Ambient-oriented Programming & AmbientTalk

Tom Van Cutsem    Stijn Mostinckx    Elisa Gonzalez Boix
Andoni Lombide    Christophe Scholliers    Wolfgang De Meuter

Software Languages Lab
Brussels, Belgium

donderdag 15 oktober 2009
Agenda

• Context: Mobile & Ubiquitous computing

• Approach: Ambient-oriented programming

• Tool: AmbientTalk

• Experiments: Demo applications
Context

Mobile & Ubiquitous Computing
Ubiquitous Computing

- Research vision postulated by Mark Weiser (1988, Xerox PARC)
Today’s Applications

Smart Homes/Domotics

RFID Inventory Management

Tourism/City Guide Software

Personal Area Networks
Issues

• Hardware Issues:
  • Miniaturisation
  • Device Autonomy
  • Interoperability
  • Processor Speed
  • Limited Memory
  • Integration
  • Cost

• Software Issues:
  • Context-awareness
  • Interaction with real world
  • Portability
  • New user interfaces
  • Standards
  • Distributed Applications
Issues

• Hardware Issues:
  • Miniaturisation
  • Device Autonomy
  • Interoperability
  • Processor Speed
  • Limited Memory
  • Integration
  • Cost

• Software Issues:
  • Context-awareness
  • Interaction with real world
  • Portability
  • New user interfaces
  • Standards
  • Distributed Applications
Mobile Ad Hoc Networks

Networks composed of mobile devices that communicate wirelessly
Mobile Ad Hoc Networks

Networks composed of **mobile** devices that communicate **wirelessly**
Mobile Ad Hoc Networks

Networks composed of **mobile** devices that communicate **wirelessly**

Zero Infrastructure

Volatile Connections
Ambient-oriented Programming
Observation #1: interaction with proximate peers

Example: match making between proximate peers

Offer:
- type = Concert Ticket
- ticket.artist = “U2”
- ticket.location = “Hyde Park”

Demand:
- type = Concert Ticket
- ticket.artist = “U2”
- ticket.location = *
Observation #1: interaction with proximate peers

Example: match making between proximate peers

demand


type = Concert Ticket
ticket.artist = “U2”
ticket.location = *

offer

type = Concert Ticket
ticket.artist = “U2”
ticket.location = “Hyde Park”

donderdag 15 oktober 2009
Observation #1: interaction with proximate peers

Example: match making between proximate peers

Offer:
- type = Concert Ticket
- ticket.artist = “U2”
- ticket.location = “Hyde Park”

Demand:
- type = Concert Ticket
- ticket.artist = “U2”
- ticket.location = *

No reliance on fixed, always-available server infrastructure
Observation #2: intermittent connectivity
Observation #2: intermittent connectivity
Observation #2: intermittent connectivity
Observation #2: intermittent connectivity

Tolerate disconnections, because they occur frequently rather than exceptionally
Software concerns

uMaMa
Software concerns

\[uMaMa\]
Software concerns

uMaMa

Discovery  Communication

donderdag 15 oktober 2009
Software concerns

uMaMa

- Discovery (21%)
- Communication (32%)
- Synchronisation

Donderdag 15 oktober 2009
Software concerns

uMaMa

Discovery  Communication  Synchronisation
Software concerns

🎵 uMaMa

Discovery  Communication  Synchronisation  Failure handling

donderdag 15 oktober 2009
AmbientTalk: fact sheet

- Object-oriented scripting language
- Started in 2005
- Pure Java implementation
- Runs on J2ME/CDC phones
- Open source implementation

[Link to code.google.com/p/ambienttalk]
How does AmbientTalk help?

**Volatile Connections**

network connections are resilient to failures by default

**Zero Infrastructure**

service discovery protocol built into the language
def makeSong(artist, title) {
    object: {
        def printArtist() {
            if: (artist == nil) then: {
                “unknown artist”;
            } else: {
                artist;
            }
        }
    }
}
```python
def makeSong(artist, title):
    object:
        def printArtist():
            if: (artist == nil) then:
                "unknown artist";
            else:
                artist;
        
def song := makeSong("U2", "One");
song.printArtist();
```
Event Loop Concurrency

- AmbientTalk programs are event loops
- They react to events from the outside world
- They communicate asynchronously
Examples of event loops

• GUI Frameworks (e.g. Java AWT)

• Highly interactive applications (e.g. games)

• IPC in Operating Systems

• Discrete Event Modelling (e.g. simulations)

• Web servers
Event Loop Concurrency in AmbientTalk
Event Loop Concurrency in AmbientTalk

Event Loop

Message queue
Event Loop Concurrency in AmbientTalk

'local' object
Event Loop Concurrency in AmbientTalk

'de remote' object
Event Loop Concurrency in AmbientTalk
Event Loop Concurrency in AmbientTalk

obj

"do m immediately"

obj.m()
Event Loop Concurrency in AmbientTalk

```
obj <- m()
```

“do m eventually”
Event Loop Concurrency in AmbientTalk

```
obj <- m()
```

“do m eventually”
Event Loop Concurrency in AmbientTalk
Event Loop Concurrency in AmbientTalk

when: future becomes: { |value|
    // process reply
}
Event Loop Concurrency in AmbientTalk

when: future becomes: { |value|
    // process reply
}
Exporting & discovering objects
Exporting & discovering objects

deftype MusicPlayer

deftype MusicPlayer
Exporting & discovering objects

deftype MusicPlayer

export: mplayer as: MusicPlayer
Exporting & discovering objects

deftype MusicPlayer

whenever: MusicPlayer discovered: { mplayer
    // open a session
}

deftype MusicPlayer

export: mplayer as: MusicPlayer

donderdag 15 oktober 2009
Exporting & discovering objects

deftype MusicPlayer

whenever: MusicPlayer discovered: {
    mplayer
    // open a session
}

detype MusicPlayer

export: mplayer as: MusicPlayer
Far References

uploadSong

session
Far References

uploadSong

session

donderdag 15 oktober 2009
Far References

donderdag 15 oktober 2009
Far References

session

donderdag 15 oktober 2009
Far References

donderdag 15 oktober 2009
when: session<-uploadSong(s)@Due(timeout) becomes: { lack
    // continue exchange
} catch: TimeoutException using: { le
    // stop exchange
}
Leasing

lease: 10 minutes

Connected (messages are forwarded)

Disconnected (messages are buffered)

Expired (messages are dropped)

expire

disconnect

reconnect

expire

donderdag 15 oktober 2009
Reacting to failures

lease: 10 minutes

Connected (messages are forwarded)
Disconnected (messages are buffered)
Expired (messages are dropped)
Reacting to failures

when: session disconnected: {
    // pause transmission
}
Reacting to failures

when: session disconnected: {
    // pause transmission
}
when: session reconnected: {
    // resume transmission
}
Reacting to failures

when: session disconnected: {
    // pause transmission
}
when: session reconnected: {
    // resume transmission
}
when: session expired: {
    // stop transmission
}
Reacting to failures

when: session disconnected: {
  // pause transmission
}
when: session reconnected: {
  // resume transmission
}
when: session expired: {
  // stop transmission
}

when: session expired: {
  // clean up resources
}
Event Notifications

when: type discovered: { |obj| ... }  whenever: type discovered: { |obj| ... }

when: obj disconnected: { ... }  whenever: obj disconnected: { ... }

when: obj reconnected: { ... }  whenever: obj reconnected: { ... }

when: obj expired: { ... }  whenever: 5.minutes elapsed: { ... }

when: future becomes: { |result| ... }
Event Notifications

Discovery

when: type discovered: { |obj| ... }  whenever: type discovered: { |obj| ... }

when: obj disconnected: { ... }  whenever: obj disconnected: { ... }

when: obj reconnected: { ... }  whenever: obj reconnected: { ... }

when: obj expired: { ... }

when: 5.minutes elapsed: { ... }  whenever: 5.minutes elapsed: { ... }

when: future becomes: { |result| ... }

donderdag 15 oktober 2009
Event Notifications

when: type discovered: { |obj| ... }  whenever: type discovered: { |obj| ... }

when: obj disconnected: { ... }  whenever: obj disconnected: { ... }

when: obj reconnected: { ... }  whenever: obj reconnected: { ... }

when: obj expired: { ... }

when: 5.minutes elapsed: { ... }  whenever: 5.minutes elapsed: { ... }

when: future becomes: { |result| ... }
Event Notifications

when: type discovered: { |obj| ... }  whenever: type discovered: { |obj| ... }

when: obj disconnected: { ... }  whenever: obj disconnected: { ... }

when: obj reconnected: { ... }  whenever: obj reconnected: { ... }

when: obj expired: { ... }

Synchronisation

when: 5.minutes elapsed: { ... }  whenever: 5.minutes elapsed: { ... }

when: future becomes: { |result| ... }
Language Interoperability

- Scripting language on top of the JVM, cfr. JRuby, Jython, Groovy, ...
- AmbientTalk can use Java libraries, Java can use AmbientTalk scripts
def Button := jlobby.java.awt.Button;
def button := Button.new(“Click Me”);
button.addActionListener(object: {
def actionPerformed(actionEvent) {
    println(“button clicked”);
}
});
button.setVisible(true);
Reflection

• AmbientTalk code can introspect and change behavior of objects and actors

```python
def makeSong(artist, title) {
    object: {
        def printArtist() {
            if: (artist == nil) then: {
                "unknown artist";
            } else: {
                artist;
            }
        }
    }
}

def song := makeSong("U2", "One");
song.printArtist();
```
Reflection

- AmbientTalk code can introspect and change behavior of objects and actors.
Implementation

• Interactive interpreter

• ± 17,000 SLOC of Java

• UDP & TCP/IP over WLAN

• Runs on top of J2ME/CDC
  • QTek 9090 SmartPhones
  • HTC Touch Cruise SmartPhones
  • iPhone [in progress]
  • Android G1 [in progress]
Case: Musical Match Maker

- Demo

<table>
<thead>
<tr>
<th>Wolf</th>
<th>MuMaMa: Tom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200 Micrograms - Rock into the...</td>
<td>1200 Micrograms - Rock into the...</td>
</tr>
<tr>
<td>Admiral Freebee - Noorderlaan</td>
<td>Kate Bush - Wuthering Heights</td>
</tr>
<tr>
<td>Electric Six - Danger</td>
<td>Pendulum - Fasten Your Seatbelt</td>
</tr>
<tr>
<td>Hooverphonic - Club Montepulciano</td>
<td>Kate Bush - Wuthering Heights</td>
</tr>
<tr>
<td>Vivaldi - Spring</td>
<td>Electric Six - Danger</td>
</tr>
<tr>
<td>Kate Bush - Wuthering Heights</td>
<td>Metallica - One</td>
</tr>
<tr>
<td>Metallica - One</td>
<td>Vivaldi - Spring</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tom</th>
<th>MuMaMa: Wolf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200 Micrograms - Rock into the...</td>
<td>1200 Micrograms - Rock into the...</td>
</tr>
<tr>
<td>Kate Bush - Wuthering Heights</td>
<td>Admiral Freebee - Noorderlaan</td>
</tr>
<tr>
<td>Pendulum - Fasten Your Seatbelt</td>
<td>Electric Six - Danger</td>
</tr>
<tr>
<td>Hooverphonic - Club Montepulciano</td>
<td>Kate Bush - Wuthering Heights</td>
</tr>
<tr>
<td>Vivaldi - Spring</td>
<td>Metallica - One</td>
</tr>
<tr>
<td>Kate Bush - Wuthering Heights</td>
<td>Vivaldi - Spring</td>
</tr>
</tbody>
</table>

Status: transfer complete, 29% songs match

Status: transfer complete, 66% songs match
Experimental Results

- Memory management
- Concurrency control
- Failure handling
- Application

Lines of Code

Java RMI

AmbientTalk
Experimental Results

![Bar chart showing experimental results for Java RMI and AmbientTalk.]

- Java RMI:
  - Application: 32.00%
  - Failure handling: 17.00%
  - Concurrency control: 20.00%
  - Memory management: 31.00%

- AmbientTalk:
  - Application: 77.00%
  - Failure handling: 7.00%
  - Concurrency control: 8.00%
  - Memory management: 8.00%
Causes

• Explicit encoding of
  • buffered, asynchronous communication using threads
  • remote messages using objects
  • timeouts using timer threads
  • event notifications (lease expiration & renewal, calls & callbacks) using listeners + event loop threads

• Java RMI does not deal with service discovery
Case: Instant Messenger

• Demo

![Instant Messenger - Tom]

Wolf
Hi Wolf

Buddy discovered: Wolf
Tom: Hi Wolf

Send

![Instant Messenger - Wolf]

Tom
Hi

Buddy discovered: Tom
Tom: Hi Wolf

Send
Chat: Java vs AmbientTalk (LoC)
Chat: Java vs AmbientTalk (%)
A simple application but...

... we do not need to explicitly manage:

- threads & locks
- low-level socket connections
- stubs, skeletons
- name server or lookup service
- timeouts, leasing
AmbientMorphic

• Implementation of the Morphic UI framework from Self

• Demo: PortalPong
Conclusion

- Mobile ad hoc networks:
- Ambient-oriented programming:
  - AmbientTalk: 
    - Scripting language on top of the JVM
    - Reactive, event-driven programs
- Applications: chat, match maker, multiplayer game
Conclusion

- Mobile ad hoc networks: Zero Infrastructure
- Mobile ad hoc networks: Volatile Connections
- Ambient-oriented programming:
  - AmbientTalk:
    - Scripting language on top of the JVM
    - Reactive, event-driven programs
- Applications: chat, match maker, multiplayer game
Conclusion

- Mobile ad hoc networks: Zero Infrastructure Volatile Connections
- Ambient-oriented programming: Peer-to-peer Tolerate disconnections
- AmbientTalk:
  - **Scripting language** on top of the JVM
  - Reactive, **event-driven** programs
- **Applications**: chat, match maker, multiplayer game
Don’t program the hardware of the future with the software of the past

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