Ambient-Oriented Programming: OOP without Wires and Threads
Phenomenon #1: Miniaturisation

±1975

±1985

±1995

±2005
Phenomenon #2: Wireless
Ambient Applications

Wolfgang De Meuter - ECOOP’08 - Paphos (Cyprus)
A mobile ad hoc network is a transitory association of mobile nodes that communicate wirelessly and that do not depend on any fixed support infrastructure.

Fundamental characteristics:
- volatile connections
- zero infrastructure
Network failure is the rule rather than the exception. Reconnection is more frequent than connection. This is problematic for object designation as well as for communication.

The ambient is inherently concurrent and event-driven. Our applications are object-based. Objects and events do not match easily.
About Middleware

Essential Complexity ≠ Accidental Complexity

Frederick Brooks
Language Design Motivations

- Tackle essential complexity and bury accidental complexity in the interpreter
- Study interactions between ambient concepts and other language concepts
- Local culture and background
Beware!

Distribution transparency is a myth that is both misleading and dangerous.

Rachid Guerraoui
Ambient-Oriented Programming

A subset of the OO paradigm that is explicitly geared towards programming mobile ad hoc networks

Classless Object Models
Ambient Acquaintance Management
Non-blocking Communication Primitives
Reified Communication Traces
Simplicity

The Power of Simplicity

David Ungar

Innovation vs. Integration in Language Design

Sir C.A.R. Hoare
AmbientTalk

Language Laboratory

Research Language (= Toy Language)

Currently runs on mobile phones
Characteristics

- Dynamically Typed
- REPL-driven
- Reflective
- Prototype-based
- First-class messages & methods
- Traits
- Escape-continuations
- Concurrent
- Distributed
Classless Object Model

def Point := object: {
def x := 0;
def y := 0;
def init(aX,aY) {
x := aX;
y := aY;
};
def sumOfSquares() {
x*x + y*y 
};
}
def Point3D := extend: Point with:{
def z := 0;
def sumOfSquares() {
    super^sumOfSquares() + z*z;
};
}
def Point2 := isolate: {
def x := 0;
def y := 0;
def init(aX,aY) {
x := aX;
y := aY;
};
def sumOfSquares() {
x*x + y*y 
};
}
AmbientTalk Concurrency Model

- Actors
- Objects
- Message queue
- Event loop

Example:
- $o_1 \leftarrow m()$
- $o_1.m()$
- $o_2 \leftarrow m()$

Wolfgang De Meuter - ECOOP’08 - Paphos (Cyprus)
def calculator := actor: {
    def add(x,y,customer) {
        customer <- result(x+y)
    };
};

calculator<-add(1,2,object: {
    def result(sum) {
        system.println("sum = " + sum);
    };
});

Parameter passing by copy & by far reference
Non-blocking Futures

```python
import lobby.at.lang.futures

def calculator := actor: {
    def add(x,y) {
        x+y
    }
}

def sumFuture := calculator<-add(1,2);

when: sumFuture becomes: {
    |sum|
    system.println("The sum is " + sum)
} catch: {
    |exc|
    system.println("Exception: " + exc.message)
}
```
deftype InstantMessenger;

export: (object: ...) as: InstantMessenger;

deftype InstantMessenger;

when: InstantMessenger discovered: {
  |messenger|
  when: (messenger<-getName()) becomes: {
    |name|
    buddyList.put(name, messenger);
    system.println("Added buddy: " + name);
  };
};
Failures can be omnipresent
def Button := jlobby.java.awt.Button;
def b := Button.new(“Click Me”);

b.addActionListener(object: {
def actionPerformed(event) {
system.println(“Pressed!”);
} })
That concludes the language

- Little Accidental Complexity
- It's simple
- Nice exercise in integration
- Maybe a bit simplistic?
Steroids?

Onward!
Defining Network Failure

when: InstantMessenger discovered: {
    |messenger|
    ...
    whenever: messenger disconnected: {
        system.println("Buddy offline: " + name);
    };
    whenever: messenger reconnected: {
        system.println("Buddy online: " + name);
    };
};

when: buddy<-chat(text)@Due(minutes(1)) becomes:
    |ack|
    //received succesfully
}
catch: {
    |exc|
    // test whether ‘exc’ is a TimeOutException
}
Far references reconnect
• to just one object
• to the same objet

Broadcasting?
(e.g. all nearby shops)

Roaming?
(e.g. a GPS location tracker)
Ambient References

Intentional Description of Remote Object(s)

```
deftype Shop

export: Paphos_H_and_M
     as: Shop
with:  { type = "clothes";
          visa = true;
          maestro = true };

def shops:= ambient: Shop
         where: { |
                lsl
                (s.type ~= ".*cloth.*").and:
                (s.visa = true) }
```
def ShoppingAssistant := object: {
def bestPriceSoFar;
def bestShopSoFar;
def multifuture;
def startSearch() {
def shops := ambient: Shop;
multifuture := shops<-query(item)@[ All, Sustain, Reply ];
whenEach: multifuture becomes: {
|reply|
if: ((bestPriceSoFar == nil).or:{reply.price < bestPriceSoFar})
then:{
[bestPriceSoFar, bestShopSoFar] := [reply.price, reply.shopId];
updateGUI(bestPriceSoFar, bestShopSoFar); }};
def stopSearch() {
multifuture.cancel() }"
Ambient Messages: Taxonomy

- **Communication Lifetime**
  - One-way
    - @Oneway
  - Two-way
    - Unbounded
      - @Reply
    - Bounded
      - @Due(t)
- **Discovery Lifetime**
  - Instant
    - @Instant
  - Transient
    - @Transient(t)
  - Sustainable
    - @Sustain
- **Expirable**
  - @Expires(t)
- **Arity**
  - Point-to-point
    - @One
  - One-to-many
    - @All

Wolfgang De Meuter - ECOOP’08 - Paphos (Cyprus)
Ambient-Oriented Programming

Classless Object Models
- far references
- isolates
- near references
- objects

Ambient Acquaintance Management
- service discovery
- resilience to failures

Non-blocking Communication Primitives
- asynchronous send
- asynchronous receive
- actors

Reified Communication Traces
- TimeWarp
- ambient references
- ambient messages

experiments
- leasing
- defining failure
An Application: Portal Pong

Wolfgang De Meuter - ECOOP'08 - Paphos (Cyprus)
Now What?

More Onward!
The Internet of Things

500 billion nodes (2020)

new buzzword

ambient \cap shopping list

massive events
reactive sets
poster

Wolfgang De Meuter - ECOOP’08 - Paphos (Cyprus)