Ambient-Oriented Programming

Designing an Object-Oriented Language for Ambient Intelligence

Dr. Éric Tanter
University of Chile
DCC/CWR

etanter@dcc.uchile.cl
Contents

• What is Ambient Intelligence (AmI)?
• OO Language Design for AmI
• AmbientTalk - A First OO Language for AmI
• Future Directions
Credits

Programming Technology Lab - Vrije Universiteit Brussel
Prof. Dr. Theo D’Hondt

Wolfgang de Meuter
“Move Considered Harmful -- A Language Design Approach to Distribution and Mobility in Open Networks”
PhD Thesis, 2004

Jessie Dedecker, Tom Van Cutsem, Stijn Mostinckx
PhD students (VUB), AmbientTalk development

Sebastián González
PhD student (UCL)

ORION: Open Reflective Infrastructure for Open Networks
Project DCC/CWR - VUB
2006-2007
Ambient Intelligence

A Peculiar Context for Application Development

[Dedecker+05b]
Hardware Environment

- Advent of mobile networks
  - wireless technologies (wifi, bluetooth)
  - nomadic and ad hoc (aka. open networks)

- Mobile devices
  - limited connection range and computational resources

- Ambient Intelligence
  - European Council’s IST Advisory Group [ISTAG03]
  - aka. ubiquitous computing [Weiser91], pervasive computing

what is fundamentally different?
Autonomicity & Concurrency

no central server  natural concurrency

Communication

Autonomous Concurrent Devices

Interconnected via ad hoc wireless networks
Connection Volatility

cannot assume stable connection

but tasks need to be done

Volatile Wireless Connections

Failure

exceptions are no more exceptional!
Resource Discovery

resources depend on location

Ambient Resources

Devices (dis)appear in the 'Ambient'

Spontaneous Service Discovery

I require a printing service

I provide a printing service

Devices (dis)appear in the 'Ambient'
Spontaneous Service Discovery
• A robust Aml application must address these characteristics

• Current alternatives
  • Middleware
  • Distributed programming languages
Middleware

- RPC-based middleware
  - Alice [Haarh+99], Dolmen [Reynolds+99]
  - CORBA on lightweight devices
  - no support for long disconnections

- Publish-subscribe middleware [Cugola+02]
  - good for context-aware communication
  - forces callback programming

- Tuple-space based middleware [Murphy+01]
  - uncoupling in both time and space
  - not object-oriented
Distributed Languages

• Languages for LAN
  • Emerald [Jul+88], Obliq [Cardelli95], etc.
  • communication mechanisms assume reliable connection

• Languages for open networks
  • Janus [Kahn+90], Salsa [Varela+98], E [Miller]
  • pure asynchronous communication
  • no support for resource discovery
Position Statement

[Dedecker+05]

• Currently insufficient support for programming nomadic / ad hoc distributed systems
  • low-level APIs, middleware, etc.
  • can be done, but hard
  • no dedicated abstractions for AmI specificities

• Need of dedicated languages: AmOP

“abstracting without hiding”
Object-Oriented Languages for AmI
Design Dimensions

object model
- class-based vs. prototype-based
- static typing vs. dynamic typing

communication model
- concurrency model
- communication primitives
- failure handling
- acquaintance management
Why Classes Do Not Fit
and neither static typing...

[Gonzales+04]
OOP without classes??

object
= 
encapsulated state 
+ 
message passing

objects are good
encapsulation and polymorphism are fundamental for highly dynamic environments
OOP without classes???

• Classification of OOP languages [Wegner87]
  • object-based (objects + messages)
  • class-based (classes)
  • object-oriented (classes + inheritance)
  • prototype-based (objects + delegation/sharing)
  • ...

Éric Tanter - JCC 2005 - etanter@dcc.uchile.cl
**Fundamental Problem**

**Classes**

A *static, implicit sharing* mechanism of state and behaviour

*Treaty of Orlando*

**Open Networks**

Distribution transparency is impossible to achieve in practice, and precisely because of that impossibility, *it is dangerous to provide the illusion of transparency.*

*Guerraoui&Fayad99*
State Sharing

• class attributes (eg. Java statics)
  • semantics = value shared by all instances

not true under distribution!

Solution: add yet more keywords to Java...

public static-replicated synchronized Vector v;
protected static-centralized String s;

or fix a choice and make it implicit in the language.

It’s no solution: a new node may appear with the same class, but with a different value....
• Exactly same issue with methods

A node containing the **same class** as another node may appear in the network, but with a different method implementation...

**Methods are part of a class’s state, just the same as class attributes.**

If a method changes, **the class is no longer the same**, even if semantically it remains compatible.
Transitive Closures

\textit{instance-of} link + \textit{subclass-of} link

when an object is transmitted, \textbf{its class and all superclasses} must be transmitted too

(an object is not a self-contained entity!)

\textbf{static typing}

transmit argument-type classes, return-type class and exception-type classes and their transitive closures
Conclusion on Classes

Classes are dangerous under distribution

Classes (and static typing) are a barrier to adaptability

sharing is important, but should be managed explicitly

prototype-based languages!
Prototypes

- eg. Self [Ungar+87]

- Self-contained objects
  - state + code
  - created ex-nihilo or by cloning

- Extension mechanism
  - true delegation to parent

- Sharing mechanism
  - parent sharing
  - managed explicitly
  - eg. traits [Ungar+91]
Communication Model

[Dedecker+05b]
OO Concurrency Model

Thread-based

Actor model  [Agha86]
Communication Primitives

- Blocking primitives are evil
- deadlocks & diminished availability of resources

Non-blocking Asynchronous Communication Primitives

Based on Actors (Agha, 1986)
Failure Recovery

- Inconsistent states cannot be avoided
- Need for reversibility

Reified Communication Traces

E.g. rollback strategies for failure recovery

Agenda slot available?

Agenda Actor

Reservation confirmed?

Agenda Actor

Rollback strategies delivery guarantees
Acquaintance Management

- No third party for communication (P2P)
- no explicit reference: *distributed naming*

**Discovery**

Reified Environmental Context

Provided service patterns

Required service patterns

description of service interface

actors specify services required from the ambient provided to the ambient
Design Conclusions

- ✔ Classless language, dynamically-typed
- ✔ Actor model
- ✔ Non-blocking communication primitives
- ✔ Reified communication trace
- ✔ Reified ambient context
AmbientTalk
A First AmOP Language

[Dedecker+05a]
AmbientTalk

► Dynamically typed language
► Prototype-based object model
► Actor-based concurrency model
► VM runs on Java J2SE and J2ME platforms
► VMs communicate via sockets over WLAN
► Plug-and-play: no additional setup
► Tested and deployed on QTek9090 PocketPC smartphones

http://prog.vub.ac.be/amop
Communication Mailboxes
reify actor-actor interaction
Discovery Mailboxes reify actor-ambient interaction
First-class Mailboxes

interaction between mailboxes governed by a metaobject protocol

Meta-level events trapped using mailbox observers
PrintFileBehavior :: object({
  printer : void;
  file : void;

  new(aFile,dpi) :: copy({
    printer := AmbientRef("Printer");
    file := aFile
  });

  printIt() :: {
    when(printer#print(file), {
      display("Doc Printed!")
    }
  }
})

PrintFileActor(file,dpi) :: {
  actor(PrintFileBehavior.new(file,dpi))
}

{ p: PrintFileActor(aFile,300); p#printIt() }
Higher-level Constructs

• Aml is a moving target
  • AmbientTalk is a testbed for AmOP experimentation

• Some higher-level constructs
  • Non-blocking futures
  • Application-specific message routing
  • Group communication
  • ...

[Dedecker+05a]
Language-level Security

[DeMeuter+05]

• Security must be addressed at all levels
  • comm. channels, OS, language, application, user

• Extreme encapsulation principle
  • objects subject to message passing alone
  • no “omnipotent operators”

• But powerful object model
  • cloning, extension, parent sharing, reflection
  • eg. mirror methods [Tanter05]
• Controversial feature
• Security-wise challenging
• A killer application? follow-me applications?

[Vallejos05]
Open Infrastructure

- Reflective Virtual Machine for AmI
- Fine-tune tradeoffs/strategies at the VM level
  - garbage collection
  - failure recovery strategy
  - partial reversible computation
  - partial object matching for resource discovery
Context Awareness

- Context-aware object references \cite{VanCutsem+05}
- Context-oriented programming \cite{Costanza+05}
- Context-aware aspects \cite{Tanter+05}
References (1)


References (2)


[ORION] Open Reflective Infrastructure for Open Networks, draft proposal for a bilateral project between the VUB Prog Lab and the DCC U. Chile, 2006-2007.


