Crosscutting Concerns in J2EE Applications

Ali Mesbah

Arie van Deursen

CWI
Research Center for Mathematics and Computer Science

October 3, 2005

Introduction

- Java 2 Enterprise Edition (J2EE)
  - Framework for distributed applications
  - Provides 3-tier architecture
  - Java Server Pages (JSP) + Servlets, Enterprise JavaBeans (EJB), Enterprise Information Systems
- Aspect-oriented Software Development (AOSD)
  - Captures crosscutting concerns
  - Reduces scattering and tangling functionality
  - Aims at improving modularity

Existing work

- Han and Hofmeister: Navigation Concerns
- Murali et al.: J2EE Patterns
- Cohen and Gil: AspectJ2EE = AOP + J2EE

Not covered: "how aspects influence evolution in J2EE Environments"

Goal

Explore evolution benefits of AOSD in J2EE settings

- Study crosscutting concerns
  - Top-down approach
  - Bottom-up approach
- Resolve crosscutting nature by adopting aspect-oriented code
- Evaluate evolution

Case Study

- Pet Store
  - Demo Web Application
  - 17,000 LOC, 283 classes/interfaces
  - Well-known, open-source
  - Well-designed, numerous design patterns
  - Latest J2EE specifications
- Real-world Web applications
  - Not as well-designed
  - More crosscutting concerns possible

Top-down concerns

Concerns that are known to have a crosscutting nature:

- Security
- Persistence
- Transaction Management
- Tier-cutting concerns (compression, encryption, ...)
- Logging
- ...

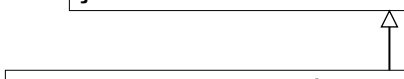
Transaction Management

- Declarative
  - EJB's deployment descriptor
  - Container is responsible
  - No support for non-EJB parts
  - Not a crosscutting concern
- Programmatic
  - Non-EJB parts: Servlets, Business Logic, …
  - Developer is responsible
  - Possible aspect candidate

Transaction Management

```
void insertTemplate(HttpServletRequest request,
 HttpServletResponse response, String templateName)
 throws IOException, ServletException {
  try {

    InitialContext ic = new InitialContext();
    UserTransaction ut = (UserTransaction)
      ic.lookup("java:comp/UserTransaction");
    ut.begin();

    context.getRequestDispatcher(templateName)
     .forward(request, response);

    ut.commit();

  }
  ...
}
```


```
abstract aspect AbstractTransAspect {
  abstract pointcut transactionOperations();
  void around() : transactionOperations() {
    try {
      InitialContext ic = new InitialContext();
      UserTransaction ut = (UserTransaction)
      ic.lookup("java:comp/UserTransaction");
      ut.begin();
      proceed();
      ut.commit();}...
}
```



```
aspect PetStoreTransactionAspect extends AbstractTransAspect{
  pointcut transactionOperations() :
   execution(* com..TemplateServlet.insertTemplate(..)
     throws IOException, ServletException)
   ||
   ...
}
```


Bottom-up concerns

Aspect mining:

- Fan-in analysis
  - Exception Wrapping (20% code reduction)
  - Service Locator & Singleton
  - Precondition Checking (removed from 9 classes)
- Interface concerns
  - Extracting Interface implementations
  - Interface migration

  (Serializable: 29 classes, 6 interfaces)

M. Marin, A. van Deursen, L. Moonen - Identifying Aspects using Fan-in Analysis. WCRE 2004

P. Tonella and M. Ceccato – Migrating Interface Implementation to Aspect-oriented Programming. ICSM 2004

Discussion

Concern	Code Reduction	Oblivious	Reliability	Modularity	Evolvability
Transaction Management	√	√	√	√	√
Exception Wrapping	√	√		√	√
Service Locator		√			√
Precondition Checking		√	√	√	√
Interface Extraction				√	√
Interface Migration				√	√

Singleton Nature Hidden

20% in affected classes

Check code in a single location

Conclusions

- Top-down and bottom-up approach
- Concerns handled by container: well-addressed
- Concerns NOT handled by container: AOP does offer evolutionary benefits
- Concerns in J2EE systems include generic ones
- + 25 aspect candidates in Pet Store
- Real-world applications
  - Circumvent container mechanism (e.g., EJB)
  - More benefits from utilizing AOP

Questions

Postdoc Position

In

Web Application Reengineering

At

Delft University of Technology

Service Locator & Singleton

```
privileged aspect LocatorAspect {
  private static ServiceLocator service;
  pointcut serviceLocator(): call(*.ServiceLocator.new())
    && !within(LocatorAspect);

  Object around() throws ServiceLocatorException: serviceLocator() {
    synchronized (service) {
      if (service == null) {
        service = new ServiceLocator();
        try {
          service.ic = new InitialContext();
          ...
        } catch (NamingException e) {
          service = null;
          throw new ServiceLocatorException(e);
    } } }
    return service;
} }
```