Writing Concurrent Desktop Applications in an Actor-Based Programming Model

Jan Schäfer  Arnd Poetzsch-Heffter
jschaefer@cs.uni-kl.de

University of Kaiserslautern
Department of Computer Science
Software Technology Group

IWMSE10
1th May 2010
Talk Outline

1. OO-Programming Models
2. The CoBox Model
3. JCoBox
4. Example Application
5. Conclusions
OO-Programming Models
An imperative program manipulates its world (e.g. memory) directly. It is founded on a now-unsustainable single-threaded premise - that the world is stopped while you look at or change it. You say “do this” and it happens, “change that” and it changes. Imperative programming languages are oriented around saying do this/do that, and changing memory locations.

Rich Hickey
Sequential OOP

- The world is an object-heap
- Objects exchange messages?
- Objects communicate by **synchronous method calls**
  - Tightly couples caller to callee
  - Caller has to wait until callee terminates
Multi-Threaded OOP

- Multiple threads run **concurrently in one, shared world**
- Sequential programming invariants **lost**
  - Unsafe programming is default
- Locks and monitors to regain invariants
- Error-prone, difficult to test and debug
- Not very modular
- Synchronous communication now become a large problem
  - “Nested monitor problem”
  - Deadlocks
Active Objects

- Multiple worlds with single objects
- Active objects run concurrently
- **Standard sequential programming** inside an object
- Communication by **asynchronous method calls**
The CoBox Model
The CoBox Model

- Multiple worlds with multiple objects
- CoBoxes run concurrently
- **Sequential OOP** inside a cobox
- Communication by *asynchronous method calls*
Cooperative Multitasking

- Multiple Task may run in a single CoBox
- Scheduling is cooperative
- Sequential invariants are still valid
JCoBox
**JCoBox**

- Extends Java 6
- CoBox classes: `@CoBox`
- Asynchronous method calls: `x ! m()`
- Futures: `Fut<T>`
- Cooperative Scheduling: `JCoBox.yield()`

**Implementation**

- Polyglot extension
- Generates Java code
- Runtime library
Example Application
Example: Concurrent Music Manager

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Interpret</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hanging By A Thread</td>
<td>Letter Black</td>
<td>buy ($0.99)</td>
</tr>
<tr>
<td>2</td>
<td>Everybody Hurts</td>
<td>R.E.M</td>
<td>buy ($0.99)</td>
</tr>
<tr>
<td>3</td>
<td>The Future</td>
<td>Prince</td>
<td>bought</td>
</tr>
<tr>
<td>4</td>
<td>Parallel</td>
<td>Bad Religion</td>
<td>buy ($0.99)</td>
</tr>
<tr>
<td>5</td>
<td>Wind of Change</td>
<td>The Scorpions</td>
<td>buy ($0.99)</td>
</tr>
<tr>
<td>6</td>
<td>There is Hope</td>
<td>Alabaster Box</td>
<td>bought</td>
</tr>
<tr>
<td>7</td>
<td>Stacked Actors</td>
<td>Foo Fighters</td>
<td>downloading (42%)</td>
</tr>
<tr>
<td>8</td>
<td>The Promise</td>
<td>Tracy Chapman</td>
<td>buy ($0.99)</td>
</tr>
<tr>
<td>9</td>
<td>A Box Full Of Sharp Objects</td>
<td>The Used</td>
<td>downloading (26%)</td>
</tr>
<tr>
<td>10</td>
<td>Cooperate</td>
<td>John Reuben</td>
<td>buy ($0.99)</td>
</tr>
<tr>
<td>11</td>
<td>The Logical Song</td>
<td>Supertramp</td>
<td>buy ($0.99)</td>
</tr>
</tbody>
</table>
CoMusic: CoBox Design

- **GUI** interacts with:
  - **Song Model**
    - get song info
    - notify about changes
  - **Download Mngr**
    - update download status
    - control downloads
  - **Download Process**
    - status updates

- **Song Model** sends:
  - player events

- **Song Player** receives:
  - player events

- **Download Mngr** receives:
  - download events
class Main { }

@CoBox class SongModel { ... }

@CoBox @Swing class GUICtrl { ... }

@CoBox class DLCtrl { ...
    @CoBox class DlProcess { ... }
}

@Immutable class Song { ... }

@Transfer class SongStatus { ... }
class Main {
    public static void main(String[] args) {
        JCoBox.setAutomaticShutdown(false);

        GUICtrl gui = new GUICtrl();
        gui.init().await();

        DLCtrl dl = new DLCtrl(model);
        gui.registerBuyListener(dl);

        SongModel model = new SongModel();
        model.registerListener(gui);
    }
}
@Immutable class Song {
    int ID;
    String name;
    String interpret;
    Song(int id, String n, String i) {
        ID = id; name = n; interpret = i;
    }
}

@Transfer class SongStatus {
    int progress;
    Song song;
    SongStatus(Song s) {
        song = s;
    }
    ... // setters and getters
}
@CoBox class SongModel {
    private Map<Integer, SongStatus> songs = ...;
    private List<SongModelListener> listeners = ...;
    ...
    public void updProgress(Song s, int p) {
        SongStatus stat = songs.get(s.ID);
        stat.setProgress(p);
        for (SongModelListener lis : listeners) {
            lis.statusChanged(stat);
        }
    }
}
@PlainJava class SongTable extends AbstractTableModel {
    ...
}

@CoBox @Swing class GUICtrl implements SongModelListener {
    private DLCtrl dlctrl;
    private SongTable table = new SongTable();
    private JButton buyBtn;

    public void init() {
        ...
        buyBtn = new JButton("Buy");
        buyBtn.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent a) {
                Song s = table.getSelectedSong()
                dlctrl!buy(s); }});
        ...
    }

    public void statusChanged(SongStatus s) {
        table.updateStatus(s);
    }

    ...
}
@CoBox class DLProcess {
    DLCtrl ctrl;
    boolean canceled;
    DLProcess(DLCtrl c) {
        ctrl = c;
    }
    void cancel() {
        canceled = true;
    }
    void start(Song s) {
        while (!canceled) {
            ... // download next piece
            JCoBox.yield(); // process cancel calls
        }
        ctrl!finished(s);
    }
    ... // further code omitted
}
CoMusic: Behavior

- **BuyButton**
  - `actionPerformed(a)`
  - `getSelectedSong()`
  - `song`

- **SongTable**
  - `song`

- **GUICtrl**

- **DLCtrl**
  - `buy(song)`
  - `<create>`
  - `DLProcess`
  - `start(song)`
  - `updProgress(song,p)`
  - `statusChanged(s1)`
  - `statusChanged(s2)`
  - `finished(song)`

- **SongModel**

- `updateStatus(s1)`
- `updateStatus(s2)`
Conclusion
Problem

Threads

- Unsafe by default
- Sequential invariants become invalid
- Synchronous method calls lead to strong coupling
## Solution

### CoBox Model

- Concurrent, isolated object-groups
- No data-races
- Communication by asynchronous method calls
- Keeps standard sequential OOP
- Has cooperative multitasking

### JCoBox

- Minimal Java extension
- Working compiler and runtime system
- Several GUI-applications written
Compiler, Examples, etc.

http://softech.cs.uni-kl.de/~jcobox

Paper

Jan Schäfer and Arnd Poetzsch-Heffter
JCoBox: Generalizing Active Objects to Concurrent Components
ECOOP 2010, June, 2010, to appear
Thanks!

Jan Schäfer, jschaefer@cs.uni-kl.de