Prototype-based languages

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

Tomas Bures

bures@d3s.mff.cuni.cz

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

IO language

Department of
Distributed and
Dependable
Systems

IO

- Dynamic prototype-based programming language
  - All values are objects
  - No classes
  - Differential inheritance
  - Code is a runtime inspectable / modifiable tree
    - Essentially a list of messages

Department of Distributed and Dependable Systems

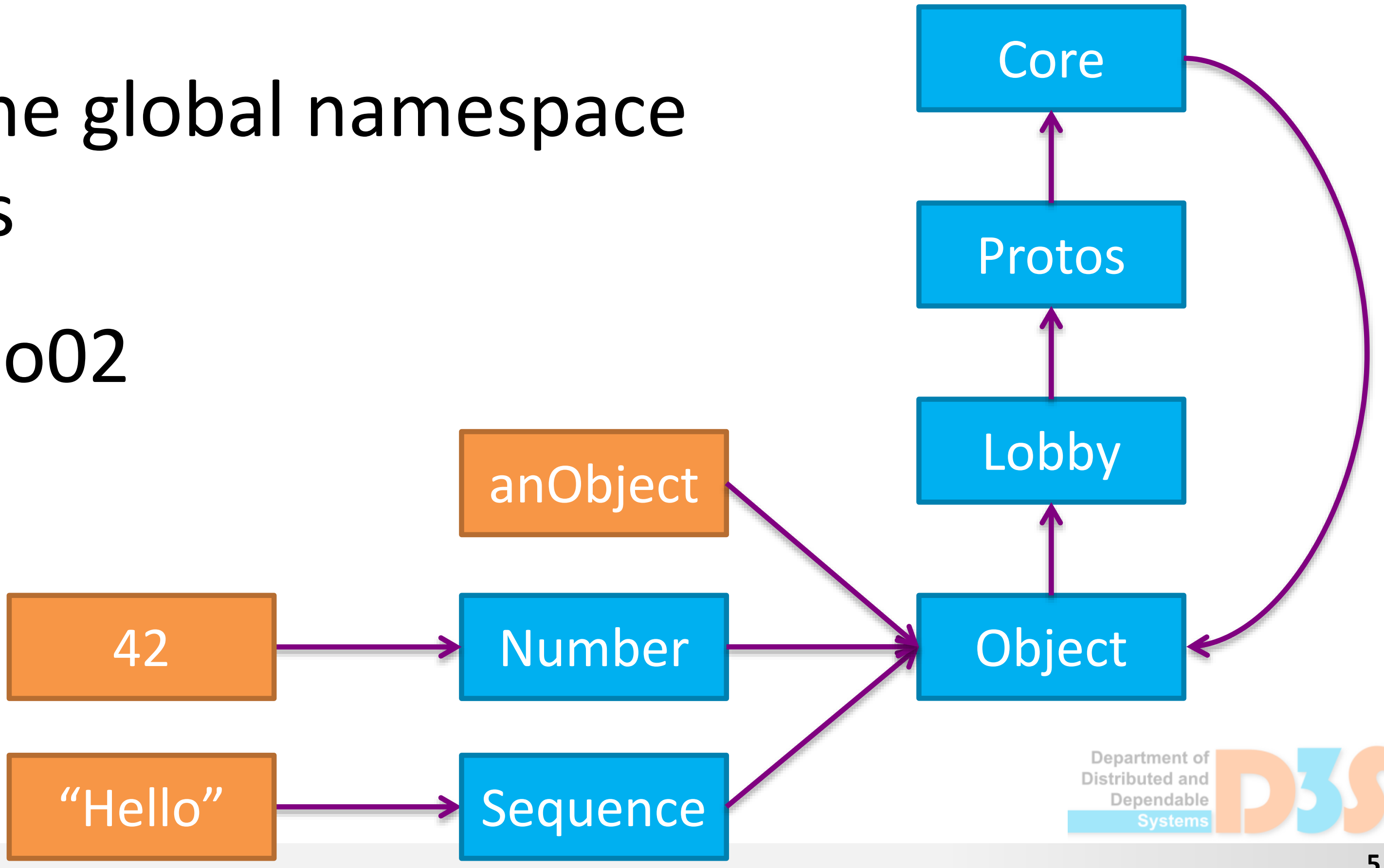
Basic concepts

- An object is a set of slots
- Object responds to messages
  - Messages handled by anonymous function stored in a slot with the name of the message
  - Properties are accessed via messages getSlot, setSlot, and updateSlot
  - :=, = are short-hand forms of updateSlot and setSlot

- Example: io01

Basic concepts

- Each object has a list of prototypes
  - Consulted in depth-first search order when a lookup in the object table fails
- Lobby is the global namespace for objects
- Example: io02



Basic concepts

- New objects created by cloning
  - Cloning creates a new object and sets the proto link to the object being cloned
- Differential inheritance
  - An object contains only attributes which are different to its prototype
- Slots can be added to any object
- Example: io03

Messages

- Code is composed of a sequence of messages
  - Each message has a name and list of arguments
  - Each argument is again a message
- Message is evaluated in a context of an object
- Example: io04

Methods / Blocks

- A block/method is a message with associated scope and parameters
- Return value is the last message in a sequence
- When invoked, activation record is created with
  - Actual parameters
  - ‘call’ object
    - ‘call target’ – target object of the call
    - ‘call sender’ – sender object
    - ‘call message’ – message used to invoke the call
  - ‘self’ – reference to the scope
  - Forward to ‘self’ for all failed lookups

- Example: io05

Methods / Blocks

- Method
  - Activatable block – called when accessed
    - Accessing without calling via getSlot(name)
  - With scope := nil – scope is set to the target object
- Block
  - Not activatable by default
  - Scope set to target of the ‘block’ message
  - Serve as local scopes within the lexical scope
- Example: io06, io07

Methods / Blocks

- Invoking a block/method means evaluating its message in a given context

- Example: io08

Control structures

- Control structures (if, while, for, ...) are ordinary methods
  - Can be implemented in the language
  - Thanks to message abstraction of the code
  - In fact ‘method’ is also an ordinary method
- IO thus has very minimal syntax and no keywords

- Example: io09

Javascript

Department of
Distributed and
Dependable
Systems

Javascript

- Prototype-based language
  - http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262%20edition%205.1,%20June%202011.pdf
  - https://developer.mozilla.org/en-US/docs/JavaScript

- Dynamically typed, first class-functions
- Used in web-browsers
- Server-side programming also possible
  - Node.js

- Example: basics.js, arrays.js, functions.js

Department of Distributed and Dependable Systems

Basics: Objects

- Object is essentially a table
- Constructed from scratch or via ‘new’ keyword and constructor function

- Example: objects.js

Basics: Prototypes

- Each Javascript object has one __proto__ slot
  - Can be accessed directly
  - Or it is automatically set by ‘new’ keyword to the value of ‘prototype’ property of the constructor function

- Examples: prototypes.js, mixins.js

Department of Distributed and Dependable Systems

Patterns: Private fields / Module

```
var Counter = (function() {
  var privateCounter = 0;
  function changeBy(val) {
    privateCounter += val;
  }
  return {
    increment: function() {
      changeBy(1);
    },
    decrement: function() {
      changeBy(-1);
    },
    value: function() {
      return privateCounter;
    }
  }
})();
```

```
alert(Counter.value()); /* Alerts 0 */
Counter.increment();
Counter.increment();
alert(Counter.value()); /* Alerts 2 */
Counter.decrement();
alert(Counter.value()); /* Alerts 1 */
```

Department of Distributed and Dependable Systems

Advanced Javascript Scripting

- NodeJS
  - Server-side Javascript interpreter
  - Webserver in Javascript

  - Asynchronous model
    - No threads
    - But asynchronous calls with a callback

Asynchronous model

```
fs.rename('/tmp/hello', '/tmp/world', function (err) {
  if (err) throw err;
  fs.stat('/tmp/world', function (err, stats) {
    if (err) throw err;
    console.log('stats: ' + JSON.stringify(stats));
  });
});
```

Immediate Two-Way Communication

- Via WebSocket
  - Abstracted by Socket.IO
- Example
  - Both client and server in Javascript
  - Server: Node.js webserver
  - Client: Javascript in HTML

Department of Distributed and Dependable Systems