Architecting Secure Software Systems Using an Aspect-Oriented Approach: A Survey of Current Research

Josh Dehlinger
Computer Science Department
Iowa State University
dehlinge@ iastate.edu

Nalin Subramanian
Computer Science Department
Iowa State University and
nvsubram@ iastate.edu

Abstract

The importance of security in the development of complex software systems has increasingly become more critical as software becomes increasingly more pervasive in our everyday lives. Aspect-orientation has been proposed as a means to handle the crosscutting nature of security requirements when developing, designing and implementing security-critical applications. This paper surveys some of the approaches and contributions of integrating an aspect-oriented approach into designing and implementing secure software systems.

Keywords: Aspect orientation, software security requirements, security framework, software architecture, security concerns in code, web applications, distributed applications.

1. Introduction

The importance of security in the development of complex software systems has increasingly become more critical as software becomes increasingly more pervasive in our everyday lives [4]. A wide range of security-critical domains (e.g., finance, national defense, etc.) rely on software applications as the major enforcement entity ensuring the security policies of the stakeholders. Although secure systems are critical to numerous domains, industry experience has shown that software developers are poor at writing secure code, often because of the complex nature of non-functional requirements such as security [17].

It is well established that security requirements are non-functional requirements that are cross-cutting in nature [7] - they "crosscut the requirements, design or implementation of several or even many building blocks" [11]. This further complicates producing secure code since the enforcement of security policies are scattered or "tangled" throughout the design and implementation. An important solution in managing the crosscutting nature of security requirements is to adopt an aspect-oriented software development (AOSD) approach in designing and implementing the system [8]. AOSD handles crosscutting concerns, such as security, by employing the separation of concerns view. In terms of security, this means that the main module(s) of a program would not need to encode security policies; rather, security policies would be separated and implemented in a separate, independent piece of code [17]. Using AOSD, the separation of concerns principle for a crosscutting concern, can be applied from the requirements engineering phase (e.g., [12]) through the software lifecycle until it is (likely) implemented as an aspect in aspect-oriented programming (AOP), for example in AspectJ [9], in the software.

In analyzing the deficiencies of the state of the art in designing, developing and implementing secure systems, Viega, Bloch and Chandra identified a number of desirable properties in any solution hoping to improve software engineers and developers in producing more secure systems. The properties in [17] include:

- The security-related properties in a system should be abstracted out of the main system to improve clarity, maintainability, manageability and reuse.
- Legacy source code with known or potential security vulnerabilities should be able to be patched with a minimal amount of new code. It should also be possible to avoid modifying the original code.
- When applicable, security-related properties should be reusable across different applications.

AOSD and AOP proponents claim that their techniques, frameworks and methodologies satisfy these properties when implementing security requirements as crosscutting concerns [17]. In addition, AOSD and AOP specifically can aid in the following security-specific activities [17]:

- Automatically perform error checking on security-sensitive calls.
- Automatically log data related to security concerns.
- Replace generic code with secure code (e.g., generic socket code with SSL socket code).
- Insert code at startup that goes through a set of "lockdown" procedures.
- Specify privileged sections a program and automatically request and return privileges when necessary.

Thus, the contributions of adopting an AOSD/AOP approach to designing and implementing the security requirements of a security-critical software system have led researchers to actively pursue this avenue as a viable approach. This paper surveys some of the contributions and their relation to designing and implementing secure software systems. Specifically, this paper investigates several differing aspect-oriented security frameworks proposed in literature, identifies problems and lessons learned from the proposed aspect-oriented security frameworks and discusses and briefly evaluates the AOSD-based security frameworks. The second part of this work provides a broad review of aspect-oriented approaches used for secure coding, modeling security concerns and resolving security concerns in distributed software applications.

2. Aspects in Software Security: Security Frameworks Employing an Aspect-Oriented Approach

This section investigates several approaches for using an aspect-oriented framework for designing secure software systems. We first review UML-based security framework for incorporating security policies as an aspect when designing a secure system. We then review a rather formal, architecture-based, aspect-oriented security framework that heavily uses the Software Architecture Model (SAM), Petri nets and temporal logic for architecting a secure software system. Third, we review a generic aspect-oriented security framework and provide the authors' learned lessons derived from its design and use. It is hoped that the lessons learned contribute to the improved design of future security frameworks. We conclude this section with some discussion and an informal evaluation of the AOSD-based security frameworks.

2.1 Designing a Secure System Using Aspects

It has been well established that the manner in which software is designed can have a significant impact on nonfunctional qualities of the system (e.g., reliability, usability, security, etc.). Therefore, it is crucial that software engineers and developers consider these nonfunctional concerns when making architectural, logical and physical design decisions. This subsection briefly covers an aspect-oriented design technique for designing a secure system as proposed Georg, Ray and France in [6]. In [6], security concerns are captured in aspects and are treated as design patterns. The authors claim that viewing security concerns in this manner during design modeling allows for the following advantages [6]:

- Aspects allow one to understand and communicate security concerns in their essential forms, rather than in terms of specific behavior.
- An aspect focuses on one concern, allowing for an easier way to model and understand its behavior.
- Security aspects may be reusable across different systems.
- Changes to security policies are made in one place (the implemented security aspect) and effected by weaving the aspects into the primary model.
- Easier to analyze the impact of security concerns on design units by weaving the aspects into the primary models and evaluating the resulting models.
- Security engineers and designers may be able to identify problems with the design of the security mechanisms even before they are implemented - potentially saving significant development cost, time and effort.

This approach uses Role Models, "a structure of roles where a (meta-)role defines properties that must be satisfied by conforming to a UML model elements" (e.g., a class or an association) [6]. Then, weaving this kind of an aspect into a primary model only involves a model transformation process where the non-conforming model is transformed to a conforming model (i.e., the model that incorporates the aspect).

A security concern as a design aspect is modeled using two aspect views: static and interaction views. The static view captures the structural properties of the aspect. The interaction view captures the interaction patterns associated with the aspect. This approach uses *Static Role Models* (SRM) and *Interaction Role Models* (IRM) to model these two views. An SRM defines the patterns of UML static structural model, such as UML Class Diagram patters, and an IRM defines UML interaction diagram patters, such as UML Collaboration Diagram patterns. Using this models, an aspect definition usually consists of an SRM and one or more IRMs. A full description of SRMs and IRMs can be found in [6] and other existing literature.

The authors of [6] then define security objectives including *confidentiality*, *integrity* and *availability* and provide a know list of potential security attacks, problems and solutions. They claim weaving strategies, to determine the constraints and the manner in which the aspects (containing the security policies), need to be developed from the expected security threats and problems expected for the proposed system. For example, if the proposed system has non-sensitive data traveling over communication links, this indicates that encryption is not need and can be omitted from the weaving strategies. However, [6] provides no insight on how to choose a strategy for different security attacks and problems and provides no listing or evaluation of weaving strategies for different weaving strategies. Presumably, this is at the discretion and expertise of the security engineer. Rather,

[6] focuses on how to model security aspects using Role Models and then how to weave the aspects into a design model.

To weave an aspect modeled by a combination of an SRM and IRM(s), the following steps suffice:

1. Map primary model elements to the roles they intend to play.
2. Merge roles with primary model elements.
3. Add new elements to the primary model.
4. Delete existing elements from the primary model.

The authors intend that the weaving strategies become "reusable forms of experience that can be used to assess the threats to a particular system and propose techniques (i.e., a combination of mechanisms) to prevent or detect related attacks" [6]. Yet, they provide no demonstration or hint how this can be done. Further, the authors claim that their approach provides the ability to easily change the weaving strategy and then re-weave them into the model to observe the impact on the system by the proposed changes. This could be quite powerful if a security engineer is intuitive enough to see the advantages and disadvantages in the application of different weaving strategies. However, this analysis would only be as good as the engineer.

This approach is somewhat problematic in that the security provided by the mechanisms in the model is only as good as the weaving strategies. That is, a good security policy may be implemented in the aspects, but a poor weaving strategy of the aspect into the primary module will yield an insecure system. Coupling this with the lack of guidance, or even several realistic examples, provided by the authors in selecting an appropriate weaving strategy for a particular security attack or problem illustrates the immaturity of this approach. Further, the lack of tool support prevents the practical use of this approach and an empirical evaluation using this approach hinders its independent evaluation.

2.2 Secure Software Architectures Using Aspect Orientation

While the security framework using aspect-orientation described in [6] uses the UML-like models along with Role Models to define a system and an aspect, the approach presented in [18] relies on the more formal methods of the Software Architecture Model (SAM), Petri nets and temporal logic to define the system and the security aspects. This approach uses SAM to define a hierarchical set of compositions of the software architecture where each composition consists of a set of components, a set of connectors and a set of constraints to be satisfied by the interacting components. The behavior of the components and connectors are modeled by predicate transition nets and the properties are specified my temporal logic formulas. This subsection describes the formal approach, described in [18], to design secure software architectures. The secure architecture derived from this approach "defines the structure of the software system, the interaction and coordination among its components, which correctly enforces the security requirement" [18]. The authors claim the following contributions of their approach:

- A formal notion for aspect-oriented modeling at an architectural level.
- An aspect-oriented approach to designing secure software architectures.

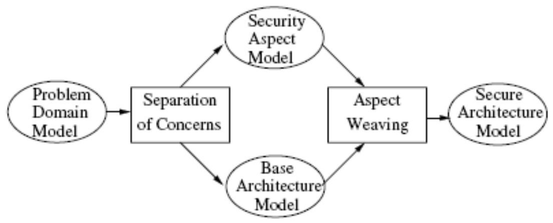

Figure 1. An AOSD Framework for Secure Software Architectures

An overview of the approach in [18] is given in Figure 1. The following provides a quick summary for each step in the framework illustrated in Figure 1:

- The *problem domain model* gives a precise description of the basic functionality and their relationship to the proposed system.
- The *base architecture model* defines the software architecture for the basic functional modules and their connections.
- The *security aspect model* describes the security requirements, defines the vulnerabilities and threats and provides mechanisms that enforce security policies into the software architecture.
- The *secure architecture model* is the software architecture model that the security polices have been correctly enforced.

The *base architecture model* in this approach is a SAM model with block grouping (a block is a part or whole of a predicate transition net that models a particular software module and is characterized by its *internal elements* and its *external elements*). Each block represents an autonomous software entity.

The *security aspect model* describe precisely the security relevant features of the proposed software system. This approach uses two language constructs to specify security aspects of software architectures. They are:

- *Architecture constructs* that define characteristics of the block-based architecture and also include attributes such as name, main task, sensitive information, etc.
- *Security constructs* that specify security policies and include LTL-like constructs for the problem domain.

Using these constructs, [18] defines how the security aspect applies to the *base architecture model* by using join points, pointcuts, and aspects. They define a pointcut as connectors that have the same security vulnerability and share the common security enforcement mechanism. An advice is a pattern that "specify the security enforcement mechanisms for pointcuts" [18]. Additionally, advice associates fragments of predicate transition nets with pointcuts, "which specify the system behaviors at every join points in particular join points" [18].

The *aspect weaving* step in this framework creates a software architecture by weaving aspect models with the base architecture model using the following steps:

1. **Locating the join points -** Pinpointing the location where the base architecture model and the aspect models (i.e., the security requirements) interact.
   - **a.** Analyze security vulnerabilities and threats to the software based on the security requirements.
   - **b.** Specify join point conditions for the connectors in the base architecture model. This typically shows what security vulnerability that the connectors in the base model are vulnerable to.
   - **c.** Check each connector in the base model to see if it meets the join point condition.
2. **Constructing advices -** Defining the behavior of the system in order to enforce security policies on the base software architecture.
   - **a.** Identify join points that have the same vulnerability and group them together as a pointcut.
   - **b.** Design a mechanism or an advice for each pointcut such that the vulnerability is mitigated.
3. **Weaving aspects -** Integrating the aspect models (i.e., security requirements) into the base software architecture.
   - **a.** Arrange a systematic way to search for joinpoints.
   - **b.** For each joinpont, modify the base architecture model according to the corresponding advice.

The authors claim that this approach offers a rigorous way to identify notion in aspect-orientation and to reason about the correctness of aspect weaving (not described here). Additionally, they claim that the join point model in their security framework has a powerful expressibility because of the hierarchical modeling ability of the software architecture, due to their use of SAM. Lastly, the authors claim that their approach supports a reusable and reliable design of secure software architectures.

In light of their claims, this paper only present preliminary results of applying their security framework using a formal aspect-oriented approach to build secure software architectures of a toy problem (a travel planner information system). This approach lacks any tool support and fails to address the scalability of their approach as the proposed system and security requirements gets larger. Lastly, the authors do not discuss the dependency between the aspect models and how to correctly partition of security aspects. Compared to the previous security framework, discussed in Section 2.1, however, this approach offers a more formal and structured process and is far more advanced and mature as a process in incorporating an AOSD approach in the design, development and implementation of security requirements.

2.3 An Aspect-Oriented Security Framework

The aspect-oriented security framework proposed in [14] is clearly not as developed as [6] in Section 2.1 and [18] in Section 2.2 but is aimed at creating a truly generic aspect-oriented framework that any specialized security framework should adhere to. That is, the approach proposed in [14] concentrates on defining the characteristics that any good AOSD security approach should contain and then how this might be achieved. In this section, we describe the generalized conclusions of this work in Section 2.3.1 and then the authors lessons learned from the development and implementation of an aspect-oriented security framework. This work, although not comparable to the previous approaches in its maturity, structure or applicability, is presented here to generalize the needed characteristics and encountered short-comings of an aspect-oriented security framework so that future proposed frameworks include what has been shown to be needed and avoid previous pitfalls.

2.3.1 An Aspect-Oriented Security Framework

The framework described in [14] identified the following primary characteristics needed in a security framework:

- *Proactive stance*. A security framework should be designed to be used as part of the development process so that security can be applied to the software system by default.
- *Global application*. A security framework should treat security as a crosscutting concern but also allow security analysts to apply security solutions globally while still giving them the flexibility to focus on only pieces of the system if necessary.

- *Consistent implementation*. A security framework should apply implementations consistently of the same solution. This should be achieved by automating the process of integrating the security solutions into the software system.
- *Adaptability*. A security framework should provide a full-featured "transformation engine and expressive but simple language for encoding generic directives for security solutions" [14]. It should ensure that the security framework can be used to implement a wide range of security solutions.
- *Seamless integration*. Any security framework or security framework tool should be easily integrated into the build process of a software system.

The authors claim that these features needed in a security framework "meld well with the strengths of the aspect-oriented program model" [14]. Surprisingly, however, the authors in [14] do not mention reusability of security policies as a desirable characteristic even though the security frameworks [6], described in Section 2.1, and [18], described in Section 2.2, mention it as a contribution characteristic of their aspect-oriented security framework.

Using these characteristics, the authors implemented a framework and tool target to address several common, implementation security problems in C programs. Specifically, the authors applied their security framework to address such prevalent security exploits as [14]:

- Buffer overruns
- Time-of-check-to-time-of-use
- Format string vulnerabilities
- Protection of communication channels
- Event ordering enforcement
- Type safety

The authors conclude, while their approach was helpful, an approach that implemented security policies at the design or architecture phases are more apt to consider globally applicable security threats or vulnerabilities. Unfortunately, few details of their approach don't allow for an adequate understanding of how to apply their security framework to other applications, much less to allow the ability to independently evaluate their approach.

2.3.2 Lessons Learned from An Aspect-Oriented Security Framework

Despite the lack of details provided for the aspect-oriented security framework [14] described in Section 2.3.1, the authors provide some lessons learned/obstacles encountered in [15]. These obstacles were derived from developing their security framework and then having developers apply it to an application in practice. The authors intend these lessons learned to be used by other practitioners when developing improved aspect-oriented security frameworks.

From [15], the lessons learned include:

1. **The KISS Principle**. The adoption by software developers and QA teams of a new software development approach or language into industry use typically requires an easy to understand, well-documented technique. This is particularly true for AOSD-based approaches since "aspects tend to invalidate the concept of well-defined, narrow interfaces" thus adding to the complexity of the software [15].
2. **Shifting Development Paradigms**. To get software professionals and the software industry to adopt a new software development paradigm demands case studies, empirical analysis and results to prove the advantages of adopting a new way of designing and developing software.
3. **Traceability**. It is necessary to have a security framework to have a mechanism to allow for traceability that a development team to maintain throughout the software development lifecycle**.** The framework in [15] did not have the ability for tracing security requirements throughout the development lifecyle, and this was the main complaint by the developers and QA teams when applying this security framework in practice.
4. **Early lifecycle security abstractions.** The approach in [15] allowed developers to separate security concerns from the program's main modules during the implementation phase rather than in the earlier development phases (requirements, design, etc.). The developers indicated that the ability to define code level security concerns during the design phase is critical to properly integrating security requirements in an AOSD-based security framework.
5. **Tool support**. As in any software engineering approach, the lack of tool support hindered the practicality, understanding, effectiveness and accuracy of using the AOSD-based security framework of [15] by developers in practice.

Although the lessons learned, listed above, may seem obvious to most software engineers, it is important to describe them since they came from comments made by actual software developers using the AOSD-based security framework of [14] in practice. Further, the lessons learned come from the mistakes made in [14] and thus should (hopefully) not be repeated in later AOSD-based security frameworks so that the AOSD and security community can quickly arrive at a practical, effective AOSD-based security framework that can be readily used in practice.

2.4 Aspect-Oriented Security Frameworks Discussion and Evaluation

Using the five lessons learned of [15] as an evaluation criteria for the AOSD-based frameworks [6], described in Section 2.1, and [18], described in Section 2.2 we see that these security frameworks make several of the same mistakes as [14] despite being published several years later.

The AOSD-based security framework in [6], we provide the following evaluation using the lessons learned of [15] as an evaluation metric:

- **The KISS Principle**. The use of UML and a UML-like way of defining security concerns is something that most software developers are familiar with allowing for a quick understanding of the framework. The description of the process, however, is not enough that it likely could not be successfully applied in practice.
- **Shifting Development Paradigms**. Again, since a UML-like language was used, software designers and developers may not be forced to make a large shift in their development paradigm to be able to incorporate security concerns as an aspect of their design and implementation.
- **Traceability**. The approach provides no mention or mechanism at how traceability could be achieved. However, we believe that the way in which they model a security concern (as a UML Collaboration Diagram), it may not be difficult to manually verify and trace a security requirement throughout the development lifecycle.
- **Early lifecycle security abstractions.** The framework is aimed at the design phase of a security-critical software application. Thus, it allows for early lifecycle security abstractions.
- **Tool support**. Does not provide any tool support although it was mentioned as future work. Note however, a current search could not find tool support for this security framework.

The AOSD-based security framework in [18], we provide the following evaluation using the lessons learned of [15] as an evaluation metric:

- **The KISS Principle**. The use of the Software Architecture Model (SAM), Petri nets and temporal logic in the definition of the software architecture and security concerns may be intimidating and difficult for those in industry that currently do not use such an approach. Yet, since [18] was looking to develop a more formal AOSD-based security framework, adhering to this principle may be difficult.
- **Shifting Development Paradigms**. Again, to ask industry to adopt the use of SAM, Petri nets and temporal logic in their development process when it is not currently used is a lot to ask without proven, empirical results showing the advantages of this framework.
- **Traceability**. Like [6], [18] provides no explicit support for traceability of security concerns from design to architecture to implementation. However, unlike [6], [18]'s heavy formalisms would complicate a manual trace of a security requirement throughout the development lifecycle.
- **Early lifecycle security abstractions.** The framework is aimed at the architecture phase of a security-critical software application. Thus, it allows for early lifecycle security abstractions, however, not as early in the development lifecycle as [6].
- **Tool support**. Does not provide any tool support although it was mentioned as future work. Tool support for this security framework is critical because of the heavy formalisms and the notation-intense definitions of a software architecture and security concerns. Again note, a current search could not find tool support for this security framework.

Thus, neither AOSD-based security frameworks measure up to the standards required by [15]. Although the security frameworks of [6] and [18] provide innovative approaches, it is clear that they need to be further developed and integrated into the development lifecycle and better supported with tools and empirical results before they are used in a software industry setting.

3. Aspects in Software Security: Other Approaches in Using an Aspect-Oriented for Software Security

This section investigates beyond proposed security frameworks employing an aspect-oriented approach. Research in adopting an aspect-oriented approach in securing coding, AOP modeling and verification of access control and distributed aspects remain active research interests in regards to software application security. This section specifically addresses the active research pursuits in these areas of adapting an aspect-oriented approach to developing, designing and implementing the security requirements of a security-critical software system.

3.1 Secure Coding

New programming paradigm promoting separation of concern is Aspect-Oriented Programming. Security information in the coding can be separated as a concern and can be encoded separately from the base code. The popular Object-oriented programming supports such kind of modularity to an extent, which doesn't provides enough flexibility and adaptability and just good in separating concepts that can be mapped easily to the objects. Modeling security in OOP's is difficult, in the sense that we can write a class for security which other objects can call this security class for each checking. This incurs complete exhaustive spreading of call code through out the application code base.

In the above case, if one forgets a critical checking, penetrate and patching process is really exhaustive and very expensive. And central security class is difficult to recover from the critical check. This leads to the separation of the security as a concern in programming base. Aspect-oriented programming gives more flexibility in addressing this concern and solving it. The paper [17] has proposed an AOP extension to the C programming language. This extension gives greater benefit in the secure coding. An AOP technique allows an application developer to just focus on the application and doesn't need to have any knowledge about the security while programming. Later, a security expert can model the secure segment and can easily weave it into the application base code.

It is also understandable that developers are not and need not be good in writing secure code. One of the popular examples is buffer overflow problem, which exploits the C code. The possible solution for this kind of known security issue is penetrate-and-patch strategy through out the base code. The paper also addresses that the reasons for such insecure code pattern are no comprehensive design time methodologies, lack of comprehensive resource tools to help write secure programs, lack of expertise with both application and security knowledge. Some of the more common problems include misuse of security protocols and unrealistic view of what a system should consider "trusted".

Tools that try to provide security assurance, vulnerability analysis help to prevent security vulnerabilities, which are after-the-fact tools. They don't address how to design and implement the secure code. The main principle of the paper [17] is to give a proactive approach by the use of AOP extension to the C language. The extension principles are minimizing the security knowledge requirement for a developer, abstraction of security related elements from the application, increase the clarity of the program, language generic security policy specification, reducing the effort of developing secure application, effectiveness and easy way of expressing policies, legacy source code with known problems should be able to benefit from this effectively, reusability of the security policies across different applications.

The language the paper [17] proposes allows inserting the advice code before the point of interest; replace the point of interest, after the point of interest. The types of locations to operate on are,

1. Calls to functions
2. Function definitions
3. Pieces of functions

One of the example for aspects that replaces the vulnerable rand() function in C Language, is given below.

```
aspects secure_random{

    int secure_rand(void) {
      /**
       * Secure call to random defined here
      **/
    }

    funcCall<int rand(void)> {
      replace {
        secure_rand();
      }
    }
}
```

In the above example, secure_rand(void) is a function definition for secure random number generation. The keyword 'funcCall' specifies that matches call to functions. In this case, the calls to rand() is caught and replaced by the secure_rand() function. The extension language weaves the aspects into the regular C program to single C program at the compile time. This extension supports three type of matching facilities, namely:

1. Name
2. Type
3. Argument

Name matching allows the programmer to give an interest in functions, files, modules whose names matches a pattern, for which they use the "?" construct to specify the wildcards in names. To support the type specification they used "any?" or "any*" to specify one type or all type. In order to match the variable argument, "…" operator is used.

This paper ignores the problem of order and precedence concern that is the order in which the aspects are weaved to the base code. It disallows all the conflicts. Applying the AOP to security has various usages, namely:

1. automatic error checking on security critical calls
2. implement buffer overflow protection techniques
3. automatic logging of security relevant data

4. replacing generic socket code with SSL socket code
5. specifying privileged sections in the program to go through set of lock-sown procedures

They also mention aspect weaver with suite of security aspects is language independent. They have also given the example above a complete implementation, and how the code looks after woven. So the paper identifies some of the major problem in software security and proposes an extension for C language to use AOP concept to alleviate those security problems.

3.2 AOP Modeling and Verification of Access Control

In the paper [16], they address the inadequate support of access control for web applications, and propose how the use of AOP techniques solves the problem. They give an extension of UML based web engineering (UWE). In the web application, implementation of complex business processes faces the problem of access control over the pages which the user can access. Access control is commonly modeled as the part of web navigation in each and every element, introducing redundancy into the models. Access control is a cross cutting concern in web applications, applies to several classes of web pages. UWE separates the web application as the content, the navigation structure, business process and presentation. Based on navigation model of UWE, they use the UML state machine to model access control in web applications.

In web application, if the navigation nodes need to be given access control then the link-based access control is given in traditional method. But the navigation node can be accessed via external link, under which case the link based access control fails. Therefore the access control should be a part of the behavior of the protected nodes. This paper extends the UWE by associating to each navigation node one state machine which specifies the detailed behavior of the navigation node (Figure 2). This a naïve approach.

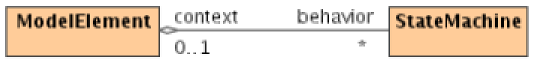

Figure 2. UML Metamodel: Model Element & State Machine

In this basic approach it is very difficult to associate same statemachine for multiple model elements. Say, if we have same security policy state machine, it introduces redundancy for each model element. So they give an extension of the UWE metamodel, introducing the concept of aspects into UWE. All the classes that is to be associated to same rules are put together in a single aspect AccessControl. So, similar association is done to this aspect and not to each and every classes contained in the AccessControl aspect. The access control rules are defined in the aspect that contains all the navigation nodes of the same access control rules. Modeling of access control in web application is modularized this way, and redundant specifications can be avoided.

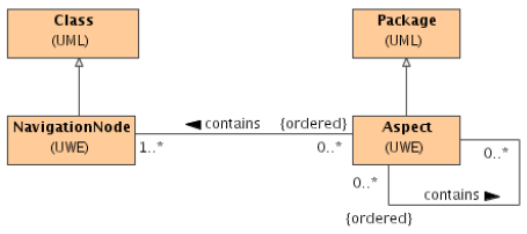

Figure 3. Extension of UWE Metamodel by Aspects

This can also be nested Aspects of Aspects and also can be extended to multiple aspects in Aspects.

An example of web application is a publication library, where each node needs to be protected by access control rules. This can be modularized by the use of this approach, which is shown below.

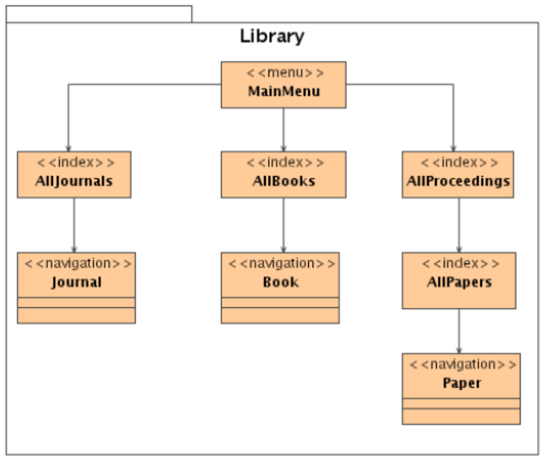

Figure 4. Library Web Application – Navigation Diagram

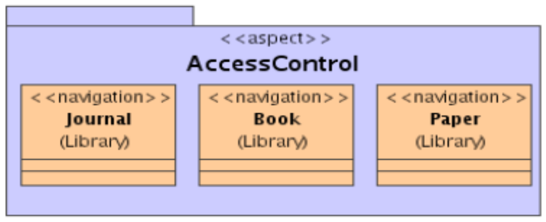

Figure 5. Aspect AccessControl Containing Concerned Nodes

This paper thus address the access control cross cutting concern in a web application, which can be easily modularized using the aspect oriented modeling approach. Also the paper [16] similarly uses the aspect concept for describing the access control properties of popular RBAC model. Also it shows how the verification composition of access control features can be supported by the use of aspect concept. But the systematic verification model doesn't gives an automatic verification model. The well known paper [5] on modeling security concern in an aspect based approach, also addresses the above discussed issues and describes the strategies to analyze the security concern in various functional concern effectively using aspect oriented approach. This approach is based on the UML templates and UML collaborations. Weaving of the aspects with the base model is primarily obtained by merging model elements with the same name.

3.3 Distributed Aspects

The paper [10] discusses the distributed related concerns. This paper proposes the notion of remote pointcuts that can match events on remote hosts, including the support for remote sequences. It also allows distributed advise execution. Finally it provides the model of distributed aspects which addresses deployment, instantiation and data sharing issues. They have extended JAsCo to support dynamic aspects. They have explained this concept taking the example of data cache and replication problem. They proposed this language as AWED which enables the matching of the remote join points by the remote pointcuts, and all corresponding associated aspects is executed in remote hosts. This gives the support for multiple host aspect execution and multiple host joint point catching. The remote sequence concept allows one to give the order of precedence and catch accordingly.

This paper also allows the advice to give the declaration of the concept of Group, where multiple hosts can be grouped together for the remote pointcut or aspects execution. Also, it allows the synchronous and asynchronous remote aspect execution. At the real implementation a remote proxy aspect is generated at the joint pointcut host and redirecting the catch to the remote aspects. This paper addresses and shows how this approach helps greatly improving the complete cache replication and solving the issues effectively.

3.4 Discussion

As we have seen in all the papers above the security is a concern which prevails in any context. When we take a scenario in the context of application, the security concern is present across various code segments. It is well known crosscutting concern, which can be effectively and efficiently handled using the AOP techniques.

Also, consider the implementation of a complete system in OSI architecture (Fig. 7). There will be various layer dependent security issues which runs across layers. Security is also a cross layer concern, using AOP we can handle more easily

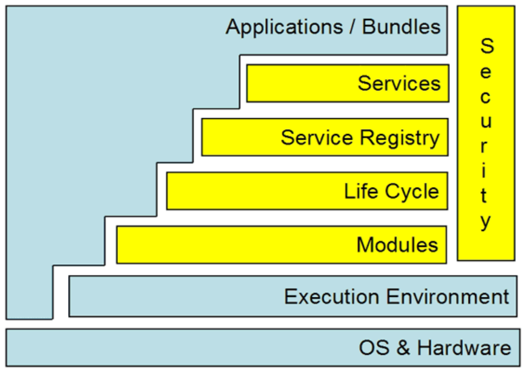

Figure 6. Security - A Crosscutting Concern

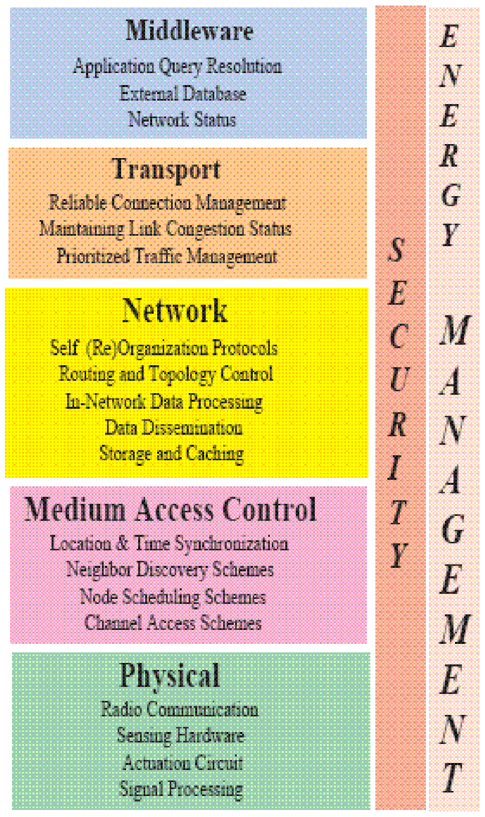
Middleware
Application Query Resolution
External Database
Network Status
Transport
Reliable Connection Management
Maintaining Link Congestion Status
Prioritized Traffic Management
Network
Self (Re)Organization Protocols
Routing and Topology Control
In-Network Data Processing
Data Dissemination
Storage and Caching
Medium Access Control
Location & Time Synchronization
Neighbor Discovery Schemes
Node Scheduling Schemes
Channel Access Schemes
Physical
Radio Communication
Sensing Hardware
Actuation Circuit
Signal Processing
SECURITY
ENERGY MANAGEMENT

Figure 7. Cross Layer Security Concern

Similarly, security is not a localized concern. It is a distributed concern which is present across the entire network globally. It is very difficult and highly expensive to implement this kind of concern in a regular programming technique. The Aspect Oriented technique gives a greater flexibility to address this problem and solve this security concern across different machine. As proposed by the paper [10], the distributed concerns can be solved by the use of aspects effectively, the security concern which runs over the machines can be solved very effectively in a cost efficient way.

4. Concluding Remarks

Clearly, assuring stakeholders that a security-critical software system correctly satisfies the security properties placed upon it will continue to be an important task for successful applications. The crosscutting nature of security requirements complicates the design, development and implementation of software systems with many security requirements (e.g., security requirements being tangled in requirements and design documents and in the actual implementation, traceability of security requirements from requirements and design into actual implementation, etc.). Fortunately, adopting an aspect-oriented software development (AOSD) approach in developing, designing and implementing eases the complexities of crosscutting requirements, such as many security requirements. AOSD proposes solutions to better modularize crosscutting requirements (i.e., concerns) by removing them from the main modules and allotting them into a separate module that then applies to certain points of execution in the main modules. This then detangles the crosscutting concerns and allows for a more modularized, manageable software architecture.

This paper described several approaches to incorporating an aspect-oriented viewpoint when developing, designing and implementing security requirements in a software system. Several AOSD-like security frameworks were reviewed as well as other approaches using AOSD when handling security concerns. The high number of and wide-ranging approaches indicate that current state of research in adopting and AOSD approach into the implementation of security requirements is in its infancy and that no agreement within the AOSD or security community has been reached as to which approach is most suitable. Thus, research in this area will likely continue until a suitable approach or approaches are published and agreed upon by the AOSD and security community with enough empirical results to prove that it, indeed, provides a superior solution.

5. Acknowledgements

This survey was completed to meet the final project requirement of COM S 610 - Seminar on Aspect-Oriented Software Development led by Dr. Hridesh Rajan, offered by the Department of Computer Science at Iowa State University during the Spring 2006 semester. The authors would like to thank Dr. Rajan for his guidance throughout the semester as well as our classmates for providing feedback on an earlier draft of this work.

6. References

[1] Baumeister, Z. and K. Knapp, “Aspect-Oriented Modeling of Access Control in Web Applications”. In *Workshop on Aspect Oriented Modeling* (AOM’05), 2005.

[2] R. Bodkin, "Enterprise Security Aspects", In *AOSD Tech. for Application-Level Security* (AOSDSEC'04), 2004.

[3] Chargi, A., and M. Mezini, “Using Aspects for Security Engineering of Web Service Compositions”, In *Proc. IEEE Int’l Conf. on Web Services* (ICWS’05), pp. 59-66, 2005.

[4] Devanabu, P. and S. Stubblebine, "Software Engineering for Security: A Roadmap", In *Proc. Conf. Future of Software Eng., ICSE'00, Special Volume*, pp. 227-239, 2000.

[5] Georg, G., France, R. and I. Ray. “An Aspect-Based Approach to Modeling Security Concerns”. In *Proc. Workshop Critical Systems Development with UML*, pp. 107–120, 2002.

[6] Georg, G., Ray, I., and France, R., “Using Aspects to Design a Secure System”, In *Proc. 8th IEEE Int’l Conf. on Eng. of Complex Computer Systems* (ICECCS’02), pp. 117-128, 2002.

[7] Haley, C.B., Laney, R.C. and B. Nuseibeh, "Deriving Security Requirements from Crosscutting Threat Descriptions", In *Proc.3rd Int'l Conf. Aspect-Oriented Software Development* (AOSD'04), pp. 112-121, 2004.

[8] Kiczales, G. et. al., "Aspect-Oriented Programming", In *Proc. European Conference on Object-Oriented Programming* (ECOOP'97), 1997.

[9] Kiczales, G. et. al., "Overview of AspectJ", In *Proc. European Conference on Object-Oriented Programming* (ECOOP'01), 2001.

[10] Navarro, L., et. al., “Explicitly Distributed AOP Using AWED”. In *Proc. 5th Int’l Conf. Aspect-Oriented Software Development* (AOSD’06), pp. 51-62, 2006.

[11] Rosenhainer, L., "Identifying Crosscutting Concerns in Requirements Specifications", In *Early Aspects 2004: Aspect-Oriented Requirements Eng. and Architecture Design Workshop*, pp. 49-58, 2004.

[12] Rashid, S., Moreira, A. and J. Araujo, "Modularisation and Composition of Aspectual Requirements", In *Proc. 2nd Int'l Conf. Aspect-Oriented Software Development* (AOSD'03), pp. 11-20, 2003.

[13] Ray, I., France, R., Li, N. and G. Georg, "An Aspect-Based Approach to Modeling Access Control Concerns", *Journal of Info. and Software Tech.*, 46(9), July 2004, pages 575-587.

[14] Shah, V. and F. Hill, "An Aspect-Oriented Framework", In *Proc. DARPA Info. Survivability Conf. and Exposition* (DISCEX'03), pp. 22-24, 2003.

[15] Shah, V. and F. Hill, “An Aspect-Oriented Security Framework: Lessons Learned”, In *AOSD Techn. for Application-Level Security* (AOSDSEC), 2004.

[16] Song, E., Reddy, France, R., Ray, I., Georg, G. and R. Alexander, “Verifiable Composition of Access Control and Application Features”, In *ACM Symposium on Access Control Models and Technologies* (SACMAT’05), 2005.

[17] Viega, J., Bloch, J.T., and P. Chandra, "Applying Aspect-Oriented Programming to Security", In *Cutter IT Journal*, 14(2):31-31, 2001.

[18] Yu, H. et. al., “Secure Software Architectures Design by Aspect Orientation”, In *Proc. 10th Int’l Conf. on Eng. of Complex Computer Sys* (ICECCS’05), pp. 45-57, 2005.