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

ACM/IEEE 32nd International Conf. on Software Engineering
2 – 8 MAY 2010 CAPE TOWN, SOUTH AFRICA

Talk Outline

1. OO-Programming Models
2. The CoBox Model
3. JCoBox
4. Example Application
5. Conclusions

OO-Programming Models

Sequential Imperative Programming

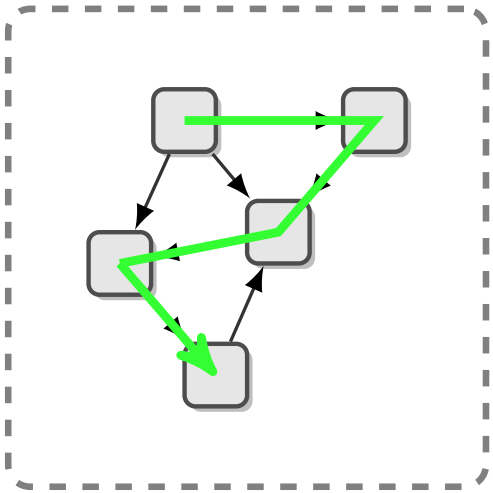
An imperative program manipulates its world (e.g. memory) directly. It is founded on a now-unsustainable single-threaded premise - that the world is stopped while you look at or change it. You say "do this" and it happens, "change that" and it changes. Imperative programming languages are oriented around saying do this/do that, and changing memory locations.

Rich Hickey

Sequential OOP

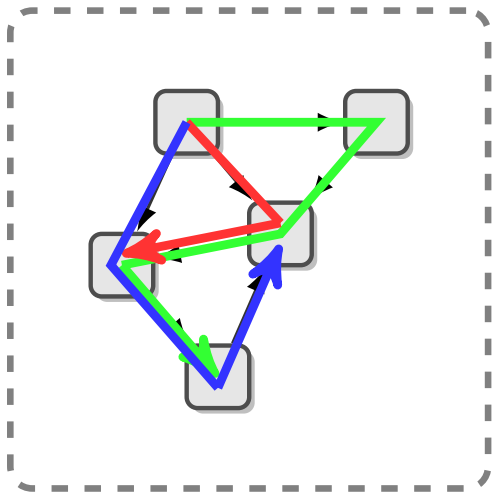
Sequential OOP

- 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

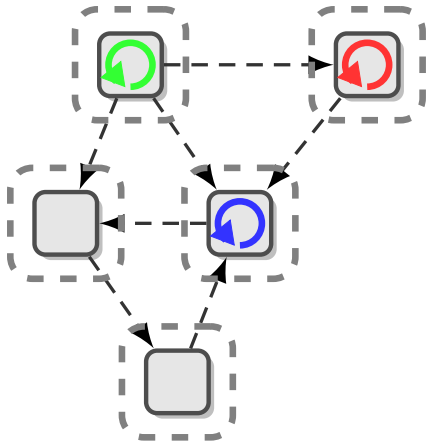
- Multiple threads run concurrently in one, shared world
- Sequential programming invariants lost
  - Unsafe programming is default
- Locks and monitors to regain invariants
- Error-prone, difficult to test and debug
- Not very modular
- Synchronous communication now become a large problem
  - “Nested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



Actor-Based Programming (Active Objects)

Active Objects

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

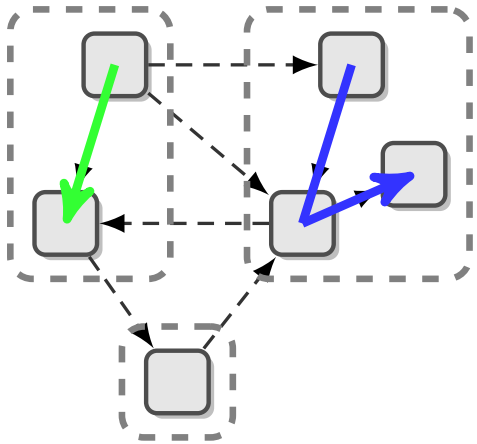


The CoBox Model

The CoBox Model

The CoBox Model

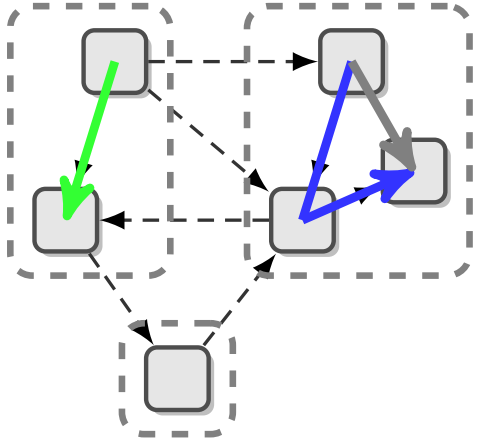
- Multiple worlds with multiple objects
- CoBoxes run concurrently
- Sequential OOP inside a cobox
- Communication by asynchronous method calls



Cooperative Multitasking

Cooperative Multitasking

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




JCoBox

JCoBox

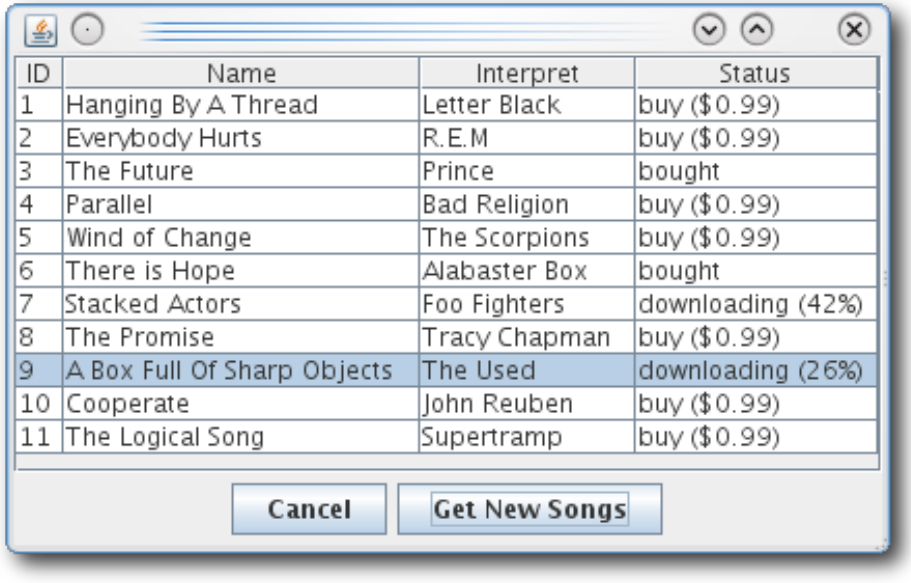
- Extends Java 6
- CoBox classes: `@CoBox`
- Asynchronous method calls: `x!m()`
- Futures: `Fut<T>`
- Cooperative Scheduling: `JCoBox.yield()`

Implementation

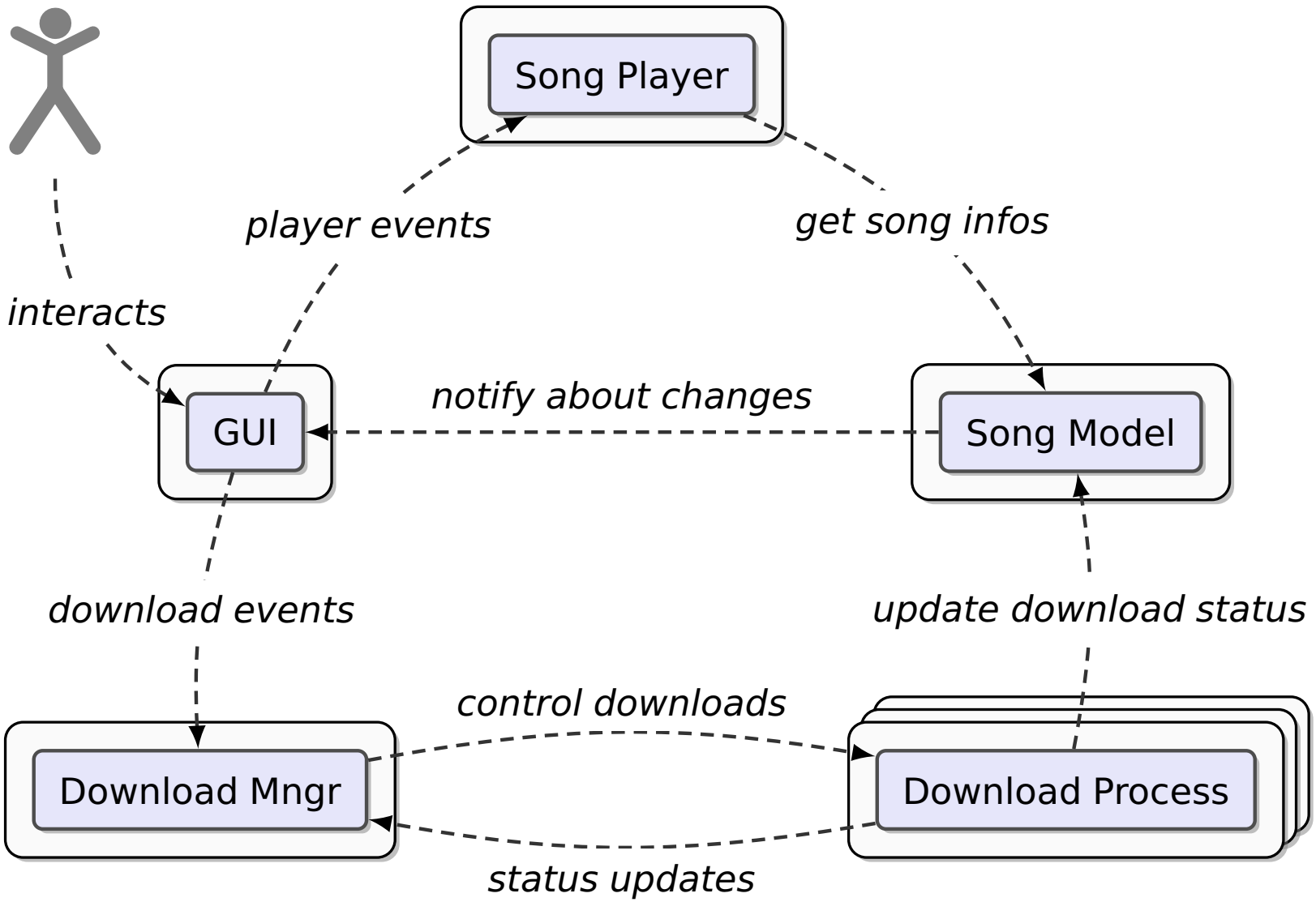
- Polyglot extension
- Generates Java code
- Runtime library

Example Application

Example: Concurrent Music Manager



CoMusic: CoBox Design



CoMusic: Implementation

```
class Main { }

@CoBox class SongModel { ... }

@CoBox @Swing class GUICtrl { ... }

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

@Immutable class Song { ... }

@Transfer class SongStatus { ... }
```

The Main Class

```
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);
  }
}
```

Data Classes

```
@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
}
```

The SongModel Class

```
@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);
    }
  }
}
```

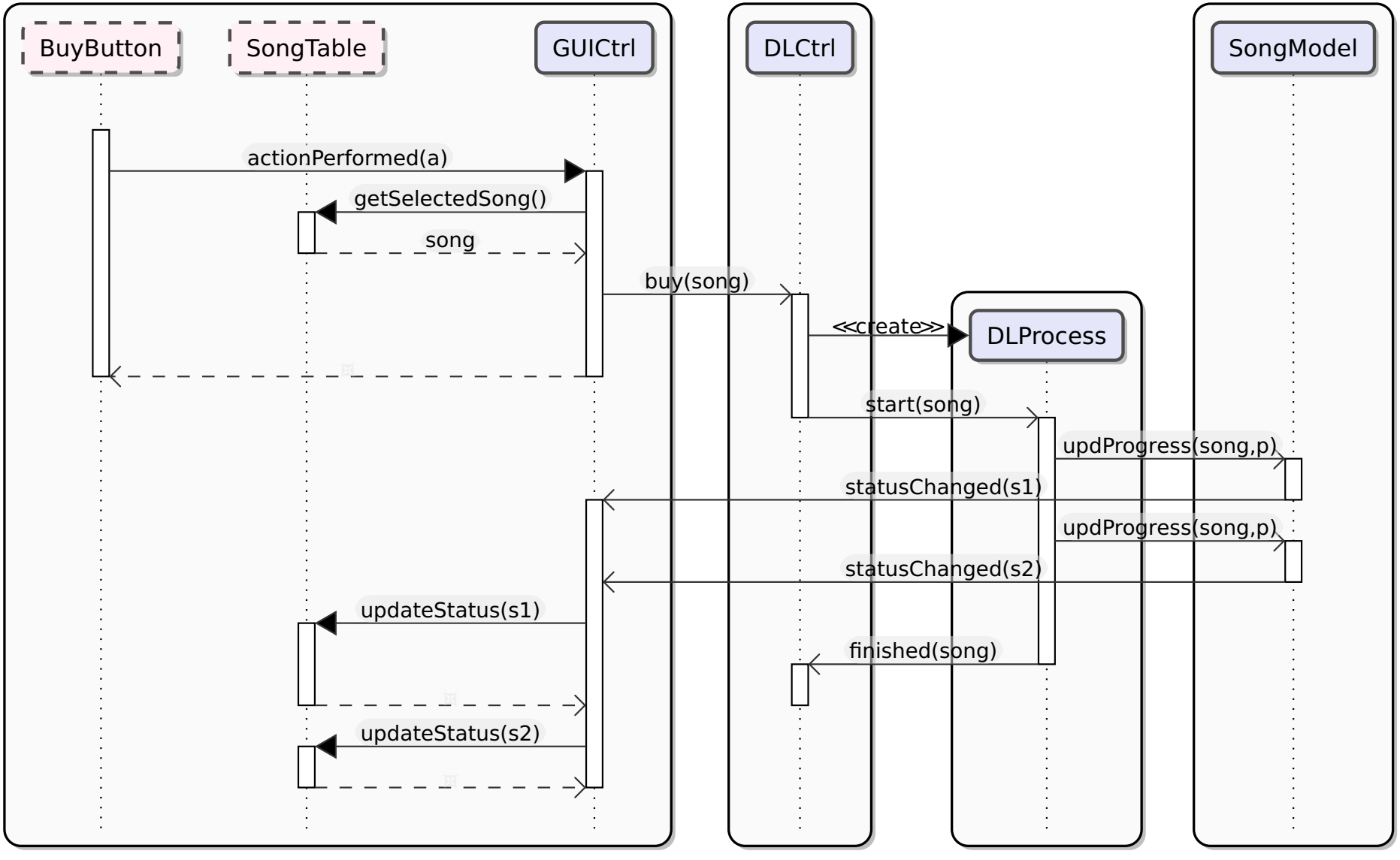
GUI classes

```
@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);
   }
   ...
}
```

The DLProcess class

```
@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



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

Further Information

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