CoBoxes: Distributed Heaps with Cooperative Tasks

Jan Schäfer  Arnd Poetzsch-Heffter

University of Kaiserslautern

HATS WP1 Task 1.1 Meeting
June 29th, 2009

http://www.hats-project.eu
Main Features

- Object-Oriented
- Class-Based
- Cooperative tasks
- Asynchronous method calls
- Futures
- Distributed object heaps (CoBoxes)

Main Language Mechanism

- CoBox classes
Main Features

- Object-Oriented
- Class-Based
- Cooperative tasks
- Asynchronous method calls
- Futures
- Distributed object heaps (CoBoxes)

Main Language Mechanism

- CoBox classes
Why CoBoxes?

Unit of Behavior (Runtime Component)
- Creol: Object
- JCoBox: CoBoxes (hierarchical object groups)

Advantages
- Complex mutable state (with atomic access)
- Multiple interface objects
- Hierarchical structure
  - Abstraction from internal details
  - Internal concurrency
  - Modular semantics for runtime components
- Active Objects are a special case
- Sequential OOP is a special case
Advantages by Example
Example: org.w3c.dom.Document

```java
interface Document {
    Attr createAttribute(String name);
    Element createElement(String tagName);
    ...
}

interface Node {
    Document getOwnerDocument(); ...
}

interface Element extends Node {
    Attr getAttributeNode(String name); ...
}

interface Attr extends Node {
    Element getOwnerElement(); ...
}
```

- Implement as a single active object?
Example: org.w3c.dom.Document (2)
Example: SwitchGroup

SwitchGroup - Specification

- A component which guarantees mutual exclusion for a set of switches, such that only one switch is on and all others are off
- If a switch in a switch group is turned on all other switches in that group are turned off
- Switches can only be directly turned on by the client

Problem

How to guarantee that invariant and allow direct access to switch objects?
SwitchGroup - With Active Objects - No Direct Access

- :SwitchGrp
  - :Switch
    - on:true
  - :Switch
    - on:false

:Client

Jan Schäfer
SwitchGroup - With Active Objects - No Direct Access

- **Client**: turnOn(1)

- **SwitchGrp**:
  - :Switch
    - on: true
  - :Switch
    - on: false
SwitchGroup - With Active Objects - No Direct Access

:Client

:SwitchGrp

:Switch
on:true

:Switch
on:false

Jan Schäfer
CoBoxes
WP1 Task 1.1 Meeting
SwitchGroup - With Active Objects - No Direct Access

Client

:Client

:SwitchGrp

:Switch
on:false

turnOn(1)

turnOff()

:Switch
on:false
SwitchGroup - With Active Objects - No Direct Access

Client

:Client

:Switch Grp

:Switch
on:false

:Switch
on:false

turnOn(1)
turnOn()
SwitchGroup - With Active Objects - No Direct Access

:Client

:SwitchGrp

:Switch
on:false

:Switch
on:true

turnOn(1)

turnOn()
SwitchGroup - With Active Objects - No Direct Access

Jan Schäfer
CoBoxes
WP1 Task 1.1 Meeting

SwitchGroup - With Active Objects - No Direct Access

:Client

:SwitchGrp

:Switch
  on:false

:Switch
  on:true

turnOn(1)
SwitchGroup - With Active Objects - Direct Access

- :Client
- :SwitchGrp
  - :Switch
    - on: true
  - :Switch
    - on: false
SwitchGroup - With Active Objects - Direct Access

:Client

:SwitchGrp

:Switch
  on:true

:Switch
  on:false

```
Jan Schäfer
CoBoxes
WP1 Task 1.1 Meeting
```
SwitchGroup - With Active Objects - Direct Access

- :SwitchGrp
  - :Switch
    - on:true
  - :Switch
    - on:false

- :Client
  - turnOn()
  - onReq(this)

Jan Schäfer
CoBoxes
WP1 Task 1.1 Meeting
SwitchGroup - With Active Objects - Direct Access

- SwitchGrp
  - Switch: on:true
  - Switch: on:false
- Client
  - turnOn()
  - onReq(this)
  - turnOff()
  - turnOn()
SwitchGroup - With Active Objects - Direct Access

:Client

:SwitchGrp

:Switch
  on:false

:Switch
  on:false

Jan Schäfer

CoBoxes

WP1 Task 1.1 Meeting
SwitchGroup - With Active Objects - Direct Access

**Client**

**SwitchGrp**

**Switch**
- on: false

**Switch**
- on: false

Jan Schäfer
CoBoxes
WP1 Task 1.1 Meeting
SwitchGroup - With Active Objects - Direct Access

:Client

:SwitchGrp

:Switch
on:false

:Switch
on:true

Jan Schäfer

CoBoxes

WP1 Task 1.1 Meeting
SwitchGroup - With CoBoxes

:Client

:SwitchGrp

:Switch
on:true

:Switch
on:false

Jan Schäfer
CoBoxes
WP1 Task 1.1 Meeting
SwitchGroup - With CoBoxes

:Client

:SwitchGrp

:Switch
on:true

:Switch
on:false

Jan Schäfer
CoBoxes
WP1 Task 1.1 Meeting
SwitchGroup - With CoBoxes

:Client

:SwitchGrp

:Switch
  on: false

:Switch
  on: false

Jan Schäfer
CoBoxes
WP1 Task 1.1 Meeting 8 / 23
SwitchGroup - With CoBoxes

:Client

:SwitchGrp

:Switch
on:false

:Switch
on:true

Jan Schäfer
CoBoxes
WP1 Task 1.1 Meeting
SwitchGroup - With CoBoxes

:Client

:SwitchGrp

:Switch
on:false

:Switch
on:true
SwitchGroup - With a CoBox Facade

SwitchGrp:
- Switch on:true
- Switch on:false

SwitchProxy:
- :Switch on:true
- :Switch on:false

Client

WP1 Task 1.1 Meeting 9 / 23
SwitchGroup - Nested CoBoxes

:Client

:SwitchGrp

:SwitchProxy

:Switch
on:true

:SwitchProxy

:Switch
on:false
Summary of Advantages
Complex Mutable State

Every non-trivial object has a logical state represented by further objects.

**In Creol**
- represent the whole state as ADT
  - only functional programming inside an object
- use other active objects to hold state
  - more difficult because of distributed state
- sequential programming cannot be completely simulated
  - always use: `x!m().get` ⇒ problems with reentrancy
  - always use: `x!m().await` ⇒ problems with concurrent accesses

**In JCoBox**
- Sequential OOP within a single CoBox
- ADTs still useful!
Sequential OOP within a CoBox

- Well-known programming technique
- Easy for programmers
- Existing tools/knowledge
  - Analysis
  - Verification
  - ...
- Encapsulation of existing legacy code
  - Allows for a smooth migration path
Multiple Interface Objects

- Multiple objects as entry points to an object structure
- Very important for OOP
- Allows for scalable OO interfaces
- Examples
  - List: Iterator
  - DOM
  - File System
  - GUI-Frameworks
Hierarchical Structure

- Treat groups of CoBoxes semantically as a single unit of behavior
- Allows for a modular semantics of CoBoxes
- Does *not* influence the behavior of individual CoBoxes
  - Like Boxes for sequential OOP
  - Different to published paper version
Practical Experience
Additional Features

- Promises (explicitly resolvable futures)
- Immutable classes
- Transfer classes
- Java 5: Generics, annotations, ...
- Plain Java classes
- Static fields and methods
- Exceptions
- Configurable Task Scheduler (very limited)
- Configurable message ordering guarantees
- Distribution with RMI
Implemented Examples

- **FourWins**
  - Game with a Swing-GUI
  - Computer player utilizing multiple cores

- **CoCoME**
  - Distributed cash desk system
  - Implemented as a product line

- **Classical Examples**
  - Philosophers
  - Dating Service
  - ...
Experiences

- The CoBox model has proven to be useful in practice
- Deadlocks rarely occur and are easy to find
- High-Level data-races rarely occur and are easy to find
- Order guarantee very useful
- Immutable and Transfer objects are needed
  - ADTs could be used instead
- Design decisions usually clear
  - Which class should be plain, cobox, transfer, immutable?
Maude Specification
Goal: Executable paper rules
⇒ not optimized for performance
Can be used to model-check simple programs
  • Found deadlock in 3-Philosopher example
Currently very different to Maude specification of Creol
  • Integration of CoBoxes would be possible, however
Questions concerning ABS
Question Concerning Concurrency Model

- Futures yes, but Promises? Channels?
- Only ADTs, or also Transfer Objects?
- Message ordering guarantees?
- Configurable Scheduling?
  - specified by join patterns?
General Questions for the ABS (from our perspective)

- Process-independent data description
  - How to represent object references?

- Modularity (Notion of a component)
  - Clear behavioral interface description
    - declarative specification language
  - Notion of composition

- Notion of Refinement
  - Relation to implementation
  - Relation between different models (fully abstract?)

- Support for features/deltas on the modeling level

- Well-defined notion concurrency with a pluggable scheduling concept