Aspect-Oriented Secure Connectors for Implementation of Secure Software Architecture

Chase Baker
Department of Computer Science
Texas Tech University
Lubbock, USA
chase.baker@ttu.edu

Michael Shin
Department of Computer Science
Texas Tech University
Lubbock, USA
michael.shin@ttu.edu

Abstract - This paper describes aspect-oriented secure connectors for implementing secure software architecture for distributed business applications. A secure connector for secure software architecture can be designed separately from application business components by considering different communication patterns between the components as well as security services required by application components. In this paper, the secure connector is implemented as an aspect-oriented secure connector separately from application business components. The aspect-oriented secure connector is structured with both a security service aspect and a communication pattern aspect that are separated from each other. Once aspect-oriented secure connectors are implemented, they can be reused for different applications with the security service and communication pattern aspects required between application components. In this paper, aspect-oriented secure connectors are used to implement an electronic commerce application.

Keywords – aspect-oriented secure connector; security service aspect; communication pattern aspect; separation of concerns; aspect oriented programming

I. INTRODUCTION

Software solutions for distributed business applications are becoming increasingly complex in nature, due to the inclusion of various crosscutting concerns such as security, synchronization, and logging. These concerns are necessary for implementation in such applications because of the progressively more significant concerns that may be required for their intended uses. As a result, it is vital that these software solutions be implemented in such a manner that their implementation is as untangled as possible in order to reduce complexity and increase reusability. One way to avoid tangling requirements is to design and implement the software such that there is a clear separation of crosscutting concerns from application concerns.

Crosscutting security concerns can be designed as secure connectors [15] separately from application components. Security mechanisms are encapsulated in secure connectors that provide security services as well as communication patterns for the applications. These secure connectors are modular and more maintainable and reusable than tangled security and application logic in design found in systems without a clear separation.

To achieve this clear separation of concerns in implementation, secure connectors need to be implemented separately from application logic. Aspect-Oriented Programming (AOP) provides a powerful way to implement such crosscutting concerns in implementation. Crosscutting security concerns can be implemented as security aspects in AOP, which are completely separated from the application logic of applications. Secure aspects are only executed when the services they provide are needed by the application.

This paper describes the mapping and implementation of aspect-oriented secure connectors from secure connectors designed for secure software architectures in distributed business applications. A mapping scheme is described to demonstrate how aspect-oriented secure connectors are implemented from secure connectors in design. The secure connector is implemented as an aspect-oriented secure connector separately from application business components. In particular, the aspect-oriented secure connector is structured with both a security service aspect and a communication pattern aspect that are separated each other. The aspect-oriented secure connectors are validated with the implementation of an electronic commerce application.

This paper is organized as follows. Section II covers related works to the current research. Section III describes the mapping of secure connectors in design to aspect-oriented secure connectors in implementation. Section IV contains the validation of our approach with a case study. Finally, section V serves as a conclusion to the paper and outlines future work.

II. RELATED WORK

Separation of concerns is one of the major topics in software engineering today. Past research has proven there is a need to clearly separate concerns in complex systems in order to promote modularity, reusability, and maintainability [1]. The work provided an approach to modeling complex systems by ensuring that security and core business logic was separated in the requirements and design models during early stages of software development. Other work has begun the discussion on how to model complex systems with clear crosscutting concerns [2]. One of the proposed methods of modeling such systems includes utilizing aspect-oriented techniques to ensure a separation of crosscutting concerns. However, the work stresses the difficulty of finding a standard design model because of UML’s informal nature. The benefits of an aspect-oriented approach have been studied extensively, mainly focusing on the maintainability and modularity of such systems [3]. The results of the research discovered that an aspect-oriented approach results in smaller, less complex, and far more modular implementation as well as shortened maintenance cycles. For
Java specifically, the AspectJ extension has been developed to provide Java support for modular designs to handling crosscutting concerns throughout a system with aspect-oriented methodologies [4].

However, there currently is no standard for representing systems with an aspect-oriented notation, and work has been done to prove the need for formal modeling of such systems to ensure the accuracy of the designs [5]. Extensions to the current UML framework with aspect-oriented concepts have been proposed, specifically with crosscutting concerns being considered early in design and then mapped to programming models with automatic techniques [6]. These extensions require aspects to be explicitly defined early in the design process in order to avoid a divergence during later mapping. Other approaches have been proposed, including aspect-oriented approaches to model-driven development [7] and even for larger, more general extensions to UML such as the Aspect Modeling Language [8]. The Aspect Modeling Language offers approaches for modeling aspect-oriented concepts in early design, requirements, and architecture models through code generation.

Other work has been done to propose aspect-oriented modeling notation at specific points during the software engineering process including early design [9], the requirements level [10], and even for general activity modeling [11].

While there exist proposals for modeling aspect-oriented systems in design, there is still a lack of thorough mapping schemes to evolve designs into implementation. Research has been done to stress the need for new mapping schemes for extending these early design models into implementation [12]. Developing such mapping schemes is not a trivial task, and work has been done to address the challenges one may face when mapping design models to implementation code [13].

Previous work directly related to the research in this paper has proposed a method for separating security concerns in design to security aspects in code [14]. This work provides an approach to map security components in design to secure aspects in implementation to promote reuse, reduced complexity, and increased maintainability during the implementation phase. Work has also been done to separate crosscutting concerns in the design phase using secure connectors while developing secure software architectures [15]. Such an approach promotes the separation of crosscutting concerns from core business logic by keeping the crosscutting concerns isolated in secure connectors. The work done in this paper focuses on an extension of that idea, specifically when secure connectors encapsulate multiple perspectives, such as security service and communication pattern perspectives, and there is a need to map such perspectives into separate aspects in implementation.

### III. From Secure Connectors to Aspect-Oriented Secure Connectors

Aspect oriented programming provides developers with a tool to separate various crosscutting concerns in their systems from core application functionality. It does this by promoting the separation of each crosscutting concern into its own aspect. An aspect in AOP can be broken down into four parts: the aspect itself, a join point(s), a pointcut(s), and advice.

An aspect can be viewed as a feature of a system that might be used at various points throughout a system’s implementation.

![Figure 1. Applying Secure Synchronous Connector with Confidentiality Security Service in the Business to Business (B2B) Electronic Commerce Application](image-url)

---

188
For instance, confidentiality is used at multiple points in systems that require encryption and decryption of data to keep information secure. In this case, confidentiality would be a crosscutting concern of the system, as it deals more with security logic than the application logic of whatever the system might be.

An aspect has two important components that provide it with its functionality, namely the pointcut(s) and advice. Pointcuts refer to well-defined moments of execution in a system, commonly referred to as join points. These join points can be object initializations, method calls, or method executions for instance. A pointcut clearly defines the collection of join points at which an aspect should be injected for use. The advice of an aspect provides the additional code or methods required to give the aspect the functionality desired. In the confidentiality aspect example, the advice would include the algorithms and methods needed to apply encryption and decryption to the necessary data.

Now that our definition of an aspect is defined, we can determine how to map secure connectors in software architecture into aspect-oriented secure connectors in implementation. Consider a use case in a business to business (B2B) electronic commerce application that is responsible for placing a requisition order. This use case may require that the requisition order be encrypted before it is packaged and then sent to the server for storage via some communication pattern, such as synchronous communication with reply.

Fig. 1 depicts the UML communication diagram for placing a requisition order. An application component, aCustomerComponent, calls the Place Requisition method to send its order to a location on the network. It then passes through a secure connector that determines the requisition order requires confidentiality. The system then applies encryption to the requisition order. Upon receiving encryption, the Place Requisition method resumes and the secure connector determines the requisition order requires a communication pattern be applied to it. The system prepares the requisition order for a synchronous communication with reply protocol and then sends the requisition order to the network where it can be taken to the server.

The mapping scheme of secure connectors to aspect-oriented secure connectors can now be considered. Notice that the security service is applied when the security coordinator in the secure connector has determined that the requisition order requires encryption or decryption. The responsibility of the security coordinator is identical to the behavior of an aspect’s pointcut, which simply is used to determine when an aspect is required during runtime. Also, the security service, aSecurityService for example, is the implementation for carrying out that security service. This is identical to the contents of an aspect’s advice. Similarly, the second perspective of the secure connector, a communication pattern, can be divided into two parts to represent it in a form of an aspect. The communication pattern concern has to decide if it is necessary to apply the pattern (pointcut) and if so, it should apply the communication pattern (advice). Thus we can notice that the contents of a secure connector, aSecureSenderConnector, are mapped to multiple aspects such as a security service aspect and a communication pattern aspect.

Taking what we just discussed, we can now construct a mapping scheme from secure connectors in software architecture to aspect-oriented secure connectors in implementation. Fig. 2 depicts the mapping scheme for this approach. Step one consists of mapping the secure connector’s security perspective into an aspect object. Step two maps the security coordinator from the security perspective to security pointcuts in the aspect. Step three maps the service from the security perspective to security advice in the aspect. Finally,
step four divides the communication perspective into a communication pointcut to determine when to apply the communication pattern in the newly created communication advice.

With the mapping scheme developed, we can focus on mapping the secure connector from Fig. 1 to an aspect-oriented secure connector. In this model, we can see a number of crosscutting concerns that are represented by secure connectors with multiple perspectives. Within the customer-side connector, we have identified both security and communication perspectives. Within the server-side, we can identify the same security and communication perspectives. Therefore in this model, we have identified two crosscutting concerns that are candidates for evolving into aspects: security (confidentiality) and communication (synchronous with reply).

Looking closer at security, we can notice two different security services it offers: encryption and decryption. Therefore we will need to have two different advice blocks with their respective pointcuts featured in the mapped model. Encryption will be applied when an encryption pointcut determines the information requires encryption, and similarly decryption will be applied if a decryption pointcut determines the information requires decryption. This is applicable for both the sender and receiver sides of the routine.

For communication on the customer side, notice there exists a communication pattern object, aSynchronousMCWithReplySender. It is important to investigate what this component is trying to do. Looking at our diagram in Fig. 1, the communication pattern object will take the encrypted order and package it according to a predefined communication style (here being synchronous message communication with reply). Therefore, we can split this component into two parts during mapping: a pointcut to determine when to apply a specific communication style and a block of advice containing the methods necessary to apply the communication style to the order. Similarly on the receiver side, aSynchronousMCWithReplyReceiver will be broken down into the same two objects, a pointcut to determine when a communication style is needed and advice to determine how to handle data that is being received.

Fig. 3 depicts the communication diagram with aspects mapped from the communication diagram with secure connectors in Fig. 1. The secure connector is broken down into two secure aspects, as we determined there were two separate crosscutting concerns contained within the connector. For each side of the system, we end up with a security aspect (cSecurityAspect, rSecurityAspect) and a communication aspect (cCommunicationAspect, rCommunicationAspect). For the security aspects, we have a pointcut to determine if decryption is required by the information being received (decryptionPointcut) and also a pointcut to determine if encryption is required by the information being sent (encryptionPointcut). Similarly in the communication aspects, we have two pointcuts to determine how to handle specific data types, orders (orderPointcut) and statuses (statusPointcut).

Immediately one can see the multiple benefits to this approach. The model now has all crosscutting concerns completely separated from any core application logic in the system. The modeled aspects can be easily removed and reused in other parts of the application or even in entirely new applications with very little modification required. This structure also allows the reader to clearly see the various crosscutting concerns contained in the system without much effort. This greatly reduces the effort required to comprehend more complex architectures.

IV. VALIDATION WITH CASE STUDY

It is important that the separation of concerns be prevalent in both the design and the implementation of the system, otherwise
the benefits of the approach would be lost. With that in mind, a case study with a distributed business to business (B2B) electronic commerce application is implemented using the Java programming language and AspectJ, an aspect-oriented extension to assist in the separation of crosscutting concerns.

We will focus on the previous scenario of a customer component contacting a requisition server in order to send a requisition order in the following example. Specifically, we will be implementing the scenario that we previously mapped in Fig. 3, which is the synchronous message communication with reply and confidentiality security service.

This scenario makes use of a secure connector with two perspectives on both the sender and receiver sides. The perspectives are communication (synchronous message communication with reply) and security (confidentiality). We previously discussed the process used to map the perspectives from the secure connectors in Fig. 1 to the aspect-oriented secure connectors in Fig. 3. Using the mapping concepts, we can develop aspect-oriented secure connector code implementing our design.

```
public aspect cSecurityAspect {
    RequisitionOrder Encrypt(RequisitionOrder Order) []

    pointcut encryptionPointcut(RequisitionOrder Order) :
        execution("aCustomerComponent.PlaceRequisition(RequisitionOrder)") & args(Order);

    RequisitionOrder around() : encryptionPointcut(RequisitionOrder) && within(cSecurityAspect +)
    {
        proceed();
        Object[] args = thisJoinPoint.getArgs();
        RequisitionOrder Order = (RequisitionOrder) args[0];
        RequisitionOrder EncryptedRequisitionOrder = Encrypt(Order);
        return(EncryptedRequisitionOrder);
    }
}
```

Fig. 4 depicts the mapped implementation for the security aspect on the customer side, specifically the objects pertaining to the encryption of the order. In code, we have the security aspect, cSecurityAspect. This aspect will contain pointcuts and advice for both encryption and decryption. First, we define the pointcut for encryption. This representation of the pointcut reads "at the execution of the PlaceRequisition() method from the aCustomerComponent class with the parameter Order of RequisitionOrder type, run the advice for the encryptionPointcut."

The advice is contained within the block of code. In this example, around-type advice is used. Around-type advice is powerful in AspectJ, as it allows the passing of objects between aspects and other java packages. In this example, we have a proceed() statement that allows the PlaceRequisition() method to finish processing before we apply our advice. Once the PlaceRequisition() method has prepared the order, our advice then retrieves it from aCustomerComponent for use within the aspect. When it has been retrieved, our advice applies encryption to the order using an Encrypt() method we define elsewhere within our aspect. When encryption has been completed, we then

```
public aspect cCommunicationAspect {
    RequisitionOrder Pack(RequisitionOrder EncryptedRequisitionOrder) []

    pointcut orderPointcut(RequisitionOrder EncryptedRequisitionOrder) :
        execution("aCustomerComponent.PlaceRequisition(RequisitionOrder)") & args(EncryptedRequisitionOrder);

    RequisitionOrder around() : orderPointcut(RequisitionOrder) && within(cCommunicationAspect +)
    {
        proceed();
        Object[] args = thisJoinPoint.getArgs();
        RequisitionOrder EncryptedRequisitionOrder = (RequisitionOrder) args[0];
        RequisitionOrder PackedEncryptedRequisitionOrder = Pack(EncryptedRequisitionOrder);
        return(PackedEncryptedRequisitionOrder);
    }
}
```

Figure 4. Security Aspect Implementation with Confidentiality Mapped From Security Perspective in Secure Connector Design

Figure 5. Communication Aspect Implementation Mapped From Communication Perspective in Secure Connector Design
return the EncryptedRequisitionOrder back into the flow of the system where we retrieved it from. This same process is used to map the decryption pointcut within the aspect.

Similarly, we map the implementation for the communication aspect on the customer side, specifically the objects pertaining to the packing and sending of a service request. The mapped implementation for the communication aspect on the customer side is shown in Fig. 5. In code, we have the communication aspect, cCommunicationAspect. Let's focus on the pointcut/advice combination required for packing a requisition order with a certain communication style before it is sent to the network (here it is synchronous message communication with reply). The representation of the pointcut reads "at the execution of the PlaceRequisition() method from the aCustomerComponent class with the parameter EncryptedRequisitionOrder of RequisitionOrder type, run the advice for the orderPointcut." Similar to the customer side, the server side in Fig. 3 can be implemented based on the mapping scheme described in Section III.

V. CONCLUSION

The mapping scheme proposed shows that it is possible to map secure software architectures, specifically those using secure connectors as a way to separate crosscutting concerns from application logic, to an implementation that features aspect-oriented concepts. The various perspectives found in the secure connectors are mapped to individual aspects in code, while maintaining the intended separation of concerns in design with added benefits in implementation.

The results of the case study using the proposed approach show that mapping the various perspectives in secure connectors to aspect-oriented secure connectors in code results in a higher degree of the separation of crosscutting concerns. This approach achieves a much greater degree of modularity and reusability from the separation of crosscutting concerns. The new aspects can also be applied to other applications that require the same crosscutting concerns, requiring only minor changes for use in such applications. The resulting implementation is also far easier to maintain, as all crosscutting concerns are isolated to their own individual aspects.

Future work could include an automated or semi-automated mapping from design models to skeleton aspects in implementation. The mapping scheme described in section 3 can be implemented as a tool. For this, the mapping scheme described in this may need to be refined. The automated or semi-automated mapping tool can be helpful for large-scale applications.

REFERENCES