JavaScript 2.0: Evolving a Language for Evolving Systems

Waldemar Horwat
Netscape

Outline

- History
- Motivation
- Challenges of evolving interfaces
  - Versioning
  - Integrity
  - Convenience
- Evolving a language
- Tools

JavaScript History

- Web page scripting language invented by Brendan Eich at Netscape
- Used in 25% of web pages
- More than an order of magnitude more widely used than all other web client languages combined

JavaScript Evolution

- JavaScript 1.0 in Navigator 2.0
- JavaScript 1.5 is most recent
- JavaScript 2.0 in 2002
- Open source (NPL or LGPL)

JavaScript and ECMAScript

- JavaScript standardized by the ECMA TC39 committee
  - ECMA 262
  - ISO 16262
- JavaScript 1.5 = ECMAScript Edition 3
- JavaScript 2.0 = ECMAScript Edition 4
- TC39 leading the design of Edition 4

JavaScript 2.0 Motivation

- Evolving programs
  - Compatibility
  - Modularity
  - Safety
  - Interoperability with other languages
- Common Patterns
  - Class declarations
- Efficiency

Evolving Programs

- “Programming in the large”
- Programs written by more than one person
- Programs assembled from components (packages)
- Programs that use evolving interfaces

Evolution Problem 1

Name conflicts

Library

```
package BitTracker;

public class Data {
  public var author;
  public var contents;
  function save() {…}
};


function store(d) {
  ...
  storeOnFastDisk(d);
}
```

Library

```
package BitTracker;

public class Data {
  public var author;
  public var contents;
  function save() {…}
};


function store(d) {
  ...
  storeOnFastDisk(d);
}
```

Web page

```
import BitTracker;

class Picture extends
                    Data {
  function size() {…}
  var palette;
};


function orientation(d) {
  if (d.size().h >=
      d.size().v)
    return "Landscape";
  else
    return "Portrait";
}
```

Library rev2

```
package BitTracker;

public class Data {
  public var author;
  public var contents;
  public function size()
    {…}
  function save() {…}
};


function store(d) {
  ...
  if (d.size() > limit)
   storeOnSlowDisk(d);
  else
   storeOnFastDisk(d);
}
```

Web page

```
import BitTracker;

class Picture extends
                    Data {
  function size() {…}
  var palette;
};


function orientation(d) {
  if (d.size().h >=
      d.size().v)
    return "Landscape";
  else
    return "Portrait";
}
```

Library rev2

```
package BitTracker;

public class Data {
  public var author;
  public var contents;
  public function size()
    {…}
  function save() {…}
};


function store(d) {
  ...
  if (d.size() > limit)
   storeOnSlowDisk(d);
  else
   storeOnFastDisk(d);
}
```

Web page

```
import BitTracker;

class Picture extends
                    Data {
  function size() {…}
  var palette;
};


function orientation(d) {
  if (d.size().h >=
      d.size().v)
    return "Landscape";
  else
    return "Portrait";
}
```

Library rev2

```
package BitTracker;

public class Data {
  public var author;
  public var contents;
  public function ?()
    {…}
  function save() {…}
};


function store(d) {
  ...
  if (d.?() > limit)
   storeOnSlowDisk(d);
  else
   storeOnFastDisk(d);
}
```

Web page

```
import BitTracker;

class … extends Data {

   ?

};
```

Non-Solutions

- Assume it won't happen
  - It does, especially in DOM
- Have programmer detect/fix conflicts (C++)
  - Old web pages linked with new libraries
- Have compiler bind identifier to definition (Java)
  - Programs distributed in source form
- Explicit overrides + static type system (C#)
  - Doesn't work in dynamic languages
  - Burden on library user instead of library developer

Solution: Namespaces

- Each name is actually an ordered pair namespace::identifier
- A `use namespace(n)` statement allows unqualified access to n's identifiers within a scope
- Default namespace is `public`
- Namespaces are values

Library

```
package BitTracker;

public class Data {
  public var author;
  public var contents;
  function save() {…}
};


function store(d) {
  ...
  storeOnFastDisk(d);
}
```

Web page

```
import BitTracker;

class Picture extends
                    Data {
  function size() {…}
  var palette;
};


function orientation(d) {
  if (d.size().h >=
      d.size().v)
    return "Landscape";
  else
    return "Portrait";
}
```

Library rev2

```
package BitTracker;
namespace v2;
use namespace(v2);

public class Data {
  public var author;
  public var contents;
  v2 function size() {…}
  function save() {…}
};

function store(d) {
  ...
  if (d.size() > limit)
   storeOnSlowDisk(d);
  else
   storeOnFastDisk(d);
}
```

Web page

```
import BitTracker;

class Picture extends
                      Data {
  function size() {…}
  var palette;
};


function orientation(d) {
  if (d.size().h >=
      d.size().v)
    return "Landscape";
  else
    return "Portrait";
}
```

Library rev2

```
package BitTracker;
namespace v2;
use namespace(v2);

public class Data {
  public var author;
  public var contents;
  v2 function size() {…}
  function save() {…}
};

function store(d) {
  ...
  if (d.size() > limit)
   storeOnSlowDisk(d);
  else
   storeOnFastDisk(d);
}
```

Web page rev2

```
import BitTracker;
use namespace(v2);

class Picture extends
                      Data {
  function dims() {…}
  var palette;
};

function orientation(d) {
  if (d.dims().h >=
      d.dims().v)
    return "Landscape";
  else
    return "Portrait";
}

…  d.size() …
```

Library rev2

```
package BitTracker;
explicit namespace v2;
use namespace(v2);

public class Data {
  public var author;
  public var contents;
  v2 function size() {…}
  function save() {…}
};

function store(d) {
  ...
  if (d.size() > limit)
   storeOnSlowDisk(d);
  else
   storeOnFastDisk(d);
}
```

Web page rev2

```
import BitTracker,
         namespace(v2);

class Picture extends
                     Data {
  function dims() {…}
  var palette;
  function size() {…}
};


function orientation(d) {
  if (d.dims().h >=
      d.dims().v)
    return "Landscape";
  else
    return "Portrait";
}
```

Versioning

- Namespaces distinguish between accidental and intentional identifier matches
- Library publisher annotates new functionality with new namespaces
- Web page imports a specific namespace
- Versioning affects name visibility only — there is only one library implementation

Other Uses of Namespaces

- `private`, `internal`, etc. are syntactic sugars for anonymous namespaces
- Export private functionality to privileged clients
- Can use namespaces to add methods to an already existing class

Classes

```
class A {
  function f() // fA
}

class B extends A {
  function f() // fB
}

class C extends B {
  function f() // fC
}
```

Lookup

```
use namespace(public);

c = new C;
c.f();  // fC
```

Classes

```
class A {
  N function f() // fA
}

class B extends A {
  N function f() // fB
}

class C extends B {
  function f() // fC
}
```

Lookup 1

```
use namespace(public);

c = new C;
c.f();  // fC
```

Lookup 2

```
use namespace(public);
use namespace(N);

c = new C;
c.f();  // fB, not fC!
```

Member Lookup `c.f`

- Find highest (least derived) class that defines a member `f` that's visible in the currently used namespaces to pick a namespace *n*
- Find lowest (most derived) definition of `c.`*n*`::f`
  - Gives object-oriented semantics

Evolution Problem 2

Leakage of implementation details

Library

```
public function find(x) {
  while (x >= 0)
    if (a[x])
      return a[x];
    else
       x -= 2;
  return null;
}
```

Library

```
public function find(x) {
  while (x >= 0)
    if (a[x])
      return a[x];
    else
       x -= 2;
  return null;
}
```

Web page

```
var c = find(34);
var d = find("42");
```

Library

```
public function find(x) {
  x += 2;
  while ((x -= 2) >= 0)
    if (a[x])
      return a[x];
  return null;
}
```

Web page

```
var c = find(34);
var d = find("42");
```

Library

```
public function find(x) {
  x += 2;
  while ((x -= 2) >= 0)
    if (a[x])
      return a[x];
  return null;
}
```

Web page

```
var c = find(34);
var d = find("42");
// same as find(420)
```

Solution: Type Annotations

Library

```
public function
       find(x: Integer) {
  x += 2;
  while ((x -= 2) >= 0)
    if (a[x])
      return a[x];
  return null;
}
```

Web page

```
var c = find(34);
var d = find("42");
// now same as find(42)
```

Type Annotations

- Optional
- Language is *not* statically type-checked
- No concept of a static type of an expression; only dynamic type of an expression matters

```
class A {
  function f()
}

class B extends A {
  function g()
}
```

```
var a:A = new B;

a.f();  // OK
a.g();  // OK
```

Supportive Features

- Package syntax
- Class definition syntax
- Attributes
- Function signature checking
- Named function parameters
- Native integer and floating-point types
- Decimals and bignums
- Operator overriding and units

Class Definition Syntax

```
class Dance {
  const kind;
  var name: String;
  static var count;
  constructor function start(partner,
      length: Integer) {…}
};

class Swing extends Dance {
  var speed: Integer;
  function acrobatics(): Boolean {…}
};
```

Why Classes?

- Ubiquitous idiom
- Few users get it right with prototypes
- Difficult to evolve a program that uses prototypes
  - Little knowledge of invariants
- Classes model cross-language interaction
- Prototypes poorly supported by CLI

JS2 classes will co-exist with prototypes

Attributes

```
static debug v3 final foo(3) bar var x:Integer = 5;
```

Restricted expressions that evaluate to:

- Built-in attributes such as `final` and `static`
- Namespaces
- `true` or `false`
- User-defined attributes

Attributes

```
const debug = true;

static debug v3 final foo(3) bar var x:Integer = 5;

const A = static debug v3;

A bar var y:Integer;

private {
  var z;
  function f();
}
```

Named Function Parameters

```
function f(a: Integer, b = 5,
  named c = 17, named d = 3)


f(4)
f(4, d:6)
```

- Why is the `named` attribute needed?

Omitted Features

- Overloading
- Interfaces
- Lots of built-in libraries

Evolving the Language

An implementation will run three kinds of programs:

- JavaScript 1.5
  - Run unchanged
- Non-strict JavaScript 2.0
  - Almost all JavaScript 1.5 programs run unchanged
- Strict JavaScript 2.0
  - Turns off troublesome quirks (scope hoisting, newline dependencies, etc.)

Tools

- ECMAScript standards use semantic descriptions of the language
- Use extended typed lambda calculus with Algol-like syntax for ECMAScript Edition 4
- Wrote Common Lisp engine to:
  - Run and test grammar and semantics directly
  - Auto-generate web page descriptions
  - Auto-generate chapters of draft ECMAScript standard
- Freely available

More Information

www.mozilla.org/js/language

Waldemar Horwat

waldemar@acm.org