Object Explorer for the Lively Kernel

Meta Programming & Reflection 2009/ 2010

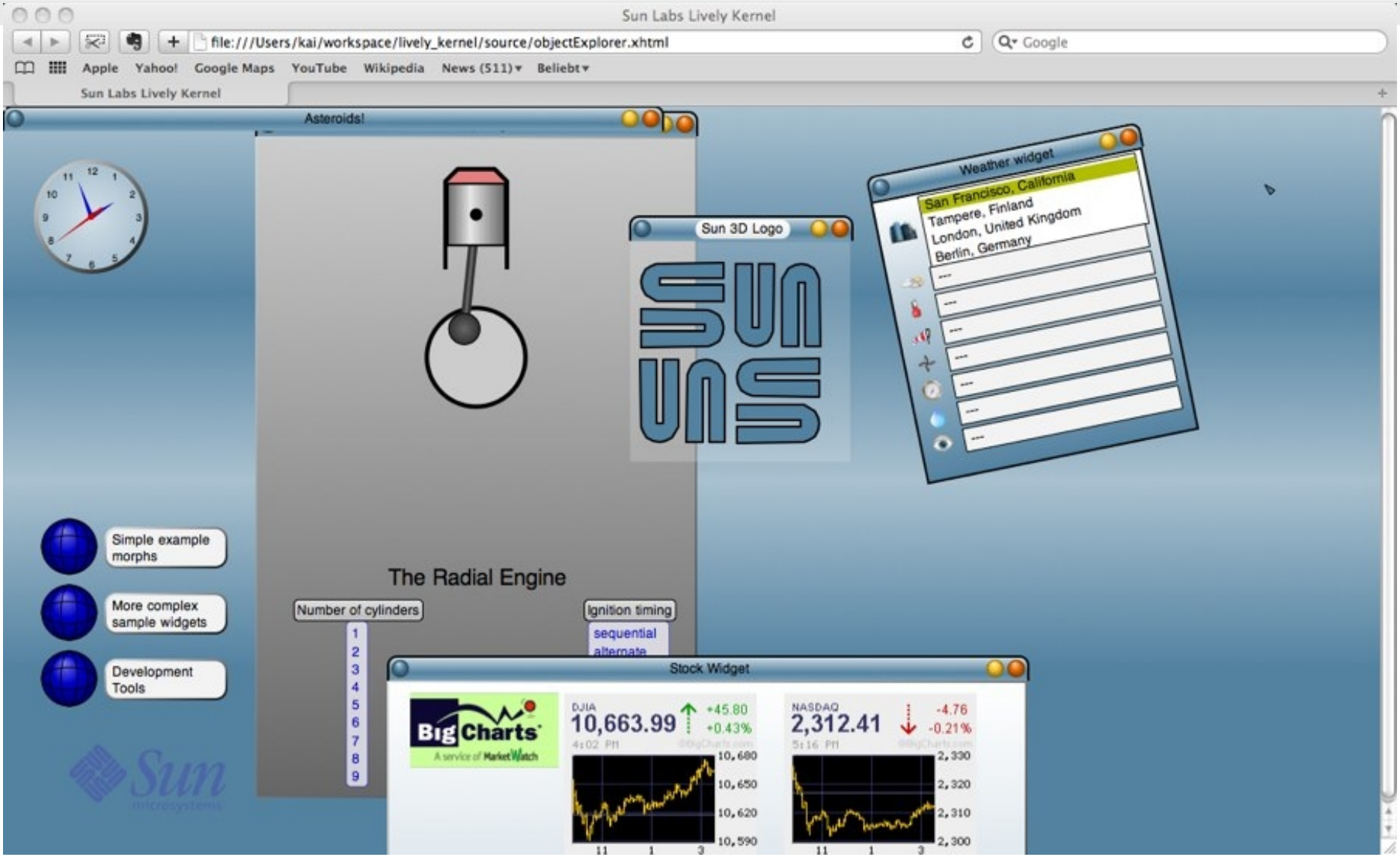
Alexander Lüders, Richard Metzler, Kai Schlichting

January 19th 2010

Agenda

- **Goal & Introduction**
- Introspection: Exploring Objects
- Demo Part I
- Intercession: Live Updates
- Demo Part II
- Summary & Outlook

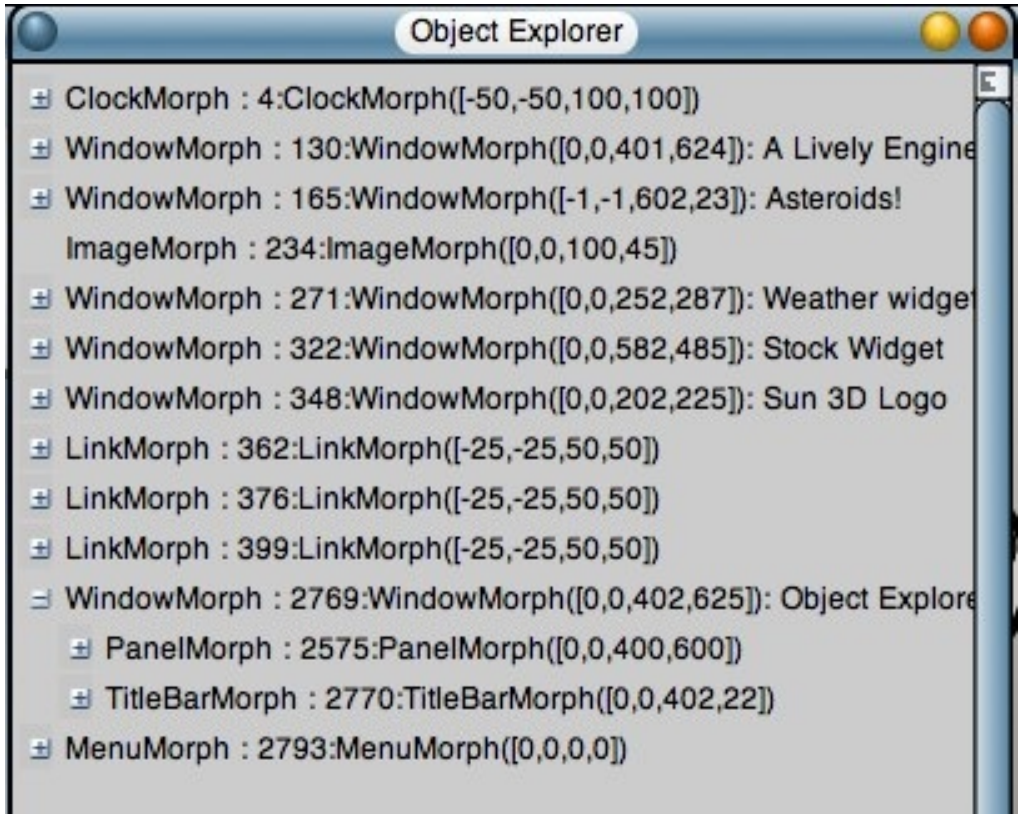
Motivation



Exploring morph relationships is tedious

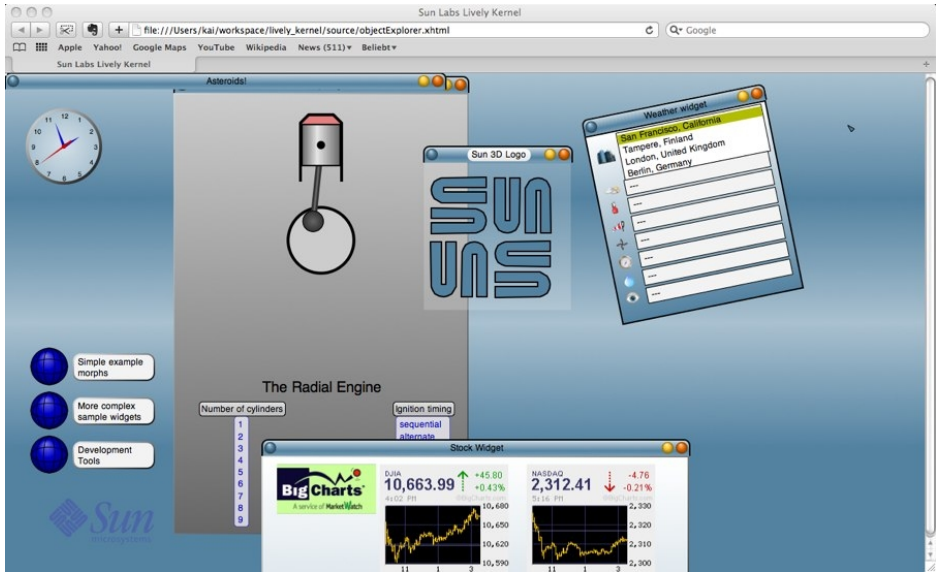
Goal

- Object explorer for morph/ submorph relationships
- It should reflect all updates to the objects immediately
- The explorer should allow to explore it using itself



Sun Labs Lively Kernel

- Suspended research project from Sun Labs
- Desktop-style applications in the browser
- Based on SVG, written in JavaScript
- "Squeak for Javascript": Lively itself as IDE for modifying and creating applications
- Supported browsers: Safari, Google Chrome(, Firefox)



Agenda

- **Goal & Introduction**
  - **JavaScript Introduction**
- Introspection: Exploring Objects
- Demo Part I
- Intercession: Live Updates
- Demo Part II
- Summary & Outlook

Objects

- Objects are associative arrays

```
var book = { title : 'Faust' };
book.author = "Goethe";
book['year'] = 1808;

book.wrote = function() {
  return this.author + " wrote " + this.title;
}

delete book.year;
keys(book); // ["title", "author", "wrote"]
```

Constructors

- Functions can act as constructors

```
function Book(author, title) {
  this.author = author;
  this.title = title;
  this.wrote = function() {
    return this.author + " wrote " + this.title;
  };
};
var hamlet = new Book("Shakespeare", "Hamlet");
hamlet.wrote(); //=> "Shakes wrote Hamlet"
hamlet.constructor === Book;
```

"Object-Oriented" Inheritance


```
// native JavaScript
function Book(author, title) {
  this.author = author; this.title = title;
  this.wrote = function(){...};
};
DrawingBook = function() {};
DrawingBook.prototype = new Book;
DrawingBook.prototype.constructor = DrawingBook;
```

vs

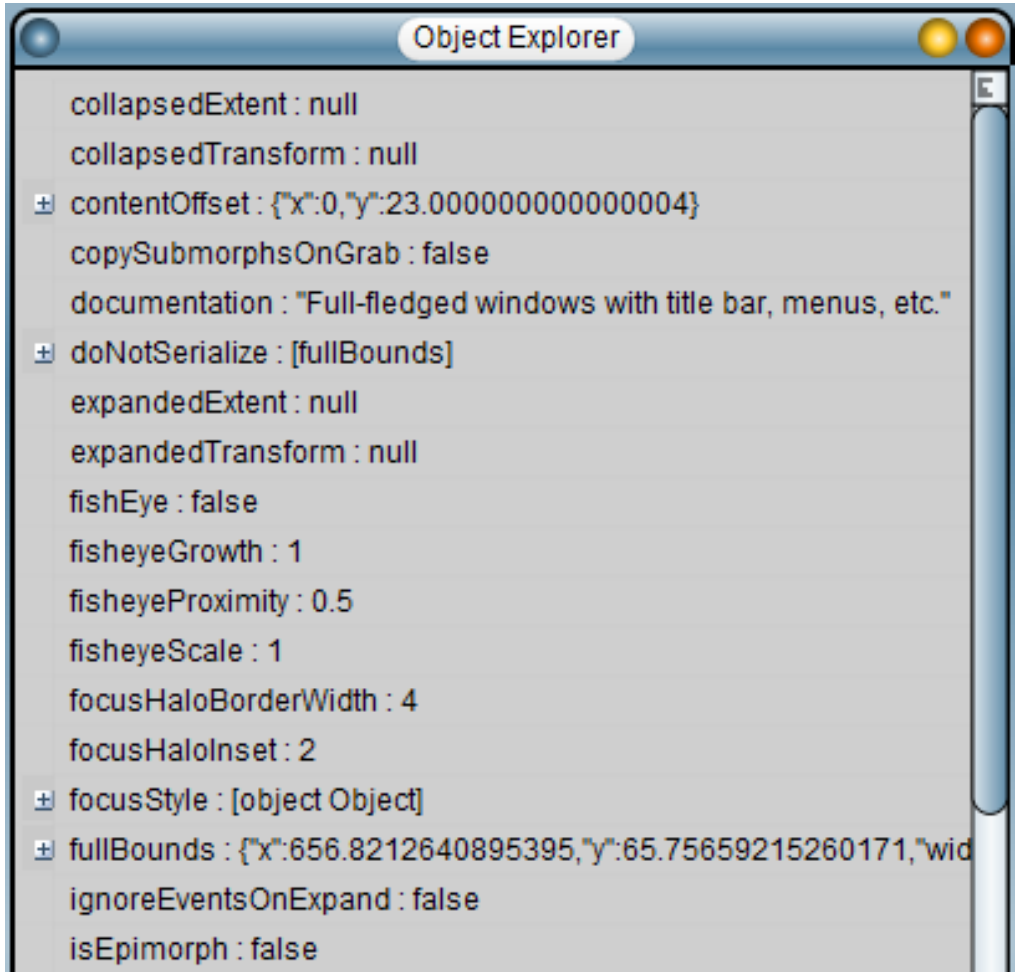
```
// Lively
Object.subclass("Book", {
  initialize: function(author, title){
    this.author = author; this.title = title;
  },
  wrote: function(){...}
});
Book.subclass("DrawingBook", {});
```

Agenda

- Goal & Introduction
- **Introspection: Exploring Objects**
- Demo Part I
- Intercession: Live Updates
- Demo Part II
- Summary & Outlook

Object Explorer

- Widget in the Lively Kernel
- Opened via context menu on graphical objects
- Displays object hierarchy lively



Browsing Object Hierarchy

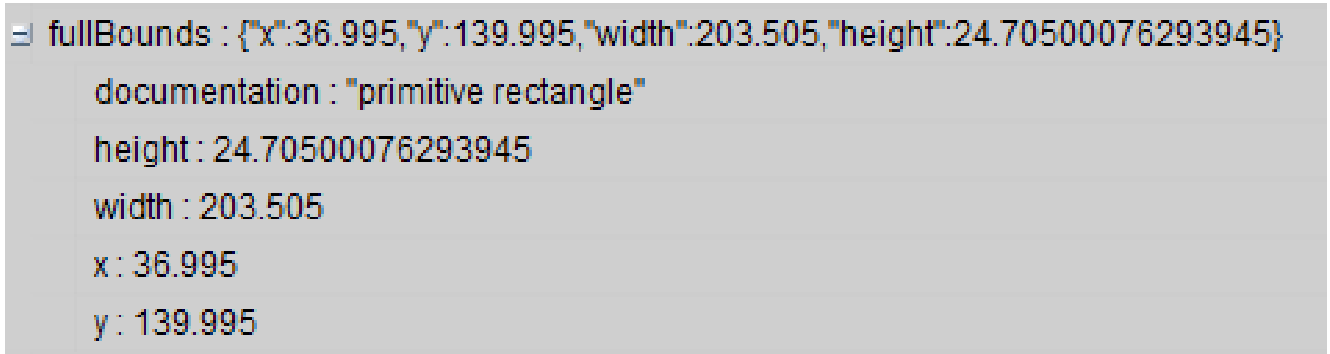
- Collect object information simply by iterating over array

```
var printObjectProperties = function(obj){
  for(var property in obj){
    if(typeof obj[property] === 'function') continue;
    alert(obj[property]);
    if(obj[property] && (typeof obj[property]=='object'))
      printObjectProperties(obj[property]);
  }
}

printObjectProperties({foo: {bar: 3}});
```

Displaying Object Information

- Values of interest for different types
  - ... a number : value (e.g. hex, octal, decimal)
  - ... a boolean: 'true' / 'false'
  - ... a string : string itself
  - ... an object: a describing string similar to Java's toString http://mckoss.com/jscript/object.htm
- `typeof` keyword determines the object's type



Displaying Object Information

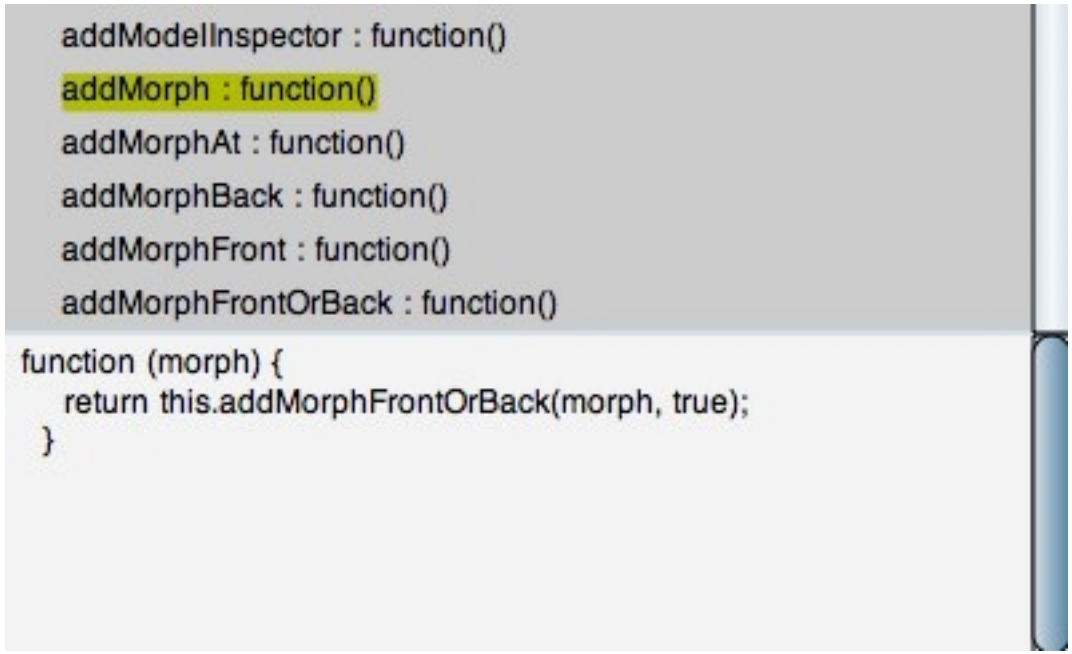

```
switch(typeof object) {
  case 'string':
    return '"' + object + '"'; // Return the string enclosed in quotes
  case 'function':
    return 'function()'
  case 'object':
    if(object == null) return "null";
      return Object.inspect(object);
  default: //e.g. case 'undefined', 'number', 'boolean'
    return ''+object;
}
```

- Works for objects which provide proper inspect or toString
- For the others build object string manually
  - Get object's type, collect property values

Manipulate Observed Object

- Console is a TextMorph
- TextMorphs in Lively Kernel support evaluation of entered text
  - Able to execute javascript code
- Setting target object to the observed object enables manipulation of the same
  - `this` keyword refers to target object
  - E.g. `this.documentation = 'Some sample text.'`

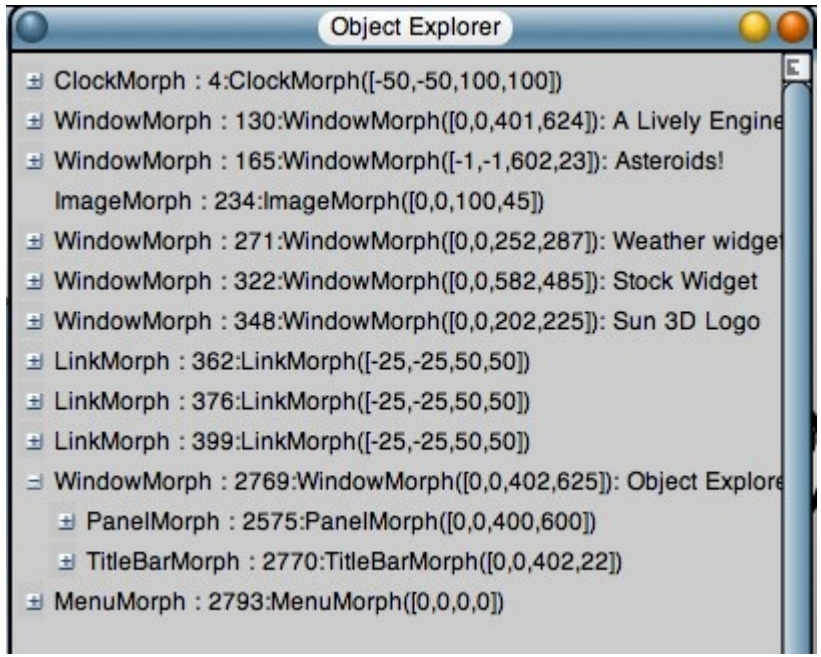
Source Code of Functions




```
var a = new Function() {alert('foo');}
a.toString() //=> 'function () {alert('foo');}'
```

Browsing Morph Hierarchy & Filters

- Morph
  - Base class for every graphical, manipulatable object in the Lively Kernel
  - May have arbitrary number of submorphs



- For exploring the graphical Lively World, the internal structure may be inexpedient
  - Object Explorer provides domain specific view

Agenda

- Goal & Introduction
- Introspection: Exploring Objects
- **Demo Part I**
- Intercession: Live Updates
- Demo Part II
- Summary & Outlook

Agenda

- Goal & Introduction
- Introspection: Exploring Objects
- Demo Part I
- **Intercession: Live Updates**
- Demo Part II
- Summary & Outlook

Installing an Observer (1/2)


```
var object = {
  foo: null
};

object.foo = 'bar'; // this should be recognized somehow
```

Installing an Observer (2/2)


```
var object = {
  foo: null
};

new ObjectObserver(object, 'foo', function(newV, oldV){
  alert('changed from ' + oldV + ' to ' + newV);
});

object.foo = 'bar'; //=> 'changed from null to bar'
```

Object.prototype.watch


```
Object.subclass('ObjectObserver', {
  initialize: function(obj, prop, fn){
    obj.watch(prop, fn);
  }
});
```

Problem: Only available in Mozilla Firefox

Polling for Changes


```
Object.subclass('ObjectObserver', {
  initialize: function(obj, prop, fn){
    var oldValue = obj[prop];
    var check = function(){
      if(oldValue !== obj[prop]){
        fn(obj[prop], oldValue);
        oldValue = obj[prop];
      }
      check.delay(1);
    };
    check();
  }
});
```

Problem: Delay in update

Getter & Setter


```
Object.subclass('ObjectObserver', {
  initialize: function(obj, prop, fn){
    var value = obj[prop];
    obj.__defineSetter__(prop, function(v) {
      fn(v, value);
      value = v;
    });
    obj.__defineGetter__(prop, function() {
      return value;
    });
  }
});
```

Problem: Overwrites existing setters

Wrapped Getter & Setter


```
Object.subclass('ObjectObserver', {
  initialize: function(obj, prop, fn){
    [...] // Define setter and getter, if not available
    var setter = obj.__lookupSetter__(prop);
    var wrappedSetter = setter.wrap(function(wrapped, val){
      fn(val, obj[prop]);
      wrapped.apply(null, val);
    });
    obj.__defineSetter__(prop, wrappedSetter);
  }
});
```

That's it! Any limits?

Intercession Limits

- Observing assignment in arrays

```
var a = [1];
a.__defineGetter__('0', function(){return 'foo'});
a[0]; // Getter does work #=> 'foo'
a.__defineSetter__('0', function(){alert("changed")});
a[0] = 1; // Setter does not work, no alert :(
```

- Observing new properties added "dynamically"

```
var a = {};
for(var prop in a){
  new ObjectObserver(a, prop, fn);
}
a.foo = 'bar'; // No observing ...
```

- Conclusion: Polling can't be avoided ...

Tidying Up: Removing Wrappers (1/2)

Hasso Plattner Institut

27

```
var object = {
  foo: null
};

var o = new ObjectObserver(object, 'foo', function(newV, oldV){
  alert('changed from ' + oldV + ' to ' + newV);
});

object.foo = 'bar'; //=> 'changed from null to bar'

o.unregister();
```

Tidying Up: Removing Wrappers (2/2)


```
Object.subclass('ObjectObserver', {
  initialize: function(obj, prop, fn){
    [...] // Define setter and getter, if not available
    var setter = obj.__lookupSetter__(prop);
    var wrappedSetter = setter.wrap(function(wrapped, val){
      fn(val, obj[prop]);
      wrapped.apply(null, val);
    });
    obj.__defineSetter__(prop, wrappedSetter);
    this.unregister = function(){
      obj.__defineSetter__(prop, setter);
    }
  }
});
```

Reflexive Browsing

- Requirement: Object explorer should be able to explore itself
- Challenge
  - Infinite recursion when expanding tree
  - If the object explorer's tree morph contains the object explorer itself
- Solution: Block exploring the tree morph, determine recursion

Agenda

- Goal & Introduction
- Introspection: Exploring Objects
- Demo Part I
- Intercession: Live Updates
- **Demo Part II**
- Summary & Outlook

Agenda

- Goal & Introduction
- Introspection: Exploring Objects
- Demo Part I
- Intercession: Live Updates
- Demo Part II
- **Summary & Outlook**

Summary Metaprogramming

- Metaprogramming is a language feature
  - *You do it all the time*
  - E.g. `foo['bar']();` vs `foo.bar();`
- Intercession limitations
  - Restricted intercession possibilities for arrays
  - Difficult to recognize a new property of an object (just polling)
- No consistent cross-browser support

Outlook

- Tidy up explored objects when closing explorer (e.g. remove wrappers)
- Explore added and removed properties
- Performance & memory analysis
  - Which impacts have the property wrappers?
  - Memory leaks

- Improve object explorer morph performance (e.g. by drawing SVG directly without creating morph objects)

Resources

- Lively Kernel project page: http://livelykernel.sunlabs.com/
- ECMAScript third + fifth edition http://www.ecmascript.org/docs.php
- *Proposed ECMAScript 3.1 Static Object Functions: Use Cases and Rationale* (Allen Wirfs-Brock, 2008)
  - http:// wiki.ecmascript.org/lib/exe/fetch.php?id=es3.1%3Aes3.1_proposal_working_draft&cache=cache&media=es3.1:rationale_for_es3_1_static_object_ methodsaug26.pdf
- http://www.slideshare.net/robnyman/javascript-from-birth-to-closure
- http://mckoss.com/jscript/object.htm
- https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Global_Objects/Object/watch

Thank you for your attention!