Edit Transactions

Dynamically Scoped Change Sets for Controlled Updates in Live Programming

Toni Mattis, Patrick Rein, Robert Hirschfeld
Software Architecture Group
HPI, University of Potsdam, Germany

Programming 2017, Brussels
Live Programming

- Change & introspection of running programs
- Immediate & continuous feedback

("working definition" in the context of this talk)
Live Programming: Example

Ball >> step
   self reflect; move

Ball >> move
   self position: (self position + self speed)

Ball >> reflect "Check bounds and reflect"
Live Programming: Example

Ball >> **step**
   ```ruby
   self reflect; move; gravitate
   ```

Ball >> **move**
   ```ruby
   self position: (self position + self speed)
   ```

Ball >> **reflect** "Check bounds and reflect"

"gravitate" undefined
Live Programming

- Change & introspection of running programs
- Immediate & continuous feedback
- Fragility

We propose "Edit Transactions"

- Collect a group of changes
- Activate and deactivate atomically
**Edit Transactions: Collecting Changes**

Ball >> **step**
   self reflect; move; **gravitate**

Ball >> **move**
   self position: (self position + self speed)

Ball >> **reflect** "Check bounds and reflect"

**Edit Transaction**

---

T. Mattis, P. Rein, R. Hirschfeld | Software Architecture Group | Hasso Plattner Institute Potsdam
Edit Transactions: Collecting Changes

Ball >> step
  self reflect; move; gravitate

Ball >> gravitate [...]

Ball >> move
  self position: (self position + self speed)

Ball >> reflect "Check bounds and reflect"

Ball (class) step gravitate

Ball (class) position step move
Edit Transactions: Activation

Ball >> step
  self reflect; move; gravitate

Ball >> gravitate [...]

Ball >> move
  self position: (self position + self speed)

Ball >> reflect "Check bounds and reflect"

Ball (class) position step move gravitate

"view"

Ball (class) step gravitate

Ball (class) position step move

Base System
Edit Transactions: Undo

Ball >> step
   self reflect; move; gravitate
Ball >> gravitate [...] insert call
Ball >> move
   self position: (self position + self speed) implement method
Ball >> reflect "Check bounds and reflect"

Ball (class) position step move

Ball (class) step gravitate

Ball (class) position step move

"view"

Base System
Edit Transactions: Operations

Run-time View (active)

Edit Transaction

Base System

Atomc Activation

Capturing Changes

Remains Effectively Unmodified
Edit Transactions as Dispatchers

**Un-map** field "step", lookup stops here

**Map** name "reflect" to a new method

**Edit Transaction**

- Ball
- **x** step
- + reflect

**Edit Transactions** *(re-)*map or un-map late-bound meta-objects (Classes, Methods, Fields, ...).

**Changes** stored as **executable meta-objects**.
Adaptation:
Translation of an edit interaction into executable representation

Emergence:
Changes to executable representation taking effect
**Edit Transactions**

- decouple adaptation from emergence
- support semantic change granularity

**Adaptation:** Translation of an edit interaction into executable representation

**Emergence:** Changes to executable representation taking effect
Edit Transaction Lifecycle

Capture & View Changes
map/un-map names, use mapping in editor

Visible to Control Flow
use mapping in dispatch

Create

Staged
unstage stage

Unstaged

Active
deactivate activate

Inactive

Delete

Per Editor

Per Thread/Process

Merge
Edit Transaction Scoping

Staged only
Review code (e.g. imported from version control), check if it compiles, run static analyses ...

Control-flow Activation
Run unit tests (auto-tester), experiment with change

Thread/Global activation
Change considered "stable", asynchronous

[ block ] withTransactions: et.
et activateFor: aProcess.
et activateGlobally.
# Types of Changes (Class-based OOP)

<table>
<thead>
<tr>
<th>Meta-object</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>add</td>
</tr>
<tr>
<td></td>
<td>remove</td>
</tr>
<tr>
<td></td>
<td>update hierarchy</td>
</tr>
<tr>
<td></td>
<td>rename</td>
</tr>
<tr>
<td></td>
<td>[•••••] remove + add</td>
</tr>
<tr>
<td>Method/Function</td>
<td>add</td>
</tr>
<tr>
<td></td>
<td>remove</td>
</tr>
<tr>
<td></td>
<td>update implementation</td>
</tr>
<tr>
<td></td>
<td>rename</td>
</tr>
<tr>
<td></td>
<td>[•••••] add (overwrite)</td>
</tr>
<tr>
<td>Field (Instance Variable)</td>
<td>add</td>
</tr>
<tr>
<td></td>
<td>remove</td>
</tr>
<tr>
<td></td>
<td>rename</td>
</tr>
</tbody>
</table>
Challenges:

1. Identify **safe point** to activate a composite change
2. Ensure safe **composition** of multiple edit transactions
3. Late-bound, scoped **state** (implementation-specific)
4. Intercepting **message dispatch** (implementation-specific)
Edit Transactions

Activation

Run-time call tree

Edit Transaction
Method atomicity
(mixed-version call stack)

Re-entrant atomicity
(single-version call stack)

Explicit boundaries
activate call,
atomic do: [ ... ]
Challenges:

1. Identify **safe point** to activate a composite change
2. Ensure safe **composition** of multiple edit transactions
3. Late-bound, scoped **state** (implementation-specific)
4. Intercepting **message dispatch** (implementation-specific)
Composition

Ball
step
move

Ball
× step

Ball
+ step
× move

Ball
× step
+ move

Ball
step

Deactivation?
Composition: Conflicting Activation

Minimize friction during composition:

Adding existing meta-objects shadows the underlying one (even if inherited)

Removing non-existing meta-objects should not have an effect (also does not block inherited ones)
Edit Transactions

Composition: Least Surprise

Prioritize most recent intent:

Removing doubly added meta-objects blocks all of them and prevents old methods or field data from becoming readable again.
Challenges:

1. Identify safe point to activate a composite change
2. Ensure safe composition of multiple edit transactions
3. Late-bound, scoped state (implementation-specific)
4. Intercepting message dispatch (implementation-specific)
Edit Transactions

**Same instance of Ball:**

```
x = nil
pos = nil
```

```
x := 10
```

```
x = nil
pos := 1@2
```

```
x = nil
pos = 1@2
```

```
x := 1
```

```
x = 10
```

```
x = nil
```
Edit Transactions

Instance Layout:

Dynamic Instance Variable Store
Concurrent Implementation

L-Value:

\[
x := \text{expr}
\]

Storage for: self at: \#x put: (expr)

Compile-time Transform

R-Value:

\[
x
\]

Storage for: self at: \#x

Storage

Process (activeProcess)

ET2: instance: \#x: 10

ET2

ET1

Base

T. Mattis, P. Rein, R. Hirschfeld | Software Architecture Group | Hasso Plattner Institute Potsdam
Challenges:

1. Identify **safe point** to activate a composite change
2. Ensure safe **composition** of multiple edit transactions
3. Late-bound, scoped **state** (implementation-specific)
4. Intercepting **message dispatch** (implementation-specific)
Class "Ball" (meta-object)

methodDict

#step ● CompiledMethod
#move ● CompiledMethod

Binary op-codes, stack frame size, ...

Recompile
"Ball >> move" when Edit Transaction "ET1" is staged

MultiVersionMethod

#ET1 ● CompiledMethod
#Base ● CompiledMethod
Class "Ball" (meta-object)

methodDict

#step ➔ CompiledMethod

#move

MultiVersionMethod

#ET1 ➔ CompiledMethod

#Base ➔ CompiledMethod

Process (activeProcess)

ET2 ➔ ET1 ➔ Base

Call Ball >> move
Case Study:

1. Tooling: Transaction-aware editor in Vivide/Smalltalk
2. Setup/Task:
   Adding features to a multi-agent simulation
3. Observed changes in programming workflow
Edit Transactions

Transactions

Unstaged

Staged

Categories

Classes

Methods

Source

step

| targetPosition |
self stepInfection.
self infection healIn: self.

targetPosition := self position + self randomPoint.

(self owner isPositionWalkable: targetPosition)
ifTrue: [self position: targetPosition.
self stepIntoAirport]
**Task:**
Agents in infected state should recover after some amount of time.

**Airport:** transports agents to other airports.

**Red:** infected agent

**Green:** healthy agent
Observation 1: Debuggers
step

| targetPosition |
| self stepInfection. |
| self infection healIn: self. |

targetPosition := self position + self randomPoint.

(self owner isPositionWalkable: targetPosition)
Debugger catching an Edit Transaction

Agent (Class)

Agent

Agent

World

Debugger

stack unwinding

step

step

step
```
<table>
<thead>
<tr>
<th>targetPosition</th>
</tr>
</thead>
<tbody>
<tr>
<td>self stepInfection.</td>
</tr>
<tr>
<td>self infection healIn: self.</td>
</tr>
<tr>
<td>targetPosition := self position + self randomPoint.</td>
</tr>
</tbody>
</table>
```

(self owner isPositionWalkable: targetPosition)
Debugger catching an Edit Transaction

Agent (Class)

Agent

Agent

Agent

World

Debugger

(Active) Edit Transaction

stack unwinding

step

step

step

T. Mattis, P. Rein, R. Hirschfeld | Software Architecture Group | Hasso Plattner Institute Potsdam
Observation 2: Lazy Initialization

**Problem:** Adding a field will leave it *uninitialized* for all instances.

**Solution:**

```
Ball >> x
  x ifNil: [x := 0].
^ x
```
Observation 3: Less Editor Windows

Normal "composite" workflow:
- Edit Method 1
- Discover dependency and switch to other window
- Edit Method 2
- Save both in correct order

With Edit Transactions:
- Stage new Edit Transaction
- Edit Method 1
- Edit Method 2
- Activate
Conclusion

Edit Transactions

- decouple adaptation from emergence in live programming
- allow semantic granularity of emergence without sacrificing live adaptation and test feedback
- trade immediacy for correct granularity of feedback
Additional Slides
Return-frame Insertion

1. mainloop | sender | pc | locals
2. step | sender | pc | locals
3. move | sender | pc | locals

[update] | sender

4. mainloop | sender | pc | locals
5. step | sender | pc | locals
6. move | sender | pc | locals

7. step_2
8. move_2
## Call and State Access Overhead

<table>
<thead>
<tr>
<th>No of Transactions</th>
<th>Results in ms (Standard Deviation)</th>
<th>Slow-down</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Without</td>
<td>Call</td>
</tr>
<tr>
<td>0</td>
<td>14 (6.26)</td>
<td>579 (5.21)</td>
</tr>
<tr>
<td>1</td>
<td>15 (0.00)</td>
<td>610 (2.74)</td>
</tr>
<tr>
<td>2</td>
<td>16 (0.42)</td>
<td>654 (1.49)</td>
</tr>
<tr>
<td>3</td>
<td>17 (0.00)</td>
<td>683 (1.23)</td>
</tr>
<tr>
<td>4</td>
<td>17 (0.53)</td>
<td>696 (1.52)</td>
</tr>
<tr>
<td>5</td>
<td>18 (0.42)</td>
<td>717 (0.85)</td>
</tr>
<tr>
<td>6</td>
<td>20 (0.00)</td>
<td>737 (1.51)</td>
</tr>
<tr>
<td>7</td>
<td>21 (0.00)</td>
<td>839 (0.82)</td>
</tr>
<tr>
<td>8</td>
<td>21 (0.00)</td>
<td>795 (1.08)</td>
</tr>
<tr>
<td>9</td>
<td>22 (0.00)</td>
<td>822 (3.20)</td>
</tr>
</tbody>
</table>