AmbientTalk: Object-oriented Event-driven programming in Mobile Ad hoc Networks

Tom Van Cutsem

Programming Technology Lab
Vrije Universiteit Brussel
Brussels, Belgium

LAMP - EPFL, July 25th 2007, Lausanne
Object-oriented programming languages

Context

Hardware

Software

Pervasive Computing
(Mobile Networks)
Object-oriented programming languages

Hardware

Software

Pervasive Computing (Mobile Networks)
Mobile Ad hoc Networks
Mobile Ad hoc Networks

Intermittent Connectivity
Mobile Ad hoc Networks

- Scarce Infrastructure
- Intermittent Connectivity
Loose Coupling

[Eugster et al. 03]

Decoupling communication in *Time & Synchronisation*
reduces impact of volatile connections
Loose Coupling

[Eugster et al. 03]

Decoupling communication in *Time & Synchronisation* reduces impact of volatile connections
Loose Coupling

[Euqster et al. 03]

Decoupling communication in *Time & Synchronisation*

reduces impact of volatile connections
Loose Coupling

Decoupling communication in *Time & Synchronisation* reduces impact of volatile connections

[Eugster et al. 03]
Loose Coupling

Decoupling communication in *Time & Synchronisation* reduces impact of volatile connections

[Ingram et al. 03]
Loose Coupling

[Lawes et al. 03]

Decoupling communication in *Time & Synchronisation*

reduces impact of volatile connections
Loose Coupling

[Eugster et al. 03]

Decoupling communication in *Time & Synchronisation* reduces impact of volatile connections
Loose Coupling

Decoupling communication in *Space*

enables ad hoc anonymous collaborations
Loose Coupling

Decoupling communication in *Space*

enables ad hoc anonymous collaborations
Loose Coupling

Decoupling communication in Space enables ad hoc anonymous collaborations

provide service
Decoupling communication in *Space* enables ad hoc anonymous collaborations.
Example: music player
Example: music player
Example: music player
Example: music player

21%  32%
AmbientTalk: the language

- Distributed prototype-based object-oriented language
- Event-driven concurrency based on actors [Agha86]
- Future-type asynchronous message sends
- Built-in publish/subscribe engine for service discovery of remote objects
• Started in 2005
• Small team: 3-6 people
• Interpreter (not optimised)
• Pure Java implementation
• Runs on J2ME/CDC phones
def Window := object: {
    def title := "Untitled";
    def init(t) {
        title := t;
        super := ClosedWindow;
    }
    def show() {
        super := OpenWindow;
    }
}

def OpenWindow := object: {
    def draw() { ... }
}
def ClosedWindow := object: {
    def draw() { ... }
}

def w := Window.new("Test");
w.draw();

- Prototypes
- Delegation
- Trait composition
- First-class delegation
Extensible language

```python
def fac(n) {
    if: (n = 0) then: {
        1
    } else: {
        n * fac(n-1)
    }
}
```
def fac(n) {
    if: (n = 0) then: {
        1
    } else: {
        n * fac(n-1)
    }
}
def Button := jlobby.java.awt.Button;
def b := Button.new("test");
b.addActionListener(object: {
    def actionPerformed(ae) {
        system.println("button pressed");
    }
});
Extensible language

- Block closures
- Keyworded messages
- Interfacing with JVM
- Reflection

```python
def fac(n):
    if n == 0:
        return 1
    else:
        return n * fac(n-1)

def Button := jlobby.java.awt.Button;
def b := Button.new("test");
b.addActionListener(object: {
    def actionPerformed(ae) {
        system.println("button pressed");
    }
});
```
Event loop concurrency

Based on E programming language [Miller05]

Actor

Message queue

Event loop
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]
Event loop concurrency

Based on E programming language [Miller05]

Actor

‘local’ object

‘remote’ object

Message queue

Event loop
Event loop concurrency

Based on E programming language [Miller05]

Actor

‘local’ object

‘remote’ object

Message queue

Event loop

obj <- m()
Event loop concurrency

Based on E programming language [Miller05]

Actors cannot cause deadlock
No race conditions on objects
def future := mplayer<-numSongsInLibrary()
def future := mplayer<--numSongsInLibrary()
def future := mplayer<-numSongsInLibrary()
def future := mplayer<->numSongsInLibrary()
def future := mplayer<--numSongsInLibrary()
def future := mplayer<-numSongsInLibrary()

when: future becomes: { num |
    system.println("user shares "+ num + " songs.")
}
def future := mplayer<-numSongsInLibrary()

when: future becomes: {
    system.println("user shares "+ num + " songs.")
}
deftype MusicPlayer;

def interface := object: {
  def openSession() {
    ...
  }
}

export: interface as: MusicPlayer;
deftype MusicPlayer;

def interface := object: {
  def openSession() {
    ...
  }
}

export: interface as: MusicPlayer;
def mplayerFuture := ambient: MusicPlayer;

• Initiates service discovery
• Immediately returns future for object to be discovered
def mplayerFuture := ambient: MusicPlayer;

- Initiates service discovery
- Immediately returns future for object to be discovered
def mplayerFuture := ambient: MusicPlayer;
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
ambient References

```python
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
```
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();

when: mplayerFuture becomes: { |ambientRef|
    println("music player found")
}
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();

when: mplayerFuture becomes: {
    |ambientRef|
    println("music player found")
}

Ambient References
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();

when: mplayerFuture becomes: {
  println("music player found")
}
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();

when: mplayerFuture becomes: {
  |ambientRef|
  println("music player found")
}
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();

when: mplayerFuture becomes: { |ambientRef|
  println("music player found")
}
def mplayerFuture := ambient: MusicPlayer;
def sessionFuture := mplayerFuture<-openSession();
Event handlers on ambient references:

```java
when: ambientRef disconnects: {
    println("music player disconnected")
}

when: ambientRef reconnects: {
    println("music player reconnected")
}
```
• Event handlers on ambient references:

```java
when: ambientRef disconnects: {
    println("music player disconnected")
}

when: ambientRef reconnects: {
    println("music player reconnected")
}
```
Event handlers on ambient references:

```java
when: ambientRef disconnects: {
  println("music player disconnected")
}

when: ambientRef reconnects: {
  println("music player reconnected")
}
```
Failure Handling

- **Leased** references which eventually expire
- **Futures + timeouts**

```when: o<-m()@Timeout(minutes(10)) becomes: { |v|
  // process return value
} catch: TimeoutException using: { |e|
  // deal with timeout
}```
Variations

- Past experiments:
  - ambient ‘omni’-references: broadcasting
  - content-based discovery

- Future experiments:
  - Customisable message delivery guarantees
  - First-class proximity: restrict spatial scope of ambient references
Lessons Learned

• AmbientTalk = OO + Events:
  • Block closures as nested event handlers
  • Futures: non-blocking synchronisation
  • Buffered asynchronous messaging abstracts over intermittent connectivity
  • Ambient references: space-decoupled remote object references
Conclusion

- MANETs: loosely coupled collaboration
- AmbientTalk: actor-based OO language
- Deal with universal MANET characteristics at the language level:
  - Intermittent Connectivity: time & sync-decoupled references
  - Scarce Infrastructure: space-decoupled references

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