

The Impact of Aspect-Oriented Programming on Future Application Design

Andrei Popovici

Information and Communication Research Group

ISSeminar Jan. 17th, 2001

Outline

- The software world
 - components, objects, languages
- Aspects in the software world
 - the need for aspects
 - what are aspects
 - when to use them
 - how to work with aspects
 - implementation techniques
 - available tools
- The impact of aspects
 - throughout a component's lifetime
 - community-specific adaptations
 - application-aware environments (or contexts)
- Related
 - research areas and CS topics



SE, PL

Ubi

The software world

- Component:
 - unitofindependentdeployment
 - hasnopersistentstate
 - subjectto3rdpartycomposition
 - » [1]
- Object:
 - unitofinstantiation
 - encapsulatesstate&behavior
 - uniqueidentity

Decomposing components

- Component: typically constructed using one language
- Main decomposition paradigm
 - functional, object-oriented
 - the tyranny of the dominant decomposition
 - » [2]
- In practice: a deployed component consists of many objects

Deployment \in
ComponentLifecycle

Preamble

Aspect~ad specere (lat.)

- to look at
- particular appearance to eye or mind

1. Software world,
Component

2. Software aspect

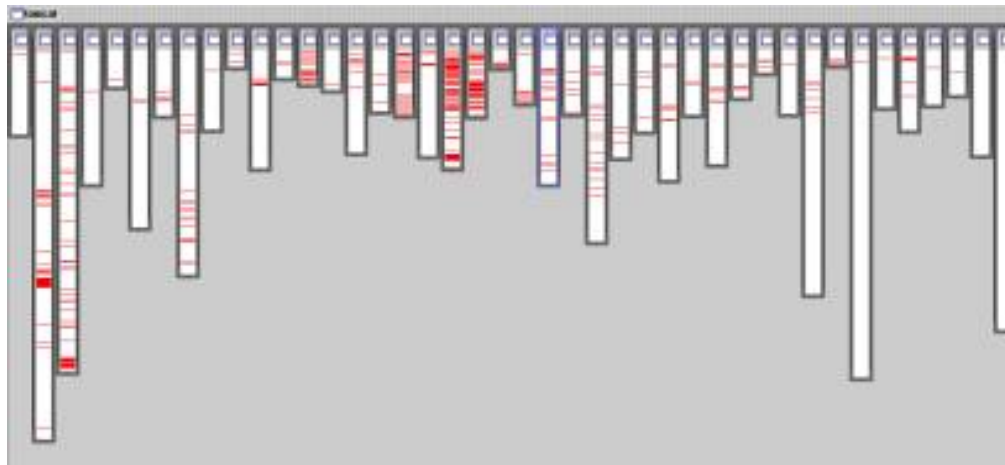
3. Component
lifetime

What is the real world security- aspect of my life ?

- lock the door of my flat in the morning
- unlock office, unlock terminal at work
- lock screen & lock office in the evening
- unlock the door of my place..

Aspectsmotivation

- tyrannyofthemaindecompositionunit(OO)
- Someconcernscannotbeexpressedina modularway
 - Whereisloggingin `org.apache.tomcat?`

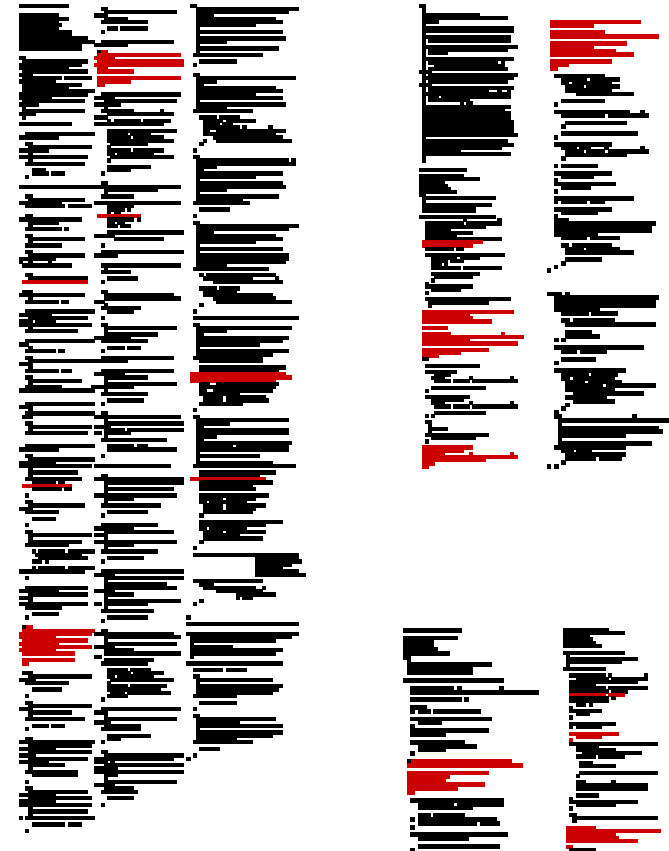


- Notinoneplace
- Noteveninafew
- Notinrelatedplaces

Cost of tangled code

- Redundancy
 - same fragment in more places
- Difficult to reason about
 - the big-picture of tangling is not clear
- Difficult to change
 - have to find all the code involved
 - then to be sure to change it consistently

» [3]

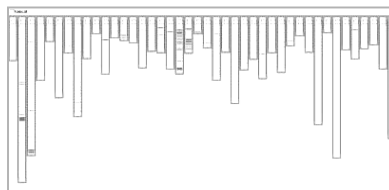


A OP=..

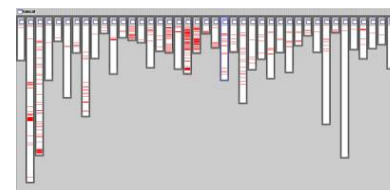
- Aspects are well-modularized crosscutting concerns
- Crosscutting concerns
 - have a clear purpose
 - define module boundaries, lines of data-flow, set of methods, points of resource utilization
- Weaving: add aspect functionality to existing component
 - hundreds of places changed at a time

Aspect

+



=



AOPexample

- Problem: track context changes (tomcat example)

ContextManager
addContext

BaseInterceptor
addContext

Each time ContextManager or BaseInterceptor receives an addContext call

Everything which happens in the class ContextManager

```
pointcut touche(ctx): (ContextManager | BaseInterceptor) &
    receptions( * addContext(Context ctx))

after trackCtxChanges: touche(ctx)
    { Logger.log("Context changed to" + ctx); }
```

Action to take every time a point defined by the crosscut ' touche' is reached

AOPtaxonomy

- Joinpoints=allrelevantpointsinthe executionofaprogram
- Pointcuts=anamedsetofjoin-points(S)
 - $S \& S, S | S, (S), !S$
- Aspect= pointcut +adviceaction
- Adviceaction=similartocomponentblock

Pointcut definitions

Some examples of primitive pointcuts:

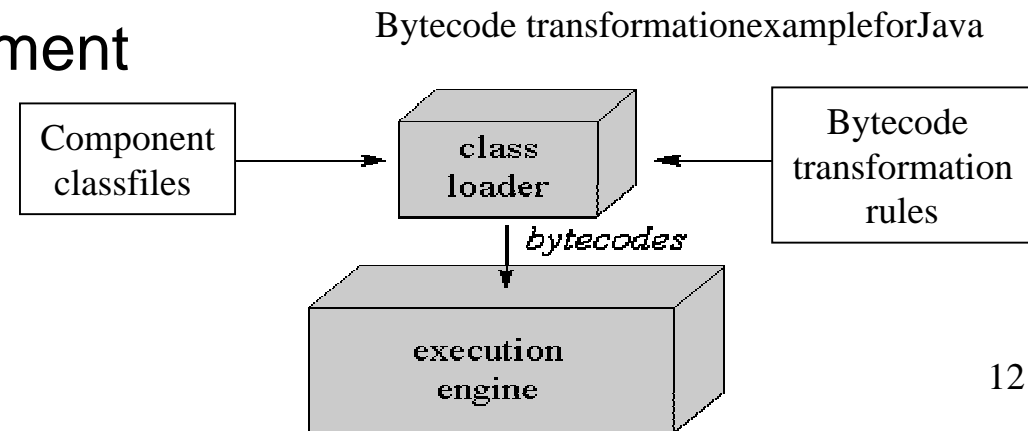
- `* addContext(Context)` matches `CtxMgr.addContext(Context)`
- `** (Context)` matches `CtxMgr.setContext(Context)` too
- `** (..)` matches all methods of all classes
- `public* (..)` matches all public methods
- `CtxMgr` matches everything which happens in the class `CtxMgr`

Action specifiers:

- `(around(before(normal execution)after)around)finally`

AOP Implementation

- Most implementations assert Java as component language
- Source preprocessing
 - semantically aware preprocessor(AspectJ[5], HyperJ[4])
- Object instrumentation
 - change object-code at load-time
 - e.g., exchange class-loader in the JVM (JOIE[6])
- Monitored runtime environment
 - Prose



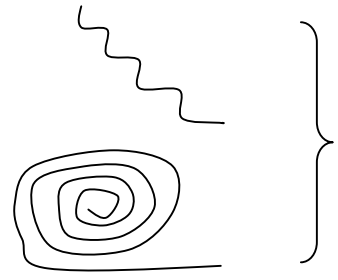
AOP usage

- AOP is used for:
 - synchronization (e.g., COOL[7])
 - logging, error handling
 - distribution concerns (e.g., RIDL[7])
 - contracts (pre- and post-conditions)
- ..and could be used for:
 - context sensitivity
 - transactional processing
 - join/setup/leave/teardown actions

When to use AOP

- *the lifecycle of a component* -

Aspects are recurrently used in this stage



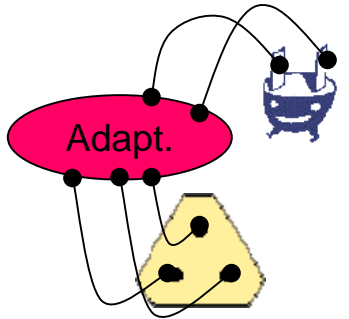
Design and implementation

release



delivery

late adaptations



Deployment environment

Deployment adaptations

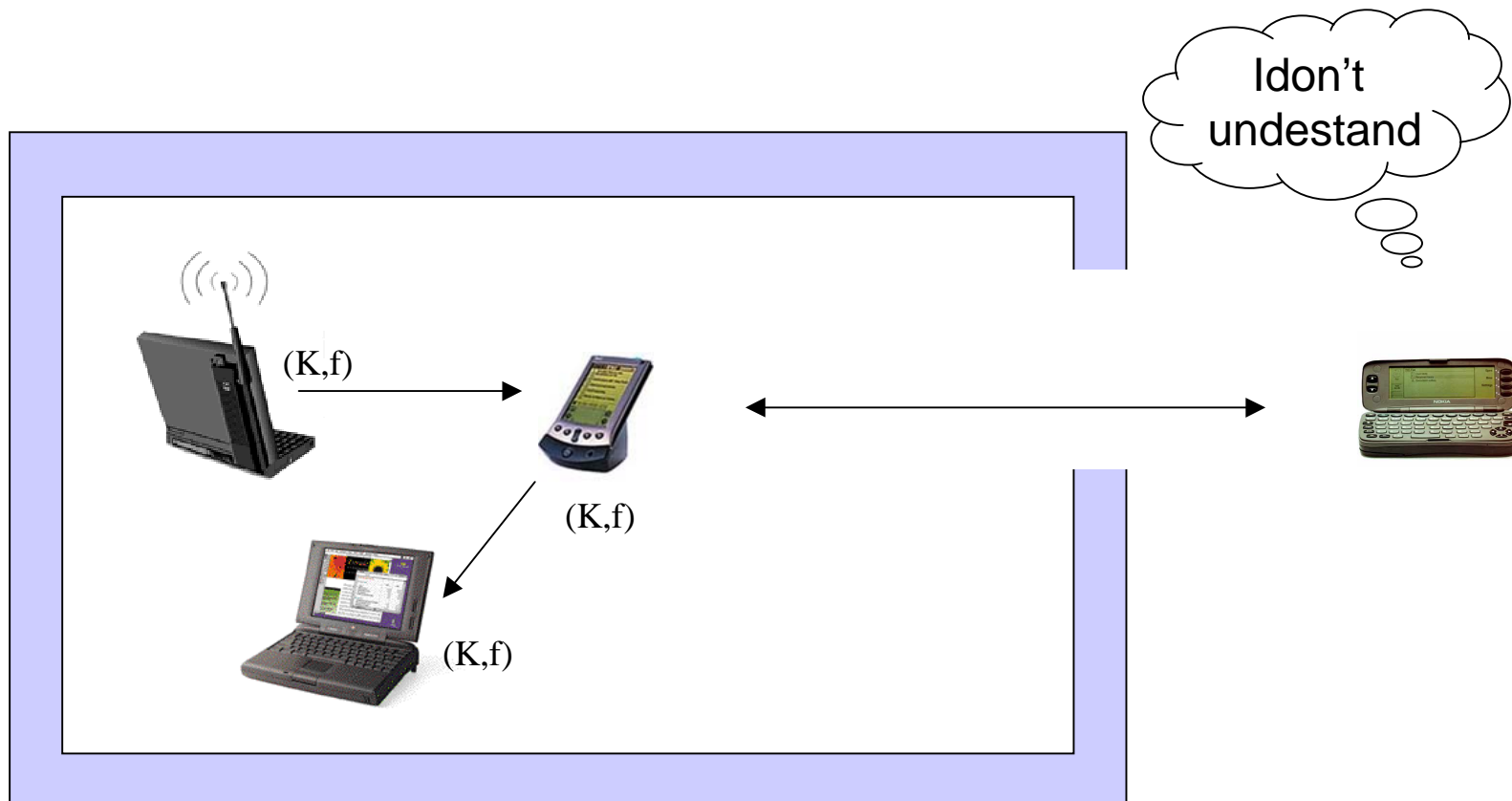
Change factors



AOP for late adaptations

- Late adaptations in response to environment changes
- Environment changes:
 - policy/organizational changes
 - access control, security, privacy
 - special case: information spaces, mobile computing
 - recurring deployment-like adaptations
 - service/usage-specific
 - principal initiating a service call
 - time and context of a service call
 - asynchronous environment changes
 - location, level of service, usage of system resources

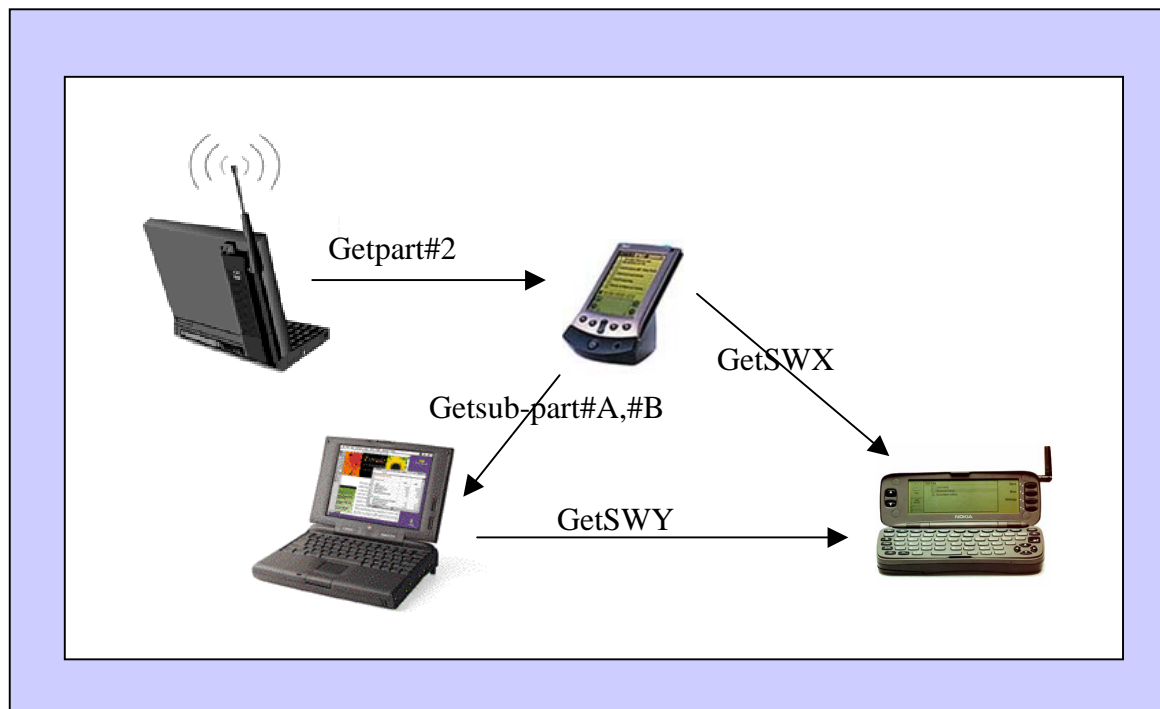
Scenarios like..(1)



Community specific keys, encryption algorithms and rules

Scenarios like..(2)

..an open-air fair-trade



- UploadMini-TM functionality in each node of the community
- Glue(normal) service calls with TM coordination
- Control resources

..by inserting a coordination aspect into the participant's component

Scenarios like..(3)



- A robotic environment:
 - remote controlled devices + sensors
- Consider JINI setup for LEGO Mindstorms
 - » [8,9]
- Weave an aspect into *all* proxies of *all* services that logs
 - what commands were issued, at what point in time, by whom
- Applications:
 - replay part of the history
 - query the past (is this a factoid aspect?)
 - perform inverse operations

A.O. Contexts

- Application-aware-context:
 - base station (associated to the context) uploads and weaves at runtime (dynamically) aspects into all components joining the environment
 - withdraw aspects at runtime

- Context awareness:

allows a mobile computing device to **adapt** to changing **environment** conditions

- Many of the current approaches :
 - intelligence (adaptation capability) located in the mobile system
 - context is passive (e.g., provides location-info)

Community-specific adaptations

- Community-specific adaptations
 - a group of nodes decides to consistently change its behavior (e.g., virtual community)
- AOP seem to be a good choice
 - consistent changes
 - do aspects have to be component-specific [11]?
- What's the competition doing?
 - design patterns for consistent changes (factory) (-)
 - replacement of implementation/libraries (--)

Relatedwork

- EnterpriseJavaBeans
 - deploymentadaptations
- Corba QOS
 - systempropertiesaroundfunctionalcalls[11]
- Meta-ObjectProtocols
 - openlanguagedefinitions(inpractice:extremely abstractandhardtounderstand)[10]
- Configurable/openoperatingsystems

Discussion

- Impact on application development?
- What adaptations are really orthogonal?
- How to deal with component-specific adaptations
- Security problems for run-time extensions?
- To what extent is AOP relevant for a world in which computer use tends to become pervasive?
- What about adaptations of (dumb) devices?

Who'swho

- Xerox Parc: AspectJ
 - Kikzales,Lopes
- U.of Twente:compositionfilters(Sina)
 - Aksit etal.
- IBM: HyperJ/multidimensional sep.ofconcerns
 - Osherr, Tarr
- N.E.U:adaptiveapplications
 - Lieberherr

References

- [1] Szypersky C: *Component Software, Beyond Object-Oriented Programming*, Addison-Wesley, 1997,
- [2] P. Tarr, H. Osherr, W. Harryson, S. Sutton: *N Degrees of Separation: Multi-Dimensional Separation of Concerns*. Proceedings of the 21st International Conference on Software Engineering, May 1999
- [3] [http:// aspectj.org/documentation/papersAndSlides/OOPSLA-2000-demo_files/frame.htm](http://aspectj.org/documentation/papersAndSlides/OOPSLA-2000-demo_files/frame.htm)
- [4] <http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm>
- [5] <http://www.aspectj.org>
- [6] *The Java Object Instrumentation Environment* -- <http://www.cs.duke.edu/ari/joie/>
- [7] *D: A Language Framework for Distributed Programming Technical Report*, Xerox Palo Alto Research Center, Number SPL97-010, P9710047, February 1997.
- [8] <http://www.legominstorms.de>
- [9] Jan Newmarch: *Jini and Mindstorms*, www.canberra.edu.au/java/mindstorms
- [10] Gregor Kikisales: *The Art of Meta-Object Programming*
- [11] J. Zinky, D. Bakken, R. Schantz: *Architectural Support for Quality of Service of CORBA Objects*. Theory and Practice of Object Systems, April 1997

Theend ?

AspectJCaseStudy

Ad-HocAccessControlonPrinter
Services