Expressive distributed and concurrent aspects

Mario Südholt
http://www.emn.fr/sudholt

OBASCO project
École des Mines de Nantes-INRIA, LINA

Microsoft, 30 Oct. 2006

Outline

1. Motivation
   - Expressive aspects
   - Aspects for non-sequential systems
2. Aspects with explicit distribution (AWED)
   - Transactional replicated caching
   - Language
   - Prototype implementation
3. Concurrent Event-based AOP (CEAOP)

A simple AspectJ example

```
aspect UpdateSignaling {
  pointcut change():
    execution(void Point.setX(int))
    || execution(void Point.setY(int))
    || execution(void Shape+.moveBy(int, int));
  after() returning: change() {
    Display.update();
  }
}
```

AspectJ characteristics

- **Pointcuts**: match execution events (**joinpoints**)
  - Mostly atomic: denote (sets of) individual execution events
    Exception `cflow`
- **Advice**: mainly full Java, `proceed`
- **Aspects**: extensive use of internal and base-program state
- Execution-related vs. static pointcuts (Inter-type declarations)
- Aspect **composition**: necessary for resolution of interactions
  AspectJ: only ordering by `dominate`

- Aspect **instantiation**: coarse-grained

The case for AOP

- Crosscutting is a **real problem**
- **Not tractable by traditional means** (OOP, components, etc.), at least without serious re-architecturing
- Corresponding programming tasks are less explicit/**done manually without AOP**

The case against AOP

- AOP as a **low-level transformation** system
- **Beyond logging**, tracing, and monitoring?
- **Semantics**? (In)formal **property analysis/verification**?
  - Properties involve base program, pointcuts, advice, non-local state of aspects and base program
- One reason: reliance on **atomic pointcuts** (e.g. AspectJ's)

Expressive aspect languages: towards robust AOP

- Make **explicit relationships** between execution events
  Eliminate use of non-local state
- Means
  - **Richer pointcut languages**
    Regular aspects, temporal logic-based aspects, logic pointcuts, etc.
  - **Domain-specific** sublanguages in pointcuts and advice

Aspects and non-sequential systems

- **Large applications**
- Large number of **different concerns** (distribution, persistence, transactions, etc.)
- **Crosscutting equally important** in concurrent and distributed systems than in sequential ones

State-of-the-art

- Surprisingly **few work**
- Predominant approch libraries, etc.
  - **Sequential AOP system with existing infrastructure** for distribution, concurrency
  - Ex.: Spring AOP, JBoss AOP
- Atomic pointcuts, no distribution-centric abstractions: subject to **same expression and correctness problems** than sequential AO programs

Outline

Transactional replicated caches

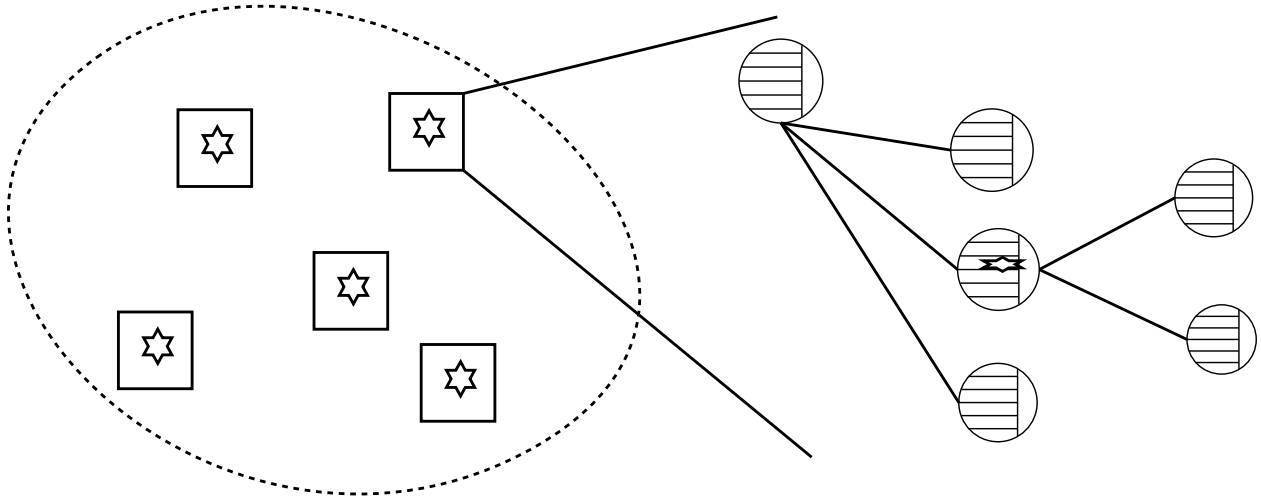


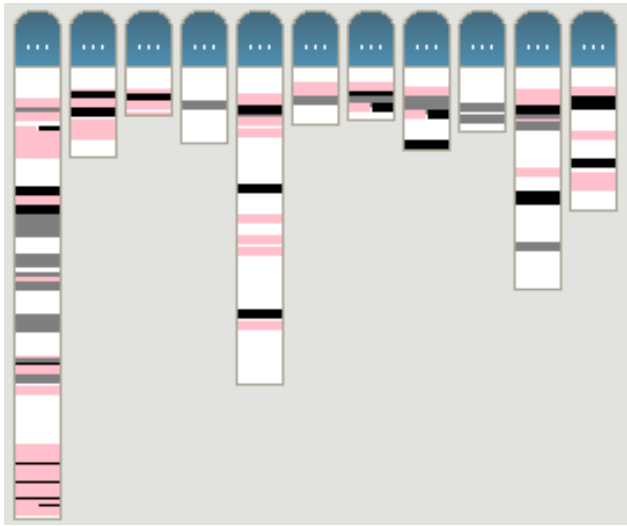
Figure: a) Replicated Caches b) Zoom of Data structure

- Cache **data structure** deployed on each node
- Data **replication** under control of **transactions**

Ex.: JBoss cache (version 1.2)

JBoss cache:

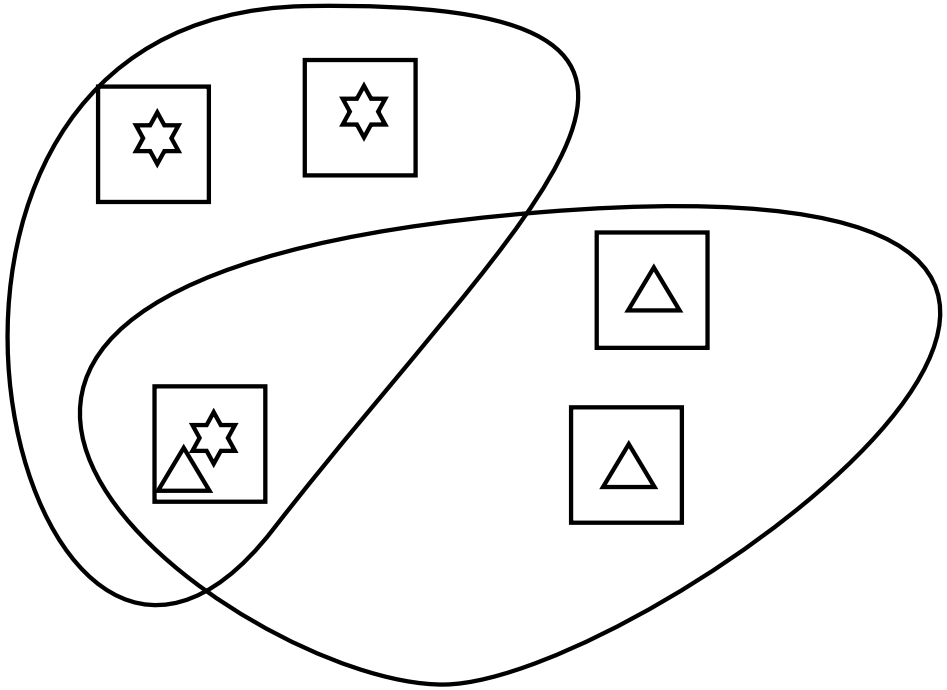
- Crosscutting concerns: **replication, transactions, interception filter**



Problems:

- **Refactor** replication of transactions
- **Extend** replication policy

Ex.: support cache evolution



- **Don’t replicate unnecessarily** huge objects
- Replicate only in case of **interest**

Modularization of distribution concerns

Distribution-specific aspect abstractions:

- Detection of remote events
- Remote execution of code
- Support for distributed state
- Distributed deployment of code

AWED language

- **Remote pointcuts**
  - References to remote hosts: `host`, `on`
  - Sequence pointcuts: `seq`, `step`
- **Remote advice**
  - Asynchronous and synchronous execution: `syncex`
  - Synchronization between interacting advice using futures
- **Distributed aspects**
  - Deployment: `single` and `all`
  - Instantiation: *e.g.*, `perthread`, `perbinding`...
  - State sharing: *e.g.*, `global`, `group(Group)`
- See [Benavides et al., AOSD'06], [Benavides et al., DOA'06]

Remote pointcuts examples

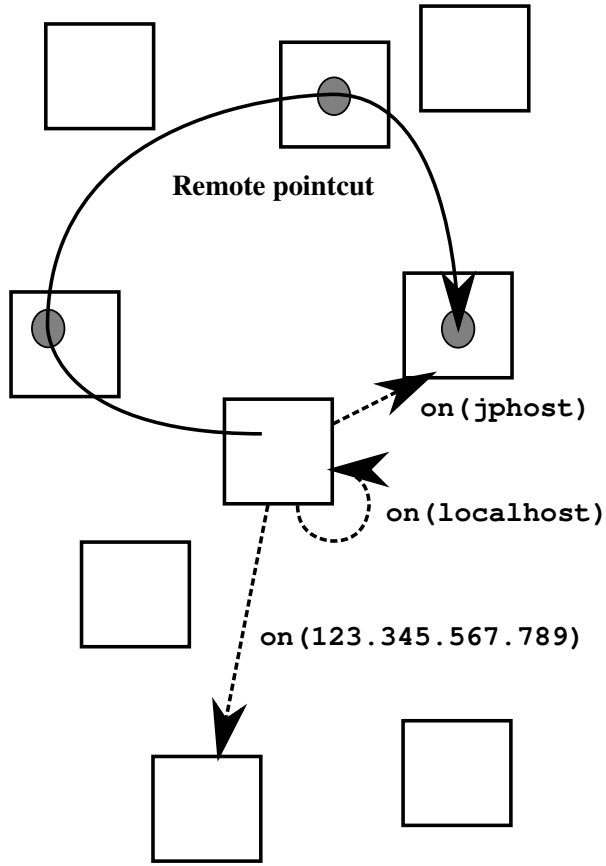
Replication pointcuts for a replicated cache application:

- Using the `host` pointcut:

  call(* Cache.put(Object,Object))
  && !**host**(**localhost**)

- Using the `on` pointcut:

  call(* Cache.put(Object,Object))
  && !**on**(**jphost**)



AWED sequence examples

Replication protocol for a lazy replicated cache (delimit via start/stop)

pointcut replPolicy(String key, Object o):
  replS: **seq**(s1: startCache() $\rightarrow$ s3 $\|$ s2,
    s2: cachePut(key, o) $\rightarrow$ s3 $\|$ s2,
    s3: stopCache() $\rightarrow$ s1)

pointcut putVal(String key, Object o):
  step(replS, s2) && **args**(key, o)

Remote Advice

- 2 synchronization **modes**: a/synchronous
  Access to result managed using **futures**
- Management of **groups**
- Ex: replication advice:

```
before(String k, Object o):
    localCachePut(k, o){
        addGroup(k);
        proceed();
}
```

Ex.: lazy replication aspect

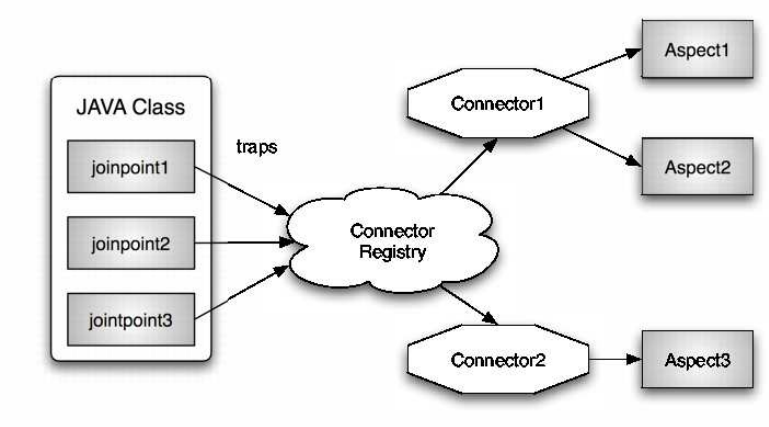
```
all aspect CacheReplication{
  pointcut cachePcut(Object key, Object o):
      call(* Cache.put(Object, Object))
      && args(key,o) && !on(jphost)
      && !within(CacheReplication);

  before(String k, Object o):
      cachePcut(k, o) && on(k){
          Cache.getInstance().put(k, o);
  }
}
```

- Aspect deployed on all hosts
- Matches `put` method in all remote hosts in the group of interest of value k
- Replicate call only if interest in that value

Implementation basis: JAsCo Infrastructure

- Tool from SSEL group at Free University of Brussels
- **Dynamic** aspect weaver for **Java**
- Supports **sequence pointcuts**
- **Connector registry** for managing aspect compositions



DJAsCo Infrastructure

- One connector registry per host
- Aspects/Connectors registered in connector registries
- **Joinpoints** propagated among connector registries

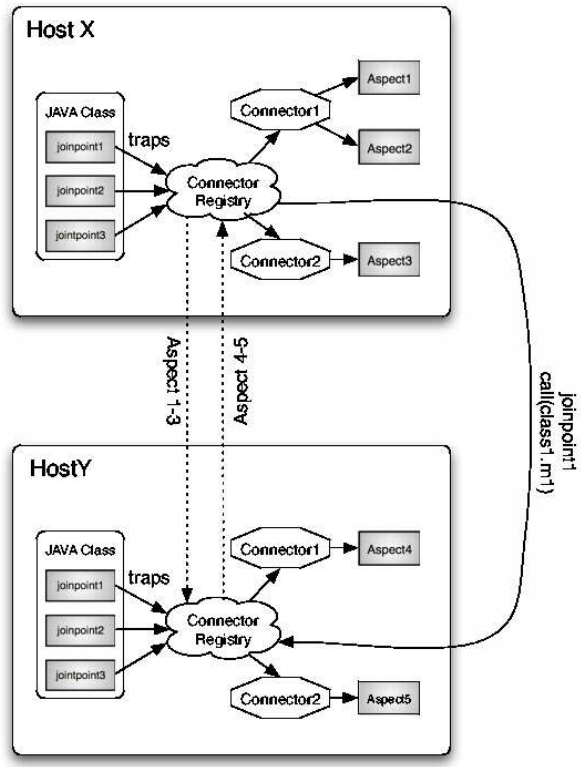


Figure: DJAsCo architecture.

Overview of implementation of features

- Remote pointcuts: JP propagation using **JGroups**
- Cflow: **customized sockets**
- Remote Sequences: extension of **JAsCo sequences**
- Remote advice: Based on **activation of deployed aspects**
- Aspect distribution: **Connector distribution** using JGroups
- Aspect state sharing, parameter passing: **AWED aspects**

On-going applications

- **Web services** [Benavides et al., DOA'06]
  - Distributed web services composition
  - Use AWED for client-side concern integration
  - Cooperation with SSEL group from VUB
- **Toll systems**
  - Use aspects for dynamic adaptation of server-side and client-side code
  - Cooperation with Siemens AG, Munich, Germany

Outline

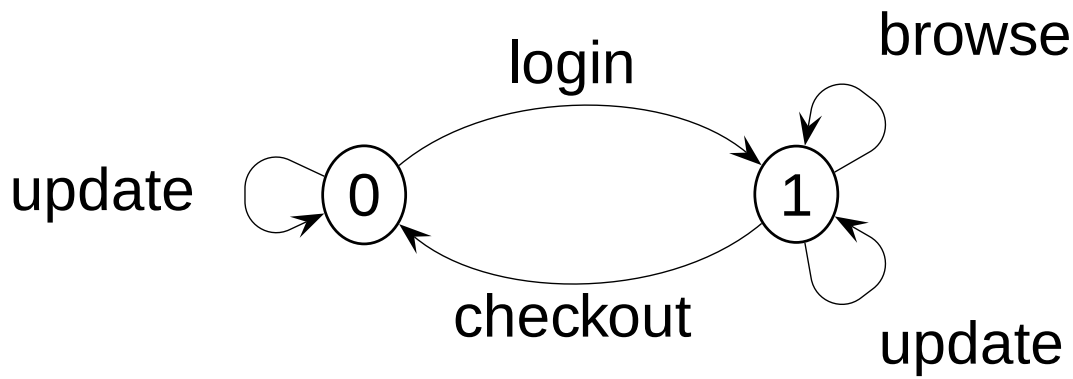
Aspects for concurrency

- Few previous approaches
  - Christa Videira Lopes (now UCLA): COOL
    Aspects define mutual exclusion relationships on base methods
  - James H. Andrew (U. Waterloo): aspects as concurrent processes
  - Use of **sequential AO systems** with concurrency libraries
- Problems
  - Several aspects not considered
  - No aspect composition

Concurrent Event-based AOP (CEAOP)

- Definition of base and aspects as **Finite State Processes** (FSP)
- Pointcuts: FSP expressions
- **Synchronization in terms of the aspect structure**:
  - Advice structure: before - **proceed/skip** - after
  - Aspect composition at one execution event
    - Synchronize on **proceed/skip**
    - Execute before, after parts sequentially or concurrently
- **Composition operators** for flexible synchronization
- **Formal weaver definition**
- **Property verification** using Labelled transition system analyzer (LTSA)
- **Prototype** implementation in Java
- See [Douence et al., GPCE'06]

Ex.: model of simple e-commerce program



$$
\begin{array}{lll}
(0) \quad \text{Server} & = & \text{login} \rightarrow \text{InSession} \\
 & | & \text{update} \rightarrow \text{Server}, \\
(1) \quad \text{InSession} & = & \text{checkout} \rightarrow \text{Server} \\
 & | & \text{update} \rightarrow \text{InSession}, \\
 & | & \text{browse} \rightarrow \text{InSession}.
\end{array}
$$

Aspect interaction and synchronization

- Ex. consistency aspect: suppress updates (price changes) in sessions

  $$\mu a.\,(\textbf{login};\,\mu a'.\,((\textbf{update} \triangleright \textbf{skip}\ \ \textbf{log};\,a') \,\Box\, (\textbf{checkout};\,a)))$$

- Ex. safety aspect: rehash and backup views after updates

  $$\mu a''.(\textbf{update} \triangleright \textbf{rehash proceed backup};\,a'')$$

- **Interactions on common execution events**
  Here: **update** events within sessions
- **Composition: synchronize parts of advice** in case of interactions
  Here: backup and logging can be executed concurrently

Instrumentation

- Instrument aspects and base program with **synchronization events**
- Goal: compose aspects and base with common parallel composition
- Ex.: Instrumentation of consistency aspect

$$
\begin{aligned}
a = (\ & login \rightarrow a' \\
& \mid eventB\_update \rightarrow proceedB\_update \rightarrow proceedE\_update \rightarrow eventE\_update \rightarrow a \\
& \mid checkout \rightarrow a \mid browse \rightarrow a\ ),
\end{aligned}
$$

$$
\begin{aligned}
a' = (\ & eventB\_update \rightarrow skipB\_update \rightarrow skipE\_update \rightarrow log \rightarrow eventE\_update \rightarrow a' \\
& \mid checkout \rightarrow a \\
& \mid browse \rightarrow a' \mid login \rightarrow a'\ ).
\end{aligned}
$$

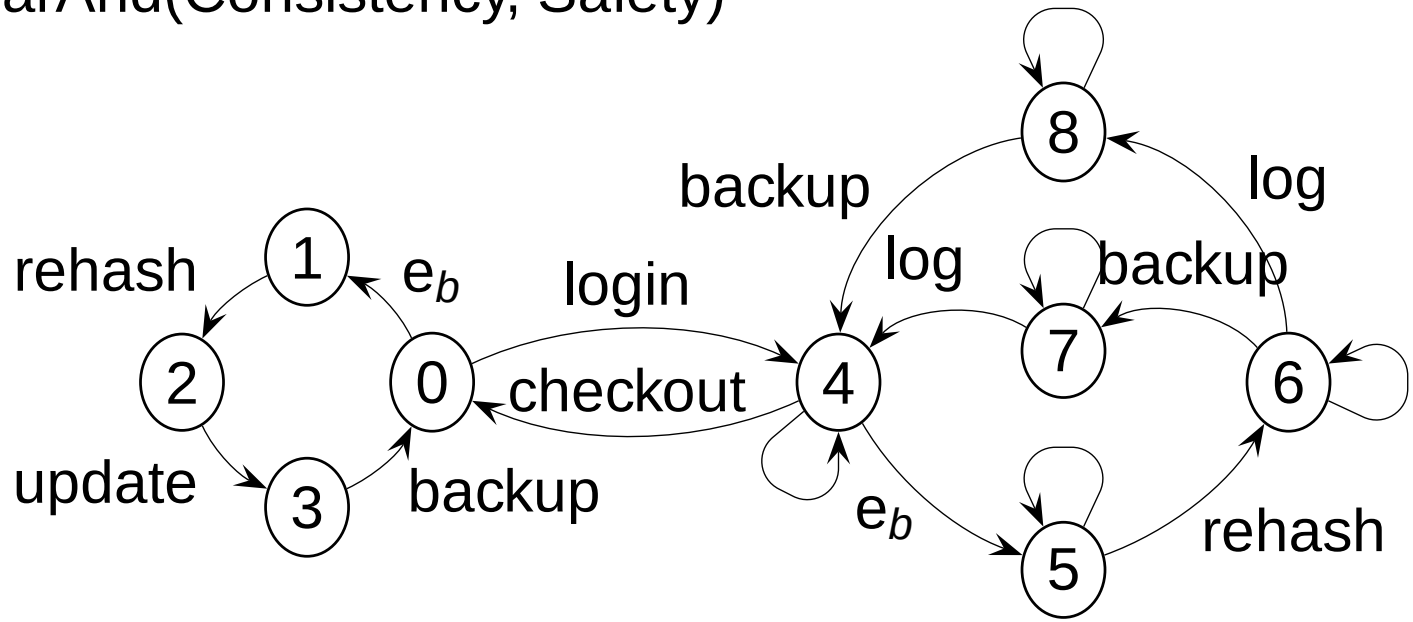
Composition operators: ex. ParAnd

- Concurrent before, concurrent after, proceed base iff both aspects proceed
- Def. (event renaming missing)

(skipB_e1 → (skipB_e2 → skipB_e → skipE_e → skipE_e1 → skipE_e2 → ParAnd
| proceedB_e2 → skipB_e → skipE_e → skipE_e1 → proceedE_e2 → ParAnd)

| proceedB_e1 → (skipB_e2 → skipB_e → skipE_e → skipE_e2 → proceedE_e1 → ParAnd
| proceedB_e2 → proceedB_e → proceedE_e →
proceedE_e1 → proceedE_e2 → ParAnd)).

- Ex.: ParAnd(Consistency, Safety)



Properties

- Safety properties
- Absence of deadlock
- Liveness
- General properties of operators, such as associativity

Conclusion

- Expressive aspects as means to tackle drawbacks of AOP
- Distributed and concurrent applications are subject to numerous crosscutting concerns.
- AOP should be useful for their modularization
- Currently, few approaches: no domain-specific support, correctness difficult to evaluate
- AWED as model for distributed AOP
  Remote pointcuts, remote advice, distributed aspects
- CEAOP
  Synchronization among aspects and base, composition operators, tool-based property support

Future work

- AWED
  - Composition operators
  - Formal semantics, property support
  - Introduce causality guarantees for distributed aspects
- CEAOP
  - Extend use of composition operators
  - Efficient implementation in mainstream languages

Further information

- `http://www.emn.fr/sudholt`
- AWED: `http://www.emn.fr/x-info/lbenavid/awed.html`
- CEAOP: `http://www.emn.fr/x-info/eaop/ceaop.html`