Executing Code in the Past: Efficient In-Memory Object Graph Versioning

Pluquet Frédéric
Stefan Langerman
Roel Wuyts

OOPSLA, October 2009
In-Memory Object Versioning
In-Memory Object Versioning

t0

<table>
<thead>
<tr>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
In-Memory Object Versioning

Client borrows the book
In-Memory Object Versioning

Objects

<table>
<thead>
<tr>
<th>aBook::Book</th>
<th>title</th>
<th>state</th>
<th>borrower</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;1985&quot;</td>
<td>&quot;clean&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>client1::Client</th>
<th>name</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;John&quot;</td>
<td>1</td>
</tr>
</tbody>
</table>

Client borrows the book

Client gives back the book in a dirty state
In-Memory Object Versioning

Client borrows the book

Client gives back the book in a dirty state

Client borrows again the book with the correct title
**In-Memory Object Versioning**

Objects:

- **t0**
  - aBook::Book
    - title: "1985"
    - state: "clean"
    - borrower: "John"
  - client1::Client
    - name: "John"
    - id: 1

- **t1**
  - aBook::Book
    - title: "1985"
    - state: "dirty"
    - borrower: "John"
  - client1::Client
    - name: "John"
    - id: 1

- **t2**
  - aBook::Book
    - title: "1985"
    - state: "dirty"
    - borrower: "John"
  - client1::Client
    - name: "John"
    - id: 1

- **t3**
  - aBook::Book
    - title: "1984"
    - state: "dirty"
    - borrower: "John"
  - client1::Client
    - name: "John"
    - id: 1

**Client borrows the book**

**Client gives back the book in a dirty state**

**Client borrows again the book with the correct title**

«If I want revisit these 3 versions, how can I do it?»
In-Memory Object Versioning

«If I want revisit these 3 versions, how can I do it?»

⇒ An ad hoc solution must be built
But...

- It is a recurring problem!
- Eiffel-like Checked Post-Conditions
- Stateful Tracer, Debugger, Google Wave
- More than 20 applications in the theoretical algorithmic domain (computational geometry, ...)
- ...

Frédéric Pluquet - fpluquet@ulb.ac.be
Our Solution

Introduce In-Memory Object Versioning in any Object-Oriented Language

- What are the challenges?
  - Be generally applicable!
  - Be expressive!
  - Be efficient in time and in space!
Intuitive approaches
Intuitive approaches

• Deep copy of the system after each change
Intuitive approaches

- Deep copy of the system after each change
  ➡ The memory fills up very quickly
Intuitive approaches

• Deep copy of the system after each change
  ➡ The memory fills up very quickly

• Deep copy of the system at each given interval of time
**Intuitive approaches**

- Deep copy of the system after each change
  - The memory fills up very quickly
- Deep copy of the system at each given interval of time
  - Some useful values can be not saved
Intuitive approaches

- Deep copy of the system after each change
  ➡ The memory fills up very quickly
- Deep copy of the system at each given interval of time
  ➡ Some useful values can be not saved
- Use database
Intuitive approaches

• Deep copy of the system after each change
  ➡ The memory fills up very quickly

• Deep copy of the system at each given interval of time
  ➡ Some useful values can be not saved

• Use database
  ➡ Not the same goal
What we really want
What we really want

• Only keep
What we really want

• Only keep

• interesting old values
What we really want

• Only keep
  • interesting old values
  • of interesting fields
What we really want

• Only keep
  • interesting old values
    • of interesting fields
    • of interesting objects
What we really want

• Only keep
  • interesting old values
  • of interesting fields
  • of interesting objects

Field selection
What we really want

- Only keep
  - interesting old values
  - of interesting fields
  - of interesting objects

Snapshots

Field selection
Example
Example

t0

\( \text{time} \)
Example

Objects

- aBook::Book
  - title: "1985"
  - state: "clean"
  - borrower

- client1::Client
  - name: "John"
  - id: 1

Time:

- t0
Example

Objects

\[ \text{t0} \]

\text{time}

\begin{align*}
\text{aBook::Book} & \\
\text{title} & \rightarrow "1985" \\
\text{state} & \rightarrow "clean" \\
\text{borrower} & \rightarrow \text{client1::Client} \\
\text{client1::Client} & \\
\text{name} & \rightarrow "John" \\
\text{id} & \rightarrow 1
\end{align*}
Example

Objects

- aBook::Book
  - title: "1985"
  - state: "clean"
  - borrower

- client1::Client
  - name: "John"
  - id: 1

Snapshots

- s1
  - aBook::Book
    - state: "clean"
    - borrower
  - client1::Client
example

Objects

- aBook::Book
  - title: "1985"
  - state: "clean"
  - borrower

- client1::Client
  - name: "John"
  - id: 1

Snapshots

- s1
  - aBook::Book
    - state: "clean"
    - borrower

- client1::Client
Example

t0

Objects

aBook::Book
- title: "1985"
- state: "clean"
- borrower

client1::Client
- name: "John"
- id: 1

aBook::Book
- title: "1985"
- state: "dirty"
- borrower

client1::Client
- name: "John"
- id: 1

time

t1

Snapshots

s1

aBook::Book
- state: "clean"
- borrower

client1::Client

s2

aBook::Book
- state: "dirty"
- borrower
Example

Objects

- aBook::Book
  - title: "1985", "clean"
  - state: "dirty"
  - borrower: client1::Client
    - name: "John"
    - id: 1

- aBook::Book
  - title: "1984"
  - state: "dirty"
  - borrower: client1::Client
    - name: "John"
    - id: 1

- aBook::Book
  - title: "1985"
  - state: "clean"
  - borrower: client1::Client
    - name: "John"
    - id: 1

Snapshots

- s1
  - aBook::Book
    - state: "clean"
    - borrower: client1::Client

- s2
  - aBook::Book
    - state: "dirty"
Example
Example

```
t0
Objects
aBook::Book
  title
  state
  borrower
client1::Client
  name
  id
s1
```

```
t1
Objects
aBook::Book
  title
  state
  borrower
client1::Client
  name
  id
s2
```

```
t2
Objects
aBook::Book
  title
  state
  borrower
client1::Client
  name
  id
s3
```

```
t3
```

Frédéric Pluquet - fpluquet@ulb.ac.be
Example

Objects

- aBook::Book
  - title: "1985"
  - state: "dirty"
  - borrower

- client1::Client
  - name: "John"
  - id: 1

- aBook::Book
  - title: "1984"
  - state: "dirty"
  - borrower

- client1::Client
  - name: "John"
  - id: 1

Snapshots

- s1
  - aBook::Book
    - state: "clean"
    - borrower
  - client1::Client

- s2
  - aBook::Book
    - title: "1984"
    - state: "dirty"
    - borrower
  - client1::Client

- s3
  - aBook::Book
    - state: "dirty"
    - borrower
  - client1::Client
### Example

<table>
<thead>
<tr>
<th>Time</th>
<th>aBook::Book</th>
<th>borrower state</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td></td>
<td></td>
<td>&quot;1985&quot;</td>
</tr>
<tr>
<td>t1</td>
<td>&quot;clean&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td>&quot;dirty&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td>&quot;dirty&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Client::Client</th>
<th>name</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>client1</td>
<td>John</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>aBook::Book</td>
</tr>
<tr>
<td>title</td>
</tr>
<tr>
<td>state</td>
</tr>
<tr>
<td>borrower</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Snapshots</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
</tr>
<tr>
<td>aBook::Book</td>
</tr>
<tr>
<td>state</td>
</tr>
<tr>
<td>borrower</td>
</tr>
<tr>
<td>client1::Client</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>id</td>
</tr>
</tbody>
</table>

| s2             |
| aBook::Book    |
| state          | "dirty" |
| borrower       |        |
| client1::Client|
| name           | John   |
| id             | 1      |

| s3             |
| aBook::Book    |
| state          | "dirty" |
| borrower       |        |
| client1::Client|
| name           | John   |
| id             | 1      |
How to efficiently store old values?

- **Fat Node Method** [Driscoll et al, 1986].
- Store the old values of a field in the field itself.
How to efficiently store old values?

<table>
<thead>
<tr>
<th>Insert a new value</th>
<th>O(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access last value</td>
<td>O(1)</td>
</tr>
<tr>
<td>Look up for a value</td>
<td>O(log(#values))</td>
</tr>
</tbody>
</table>
Complexities
## Complexities

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take a snapshot</td>
<td>O(1)</td>
</tr>
<tr>
<td>Perform an ephemeral update</td>
<td>O(1)</td>
</tr>
<tr>
<td>Perform a versioned update</td>
<td>O(1)</td>
</tr>
<tr>
<td>Read last value of a field</td>
<td>O(1)</td>
</tr>
<tr>
<td>Read old value of a field</td>
<td>O(log #values)</td>
</tr>
</tbody>
</table>
## Complexities

<table>
<thead>
<tr>
<th></th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take a snapshot</td>
<td>O(1)</td>
</tr>
<tr>
<td>Perform an ephemeral update</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
## Complexities

<table>
<thead>
<tr>
<th></th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take a snapshot</td>
<td>O(1)</td>
</tr>
<tr>
<td>Perform an ephemeral update</td>
<td>O(1)</td>
</tr>
<tr>
<td>Perform a versioned update</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
## Complexities

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take a snapshot</td>
<td>O(1)</td>
</tr>
<tr>
<td>Perform an ephemeral update</td>
<td>O(1)</td>
</tr>
<tr>
<td>Perform a versioned update</td>
<td>O(1)</td>
</tr>
<tr>
<td>Read last value of a field</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
# Complexities

<table>
<thead>
<tr>
<th>Action</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take a snapshot</td>
<td>O(1)</td>
</tr>
<tr>
<td>Perform an ephemeral update</td>
<td>O(1)</td>
</tr>
<tr>
<td>Perform a versioned update</td>
<td>O(1)</td>
</tr>
<tr>
<td>Read last value of a field</td>
<td>O(1)</td>
</tr>
<tr>
<td>Read old value of a field</td>
<td>O(log #values)</td>
</tr>
</tbody>
</table>
And of course

- Don’t lose time to save useless data
- Developer selects!
Efficiency in Size

aBook::Book
- title: "1984"
- state: t1
- borrower: t2

client1::Client
- name: "John"
- id: 1

"dirty"
"clean"
Efficiency in Size

- States are shared between snapshots
Efficiency in Size

- States are shared between snapshots
Efficiency in Size

- States are shared between snapshots
- The states are in the object itself
  - If an object is no longer reachable, it is garbage collected
  - Including all of its fields history
Implementation

History

• Early version in Java
• Stable version in Smalltalk
Smalltalk Integration

- A standard library
- Can be loaded in any application
- Existing code does not need modification
- Three primitives
  - `anObject selectFields`
  - `s := HSnapshot atNow`
  - `s execute: aBlock`
- Easy to learn and use
**Smalltalk Integration**

```smalltalk
aClient := Client named: 'John'.
aBook := Book titled: '1985'.
aBook state: 'clean'.
aBook borrower: aClient.

aBook state: 'dirty'.
aBook borrower: nil.
```
Smalltalk integration

```smalltalk
aClient := Client named: 'John'.
aBook := Book titled: '1985'.
aBook state: 'clean'.
aBook borrower: aClient.
aBook selectFields: [#state. #borrower].
aBook state: 'dirty'.
aBook borrower: nil.
```
Smalltalk integration

```smalltalk
aClient := Client named: 'John'.
aBook := Book titled: '1985'.
aBook state: 'clean'.
aBook borrower: aClient.
  aBook selectFields: [#state. #borrower].
s1 := HSnapshot atNow.
aBook state: 'dirty'.
aBook borrower: nil.
```
Smalltalk integration

```smalltalk
aClient := Client named: ‘John’.
aBook state: ‘clean’.
aBook borrower: aClient.
aBook selectFields: {#state. #borrower}.
s1 := HSnapshot atNow.
aBook state: ‘dirty’.
aBook borrower: nil.

aBook state.
```
Smalltalk integration

aClient := Client named: 'John'.
aBook := Book titled: '1985'.
aBook state: 'clean'.
aBook borrower: aClient.
aBook selectFields: {#state. #borrower}.
s1 := HSnapshot atNow.
aBook state: 'dirty'.
aBook borrower: nil.

aBook state. --> 'dirty'
Smalltalk Integration

aClient := Client named: 'John'.
aBook := Book titled: '1985'.
aBook state: 'clean'.
aBook borrower: aClient.
aBook selectFields: [#state. #borrower].
s1 := HSnapshot atNow.
aBook state: 'dirty'.
aBook borrower: nil.

`s1 execute: [aBook state].`
**Smalltalk Integration**

```smalltalk
| borrowing | returning |
|---------------------------------------------|
| aClient := Client named: ‘John’.           |
| aBook state: ‘clean’.                       |
| aBook borrower: aClient.                   |
| aBook selectFields: [#state. #borrower].   |
| s1 := HSnapshot atNow.                     |
| aBook state: ‘dirty’.                      |
| aBook borrower: nil.                       |

aBook state. → ‘dirty’
s1 execute: [aBook state]. → ‘clean’
```
Performance
## Performance

<table>
<thead>
<tr>
<th></th>
<th>Slowdown factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic worst case update</td>
<td>7</td>
</tr>
</tbody>
</table>
# Performance

<table>
<thead>
<tr>
<th></th>
<th>Slowdown factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic worst case update</td>
<td>7</td>
</tr>
<tr>
<td>Real cases update</td>
<td>1.3 to 2</td>
</tr>
</tbody>
</table>
Applications

• We implemented
  • Eiffel-like Checked Post-Conditions for Smalltalk
  • Execution tracer with states
  • Planar Point Location
Benchmarks

Stores messages sent in a collection

Stores messages sent in a collection
Benchmarks

Stores messages sent in a collection
Also stores states of receiver and arguments
Benchmarks

Stores messages sent in a collection
Also stores states of receiver and arguments
Benchmarks

Stores messages sent in a collection
Also stores states of receiver and arguments

Only 30% slower!
Conclusion

• A general and efficient model for in-memory object versioning for object-oriented languages
• We implemented it in Smalltalk
  • 3 primitives
  • Non-intrusive library
  • Efficient
Thank you!

- fpluquet@ulb.ac.be
- HISTORY on Google
