a State def with label d7
create State d7
//initialize the new State "locked"
d7 = State("locked",[Trans("unlock",d2)])
//store 2nd Trans in state "closed"
d2.out[1] = Trans("lock",d7)
//store new State
d1.states[2] = d7
```
Contributions

• Question
  – How can **textual differencing** be used to match model elements based on **origin tracking**?

• Contributions
  – TMDiff
  – Apply TMDiff to DSL programs
Objectives: Computing Deltas

• **Origin**
  - \( \text{src}_n \) has an origin relation with \( m_n \)

• **Align**
  - Use the text diff \( \Delta \) between \( \text{src}_1 \) and \( \text{src}_2 \) to align tokens of entities.

• **Objective: Identify**
  - Given textual models \( \text{src}_1 \) and \( \text{src}_2 \) determine which entities in \( m_1 \) are still in \( m_2 \)
Approach: TMDiff

- **TMDiff steps**
  - **Matching:** generate a tuple of added, removed and identified entities
  - **Added:** generate *Create* and *SetTree* operations
  - **Identified:** *difference nodes* definitions
  - **Removed:** generate *Delete* operations

```
str src1  obj m1  TMDiff  str src2  obj m2
```

```
list[Operation]
```
Matching Entities: Text diff

1 machine doors 1 machine doors
2 state closed 2 state closed
3 open => opened 3 open => opened

4 lock => locked
5 state opened
6 close => closed
7 end
8
9
10
11

--- a/doors1.sl
+++ b/doors2.sl
@@ -3,0 +4
@@ -6,0 +8,3
+ lock => locked
+ state locked
+ unlock => closed
Matching Entities: Project, Identify

**P1** =

\[
\begin{align*}
\text{machine} & \text{ doors} (d_1) \\
\text{state} & \text{ closed} (d_2) \\
\text{open} & \Rightarrow \text{ opened} \\
\text{lock} & \Rightarrow \text{ locked} \\
\text{open} & \Rightarrow \text{ opened} \\
\text{closed} & \Rightarrow \text{ closed} \\
\text{open} & \Rightarrow \text{ opened} \\
\text{locked} & \Rightarrow \text{ closed} \\
\text{open} & \Rightarrow \text{ opened} \\
\text{locked} & \Rightarrow \text{ closed} \\
\text{locked} & \Rightarrow \text{ closed} \\
\text{locked} & \Rightarrow \text{ closed} \\
\text{locked} & \Rightarrow \text{ closed} \\
\text{locked} & \Rightarrow \text{ closed} \\
\text{locked} & \Rightarrow \text{ closed} \\
\end{align*}
\]

\[
\begin{align*}
\text{P1} & = \\
\{\langle \text{doors}, \text{ Machine}, 1, d_1 \rangle, \\
\langle \text{closed}, \text{ State}, 2, d_2 \rangle, \\
\langle \text{open}, \text{ State}, 5, d_3 \rangle\}
\end{align*}
\]

\[
\begin{align*}
\text{P2} & = \\
\{\langle \text{doors}, \text{ Machine}, 1, d_4 \rangle, \\
\langle \text{closed}, \text{ State}, 2, d_5 \rangle, \\
\langle \text{open}, \text{ State}, 6, d_6 \rangle, \\
\langle \text{locked}, \text{ State}, 9, d_7 \rangle\}
\end{align*}
\]

- Calculate Matching
  - added, removed, identified entities
  - \( M_{1,2} = \langle \{d_7\}, \{\}, \{\langle d_1, d_4 \rangle, \langle d_2, d_5 \rangle, \langle d_3, d_6 \rangle\} \rangle \)
Differencing

• We now have
  – Textual sources
  – Models
  – Origin relations
  – Matching

• We now can
  – Apply well-known model differencing algorithms.
Implementation & Evaluation

• Rascal
  – Meta-programming language and language work bench
    http://www.rascal-mpl.org
  – TMDiff
    https://github.com/cwi-swat/textual-model-diff

• Evaluated on Derric
  – A DSL for digital forensics
  – Describes file formats for analyzing large amounts of unstructured data.
  – File format evolution is available on GitHub.
    https://github.com/jvdb/derric

```plaintext
format gif
extension gif

strings ascii
sign false
unit byte
size 1
type integer
_endian big

Sequence
(Header87a Header89a)
  LogicalScreenDesc
  (  
    [GraphicControlExtension? TableBasedImage
      CompressedDataBlock*]  
    [GraphicControlExtension?
      PlaintextExtension DataBlock*]  
    [ApplicationExtension DataBlock*]  
    [CommentExtension DataBlock*]  
)"
```
Towards Live Domain-Specific Languages
From Text Differencing to Adapting Models at Runtime

Journal of Software & Systems Modeling, 2017

Riemer van Rozen\textsuperscript{1,2,3} and Tijs van der Storm\textsuperscript{2,4}

@rvrozen \hspace{1cm} @tvdstorm

\textsuperscript{1} Amsterdam University of Applied Sciences (HvA)
\textsuperscript{2} Centrum Wiskunde & Informatica (CWI)
\textsuperscript{3} University of Amsterdam – Master SE (UvA)
\textsuperscript{4} University of Groningen (RUG)
Case Study: Live SML

- **LiveSML Metamodels**
  a) Static metamodel
  b) Dynamic metamodel extension:
    - Machine current state
    - State count

- **Note:** The run-time meta-model of LiveSML “extends” its static meta-model, which is not true in general
Live SML: Components & Models

- Live SML components
  a) programming environment
  b) program execution as an interactive GUI

- Live SML Models
  c) static SML model representing the textual source code
  d) dynamic SML model that is executing at run time
Live SML: State Migration

- **Creation of a new machine**
  - Initially there is no machine because we start with an empty object space.
  - We store a reference to the machine when it is first created (lines 9 and 10).

- **Creation of a new state**
  - The *count* attribute is initialized to 0 (lines 12–15).

```java
class MigrateSML extends ApplyDelta {
    private Mach machine; // run-time model to migrate

    @Override
    public void visit(Create create) {
        super.visit(create);

        Object x = create.getCreated(this);
        if (x instanceof Mach) { // new machine
            this.machine = (Mach) x;
        } else if (x instanceof State) { // new state
            Edit e = new SetPrim(reverseLookup(x),
                                  new Path(new Field("count")), 0);
            e.accept(this);
        }
    }
}
```
Live SML – State Migration

• **Insertion of an element in an uninitialized machine.**
  - When a state or group is inserted into a machine that has no current state (lines 24–29), it is initialized to the initial state (lines 43–54).
  - The initial state is the first state in the textual model.

• **Deletion of the current state**
  - When a machine’s current state is deleted (lines 36–37), it is reinitialized to the initial state (lines 43–54).

```java
@Override
public void visit(Insert insert) {
    super.visit(insert);

    Object owner = insert.getOwner(this);
    if (machine != null && machine.state == null
        && owner == machine) {
        // Added a group or state to a machine
        // without a current state.
        goToInitialState();
    }
}
```

```java
@Override
public void visit(Delete delete) {
    super.visit(delete);

    Object x = delete.getDeleted(this);
    if (machine != null && x == machine.state) {
        // Deleted the current state.
        goToInitialState();
    }
}
```
Live SML – State Migration

• Insertion of an element in an uninitialized machine.
  – When a state or group is inserted into a machine that has no current state (lines 24–29), it is initialized to the initial state (lines 43–54).
  – The initial state is the first state in the textual model.

• Deletion of the current state
  – When a machine’s current state is deleted (lines 36–37), it is reinitialized to the initial state (lines 43–54).

```java
private void goToInitialState()
{
    State s = machine.findInitial();
    Edit e1 = new Edit(reverseLookup(machine),
                       new Path(new Field("state")), s);
    e1.accept(this); // Set the current state.

    if (s != null)
    {
        Edit e2 = new Edit(reverseLookup(s),
                            new Path(new Field("count")), s.count+1);
        e2.accept(this); // Increment current state count.
    }
}
```
Live State Machine Language in Rascal

```rascal
machine doors
  state closed
    open => opened
  state opened
    close => closed
end
```

State machine

<table>
<thead>
<tr>
<th>State</th>
<th>#</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>closed</td>
<td>1</td>
<td>[open]</td>
</tr>
<tr>
<td>opened</td>
<td>0</td>
<td>[close]</td>
</tr>
</tbody>
</table>
Live SML: Modeling Scenario

- Interleaved coevolution of models $Doors_n$ and application run-time states $S_n$ over time

- Next: TMDiff deltas + migration deltas
## Live SML: Modeling Scenario

<table>
<thead>
<tr>
<th>Model</th>
<th>State</th>
<th>Event</th>
<th>Edit operation</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>s0</td>
<td>Save Doors₁</td>
<td>δ₁ create State d₂</td>
<td>TMDiff ∅ Doors₁</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ₂ d₂.count = 0</td>
<td>side effect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ₃ create State d₃</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ₄ d₃.count = 0</td>
<td>side effect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ₅ create Mach d₁</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ₆ d₂ = State(name(&quot;closed&quot;),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[Trans(&quot;open&quot;,d₃)])</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ₇ d₃ = State(name(&quot;opened&quot;),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[Trans(&quot;close&quot;,d₂)])</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ₈ d₁ = Mach(name(&quot;doors&quot;),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[d₂,d₃])</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ₉ d₁.state = d₂</td>
<td>side effect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ₁₀ d₂.count = 1</td>
<td>side effect</td>
</tr>
</tbody>
</table>

At the end of this sequence we are in Model Doors₁ and State s₁.
<table>
<thead>
<tr>
<th>Model</th>
<th>State</th>
<th>Event</th>
<th>Edit operation</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doors1</td>
<td>s1</td>
<td>Click open</td>
<td>δ11  d1.state = d3</td>
<td>user action</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ12  d3.count = 1</td>
<td></td>
</tr>
<tr>
<td>Doors1</td>
<td>s2</td>
<td>Click close</td>
<td>δ13  d1.state = d2</td>
<td>user action</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ14  d2.count = 2</td>
<td></td>
</tr>
<tr>
<td>Doors1</td>
<td>s3</td>
<td>Save Doors2</td>
<td>δ15  create State d7</td>
<td>TMDiff Doors1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ16  d7.count = 0</td>
<td>Doors2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ17  d7 = State(name(&quot;locked&quot;), [Trans(&quot;unlock&quot;,d2)])</td>
<td>side effect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ18  insert d2.transitions[1] = Trans(&quot;lock&quot;,d7)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ19  insert d1.states[2] = d7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ20  rekey d1 → d4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ21  rekey d2 → d5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ22  rekey d3 → d6</td>
<td></td>
</tr>
<tr>
<td>Doors2</td>
<td>s4</td>
<td>Click lock</td>
<td>δ23  d4.state = d7</td>
<td>user action</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ24  d7.count = 1</td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>State</td>
<td>Event</td>
<td>Edit operation</td>
<td>Origin</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>---------</td>
<td>-------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Doors2</td>
<td>s5</td>
<td>Save Doors3</td>
<td>δ25 create Group d11</td>
<td>TMDiff Doors2 Doors3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ26 d11 = Group(&quot;locking&quot;,[d6])</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ27 remove d4.states[2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ28 insert d4.states[2] = d0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ29 rekey d4 → d8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ30 rekey d5 → d9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ31 rekey d6 → d10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ32 rekey d7 → d12</td>
<td></td>
</tr>
<tr>
<td>Doors3</td>
<td>s6</td>
<td>Save Doors1</td>
<td>δ33 remove d8.states[2]</td>
<td>TMDiff Doors3 Doors1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ34 remove d9.transitions[1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ35 delete d11</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ36 delete d12</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ37 d13.state = d9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ38 d9.count = 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ39 rekey d8 → d13</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ40 rekey d9 → d14</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>δ41 rekey d10 → d15</td>
<td></td>
</tr>
</tbody>
</table>
## Discussion, Benefits and Limitations

<table>
<thead>
<tr>
<th>Feature / benefit</th>
<th>Trade-off / limitation</th>
<th>Mitigating argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit operations: record history as edit scripts for do, undo, replay</td>
<td>Large memory footprint, a potential memory leak</td>
<td>Recording differences can be turned off or limited</td>
</tr>
<tr>
<td>TMDiff is language-parametric (needs name resolution) and calculates model-based</td>
<td>The results of the differencing algorithm bleed into the language semantics (which entities live and die)</td>
<td>Facilitates rapid Live prototyping of DSLs for live and textual modeling. The default is usually OK due to small incremental changes</td>
</tr>
<tr>
<td>deltas “for free”</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMPatch helps construct DSL interpreters for live programming</td>
<td>High implementation effort. The granularity of edit scripts operations is too fine (does not scale).</td>
<td>Some languages require exact state migrations and precise steering</td>
</tr>
</tbody>
</table>


Conclusions and Future Work

Questions
1. How can textual differencing be used to match model elements based on origin tracking?
2. How can “Live DSL” construction be supported with generic reusable frameworks?

Contributions
- TMDiff and RMPatch
- Apply TMDiff to DSL programs
- LiveSML illustrative example

Current work
- Modeling extensible state migrations that scale to larger DSLs
- Live Machinations

References
Modeling with Side-Effects

current work

in the context of the Live Game Design RAAK-MKB project

Riemer van Rozen\textsuperscript{1,2,3}

@rvrozen

\textsuperscript{1} Amsterdam University of Applied Sciences (HvA)
\textsuperscript{2} Centrum Wiskunde & Informatica (CWI)
\textsuperscript{3} University of Amsterdam – Master SE (UvA)
Machinations Evolution & Approach

2009
Conceptual Game Design Aid

2013
Formal Analysis + text, modules

2014
Live Adaptations v1.0 C++

2015
A Pattern Based Game Design Assistant

2018
Live Adaptations v2.0 C# Unity

auto source s pool p at 7
flow: s -p-> p

MM Lib

Gameplay Engineer Player

Mechanics Patterns
Mechanics Pattern Language
Mechanics Design Assistant

Design Decision Alternatives

Game Mechanics
Advanced Game Design
Ernest Adams
Joris Dormans

Mechanics Design Assistant
Mechanics Design Assistant

Auto Source

pool p at 7
flow: s -p-> p

MM Lib

Gameplay Engineer Player

Mechanics Patterns
Mechanics Pattern Language
Mechanics Design Assistant

Design Decision Alternatives

Game Mechanics
Advanced Game Design
Ernest Adams
Joris Dormans

Mechanics Design Assistant
Mechanics Design Assistant

Auto Source
Machinations Evolution & Approach

Live Machinations: Model + State
Live Machinations: Model + State