Object Versioning for the Lively Kernel
Preserving Access to Previous System States in an Object-oriented Programming System

Master’s Thesis
Lauritz Thamsen

Prof. Dr. Robert Hirschfeld
Bastian Steinert

June 04, 2014
Changed States

State 1

: World
extent = aPoint
  (x: 800, y: 600)
submorphs = [...
  ...
]

State 2

: World
extent = aPoint
  (x: 800, y: 600)
submorphs = [...
  aRectangle  ...
]
How to undo changes to objects?
Command Pattern

"Encapsulate a request as an object, thereby letting you [...] queue or log requests, and support undoable operations" [Gamma, et al.]

but: each action needs to be accompanied by an undo action
What about Worlds? [Warth, et al.]

A language construct to explicitly control the scope of side-effects to support experimentation

```
A = thisWorld;
p = new Point(1, 2);

B = A.sprout();

in B { p.y = 3; }

B.commit();
delete B;
```

```
in A: p = { x = 1, y = 23 }
```
CoExist: Continuous Versioning [Steinert, et al.]

<table>
<thead>
<tr>
<th>Method</th>
<th>Branch Desc</th>
<th>Test Res</th>
<th>Test Chg</th>
</tr>
</thead>
<tbody>
<tr>
<td>MarbleMania-Game</td>
<td>MaMaBoard putBall:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Test</td>
<td>MaMaBoard allMarbles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>MaMaBoard deactivatePauseMode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>MaMaBoard skipAnimations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>MaMaBoard at</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>MaMaBoard enableAllAnimations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>MaMaBoard shuffle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>MaMaBoard activatePauseMode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>MaMaBoard createAndFillMatrixWithWidthAndHeight</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>MaMaBoard matrix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>refactor instance variable, accessors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>Added MaMaBoard gravity:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>Added MaMaBoard gravity:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>refactor method, rename</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>Modified MarbleMania createBoard</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>Modified MaMaMarble/setToTargetPosition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>MaMaBoard gravity</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>MarbleMania-Game</td>
<td>MaMaBoard ThroughwidthAndHeight</td>
<td>30/2&gt;31</td>
<td>30/2&gt;31/2&gt;32</td>
</tr>
</tbody>
</table>
CoExist: Continuous Versioning [Steinert, et al.]

- recover previous development states whenever changes turn out to be inappropriate
- concentrate on implementing ideas instead of also having to protect intermediate development states
But Continuous Versioning of code is not enough for systems like the Lively Kernel.
Project Questions

Goal: fine-grained versioning of all objects in Lively

- How can we implement fine-grained object versioning?
- Is that really practical?
Object Versioning
Versions of Objects

\[ v1 \]

\| Street \| Number \| City |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Kantstr.</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

\[ v2 \]

\| Street \| Number \| City |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Kantstr.</td>
<td>148</td>
<td>Berlin</td>
</tr>
</tbody>
</table>
Preserving Previous Versions

v1

<table>
<thead>
<tr>
<th>: Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>street=Kantstr.</td>
</tr>
<tr>
<td>number=null</td>
</tr>
<tr>
<td>city=null</td>
</tr>
</tbody>
</table>

v2

<table>
<thead>
<tr>
<th>: Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>street=Kantstr.</td>
</tr>
<tr>
<td>number=null</td>
</tr>
<tr>
<td>city=null</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>street=Kantstr.</td>
</tr>
<tr>
<td>number=148</td>
</tr>
<tr>
<td>city=Berlin</td>
</tr>
</tbody>
</table>
References to Objects

\[ v1 \]

: Person
  name=Joe

: Address
  street=Kantstr.
  number=null
  city=null

\[ v2 \]

: Person
  name=Joe

: Address
  street=Kantstr.
  number=null
  city=null

: Address
  street=Kantstr.
  number=148
  city=Berlin
**Version-aware References**

![Diagram of version-aware references]

- **Version-aware Reference**
- **System Versions (with version identifier)**
- **Current Version**

```plaintext
v1
next
v2
previous

: Person
  name=Joe

: Address
  street=Kantstr.
  number=null
  city=null

: Address
  street=Kantstr.
  number=148
  city=Berlin

aPerson.address.city === 'Berlin'
```
Version-aware References (cont.)

![Diagram of version-aware references]

- **Version-aware Reference**
- **System Versions** (with version identifier)
- **Current Version**
Explicit Creation of Discrete System Versions

aPerson.address.zip = 10623
Changing Versions

Given:

Undo:

Redo:

Commit:
Generic Object Versioning provides versioning for all kinds of applications.
Scope

JavaScript / Lively Kernel runtime

Document / HTML page

Servers
Proxy-based Design & Implementation
Version-aware References (revisited)

System Version Object (with version identifier)

Current Version

: Person
  name=Joe

: Address
  street=Kantstr.
  number=null
  city=null

: Address
  street=Kantstr.
  number=148
  city=Berlin

Version-aware Reference

next

previous
Proxies as Version-aware References

proxies delegate interactions transparently to the correct versions

lively.CurrentVersion = { ID: 'v2',
    predecessor: ..., successor: ... }

: Person
  name=Joe

: Proxy
  address

: Address
  street=Kantstr.
  number=null
  city=null

v1

: Address
  street=Kantstr.
  number=148
  city=Berlin

v2

versions : Dict

: Address

aPerson.address.city === 'Berlin'
var client = {},
    server = {openSecret: "I don't like Mondays"},
    handler = {
        get: function(target, name) {
            console.log(name + ' was read at ' + Date());
            return target[name];
        }
    }

client.server = new Proxy(server, handler);

> client.server.openSecret
openSecret was read at Sun Jun 01 2014 17:58:38 GMT+0200 (CEST)
< "I don't like Mondays"
ECMAScript 6 Proxies as Version-aware References

```
// helpers
currentVersion=function()
versionForWriteAccess=function()
...

// traps
get=function(name)
set=function(name, value)
apply=function(thisArg, args)
construct=function(args)
defineProperty=function(name, desc)
deleteProperty=function(name)
freeze=function()
isFrozen=function()
...
```

```
: Address
street=Kantstr.
number=null
city=null
```

```
: Address
street=Kantstr.
number=148
city=Berlin
```
Handler: Read-only Traps

get: function(dummyTarget, name) {
    var version = this.currentVersion();
    return version[name];
}

currentVersion: function() {
    var objectVersion,
        systemVersion = lively.CurrentVersion;

    while(!objectVersion && systemVersion) {
        objectVersion = this.versions[systemVersion.ID];
        systemVersion = systemVersion.predecessor;
    }

    return objectVersion;
}
Handler: Traps That Intercept Changes

```javascript
set: function(target, name, value, receiver) {
  var newVersion;

  if (!this.versions[lively.CurrentVersion.ID]) {
    newVersion = this.copyObject(this.currentVersion());
    this.versions[lively.CurrentVersion.ID] = newVersion;
  }

  return this.currentVersion();
}

defineProperty: function(target, name, desc) {
  var version = this.versionForWriteAccess();
  version[name] = value;
  return true;
}

deleteProperty: function(target, name) {
  var version = this.versionForWriteAccess();
  delete version[name];
  return true;
}

freeze: function(target) {
  var version = this.versionForWriteAccess();
  for (var key in version) {
    delete version[key];
  }
}

seal: function(target) {
  var version = this.versionForWriteAccess();
  for (var key in version) {
    delete version[key];
  }
}

preventExtensions: function(target) {
  var version = this.versionForWriteAccess();
  for (var key in version) {
    delete version[key];
  }
}

apply: function(target, thisArg, args) {
  var version = this.versionForWriteAccess();
  for (var i = 0; i < args.length; i++) {
    version[args[i]] = args[i + 1];
  }
}
```

Use Version-aware References

return proxies for all new objects

**Literals**

{}, [], function () {}

**Specific built-in functions**

Array(), eval(), Object.create(proto)

**Constructors**

ew Morph()
Use Version-aware References: Transform Literal Objects

\{
  ..
\}

\[
  ..
\]

\[
  \text{function (..) \{..\}}
\]

\[
  \text{\ldots}
\]

\[
\rightarrow
\]

\[
\rightarrow
\]

\[
\rightarrow
\]

\[
\rightarrow
\]

\[
\rightarrow
\]
Use Version-aware References:
Transform Specific Built-in Functions

\[ \text{new Array()} \rightarrow \text{new proxyFor(Array)()} \]

...\text{Array, Boolean, Date, Function, Number, Object, RegExp, String, Math, JSON, document, Worker, XMLHttpRequest, window...}

\[ \text{eval(..)} \rightarrow \text{eval(transformSource(..))} \]
Use Version-aware References: Return Value of Constructors

```javascript
construct: function(dummyTarget, args) {
    // construct new instance
    return proxyFor(newInstance);
}
```

```javascript
Morph = proxyFor(function(..) {..})
```

```javascript
new Morph()
```
Evaluation
Functionality

DEMO /
SCREENCAST
Memory Overhead for Proxies

Size of heap snapshots after loading the Blank world

- * without Proxies: 19 MB
- * with Proxies: 65 MB

* averaged over five runs, on 5/9/2014, Chrome 34.0.1847.131, Macbook Air, 2 GHz i7, 8 GB RAM.
Execution Overhead: Octane Benchmarks

![Bar chart showing execution overhead for various benchmarks.]

*averaged over five runs, on 5/9/2014, Chrome 34.0.1847.131, Macbook Air, 2 GHz i7, 8 GB RAM.*
Execution Overhead: Lively Interactions

Overhead (in times the time)

<table>
<thead>
<tr>
<th></th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Halos</td>
<td>47</td>
</tr>
<tr>
<td>World Menu</td>
<td>40</td>
</tr>
<tr>
<td>SCB</td>
<td>42</td>
</tr>
</tbody>
</table>

* averaged over five runs, on 5/9/2014, Chrome 34.0.1847.131, Macbook Air, 2 GHz i7, 8 GB RAM.
Future Work
Improving the Performance: Wait for ECMAScript 6

- our implementation
  (versioning proxy handler)

- harmony-reflect library
  (direct proxies)

- Experimental Harmony Features
  (catch-all proxies)

- release target for ECMAScript 6: 12/2014
Improving the Performance: Alternative Implementation

use source transformations and ordinary functions

person.name -> get(person, 'name')

person.dance() -> apply(person, get(person, 'dance'))
...

```javascript
function get(standIn, propertyName) {
    var version = lively.getCurrentVersionOf(standIn);
    return version[propertyName];
}
```
Tool Support: Continuous Versioning

continuous creation of versions corresponding to **developer actions**: direct manipulation, scripting, do-its, user-triggered events
Tool Support: Version Management Tools

- helpful information to find relevant versions of objects
- fine-grained version control to recover previous states
Related Work

Recovering Previous States

CoExist
[Steinert, et al.]

Lively Offline Worlds
[Czuchra]

Back-in-Time Debugging
[Lewis, et al.]

Software Transactional Memory
[Shavit, et al.]

Scoping Changes Dynamically

Worlds
[Warth, et al.]

Changeboxes
[Denker, et al.]

Object Graph Versioning
[Pluquet, et al.]

Practical Object-Oriented Back-in-Time Debugging
[Lienhard, et al.]

Context-oriented Programming
Conclusions

• the version-aware references approach seems to be sufficient for fine-grained versioning of objects

• implementation with ECMAScript 6 proxies is not (yet) practical

• developing the implementation for Lively helped a lot

• interesting open questions and future work
References


