Linguistic Symbiosis between Actors and Threads

Tom Van Cutsem    Stijn Mostinckx    Wolfgang De Meuter

Programming Technology Lab
Vrije Universiteit Brussel
Brussels, Belgium

International Conference on Dynamic Languages, August 27th 2007, Lugano
Overview

- **AmbientTalk**: OO DSL for mobile ad hoc networks
- Pure *event-driven* concurrency model (actors [Agha86])
- How to do a *safe* linguistic *symbiosis* between actors and threads?
Actors vs. Threads

actor: 
{
    def obj := object: 
    {
        def m() { ... }
    }
}

def button := Button.new("Click Me");
button.addActionListener(object: 
{
    def actionPerformed(ActionEvent actionEvent) 
    {
        obj.m();
    }
})

obj.m();
Actors vs. Threads

actor: {
    def obj := object: {
        def m() { ... }
    }
}

def button := Button.new("Click Me");
button.addActionListener(object: {
    def actionPerformed(actionEvent) {
        obj.m();
    }
});

obj.m();
Event Loop Concurrency

- Events are executed **serially**
- Event notification is strictly **asynchronous**
- Event loops should have **no shared state**
Event loop concurrency

Based on E programming language [Miller05]
Event loop concurrency

Based on E programming language [Miller05]

Actor

‘local’ object

Message queue

Event loop
Event loop concurrency

Based on E programming language [Miller05]

Actor

‘local’ object

Message queue

Event loop

obj

obj.m()
Event loop concurrency

Based on E programming language [Miller05]
Event loop concurrency

Based on E programming language [Miller05]

Actor

‘local’ object

‘remote’ object

Message queue

Event loop

obj<-m()

obj
Event loop concurrency

Based on E programming language [Miller05]

Actors cannot cause deadlock
No race conditions on objects

Message queue
Event loop

Actor
‘local’ object

‘remote’ object

obj←m()

obj

Based on E programming language [Miller05]
AmbientTalk/Java

Based on Inter-language Reflection [Gybels et al 05]

• AmbientTalk is implemented in Java
• Data mapping: cfr. JRuby, Jython, JScheme, LuaJava, JPiccola, ...
• Tight integration at the syntactic level

```java
def Button := jlobby.java.awt.Button;
def button := Button.new("Click Me");
button.addListener(object: {
    def actionPerformed(actionEvent) { ... }
});
button.setVisible(true);
```
Actor/Thread Mapping
Actor/Thread Mapping
def obj := object: { ... };

aJavaCollection.add(obj);
def obj := object: { ... };
ajavaCollection.add(obj);
Actors as Threads

```python
def obj := object: { ... };
ajavaCollection.add(obj);
```
def obj := object: { ... };
ajavaCollection.add(obj);
Actors as Threads

\[
def \text{obj} := \text{object:} \{ \\
    \text{def compareTo(}\text{other}) \{ ... \}
\}
\]

\text{aJavaCollection}.add(\text{obj});
def obj := object: {
    def compareTo(other) { ... }
}
aJavaCollection.add(obj);
def obj := object: {
def compareTo(other) { ... }
}

aJavaCollection.add(obj);
def obj := object: {
    def compareTo(other) { ... } }

aJavaCollection.add(obj);
def obj := object: {
    def compareTo(other) { ... }
}
aJavaCollection.add(obj);
def ambientTalkTest := object: {
    def countTestCases() { ... }
    def run(result) { ... }
}

interface junit.framework.Test {
    public int countTestCases();
    public void run(TestResult r);
}
interface junit.framework.Test {
    def ambientTalkTest := object: {
        def countTestCases() {
            ...
        }
        def run(result) {
            ...
        }
    }
}

TestSuite suite = new TestSuite();
ATObject atUnitTest = /* load ambienttalk test */;
suite.addTest((Test) wrap(atUnitTest, Test.class));
suite.addTest(aJavaUnitTest);
junit.textuitut.TestRunner.run(suite);
Threads as Actors

suite

ambientTalkTest

Actor
Threads as Actors
Threads as Actors

suite

run(result)

ambientTalkTest

Actor
Threads as Actors

suite

ambientTalkTest

Actor
Threads as Actors

suite

wrapper

ambientTalkTest

Actor
Threads as Actors

```java
suite

barrier.get()

wrapper

ambientTalkTest

Actor
```
Threads as Actors

suite

wrapper

barrier.get()

ambientTalkTest

Actor
Threads as Actors

suite

wrapper

barrier.get()

ambientTalkTest

Actor
Threads as Actors

ActionListener l = ...;
l.actionPerformed(actionEvent);

\[\text{def button := Button.new(“Click Me”);}
\text{button.addActionListener(object: {
  \text{def actionPerformed(actionEvent) {
    ...
  }
}});}\]
Threads as Actors

ActionListener l = ...;
l.actionPerformed(actionEvent);

def button := Button.new("Click Me");
button.addActionListener(object: {
    def actionPerformed(actionEvent) {
        ...
    }
});
Threads as Actors

```
ActionListener l = ...;
l.actionPerformed(actionEvent);
```

```
def button := Button.new("Click Me");
button.addActionListener(object: {
    def actionPerformed(actionEvent) {
        ...
    }
});
```
Threads as Actors

ActionListener l = ...;
l.actionPerformed(actionEvent);

def button := Button.new("Click Me");
button.addActionListener(object: {
    def actionPerformed(actionEvent) {
        ...
    }
});
Threads as Actors

ActionListener l = ...;
l.actionPerformed(actionEvent);

def button := Button.new("Click Me");
button.addActionListener(object: {
  def actionPerformed(actionEvent) {
    ...
  }
};
def button := Button.new("Click Me");
button.addActionListener(object: {
def actionPerformed(actionEvent) {
...  
}
});

ActionListener l = ...;
l.actionPerformed(actionEvent);
Threads as Actors

ActionListener l = ...;
l.actionPerformed(actionEvent);

`def button := Button.new("Click Me");
button.addActionListener(object: {
    def actionPerformed(actionEvent) {
        ...
    }
});`
interface I extends java.util.EventListener {
    public void event(...);
}

button
wrapper
buttonListener
Actor
Summary

collection.add(obj)
Summary

obj.compareTo(obj2)
Summary

unitTest.run(reporter)
Summary

listener.actionPerformed(ae)
Experience

- AmbientTalk using Java: AWT and Swing for GUI construction
- Java using AmbientTalk: JEdit plugin for 
collaborative text editing
- Self/Squeak’s Morphic UI framework in 
AmbientTalk
Conclusions

- AmbientTalk: object-oriented (distributed) event-driven programming

- Symbiotic Thread/Actor mapping:
  - AmbientTalk invocations proceed immediately
  - Automatic synchronization of Java invocations
  - Support for Java “event notifications” (listeners)

http://prog.vub.ac.be/amop