

Security with AspectJ

Emin İslam Tatlı
tatli@th.informatik.uni-mannheim.de

Department of Computer Science, University of Mannheim

Doktorandenseminar, 04 July 2006

Outline

- 1 Aspect-oriented Programming
 - Crosscutting Concerns
 - Aspects
 - After AOP
- 2 AspectJ
 - Motivation
 - AspectJ Terminology
- 3 Security with AspectJ
 - Security Requirements
 - Security with AspectJ (with demos)
- 4 Conclusion
 - Discussion

Motivation to AOP

- Modularity in programming languages
 - procedural \Rightarrow object-oriented
- Bad modularity due to crosscutting concerns
 - security, logging, exceptions, database transactions, etc.
- Approach to better modularity (*separation of concerns*)
 - object-oriented \Rightarrow **aspect-oriented**

Crosscutting Points

Definition

Crosscutting concerns are aspects of a program which affect (**crosscut**) other concerns and often cannot be cleanly decomposed from the rest of the system in both the design and implementation (*from Wikipedia*).

Crosscutting Examples

Exceptions

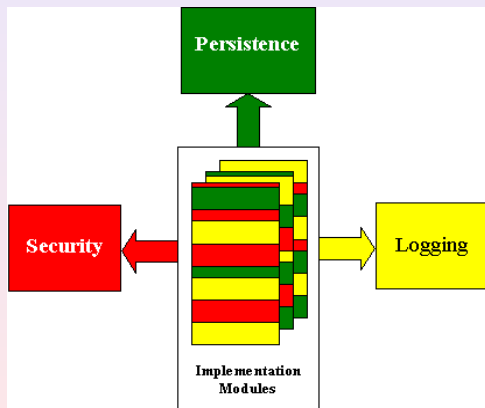
```
1 // Create a socket with a timeout
2 try {
3     SocketAddress saddr = new InetSocketAddress("web.de", 80);
4     Socket socket = new Socket();
5     socket.connect(saddr, 2000); // 2 seconds timeout
6 } catch (UnknownHostException e) {
7 } catch (SocketTimeoutException e) {
8 } catch (IOException e) {
9 }
```

Crosscutting Examples (cont.)

Authentication

```
public class Test {  
    public void methodOne() {  
        if (user.authenticated()) {  
            // Log start of the operation  
            doBusinessLogic();  
            // Log completing of the operation  
            // Do exception handling  
        }  
    }  
    public void methodTwo() {  
        if (user.authenticated())  
            doBusinessLogic();  
    }  
}
```

Crosscutting Illustration



Aspects

- Problem: **code tangling and scattering**
- Solution: **separation of concerns**
- Aspects provide:
 - separation of crosscutting concerns
 - better reusability
 - better code quality
 - easy and efficient evolution

After AOP

```
public class Test {  
  
    public void methodOne() {  
        doBusinessLogic();  
    }  
  
    public void methodTwo() {  
        doBusinessLogic();  
    }  
  
}
```

```
public aspect TestAspect {  
  
    pointcut p(): call(* Test.method*());  
  
    void around(): p() {  
        if (!user.authenticated()) {  
            }  
        else proceed();  
    }  
}
```

After AOP (cont.)

Socket Method

```
public void doSocketConn() {  
  
    SocketAddress saddr = new InetSocketAddress("web.de", 80);  
    Socket socket = new Socket();  
    socket.connect(saddr, 2000); // 2 seconds timeout  
  
}
```

After AOP (cont.)

Exception Handler Aspect for Socket Method

```
public aspect SocketAspect {  
  
    pointcut pexception(): call(void Socket.connect(..));  
  
    declare soft: java.lang.Exception: pexception();  
  
    void around(): pexception() {  
        try {  
            proceed();  
        } catch (UnknownHostException e) {  
        } catch (SocketTimeoutException e) {  
        } catch (IOException e) {  
        }  
    }  
}
```

AspectJ

Definition

an aspect-oriented programming extension to Java

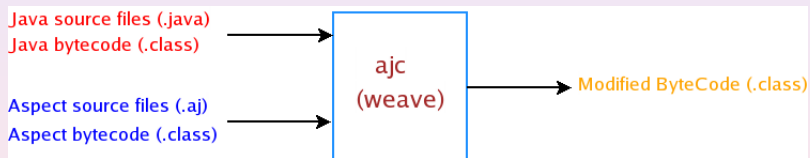
History

- **August 1998**: Gregor Kiczales and his team developed this concept at Xerox PARC.
- **December 2002**: Xerox transferred AspectJ to an openly-developed eclipse.org project (www.eclipse.org/aspectj).

Example

"Hello World" AspectJ Example

AspectJ Mechanism



AspectJ Terminology

- 1 Join point
- 2 Pointcut
- 3 Advice
- 4 Aspect

Join Point

Definition

any identifiable execution point in a system

Examples

- 1 method call/execution
- 2 constructor call/execution
- 3 object pre-initialization/initialization
- 4 read from a variable/write to a variable
- 5 exception throwing/handler execution
- 6 advice execution

Join Points in Example

Join Points

```
class BankAccount {  
  
    private int account;  
  
    public BankAccount(int account) {  
        this.account=account;  
    }  
  
    private void increaseAcc(int debit) {  
        account +=debit;  
    }  
  
    public static void main(String[] args) {  
        BankAccount bc=new BankAccount(100);  
        bc.increaseAcc(50);  
    }  
}
```

Pointcut

Definition

selection of a set of join points

Pointcut Types

call, execution, get, set, handler, within, withincode, cflow, cflowbelow, this, target, args, if

Pointcut

Examples

- 1 `call(void HelloWorld.doTest())`
- 2 `call(* HelloWorld.doTest*(..))`
- 3 `execution(* doTest*()) && within(HelloWorld)`
- 4 `call(HelloWorld.new(..))`
- 5 `set(int BankAccount.account) || get(int *.account)`
- 6 `handler(java.io.IOException)`
- 7 `handler(IOException) && cflow(execution(void doTest()))`
- 8 `call(void increaseAcc(..)) && target(bc)`
- 9 `args(String[])`

Advice

Definition

a declaration advising to execute a certain code within a matched pointcut

Advice Types

- 1 before
- 2 around
- 3 after
 - after
 - after returning
 - after throwing

Aspect

Definition

pointcuts and advices are combined within Aspects

Aspect Features

- 1 Static structure of a class can be changed
 - add fields, methods, constructors
 - extend parent classes (declare parent)
 - implement an interface
- 2 Abstract aspects and pointcuts can be defined
- 3 Introductions are supported (declare soft, declare error, declare warning)

Tracing Aspect Example

Tracing Aspect Example

```
public aspect TracingAspect {  
  
    pointcut ptrace(): call (* *.*(..));  
  
    before(): ptrace() {  
        System.out.println(thisJoinPointStaticPart+" Start Time: "+System.  
            currentTimeMillis());  
    }  
  
    after(): ptrace() {  
        System.out.println(thisJoinPointStaticPart+" End Time: "+System.currentTimeMillis  
            ());  
    }  
}
```

AspectJ Visualizer

- crosscutting points between classes and aspects are displayed visually in an Eclipse perspective.

Security Requirements

- security belongs to non-functional logic
- separation of security logic from business logic can help better management of security

Requirements

- 1 Integrity
- 2 Confidentiality
- 3 Authentication

Integrity

Definition

the condition in which data are identically maintained during any operation, such as transfer, storage, and retrieval (from Wikipedia)

Problem

- Sensitive data (i.e. messages, files, etc.) can be altered by unauthorized

Solution

- Hash functions (e.g. MD5,SHA1,RIPEMD,etc.)

Integrity and AspectJ

Demo

- Servers should store hash value of passwords, not the clear text.
- For authentication, servers should hash user's password and compare with the hashed password in the database.

Confidentiality

Definition

ensuring that information is accessible only to those authorized to have access (from ISO)

Problem

- Confidential data can be sniffed on the wire or accessed on the memory, disk, etc. illegally by unauthorized

Solution

- Encryption
 - 1 Symmetric (e.g. DES, AES, RC4)
 - 2 Asymmetric (e.g. RSA)

Confidentiality and AspectJ

Demo

- Communication over the the untrusted networks (e.g. Internet) should be encrypted.
- Use SSL sockets rather than plain sockets for confidential communication.

Authentication with AspectJ

Definition

Verification of the credentials (identity) of a principal/user

Demo

- check credentials before sensitive methods are executed.

Other Aspect Languages

AOP Languages

refer to Wikipedia:

http://en.wikipedia.org/wiki/Aspect-oriented_programming

Discussion

- Aspects are great, but ...
- Problems also do exist
 - Unintended weaving can be dangerous.
 - Aspect languages, as (*complicated*) languages, are expected to learn.
 - Aspects should be considered at design time.

Conclusion

- Bad modularity due to crosscutting concerns
- Aspects can separate business and other non-functional logics
- Better security management is possible with AOP
- But aspects should be designed and implemented carefully

References and Further Reading I

Communities

- Aspect-oriented Software Development Community.
<http://aosd.net>
- European Network of Excellence on Aspect-oriented Software Development.
<http://www.aosd-europe.net>

Mailing lists

- AspectJ user mailing list:
<https://dev.eclipse.org/mailman/listinfo/aspectj-users>
- AOSD user mailing list:
<http://aosd.net/mailman/listinfo/discuss>

References and Further Reading II

Tools

- Eclipse/AspectJ: www.eclipse.org/ajdt,
www.eclipse.org/aspectj
- Emacs/AspectJ: <http://aspectj4emacs.sourceforge.net>
- Other AOP tools:
http://en.wikipedia.org/wiki/Aspect-oriented_programming

Books

- Ramnias Laddad: AspectJ in Action, Manning Publications, 2003.
- Joseph D. Gradecki, Nicholas Lesiecki: Mastering AspectJ: Aspect-Oriented Programming in Java, Wiley Publications, 2003.