# The relevance of AOP to an Applications Programmer in an EJB environment

Howard Kim and Siobhán Clarke

Department of Computer Science, Trinity College Dublin, Ireland
howard.kim@cs.tcd.ie

**Abstract.** Many of the examples that are used to demonstrate the value of aspect-oriented programming are based on crosscutting concerns such as distribution support, remote access of objects, and synchronisation. Enterprise Java Beans (EJB), a standard component model for component transaction monitors (CTMs), provides inbuilt support for these concerns, thereby reducing the need for the applications programmer to be concerned about them. Does this make aspect-oriented programming irrelevant in an EJB environment? In this paper, we describe a distributed system developed using EJB where crosscutting concerns were managed by the EJB environment. The system is an eVoting system that allows students to vote online in Student Union elections in Trinity College

## Introduction

Most software systems consist of several concerns. Typical examples of concerns are: *logging, transaction integrity, persistence, authentication, security, and performance*. Many of these concerns do not affect one implementation module of the system but affect multiple modules; these are known as crosscutting concerns. With EJB the container handles support for *security, administration, performance and container managed persistence (CMP)*. These sound a lot like the crosscutting concerns that are used to motivate the need for aspect-oriented programming (AOP) kinds of techniques. This led us to ponder the need for aspect-oriented programming in component-based development environments such as EJB.

In this paper, we examine the crosscutting concerns that were evident in an EJB implementation of an eVoting software system. In this system students should be allowed to authenticate themselves using a given student I.D and password and then be allowed to vote online in the given election. The system has a requirement that communication between client and server is secure and how a client voted is not determinable by examining communication channels or log files, this has the implication that when a vote is entered into the database it is not logged. Security is therefore a key concern in the system. Other concerns we discuss here are persistence and transactions.

In the background of this paper we discuss the J2EE platform. We then describe how EJB were useful in the development process and what problems they solved. We also relate our development with an emphasis on how aspects may be used in conjunction with EJB to solve the problems of crosscutting concerns. We conclude that EJB does solve some of the problems dealt with by aspects but there may exist the need for aspects in a distributed environment depending on the requirements of the system.

**Background**

J2EE application servers usually support three different types of security: *authentication*, *access control* and *secure communication*. Only access control is specifically addressed by EJB.

1. Authentication confirms the identity of a particular user and allows them access to certain resources in the system.
2. Access control applies administrator defined explicit policies that regulate what a user can do in the system. Policies are particular business goals/objectives that the application server must understand before it can determine the right of access to a resource. The EJB deployment descriptor allows an administrator to identify particular groups of users and permit access to the resource.
3. At present no specification exists for the secure communication between EJBs. The most popular solution used in web-based applications is Secure Socket Layer (SSL).

The EJB architecture deals with crosscutting concerns such as: security, administration, performance and container managed persistence (CMP). An applications programmer can if they so wish handle the code of persistence this is called bean-managed persistence (BMP)[7]. With CMP a developer can concentrate on developing the business logic and the container would manage the named crosscutting concerns. With EJB most crosscutting concerns are dealt with in the XML deployment descriptor; a bean deployer or developer would mark beans as being persistent, transactional and would set the security policy. Fig 1 shows how an applications programmer would define persistence and security in an EJB environment.

```
<entity>
        <ejb-name>ISS</ejb-name>
        <home>ie.tcd.server.authentication.AuthenticateHome</home>
        <remote>ie.tcd.server.authentication.Authenticate</remote>
        <ejb-class>ie.tcd.server.authentication.AuthenticateBean</ejb-
class>
        <persistence-type>Container</persistence-type>
        <prim-key-class>java.lang.Integer</prim-key-class>
        <reentrant>False</reentrant>
……      //CMP Fields
</entity>
<security-role>
     <role-name>everyone</role-name>
</security-role>
<container-transaction>
     <method>
        <ejb-name>ISS</ejb-name>
        <method-name>*</method-name>
     </method>
     <trans-attribute>Required</trans-attribute>
</container-transaction>
```

**Fig. 1 XML Deployment Descriptor**

**The SU eVoting System**

For the development of the eVoting system, Session and Entity Beans (CMP) were needed. The system is based on the Model View Controller paradigm [1] and is also loosely based on the J2EE Front Controller pattern [2]. Fig 2 shows the architecture of the system. In the client side of the application a student enters their login details in an applet; this applet then generates a public/private key pair that is used to encrypted data sent to the server. Once the server authenticates that the user is valid, a special token or ticket is generated by the server and sent back to the client. The client can then use this ticket to vote in the given election by again encrypting their vote and sending it to the ballot server that validates if the ticket is a valid.

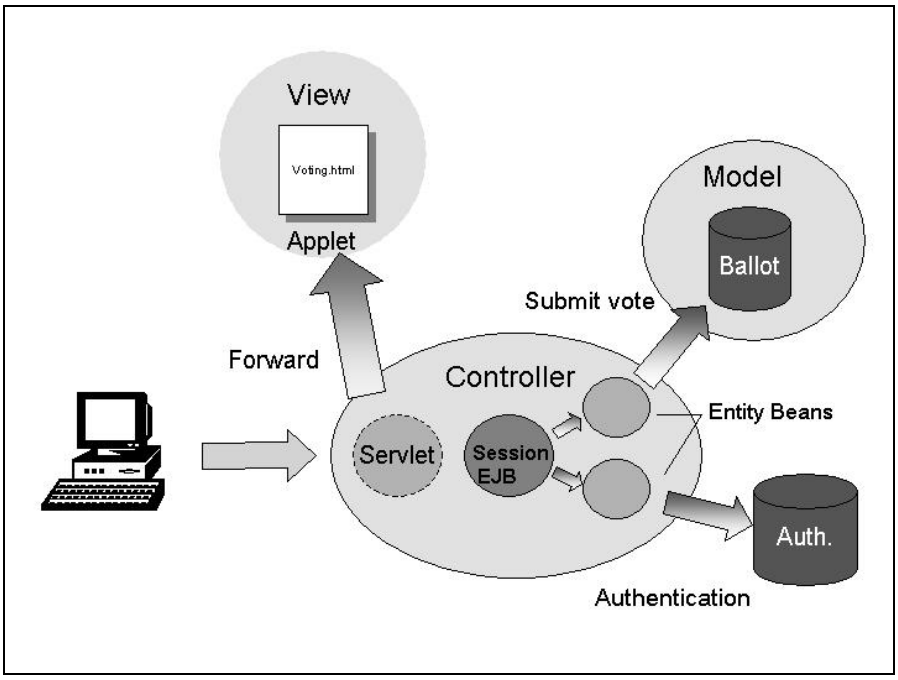


**Fig. 2: Architectural Diagram of the SU eVoting system**

We can view the system as a combination of multiple concerns; the concerns being:

1. Authentication of a given user.
2. Secure communication between client and server
3. Persistence storage of a user vote
4. Transactional vote updates
5. Secured access to the voting database.

These were probably the most important concerns of the system. Once a vote is entered into the database this vote must be persistent otherwise the loss of vote would mean a re-election being called. It was also imperative that no unauthorised access be allowed to the

vote database as you may guess the validity of the election depends on this concern. Finally authentication of users is a major concern of the system.

A more in-depth look at the concerns in the system revealed that all are crosscutting concerns. But the code remained modular because the container handled some these concerns[†] as Table 1 displays.

| Concern Name | Crosscutting in a non-EJB environment | Solved by |
|---|---|---|
| Authentication of users | Yes | Using a combination of a Session and an Entity Bean solved this problem. The code is modularised into two components and not replicated. Fig 3 shows how authentication was handled for the system.<br>No crosscutting remained for applications programmer. |
| Secure communication | Yes | By using Public Key Infrastructure (PKI), the communication channels between client and server remained secure. The code usage is scattered between client and server but this is necessary due to the nature of PKI. The code is not replicated and remained in a single class.<br>No crosscutting remained for applications programmer. |
| Persistence storage of votes | Yes | The vote Entity bean handled persistent storage of votes.<br>No crosscutting remained for applications programmer. |
| Transactional vote updates | Yes | Again the vote Entity bean/XML Deployment Descriptor handled transactional issues; by using the deployment descriptor beans were marked as being transactional.<br>No crosscutting remained for applications programmer. |
| Secure access to database | Yes | The security policy for the system was set in the deployment descriptor at deployment.<br>No crosscutting remained for applications programmer. |

**Table 1. Concerns in the system and how they were solved**

[†] It should be noted that although these concerns resulted in modular code from an applications programmer point of view, the only reason is because the EJB container handles all these issues internally; such as *object locking, persistence management and security management*.

By using the EJB technology in the application many of the concerns involved when programming distributed systems were taken away such as transactions and server-side security. In Fig 1 we stated which methods should be marked as transactional and define the security policy (who is allowed access these methods). In this declarative programming model the applications programmer who creates the bean need not be the same person as the deployer who states that bean is transactional or set the security rights. As Giese [3] describes with EJB there is a pre-defined component lifecycle that ensures a well-defined interface, but this also has the property that instead of specific modules realising a particular aspect, the application server provides a predefined list.

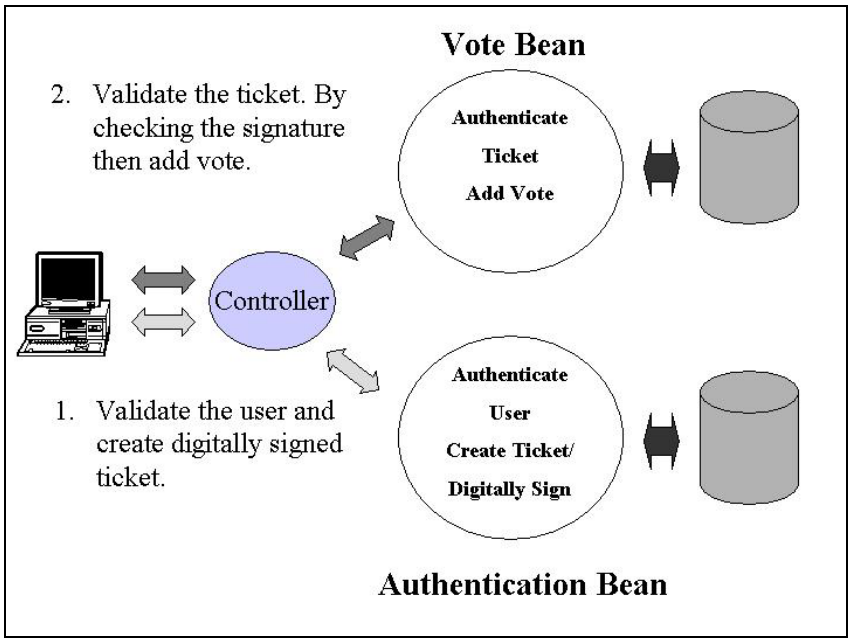


**Fig 3. Process of validating user & vote**

Although 'aspects' or 'crosscutting concerns', from the applications programmers view; were hard to find in this system we believe that they could be useful when an applications programmer needs transaction/database logging or BMP.

**Logging**

When logging in an EJB environment the EJB specification suggests that `java.io.*` classes are not used within the bean (this is to increase portability). The two main products available for logging are JLog [4] and Log4j [5]. These have been used for logging within beans. Other solutions may involve writing log data through the network rather than to disk.

**BMP**

With BMP all persistence logic issues are left to the applications programmer. The applications programmer must write the persistence handling code into the bean class implementation. The structure of the database and how the bean class's fields map to the

database must be known by the applications programming. Monson-Haefel [7] states that application programmers can use BMP to develop custom beans for their business systems. Further research is necessary to see if AOP would help with logging or BMP in an EJB environment.

**Discussions and Conclusions**

It would be a surprising if there existed a large enterprise system that had no crosscutting concerns. Which is probably why a lot of techniques have been developed to modularise the implementation of systems with crosscutting concerns; these techniques include mixin-classes, design patterns and domain-specific solutions [6] and of course AOP. EJB is an example of a domain-specific solution to crosscutting concerns.

The project architecture ensured modularity even though as discussed some modules cross cut the application. By using EJB it ensured that the application could be used with any J2EE compliant server. The main difficulties with the J2EE architecture is that there is a steep learning curve in development, but this is out weighed by the advantages it has to offer; reliability, robustness, scalability and the enterprise computing power.

We believe with our experiences in development of the S.U eVoting system that EJB does solve some of the problems associated with transactions and security; but this is only due to the nature of our specific voting problem. If other crosscutting concerns such as logging had become necessary, then EJB alone would not have met our requirements.

Our on going research is in the area of AOP and our next research is an evaluation of AOP in the Microsoft .NET environment.

**References**


1. S. Burbeck, Applications Programming in Smalltalk-80™: How to use.the Model-View-Control (MVC), 1999. http://st-www.cs.uiuc.edu/users/smarch/
2. Sun Microsystems, Sun Java Centre J2EE Patterns, 2002. http://developer.java.sun.com/developer/technicalArticles/J2EE/ patterns/
3. H. Giese. Towards Ruling Component-Based Distributed Systems with Role-Based Modelling and Cross-Cutting Aspects, University of Paderborn.
4. JLog IBM implementation of Java Logging tool http://www.alphaworks.ibm.com/tech/loggingtoolkit4j
5. Log4j Java Logger www.log4j.org
6. R. Laddad, I want my AOP (Part 1) Separate software concerns with aspect-oriented programming. http://www.javaworld.com 2002
7. R. Monson-Haefel, Enterprise JavaBeans 2nd Edition, O'Reilly publications 2000.