JCoBox: Java + Asynchronously Communicating Object Groups with Cooperative Tasks

Jan Schäfer Arnd Poetzsch-Heffter

June 26, 2012

Challenge

Model for concurrent and distributed OO-programming that

- is simpler for programmers
- is a better basis for reasoning
- scales from local concurrency to distributed programming
- integrates well with classical sequential OOP, in particular:
  - Synchronous methods calls and asynchronous messages
  - Smooth language integration

Idea

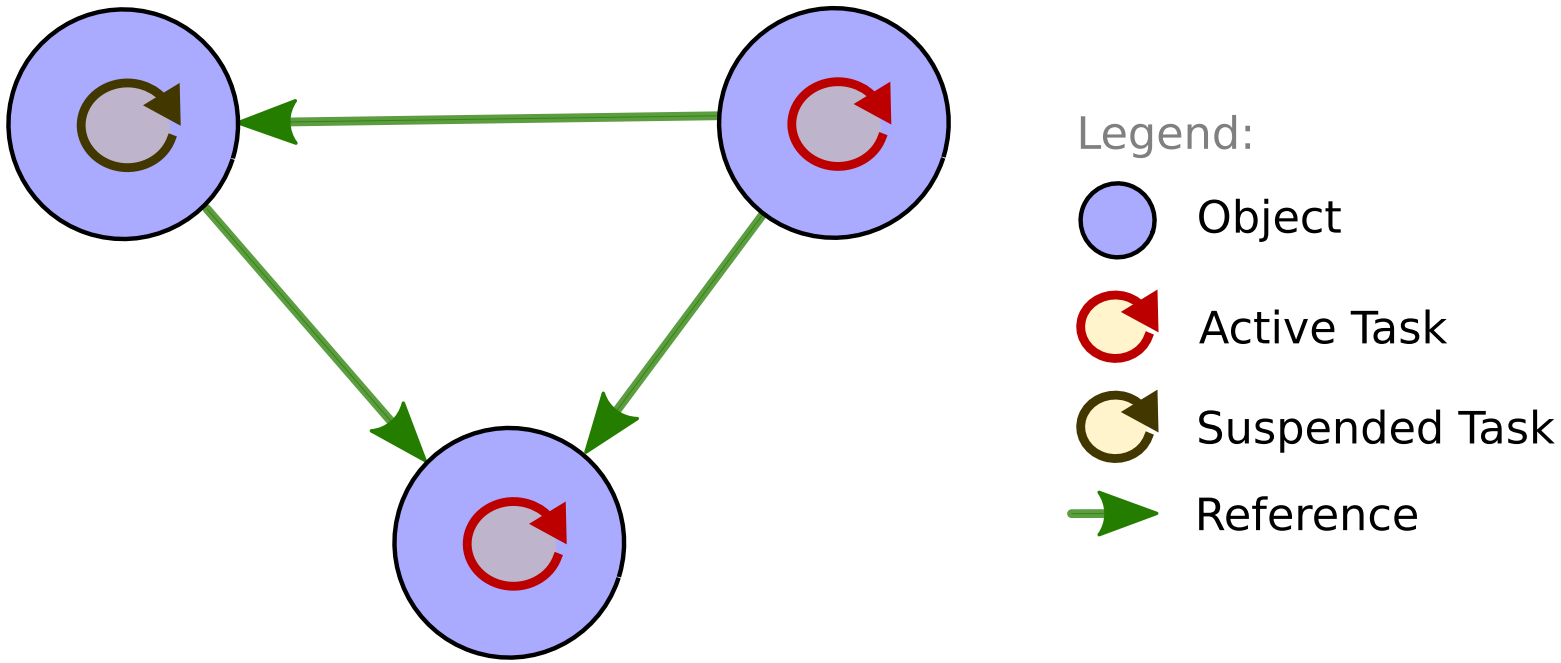
CoBoxes = Active Objects + Groups of objects

Related Work

- CreoI ($\sim$ CoBoxes of size 1)
- ASP (active object heaps with a single access object)
- E-Programming Language (VATs $\sim$ CoBoxes)
- AmbientTalk ($\sim$ ASP $+$ channels)
- Sing# (processes + channels with contracts)

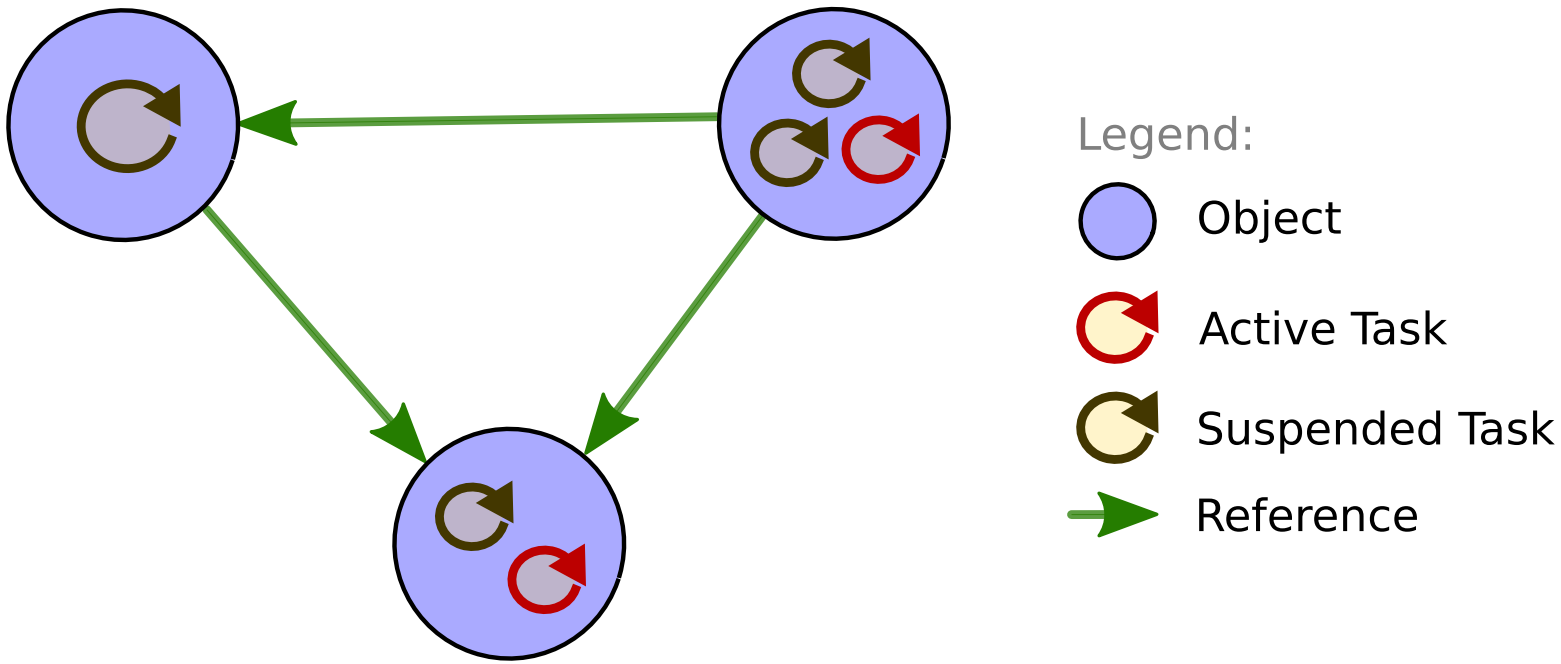
Outline

Actors[1] / Active Objects[2]



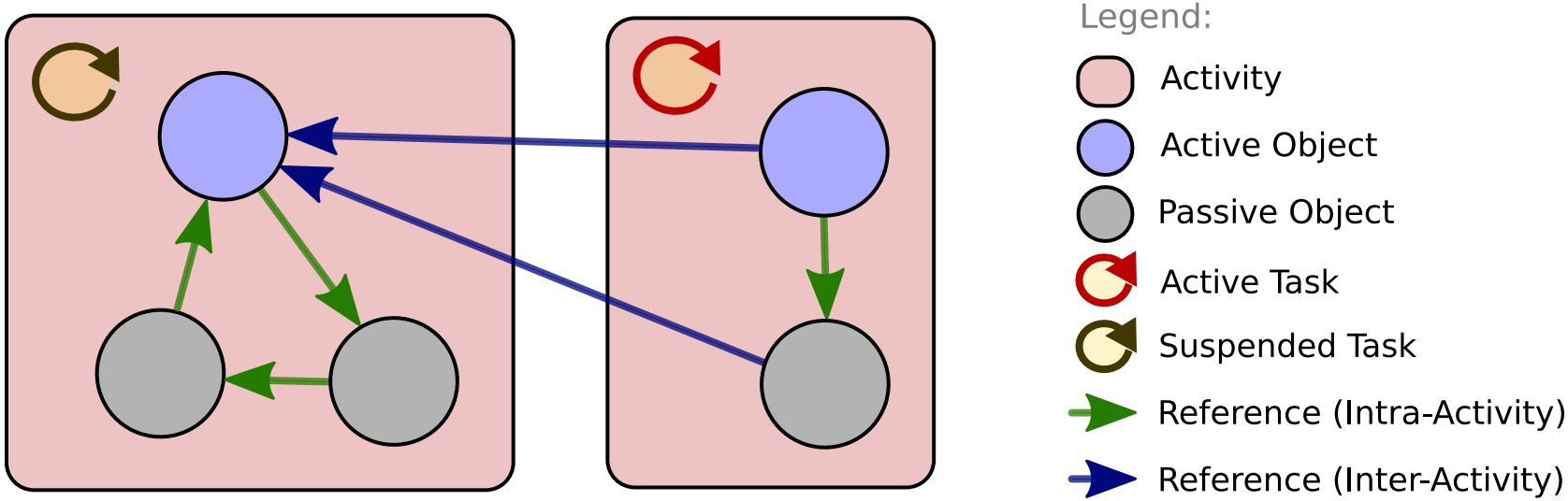
[1] Gul Agha, 1986
[2] Denis Caromel, 1993

Creol[3]



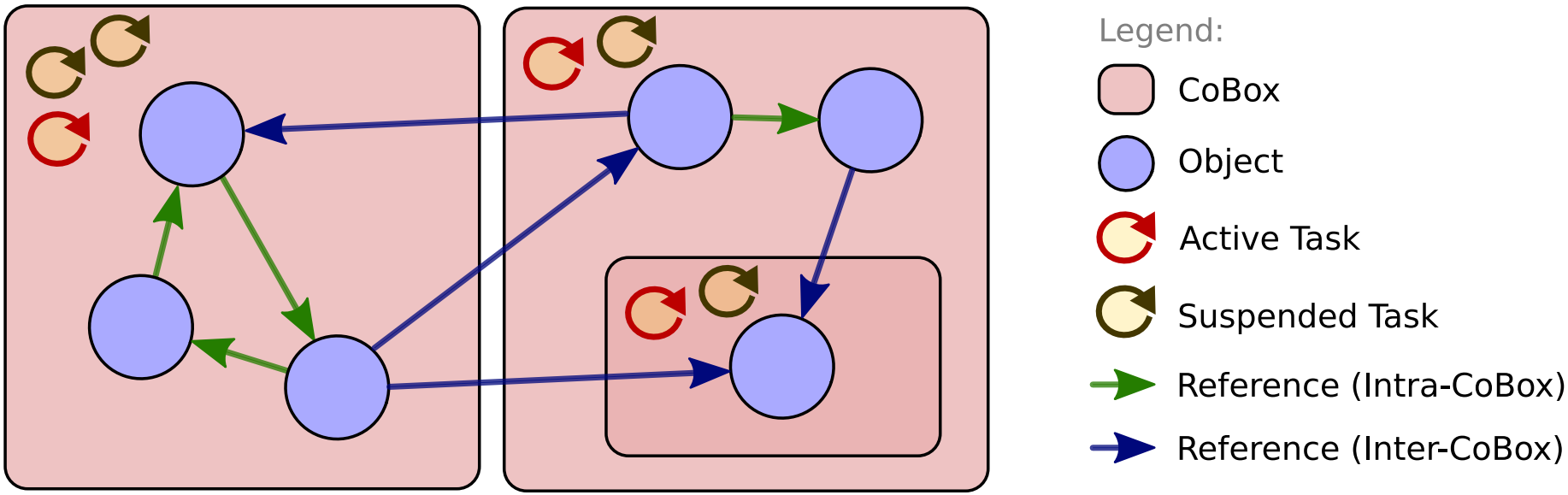
[3] Einar Broch Johnsen et al, 2006

ASP[4]



[4] Denis Caromel, 2004

CoBoxes


Legend:
CoBox
Object
Active Task
Suspended Task
Reference (Intra-CoBox)
Reference (Inter-CoBox)

Groups of Objects

Reasons for regions

- State encapsulation / modularity (ownership types[5])
- Region-based memory management
- Distribution
- Synchronization

How boxes are defined

- Created by designated classes (@CoBox class C ...)
- Other objects are created local to the box of the creator

[5] e.g. Clarke et al, 1998

Multiple Port Objects

OO-runtime components have multiple objects at the boundary:

Examples

- Linked List (Iterators)
- XML-DOM (Document $\rightarrow$ Elements $\rightarrow$ Attributes)
- GUIs (Frame $\rightarrow$ Toolbar $\rightarrow$ Buttons)
- Synchronization of concurrent sessions
- COM/OSGi: exposed services
- ...

Example: List

```
interface List {
  Iterator getIterator();
  ...
}
```

Example: XML-DOM

```
interface DOM {
  Document getDocument();
  ...
}

interface Document {
  Element getDocumentElement();
  ...
}

interface Element {
  Attribute getAttribute(String name);
}
```

Concepts and Approach

JCoBox: The Language

Implementation

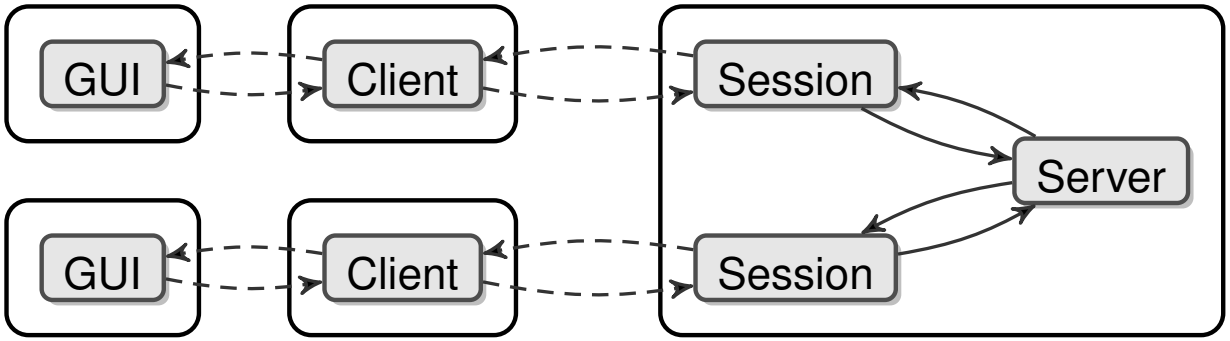
JCoBox: Sequential Java + CoBoxes

- Sequential Java
- CoBox classes: @CoBox class ...

- Asynchronous method calls: x!m()
- Futures: Fut<T> f;
- Cooperative tasks:
  - blocking: f.get
  - suspension: f.await, yield
- a.m() ≡ a!m().get

- Transfer objects: @Transfer class ...
- Immutable objects: @Immutable class ...

Example: Simple Chat Application




```
interface Client4Session {
    void onChatMsg(Msg m);
}

interface Client4GUI {
    void publishMsg(String m);
}
```

```
interface Server {
    Session connect(Client c);
}

interface Session {
    void publish(Msg m);
    void close();
}
```

Example: Server Class

```
@CoBox
class ChatClient implements Client4Session, Client4GUI {
    GUI gui = new GUI(this); Session session; String userName;

    ChatClient(String name, Server s) {
        userName = name;
        gui!init(userName);
        Fut<Session> sessFut = s!connect(userName,this);
        session = sessFut.await();
    }

    public void onChatMsg(Msg m) {
        gui!showString(m.user+": "+m.content);
    }

    public void publishMsg(String s) {
        if( session != null ) session!publish(s);
    }
}
```

Example: Server Class

```
@CoBox
class ChatServer implements Server {
    List<ChatSession> sessions = new ArrayList<ChatSession>();

    public Session connect(String userName, Client4Session c)
                                                    throws UserAlreadyExists {
        checkUserName(userName);
        ChatSession s = new ChatSession(this,userName,c);
        sessions.add(s);
        return s;
    }

    void checkUserName(String userName) throws ... {
        ...
    }
}
```

Example: Session Class

```
class ChatSession implements Session {
    private ChatServer server;
    private Client4Session client;
    String userName;
    private boolean isClosed = false;

    ChatSession(ChatServer s, String uNm, Client4Session c) { ... }

    public void publish(String msg) {
        if (isClosed) throw new DisconnectedException();
        Msg msgx = new Msg(userName,msg);
        for (ChatSession s : server.sessions) {
            s.client!onChatMsg(msgx);
    } }

    public void close() { ... }
}
```

Example: Transfer Class

```
@Transfer
class Msg {
    String user;
    String content;

    Msg(String user, String content) {
      this.user = user; this.content = content;
    }
}
```

Concepts and Approach

JCoBox: The Language

Implementation

Implementation

Language and Compiler

- based on Polyglot 1.3.5
- support for Java 5

CoBoxes

- represented by a cobox object
- objects have an additional field referencing their cobox
- objects of transfer or immutable classes have no such field

Method calls

- Asynchronous calls are wrapped into special task objects that also deal with parameter transfer
- Simple optimizations aviod that unnessary futures are created
- Where possible, synchronous calls are treated as normal method calls

Task scheduling

- The tasks of a cobox are handled by a FIFO scheduler.
- Asynchronous calls (task objects) are executed by a thread pool.