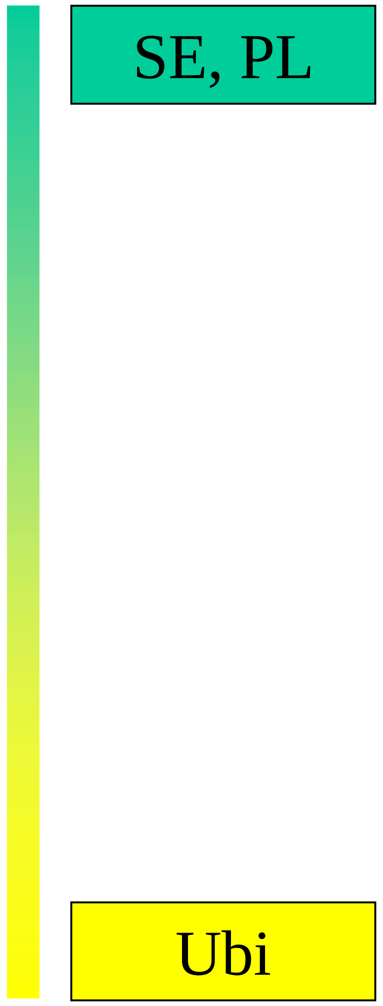
The Impact of Aspect-Oriented Programming on Future Application Design

Andrei Popovici
Information and Communication Research Group

IS Seminar 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 components lifetime
  - community-specific adaptations
  - application-aware environments (or contexts)
- Related
  - research areas and CS topics



The software world

- Component:
  - – unit of independent deployment
  - – has no persistent state
  - – subject to 3rd party composition
    - » [1]
- Object:
  - – unit of instantiation
  - – encapsulates state & behavior
  - – unique identity

Decomposing components

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

Deployment $\in$ Component Lifecycle

Preamble

Aspect ~ ad specere (lat.)
- to look at
- particular appearance to eye or mind

1. Software world, Component

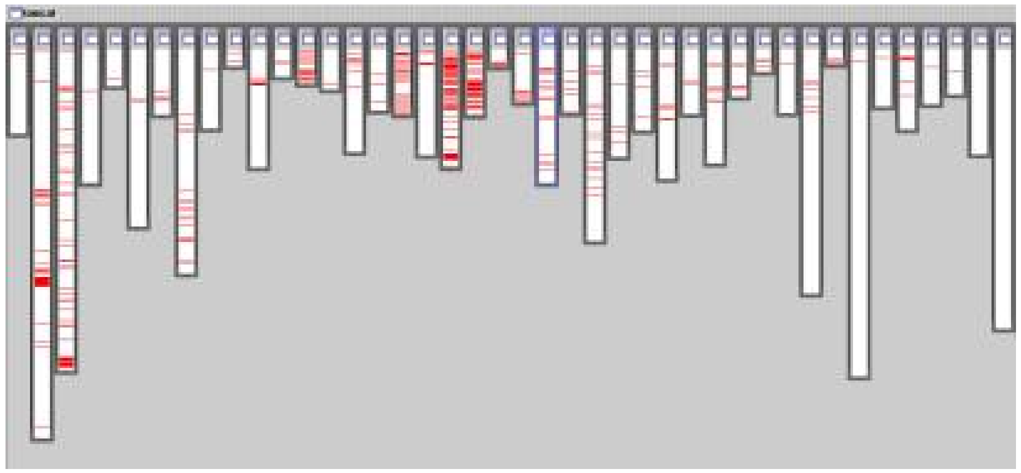
2. Software aspect

3. Component lifetime

What is the <u>real world</u> security-<u>aspect</u> of <u>my life</u>?
- 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 ..

Aspects motivation

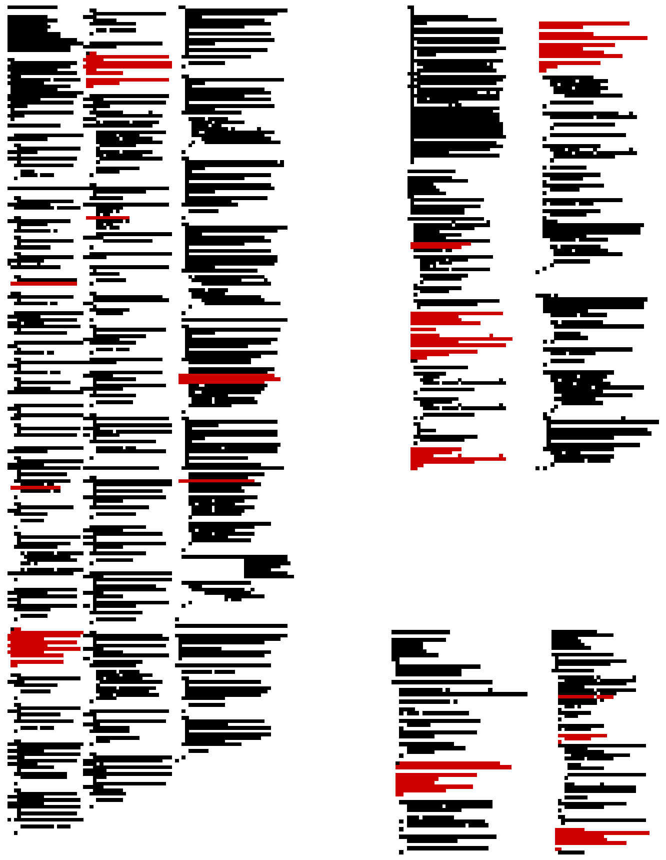
- tyranny of the main decomposition unit (OO)
- Some concerns cannot be expressed in a modular way
  - Where is logging in `org.apache.tomcat`?



- Not in one place
- Not even in a few
- Not in related places

Cost of tangled code

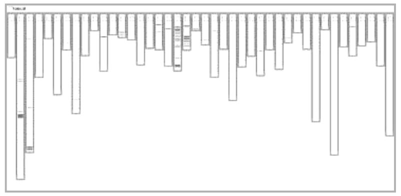
- Redundancy
  - same fragment in more places
- Difficult to reason about
  - the big-picture of tangling is not clear
- Difficult to change
  - have to find all the code involved
  - the to be sure to change it consistently
      - » [3]



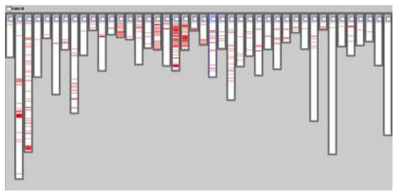
A O P = ..

- 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 +



=



AOP example

- 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

AOP taxonomy

- Joinpoints = all relevant points in the execution of a program
- Pointcuts = a named set of join-points (S)
  - S & S, S | S, (S), !S
- Aspect = pointcut + advice action
- Advice action = similar to component block

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*

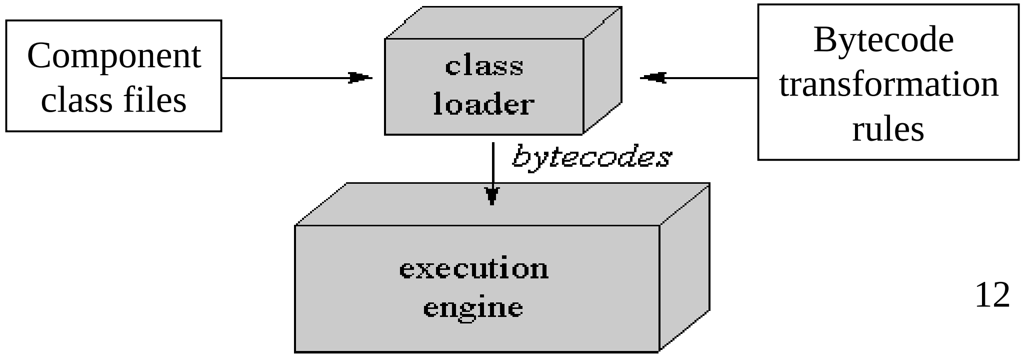
Action specificators:

– (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

Bytecode transformation example for Java

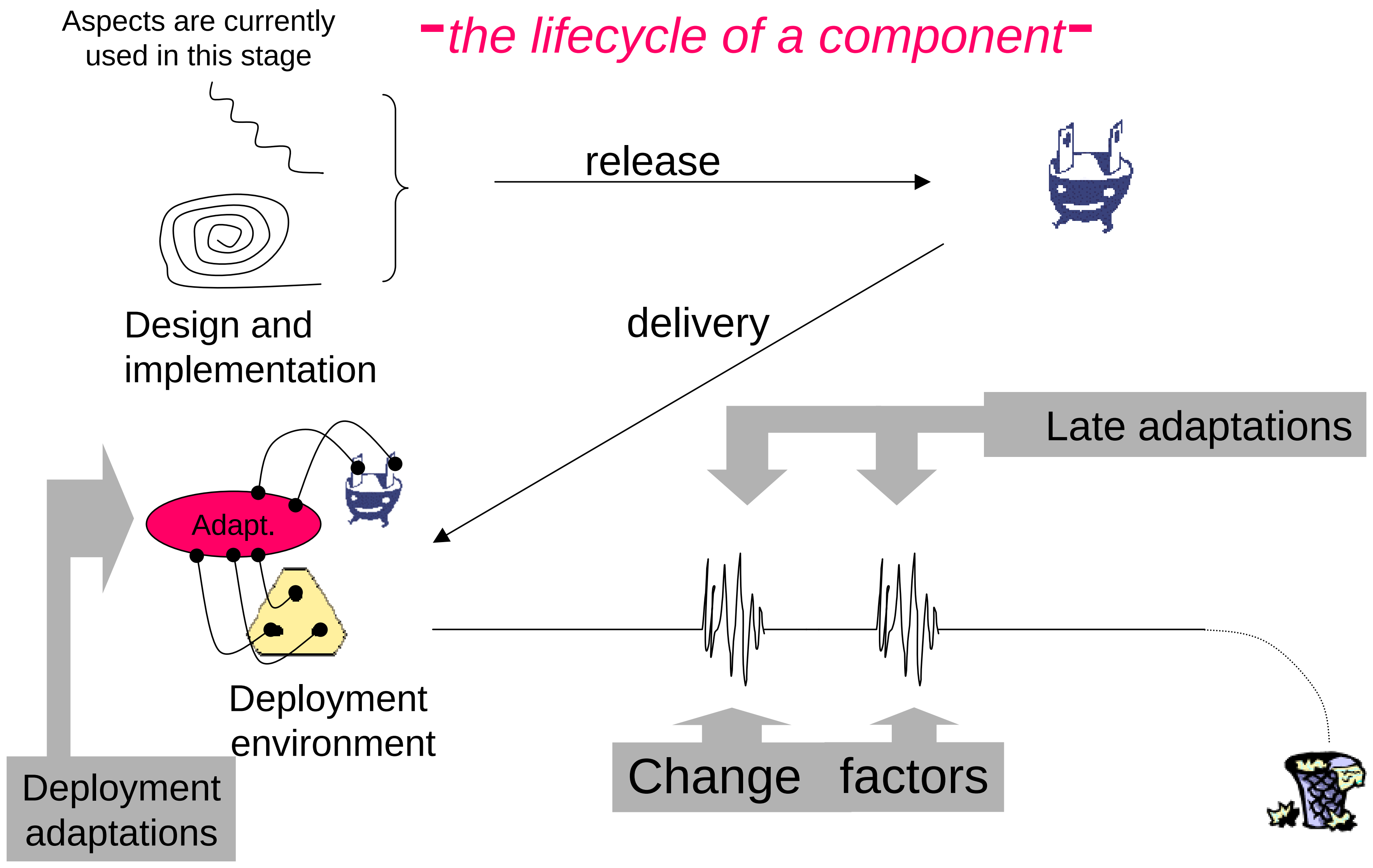


AOP usage

- 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 sensitiveness
  - transactional processing
  - join/setup/leave/teardown actions

When to use AOP

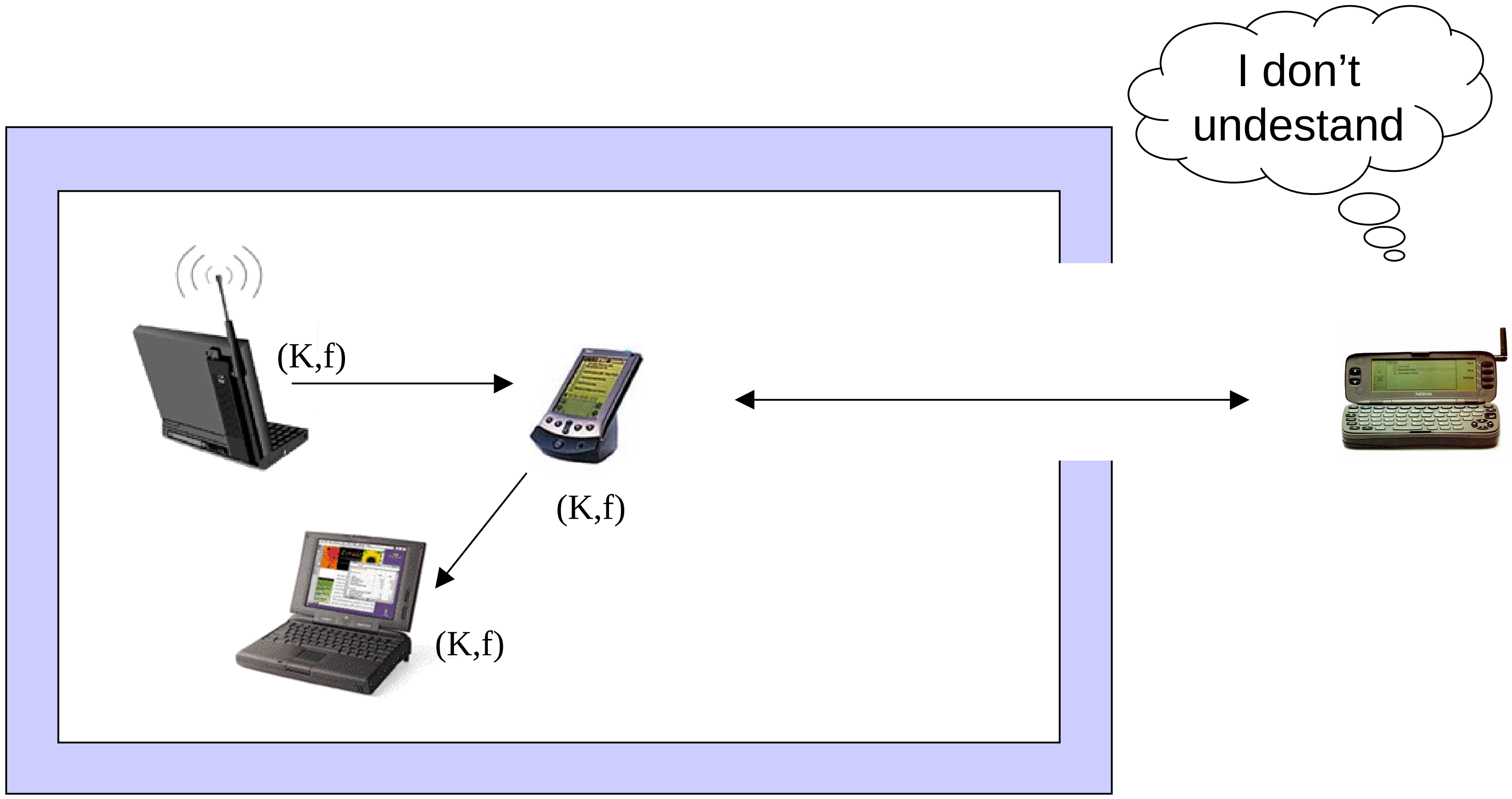
-the lifecycle of a component-



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

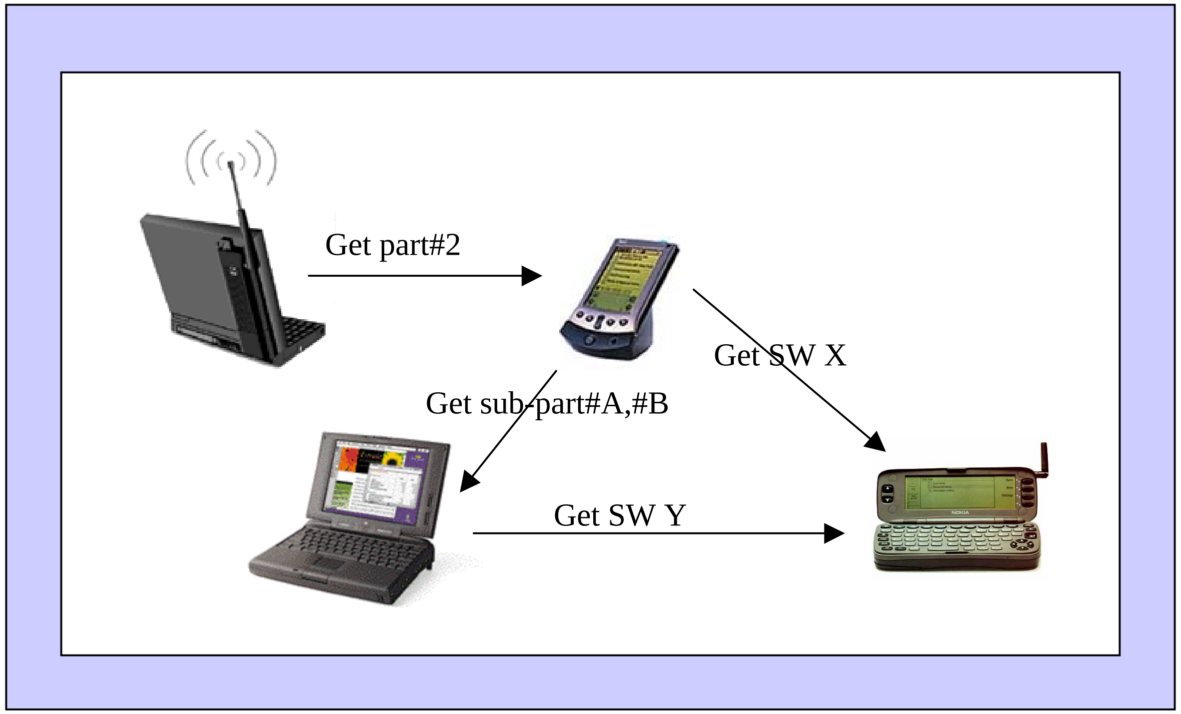
Scenarios like .. (1)



Community specific keys, encryption algorithms and rules

Scenarios like.. (2)

..an open-air fair-trade



- Upload Mini-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 parts 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 decides to consistently change its behavior (e.g., virtual community)
- AOP seems 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 (--)

Related work

- Enterprise Java Beans
  - deployment adaptations
- Corba QOS
  - system properties around functional calls [11]
- Meta-Object Protocols
  - open language definitions (in practice: extremely abstract and hard to understand)[10]
- Configurable/open operating systems

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’s who

- Xerox Parc: AspectJ
  - Kikzales, Lopes
- U. of Twente: composition filters (Sina)
  - Aksit et al.
- IBM: HyperJ/multidimensional sep. of concerns
  - Osherr, Tarr
- N.E.U: adaptive applications
  - 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
- [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 Kiksales: *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

The end?

AspectJ Case Study

Ad-Hoc Access Control on Printer Services