Distributed programming with aspect languages

Mario Südholt

http://www.emn.fr/sudholt

ASCOLA research group
École des Mines de Nantes, INRIA, LINA

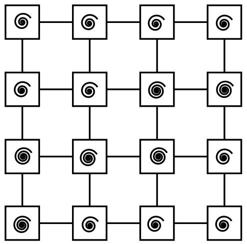
MSc Alma, module CGA, 24 Sep. 2018

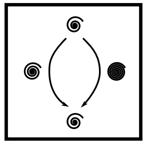
Outline

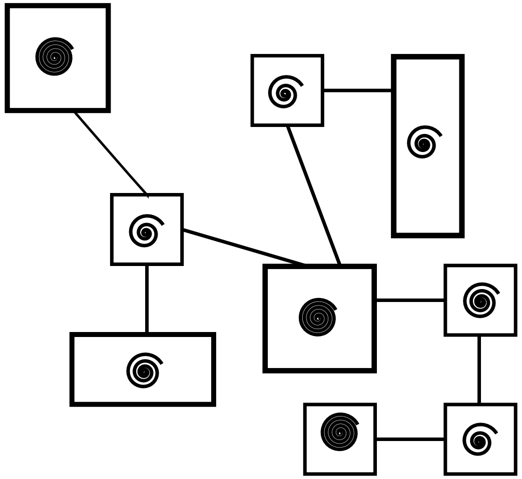
Beyond sequential programs: problems

- Improve **execution speed** of resource-intensive calculations
  E.g.: weather simulations
- Improve **reactivity** of interactive programs
  E.g.: graphical user interfaces
- Improve **resource sharing**
  E.g.: sharing of physically distributed printers

Beyond sequential programs: programming paradigms







- **Massively parallel programs**: perform similar calculations on large number of similar (equal) processors
- **Concurrent programs**: multiple executions in an interleaved fashion on one machine
- **Distributed programs**: coordinate calculations on a, possibly heterogeneous, network that links, possibly very different, machines, OSs . . .

Basic goals of distributed programming

Heart of the story: **hiding** of

Basic goals of distributed programming

Heart of the story: **hiding** of

- **Access** to distributed data
- **Locations** of distributed entities
- **Migration** and **relocation** of distributed entities
- **Concurrent** execution
- Distribution-related **failures**

Basic goals of distributed programming

Heart of the story: **hiding** of

- **Access** to distributed data
- **Locations** of distributed entities
- **Migration** and **relocation** of distributed entities
- **Concurrent** execution
- Distribution-related **failures**

Complete hiding of distribution issues is unreasonable:
remote method calls, e.g., inherently different from local ones

$\Rightarrow$ **Explicitely distributed programming**

Concrete issues

- **Sequential execution** vs.
  concurrency on one machine vs.
  parallel execution on several processors/machines
- Execute activities on **multiple processors/machines**
  - Access to **distributed state**
- **Synchronous** (blocking) vs. **asynchronous** (non-blocking) **communication**
- **Error handling** of distribution, communication and concurrency failures

Distributed and concurrent applications

Typical examples

- Graphical user interfaces
- Peer-to-peer systems
- Web-based/enterprise information systems

Distributed and concurrent applications

Typical examples

- Graphical user interfaces
  $\rightarrow$ concurrent applications
- Peer-to-peer systems
  $\rightarrow$ concurrent and distributed
- Web-based/enterprise information systems
  $\rightarrow$ concurrent and distributed

Distributed applications typically subject to concurrency issues

Languages vs. library-based approaches

Languages (this part)

- Provide language mechanisms for the expression of distributed aspects
- \+ Concise expression
- \+ Better readability and understandability
- – Potentially difficult integration with existing languages
- – Learning curve

Languages vs. library-based approaches

Languages (this part)

- Provide language mechanisms for the expression of distributed aspects

+ Concise expression
+ Better readability and understandability
– Potentially difficult integration with existing languages
– Learning curve

Libraries, frameworks, APIs (second part: Prof. Wouter Joosen)

- Design library (OO framework) representing and implementing distributed aspects

– More verbose and expression
– Readability and understandability
+ Facilitates integration with existing languages
+ Learning curve

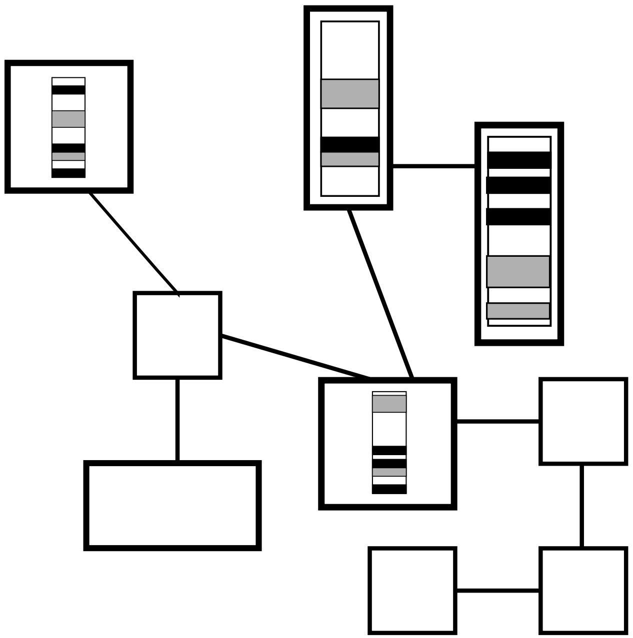
Outline

Crosscutting and distributed applications

- **Are there crosscutting concerns in distributed applications**?

Crosscutting and distributed applications

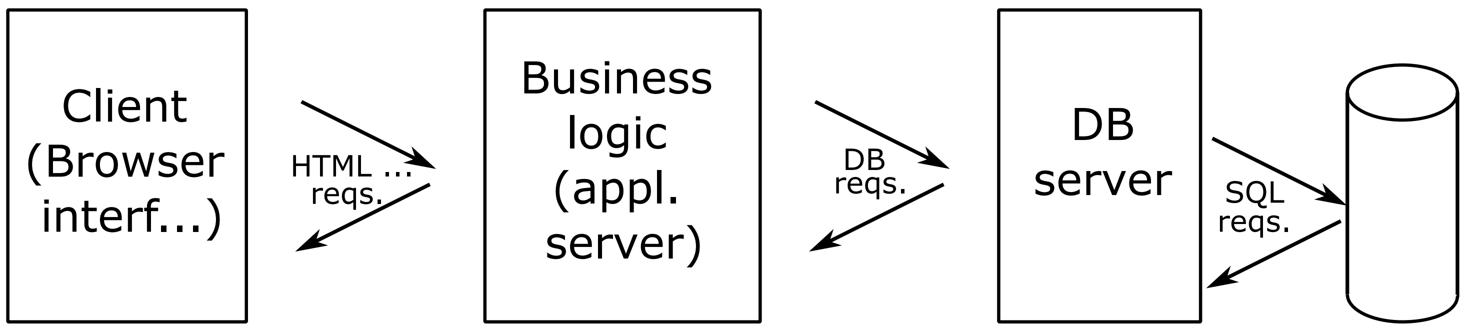
- **Are there crosscutting concerns in distributed applications**?



- YES! **Large number of different concerns**
  - Distribution, synchronization, coordination
  - Security
  - Transactions
  - Persistence
  - Testing
  - ...

Ex.: EJB-based application servers

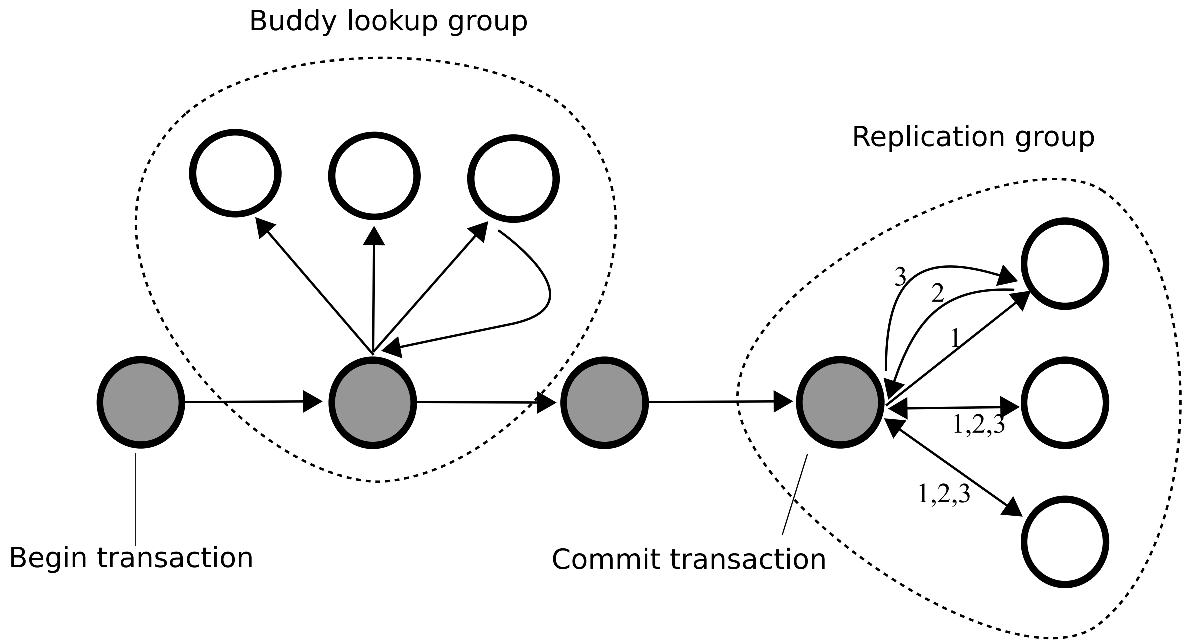
- Enterprise JavaBeans (EJB): **industrial (distributed) component model** defined by Sun Microsystems (`http://java.sun.com/products/ejb`)
  - **3-tier** distributed systems: distributed database, business logic, user-level presentation layer
  - 3 crosscutting functionalities: **persistance**, **transactional behavior**, **security**
  - Specification (ver. 2): ˜550 pages (150: components, 400: crosscutting functionalities)



Ex.: JBoss application server, JBoss Cache

- **Application servers**: development and execution platform on top of such component models
  - Ex. **JBoss** application server (`jboss.com`)
- JBoss Cache: **Data replication under transactional control**
- **Replication**: provide copies of a set of data on all machines
- **Transactions**: manage (concurrent) changes on each machine
  1. Do local changes (from begin to end of transaction)
  2. Globally coherent changes: success (transaction committed) incoherent changes: failure (transaction aborted)
  - All or nothing model: either commit all local changes or abort all changes and return to previous state

(Abstract) runtime architecture



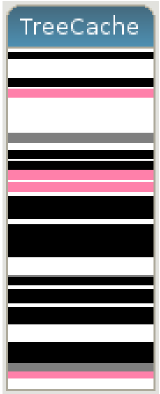
(Simple) composition of 3 standard parallel patterns

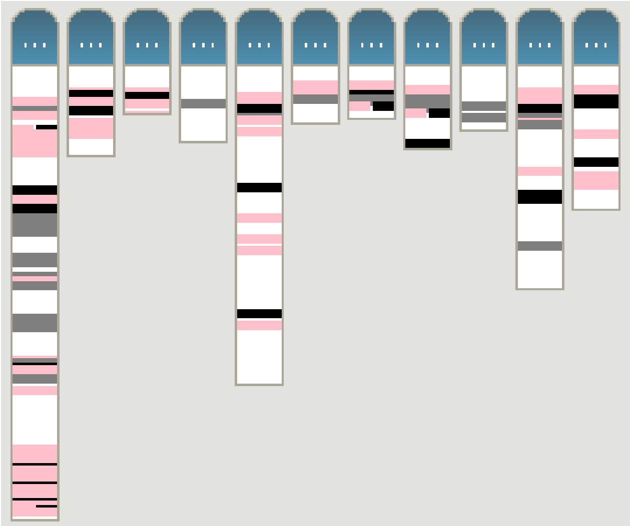
1. Transaction (pipe pattern)
2. Get data during transaction (farm)
3. Transaction (2-phase) commit protocol (farm and gather)

Crosscutting in JBoss Cache **Benavides et al., AOSD'06**

- Representation of part of codebase (JBoss Cache 1.2)
  - Class `TreeCache`: main data structure
  - `interceptor` package: modularization of replication and transaction code
- Crosscutting concerns:

  replication (black),
  transactions (dark gray),
  interception filters (light gray)





- Class `TreeCache`: 1741 lines of code (comments incl.)
- Replication code: $\geq$ 196 LOC
- Transactions: $\geq$ 228 LOC

Resulting problems

- What about
  - **nicely modularizing** transactions and replication code?
  - **adding** new replication policies?
- Requires **different modifications at multiple places** among the 424 LOC concerning replication and transactions
- Involves modifications involving **different distributed entities**, in particular machines

Outline

Overview of approaches for distributed AOP

- Distributed applications are a main target of AOP but **surprisingly few approaches**
- Most approaches **apply sequential AOP to distributed infrastructures**
  - Java RMI and AspectJ **Soares et al., OOPSLA'02**
  - Manipulation of IBM's EJB-based application server Websphere using AspectJ **Colyer, Clement, AOSD'04**
  - Frameworks for sequential AOP over distributed infrastructures: small language-level support
    - Spring AOP
    - JBoss AOP
    - AspectJ2EE **Cohen, Gil, ECOOP'04**
    - see Prof. Wouter Joosen's part
  - ...

Sequential AOP and distributed infrastructures

Ex.: AspectJ and RMI **Soares et al., OOPSLA'02**

Server-side distribution

```
aspect ServerSideHWDistribution {
  public static void
      HWFacade.main(String[] args) {
    try {
      HWFacade f = HWFacade.getInstance();
      UnicastRemoteObject.exportObject(f);
      java.rmi.Naming.rebind("/HW",f);
    } catch (Exception rmiEx) { ... }
  }
  ...
}
```

- Register server with naming service

Client-side distribution

```
aspect ClientSideHWDistribution {
  pointcut facadeCalls(HWFacade f):
    target(f) && call(* *(..))
      && !call(static * *(..))
      && this(HttpServlet);

  int around(Complaint c) throws /*...*/:
      facadeCalls() && args(c) &&
      call(int registerComplaint(Complaint)) {
    return remoteHW.registerComplaint(c);
  }
}
```

- Redirect client facade calls to server

- No modular definition of distributed functionalities

Challenges

- No explicit support for **concurrent** and **distributed** abstractions
- Each aspect only treats parts of crosscutting functionality of distributed applications
  - Ex.: separate aspects for client and server sides in client-server architectures
    Soares et al., OOPSLA’02
  - Ex.: multiple aspects for handling of distributed transactions

Approaches for AOP with explicit distribution

(Almost) exhaustive list of approaches:

- **Riddle Lopes'97**: argument passing modes
- **JAC Pawlak et al., Reflection'01**: Java framework (no dedicated language) with features for matching of distributed events
- **Djcutter Nishizawa et al., AOSD'04**: `host`, distributed `cflow`
- **ReflexD E. Tanter et al., DAIS'06**: reflective kernel for distributed AOP
- **DyMAC Lagaisse et al., Middleware'06**: middleware platform with advanced support for (aspect) composition
- **CaesarJ I. Aracic et al., TAOSD'06**: support for distributed deployment and dynamic weaving
- **AWED Benavides et al., AOSD'06**: detailed later

Expressive aspect languages

- Make **explicit relationships** between execution events
  Eliminate use of non-local state in aspects

Expressive aspect languages

- Make **explicit relationships** between execution events
  Eliminate use of non-local state in aspects
- Means
  - **“Stateful aspects”** **Douence et al., GPCE’02**
    Synonyms: history-based or trace-based aspects
  - **Richer pointcut languages**
    Regular aspects, temporal logic-based aspects, logic pointcuts, etc.
  - **Domain-specific** sublanguages in pointcuts and advice
    Especially for distribution and concurrency

Expressive aspect languages

- Make **explicit relationships** between execution events
  Eliminate use of non-local state in aspects
- Means
  - **"Stateful aspects" Douence et al., GPCE'02**
    Synonyms: history-based or trace-based aspects
  - **Richer pointcut languages**
    Regular aspects, temporal logic-based aspects, logic pointcuts, etc.
  - **Domain-specific** sublanguages in pointcuts and advice
    Especially for distribution and concurrency
- Goals
  - **Enhance expressivity**: enable expression of complex aspects
  - Support definition and **reasoning** of aspectual properties

Outline

Motivation: transactional replicated caches

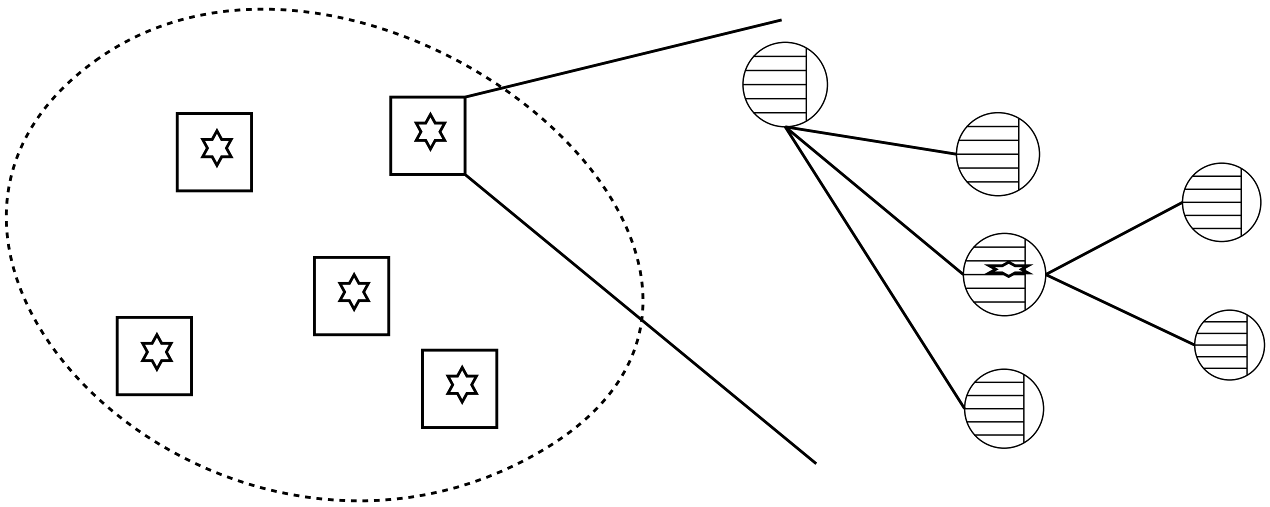
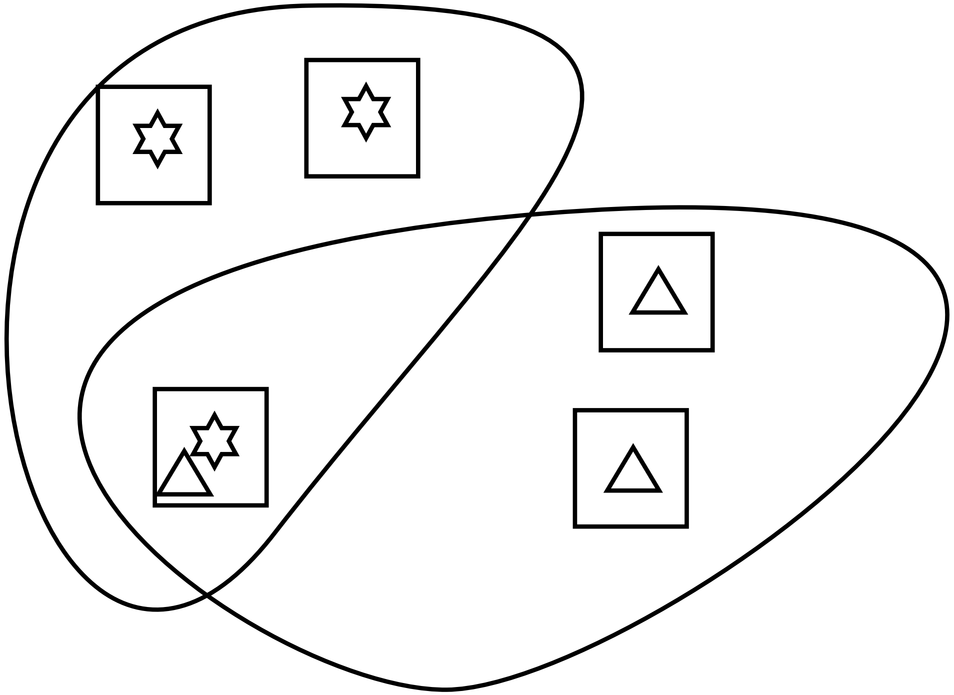


Figure: a) Replicated Caches b) Zoom of Data structure

- Cache **data structure** deployed on each node
- Data **replication** under control of **transactions**
- JBoss Cache (version 1.2):
  replicate item in `TreeCache` structure on all hosts of a cache

Ex.: support cache evolution



- New replication policies
  - **Don't replicate unnecessarily** huge objects
  - Replicate only in case of **interest**

Modularization of distribution concerns

Requirements for distribution-specific aspect abstractions:

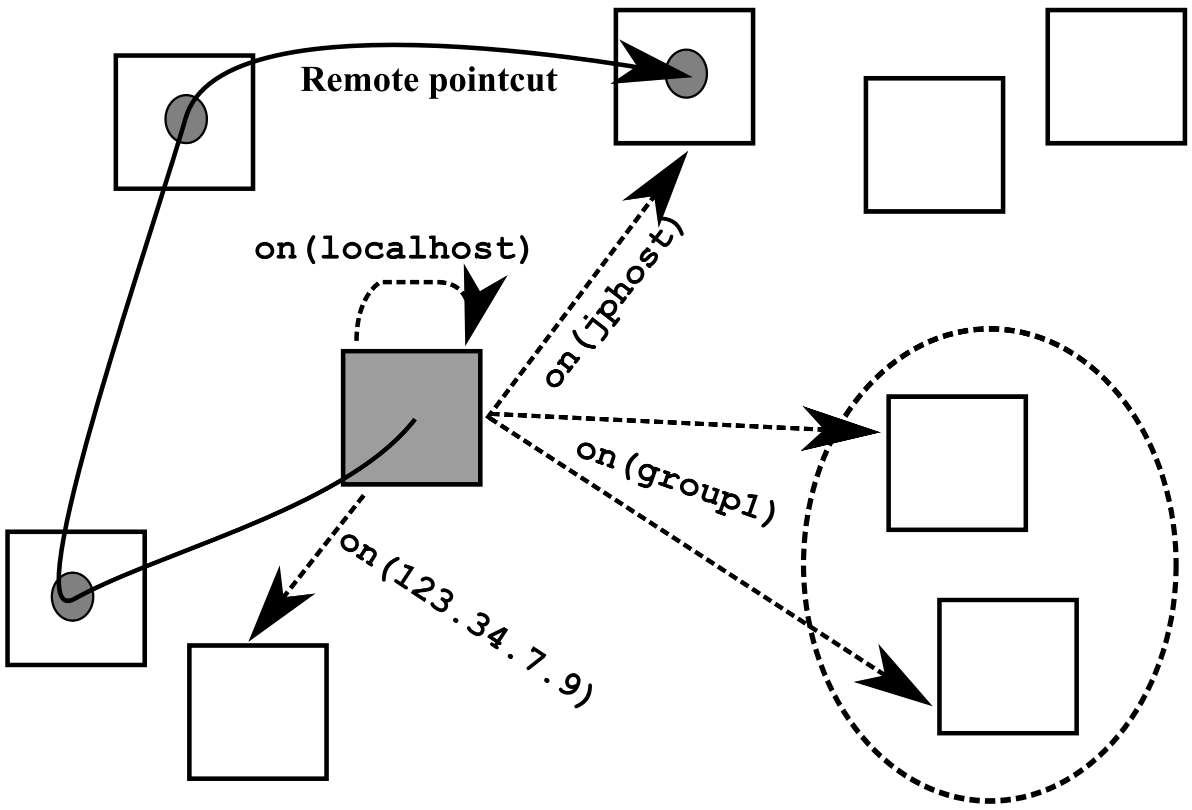
- Detection of **remote events**
- **Remote execution** of code
- Support for **distributed state**
- **Distributed deployment** of aspects

Aspects With Explicit Distribution (AWED)

- **Remote pointcuts**
  - References to remote hosts: `host`, `on`, host groups
  - 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)`

Benavides et al., AOSD'06

Remote pointcuts and advice



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() → s3 || s2,
              s2: cachePut(key, o) → s3 || s2,
              s3: stopCache() → s1)

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

- Essentially (non-deterministic) finite-state automata

Remote Advice

- 2 synchronization **modes**: a/synchronous
  Access to result managed using **futures**
- Management of **groups**:
  dynamically add hosts to or remove from groups
- Ex: replication advice:

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

Ex.: lazy replication aspect

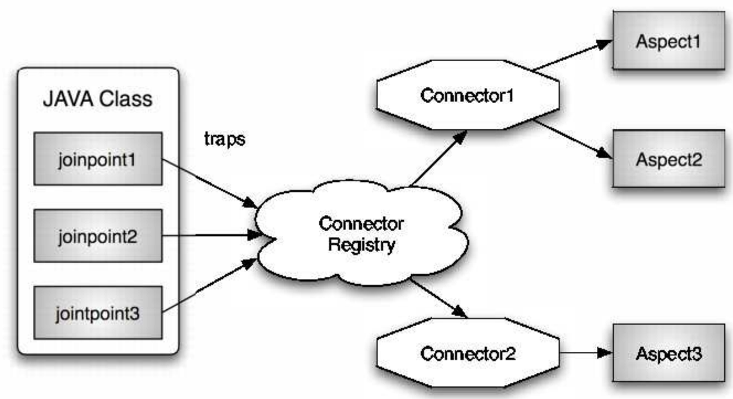
```
all aspect CacheReplication{
  pointcut cachePcut(Object key, Object o):
      call(* Cache.put(Object, Object))
      && args(key,o) && !on(jphost)
      && !within(CacheReplication);

  before(String k, Object o):
      cachePcut(k, o) && on(interestedHosts) {
          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**
  Connectors
  - control distribution of execution events that represent matched join points and
  - manage aspect compositions and resolve interactions



Distributed JAsCo Infrastructure

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

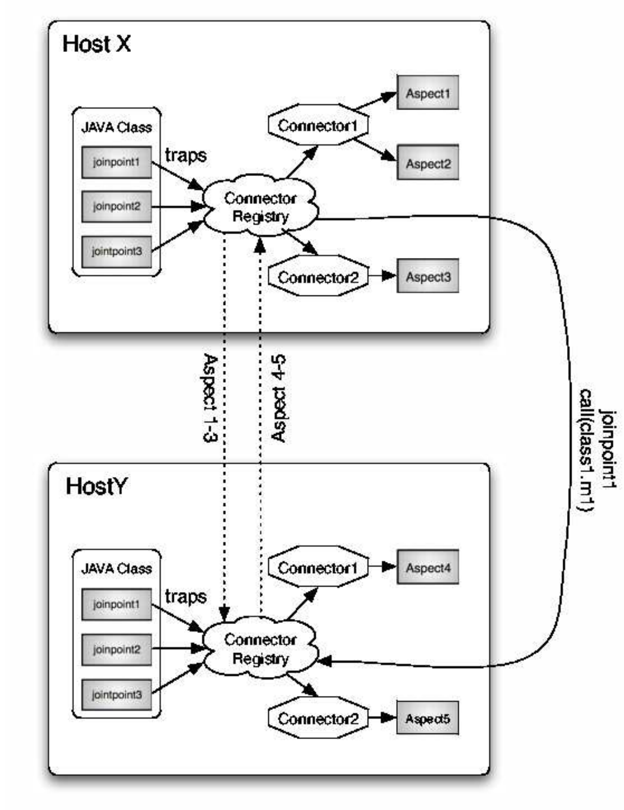


Figure: DJAsCo architecture.

Overview of implementation

- Remote pointcuts
  - JP propagation using **JGroups** (a popular Java library for multicast communication)
  - Cflow: **customized sockets** (extension mechanism of Java RMI)
  - Remote Sequences: extension of **JAsCo sequences**

Overview of implementation

- Remote pointcuts
  - JP propagation using **JGroups** (a popular Java library for multicast communication)
  - Cflow: **customized sockets** (extension mechanism of Java RMI)
  - Remote Sequences: extension of **JAsCo sequences**
- Remote advice
  - Based on **activation of deployed aspects**
  - Synchronization by futures implemented using **AWED aspects**

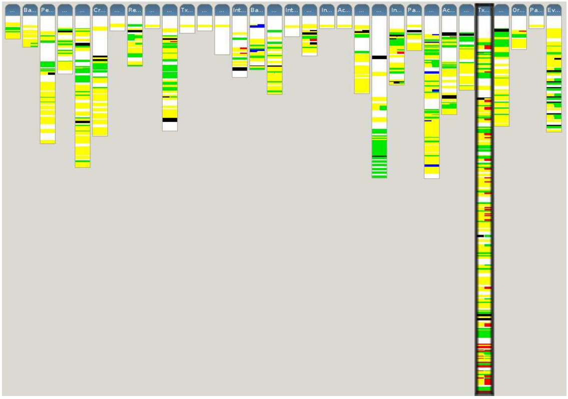
Overview of implementation

- Remote pointcuts
  - JP propagation using **JGroups** (a popular Java library for multicast communication)
  - Cflow: **customized sockets** (extension mechanism of Java RMI)
  - Remote Sequences: extension of **JAsCo sequences**
- Remote advice
  - Based on **activation of deployed aspects**
  - Synchronization by futures implemented using **AWED aspects**
- Distributed aspects
  - **Connector distribution** using JGroups
  - Aspect state sharing, parameter passing: **AWED aspects**

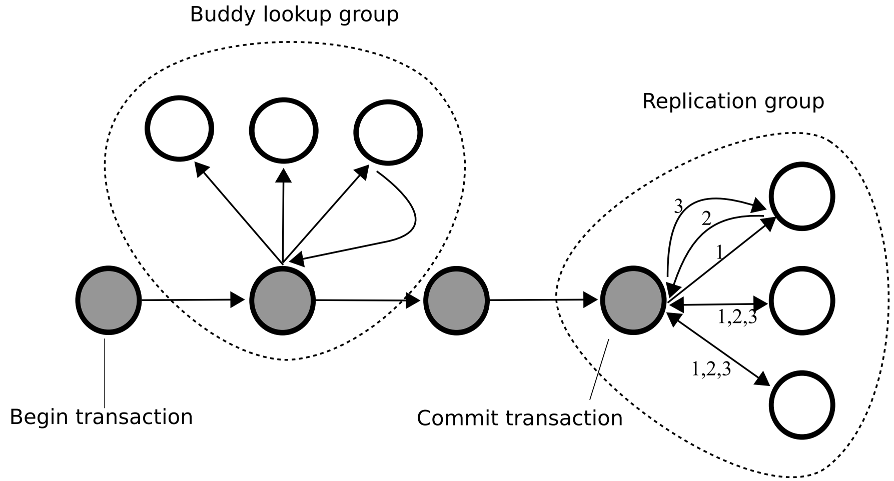
Example for **integration of diverse implementation techniques/platforms**

Outline

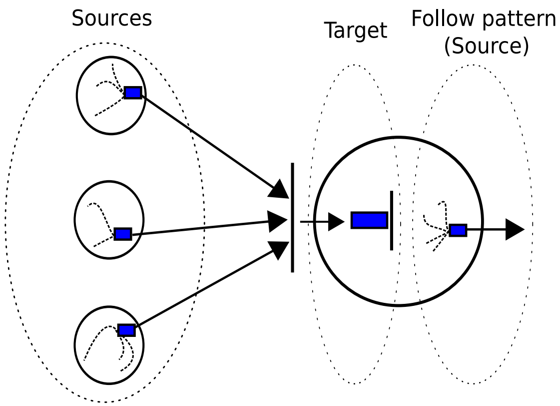
Ex.: “architectural programming” of JBoss Cache



Implementation (~50 KLOC)



Runtime architecture



Invasive pattern `gatherI`

```
pipeI( ...
    farmI(...) ...;
    farmI(...
        gatherI(...
            farmI(...)));
```

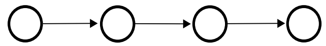
Program

Requirements for suitable notion of patterns

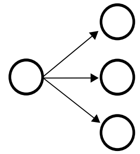
- **Modularization of similar (micro) pattern applications in different contexts**
  - Unlike standard distributed patterns
- **Remote executions and multicast communication**
  - Similar to standard distributed patterns
- **Heterogenous synchronization requirements**
  - Unlike standard distributed patterns

Approach: invasive distributed patterns

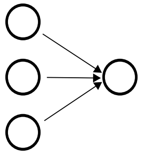
- **Extend standard computation and communication patterns**
  - Source side: context-dependent pattern applications
    - Extraction of pattern-enabling state information
    - Preparation of data and communication
  - Target side: remote executions
  - multicast communication and heterogenous synchronization requirements: cooperation of both sides
- **Invasive distributed patterns** integrate these features
- Implementation by means of **stateful distributed aspects**



a) pipe



b) farm

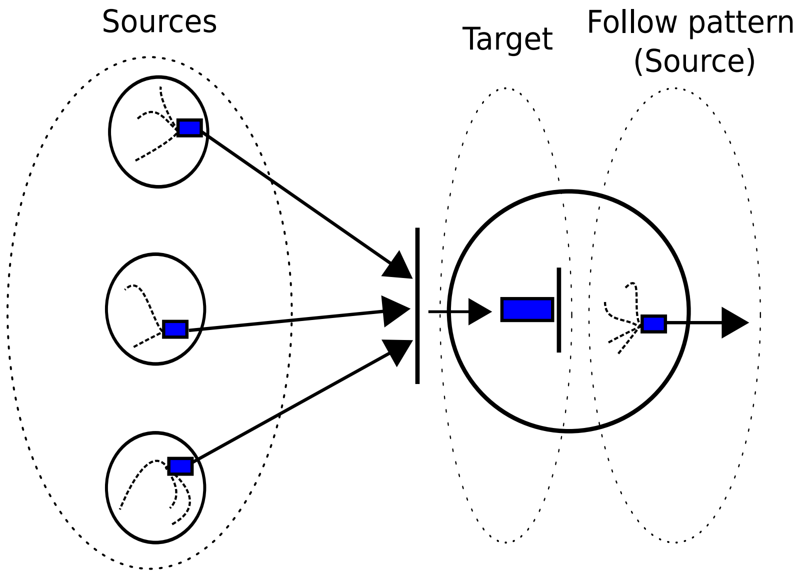


c) gather

Outline

Ex.: invasive (gather) pattern

- **n-to-1** communication and data integration
- Source and target **groups** of computing units
- **Sequences over remote events** to quantify over different pattern-enabling contexts
- **Data prepared** on source units and **integrated** in target units

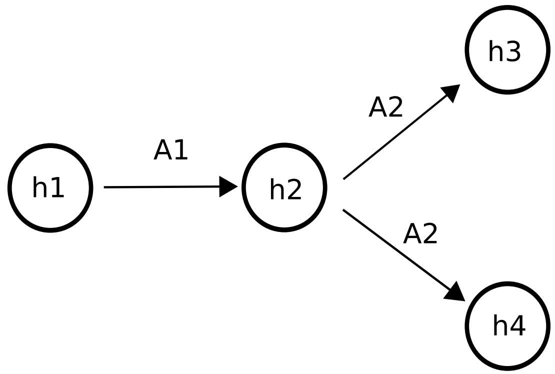


- **Asynchronous execution of source computations** and communications
- **Synchronization** between sources and host as well as target integration and follow pattern

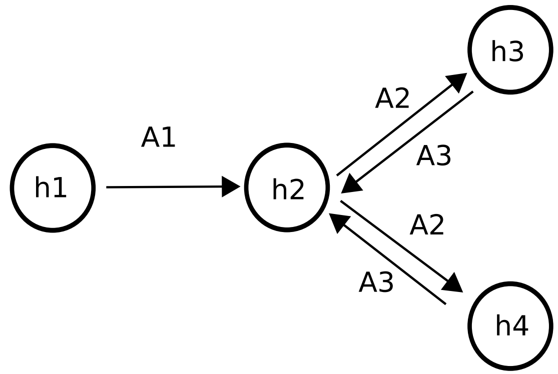
Pattern language and compositions

$$
\begin{array}{lcl}
P & ::= & \texttt{patternSeq}\ G_1\ A_1\ G_2\ A_2 \ldots G_n \\
G & ::= & H\ G \mid P\ G \mid \epsilon \\
A & ::= & \texttt{aspect \{ around}((H, \mathrm{Id}^*)^*):\ PCD\quad SourceAdvice\quad [\texttt{sync}]\ TargetAdvice\ \texttt{\}} \\
PCD & ::= & \texttt{call}(MSig) \mid \texttt{target}(Id) \mid \texttt{args}(Id+) \\
 & \mid & PCD\ \texttt{\&\&}\ PCD \mid PCD\ \texttt{||}\ PCD \mid \texttt{!}\ PCD \\
 & \mid & Seq
\end{array}
$$

- **patternSeq** $G_1\ A_1\ G_2\ A_2 \ldots G_n$
  - $G_i\ A_i\ G_{i+1}$: pattern application
    triple of source group, distributed aspect, target group
- Examples:



patternSeq(h1,A1,h2,A2,(h3,h4))



patternSeq(h1,A1,h2,A2,(h3,h4),A3,h2)

Outline

Pattern-based definition of JBoss Cache

```
gCaches = {h1, h2, h3}
pipe([h], // h is current host
     Atransac,
     farm(
       gather(
         farm([h], Aprepare, sync gCaches-[h]),
         Apresp,
         [h]),
       Acommit,
       gCaches-[h])
    );
```

- Composition of **4 pattern applications**
- **3 aspects** used to modularize crosscutting pattern-enabling state accesses
- **Directly implements overall runtime architecture** of replicated caching

Ex. Aprepare: aspectized prepare phase

```
aspect Aprepare {
  org.jboss.cache.TreeCache tc = CacheRegistry.getInstance().getCache();

  around(DataStorage d, String txId):
    call(* PrepareHelper.send(..)) && args(d,s) &&
    !cflow(call(TransactionManager.prepare(..)))

    // Source advice
    { proceed(); }

    // Target advice
    { TransactionManager tm = TransactionManager.getInstance();
      PrepareHelper ph = new PrepareHelper();
      try{
        tm.prepare(d, txId, tc);
        ph.respAgree(txId);
      } catch(Exception e) {
        ph.respNotAgree(txId);
      }
    }
}
```

Evaluation

- **Significant reduction in code size and complexity**
  - Original JBoss Code: 2674 LOC in 17 classes with intricate scattering
  - Refactored functionally-equivalent solution: 532 LOC **($\sim$ 80% reduction)** in **11 well-modularized aspects and classes**
- **Execution overhead**
  - Amortized over application typically very low
  - Benchmark on cache use only: **currently factor 3**
  - Dynamic aspect application using aspect execution infrastructure on top of J2SE
  - Use of JGroups library for multicast communication

Much optimization potential: compile patterns, optimize communication

No reason not to match original performance:
use static aspect system

Outline

Conclusion

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

Future work

- Composition operators
- Formal semantics
- Property support
  - General correctness and QoS properties
  - Causality guarantees for distributed aspects
    first approach: **Benavides et al., Middleware'08**
  - Absence of deadlocks
  - Sequentialization of aspect-based pattern compositions
  - Scoping and distributed aspects **É. Tanter et al., AOSD'09**
- New application domains
  Ex.: introduction of checkpointing for grid algorithms **Benavides et al., SBAC-PAD'08**
- Links to related domains: component-oriented programming, standard distributed and concurrent patterns . . .

References (1)

Akşit et al., Addison-Wesley'04 Mehmet Akşit, Siobhán Clarke, Tzilla Elrad, Robert E. Filman (eds.): *"Aspect-Oriented Software Development"*, Addison-Wesley Professional, Sep. 2004.

Kiczales, CIOO'96 Gregor Kiczales: *"Aspect Oriented Programming"*, Proc. of the Int. Workshop on Composability Issues in Object-Orientation (CIOO'96) at ECOOP, July 1996.

Tarr et al., ICSE'99 Peri Tarr, Harold Ossher, William Harrison, Stanley M. Sutton, Jr: *"N Degrees of Separation: Multi-dimensional Separation of Concerns"*, Proceedings of ICSE, 1999.

Soares et al., OOPSLA'02 Sergio Soares, Eduardo Laureano, Paulo Borba: *"Implementing distribution and persistence aspects with AspectJ"*, Proc. of OOPSLA, Nov. 2002.

Colyer, Clement, AOSD'04 Adrian Colyer, Andrew Clement: *"Large-scale AOSD for Middleware"*, Proc. of AOSD, Mar. 2004.

Cohen, Gil, ECOOP'04 T. Cohen, J. Gil: *"AspectJ2EE = AOP + J2EE: Towards an Aspect Based, Programmable and Extensible Middleware Framework"*, Proc. of ECOOP, Jul. 2004.

Benavides et al., AOSD'06 Luis Daniel Benavides Navarro, Mario Südholt, Wim Vanderperren, Bruno De Fraine, Davy Suvée: *"Explicitly distributed AOP using AWED"*, Proc. of AOSD, Mar. 2006.

References (2)

Douence et al., GPCE'02 Rémi Douence, Pascal Fradet, Mario Südholt: *"A framework for the detection and resolution of aspect interactions"*, Proc. of GPCE, Oct. 2002.

Lopes'97 Cristina Videira Lopes: *"D: A Language Framework for Distributed Programming"*, PhD thesis, Northeastern University, 1997.

Pawlak et al., Reflection'01 Renaud Pawlak, Lionel Seinturier, Laurence Duchien, Gérard Florin: *"JAC: A Flexible Solution for Aspect-Oriented Programming in Java"*, Proc. of Reflection, Sep. 2001.

Nishizawa et al., AOSD'04 Muga Nishizawa, Shigeru Chiba, Michiaki Tatsubori: *"Remote Pointcut - A Language Construct for Distributed AOP"*, Proc. of AOSD, Mar. 2004.

E. Tanter et al., DAIS'06 É. Tanter, R. Toledo: *"A versatile kernel for distributed AOP"*, Proc. of DAIS, June 2006.

Lagaisse et al., Middleware'06 Bert Lagaisse, Wouter Joosen: *"True and Transparent Distributed Composition of Aspect-Components"*, Proc. of Middleware, Nov. 2006.

I. Aracic et al., TAOSD'06 I. Aracic, V. Gasiunas, M. Mezini, K. Ostermann: *"An Overview of CaesarJ"*, TAOSD I, 2006.

References (3)

Andrews, Reflection'01 James Andrews: *"Process-Algebraic Foundations of Aspect-Oriented Programming"*, Proc. of Reflection, Sep. 2001.

Cunha et al., AOSD'06 Cunha, Carlos A. and Sobral, João L. and Monteiro, Miguel P. : *"Reusable Aspect-Oriented Implementations of Concurrency Patterns and Mechanisms"*, Proc. of AOSD, Mar. 2006.

Douence et al., GPCE'06 Rémi Douence, Didier Le Botlan, Jacques Noyé, Mario Südholt: *"Concurrent Aspects"*, Proc. of GPCE, Oct. 2006.

Benavides et al., DOA'07 D. Benavides, M. Südholt, R. Douence, J.-M. Menaud: *"Invasive patterns for distributed applications"*, Proc. of DOA, Nov. 2007.

Benavides et al., Middleware'08 L. D. Benavides Navarro, R. Douence, M. Südholt: *"Debugging and testing middleware with aspect-based control-flow and causal patterns*, Proc. of Middleware, Nov. 2008.

É. Tanter et al., AOSD'09 É. Tanter, J. Fabry, R. Douence, J. Noye, M. Südholt : *"Expressive Scoping of Distributed Aspects"*, Proc. of AOSD'09, Mar. 2009.

Benavides et al., SBAC-PAD'08 L. D. Benavides Navarro, R. Douence, F. Hermenier, J.-M. Menaud, M. Südholt: *"Aspect-based patterns for grid programming"*, Proc. of SBAC-PAD, Oct. 2008.