Introducing Security Aspects with Model Transformation

Jorge Fox, Jan Jürjens
Technische Universität München
Boltzmannstraße 3
D-85748 Garching
{fox,juerjens}@in.tum.de

Abstract

Aspect Oriented Programming and subsequently Aspect Oriented Software Development have received great attention recently and constitutes an interesting field of research in computer science. The goal of this paper is to propose a more precise understanding of aspects based on the idea of crosscutting concerns in view of model transformation. This proposal considers security aspects as an example of a behavior applied over a desired software product. This implies improving the actual definition of aspects. The work introduces the main current concepts of aspect, defines aspects as behavioral entities, presents examples, and outlines a method for model transformation based on the proposed definition.

1. Introduction

Aspect Oriented Programming has been the focus of an ever growing attention and research in computer science. Names as Aspect Oriented Software Develpoment (AOSD), Aspect Modeling, Early Aspects, and the like can be found recently in the literature. Despite all the work done, it still seems necessary to provide some of these concepts with a more precise meaning in relation to its practical utility.

Aspect Modeling and Model Transformation with aspects are an importante line of research within AO. We will therefore in this paper explore aspects in relation to transformation of models, particularly in relation to introduction of security aspects in UML models.

The present paper presents a definition of aspects understood as a desired behavior affecting various execution units. This approach allows us to propose a methodology for transformation of models with aspects, based on a formal transformation language (BOTL) [13], as well as a syntactic and semantic proposal for representing aspects as in UMLsec [11].

Related work is based mostly on defining aspects in terms of frameworks, roles, mixing of UML models with frameworks using OCL [20], or in some cases by techniques close to what [3] call direct manipulation as in [8, 19]. On the other hand, as [8] show, aspect composition can lead to conflicts in the resulting model, in which case, the system developer must resolve the conflict manually. By defining an aspect as we propose in section 3 and expressing them formally, we achieve a higher level of abstraction which under frameworks as BOTL and UMLsec [11] allow us to prove the properties of the desired transformation. The latter is work on progress. In this work, we focused ourselves on security aspects, in order to explore a means of representing aspects in general. It is our hypothesis that this approach is useful for other kinds of aspects, defined as in section 3, and represented as exemplified in section 4.

The rest of the paper is structured as follows. Section 2 explores the current definitions of aspect. Section 3 presents examples of aspects from the literature and gives our definition of aspect. Section 4 introduces our proposal for expressing an aspect syntactically and semantically based on the security formalism in [11] and outlines model verification against the consistency of desired security characteristics. Section 5 describes our proposal toward model transformation with aspects. Finally, Section 6 presents our conclusions.

2. Current definitions of aspects

We are now going to explore some of the current definitions of aspect found in the literature.

In [2] we find that aspects are issues not well localized in functional designs, such as:

synchronization, component interaction, persistency, security control, fault tolerance mechanisms, quality of service, and the like; these are considered concerns that constitute typical candidate aspects. In this case we need then first to define issue as well as concern so that the concept of aspect acquires meaning.

Moreover, [18] indicate that compatibility, availability and security requirements are crosscutting concerns. Also, exception handling, multi-object protocols, synchronization, and resource sharing would be extended across the source code if only using traditional implementation techniques, like Object Orientation, thus implying that those behavioral elements are candidate aspects as well.

In our view, one of the central unanswered issues in Aspect Orientation (AO), as Ossher in [5] mentions, is that “one of the hard things about crosscutting concerns is understanding just what cuts across what”.

There is a need for a clear definition of aspect, even before we aim at achieving aspect identification, “weaving”, and modeling of aspects. Take for instance, the early stage of software requirements; it is at that stage that many of the later difficulties in software development can be generated. Therefore a great effort is on progress to identify aspects at the requirements stage, see for instance [1, 7, 9, 15, 17].

Despite the efforts, in most cases it is left to the criterion of the analyst to identify software concerns, out of these, select candidate aspects and test them. As [12] stated: “Designers must rely on their discretion to decompose the problem effectively”. The later seems to be astray from a software engineering approach.

There is also research in aspect mining in code as in [9]. Research on software evolution and AO is based also on aspect mining in existing code [14, 16]. Nevertheless, no conclusive work seems to have been as yet achieved.

We believe that the problem can be traced to a deeply rooted belief in the early definitions of aspect. As long as the work is based on the notion of “crosscutting concern” and take for granted definitions such as “a concern is any matter of interest in a software system,” [22] the following kind of questions will remain open. Questions like: is a method a crosscutting concern, i.e. an aspect? If so, then how do we distinguish a clone in code from an aspect, and an aspect from just an erroneous implementation?

A bottom-up approach is not devoid of difficulties in view of the primary goal of AO, i.e. to achieve Separation of Concerns (SoC) as defined in [4, 5, 6]. The reason for the above might well be that if we consider an aspect as a collection of Advice and Join Points, or pointcut designators coupled with advice [21, 23], such an approach does allow for the creation of language extensions to the Object Oriented Paradigm (OOP). However, a programmer using this kind of aspectual language implementations may even damage an originally well designed software product, or a product being developed by different work teams, as shown in [16].

3. A definition of aspect

We believe it necessary to remember the primary motivation of AO, in our view, to help reducing complexity. Like [2] suggest “the goal of AOP is to provide methods and techniques for decomposing problems into a number of functional components as well as a number of aspects that crosscut functional components”.

In some of the early works on AOP we find the following motivations.

“AOP is based on the idea that computer systems are better programmed by separately specifying the various concerns (properties or areas of interest) of a system and some description of their relationships, and then relying on mechanisms in the underlying AOP environment to weave or compose them together into a coherent program” [6].

In this paper we propose to accept the definition presented in [10] as a point of departure in order to further define aspects. According to Ivar Jacobson [10], an aspect is a “modular unit of crosscutting implementation”. Please note that both [2, 10] specify that aspects are functional units. We will now explore the example presented in [10], which can be broadly accepted as a typical aspect example.

3.1 Call handling and traffic recording example

The example is about a Telecom Switching System. There is a Call Handling Subsystem that further requires a Traffic Recording Implementation. The latter is added in [10] as an aspect, though its author names these “Base” and “Existion”. Both are shown in Figures 1 and 2.

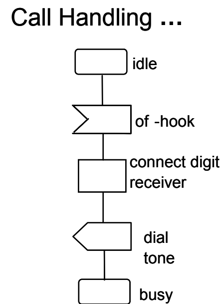

Figure 1. Call handling flow diagram. [10]

On the one hand, Figure 1 presents flow of activities associated with a Use Case "Call Handling" as proposed in [10]. We might have chosen a similar example, but this one seems neat and precise enough. On the other hand, we have in Figure 2 the activities associated with an additional desired behavior extending the "Call Handling" (flow of activities) as indicated in the figure by the so called extension points. We focus ourselves at the moment, not on the issue of extension points or the relation between what [10] calls "Base" and "Existion." We aim at drawing attention to the fact that the flow of activities represents a desired behavior. As we mentioned, this behavior is expressed in a sequence of activities, and aims at achieving a given goal. In this example, in Figure 2 the goal is to measure the average traffic from subscribers [10], another important consideration we would like to point at is that this goal is set by some stakeholder. The fact that the sequence of activities acquires meaning in respect to a given stakeholder or stakeholders provides pointers for aspect identification.

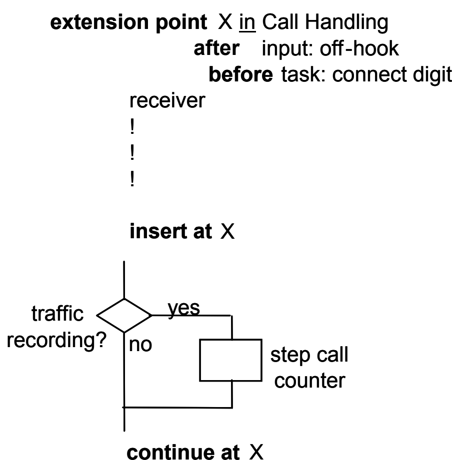

Figure 2. Traffic recording flow diagram. [10].

Both sequence diagrams can also be expressed in use cases, and also be described with state machines, which is out of the scope of this work. However, we will show the utility of our aspect definition with model transformations in state machines with an example in section 5.

3.2 Aspects in a case study

In [16] the author implements a software evolution case study in order to appraise the capabilities of aspect orientation in view of the maintainability of software systems. For the purposes of this paper, we selected two of the aspects he presents as trial case.

The author realized a system called "MySABoM", namely My Simple Address Book Manager.

In the realization of this software product the author in [16] modularized the desired behavior on the one hand in a Data Model following the Object Oriented Paradigm, on the other he identified pieces of crosscutting functionality, meaning aspects. He based himself on the following definition of aspect as "a modular unit that cross-cuts the structure of other modular units."

He defined four aspects in his work:

- Authentication
- Authorization
- Tracing
- Presentation of portions of the user interface according to the user's role

We selected the first two with the aim of achieving a more thorough understanding of the subject matter.

3.2.1. Authentication aspect. From the description of the aspect we select its goal. In this case, it has two goals. First, to define which parts of the system shall be protected. Second, to get the log on information from the user.

3.2.2. Authorization aspect. In this case, the aspect has one goal, and it will help us illustrating our definition in section 3.3. Its goal is to enforce that "a user with the role 'Reader' shall be allowed to change data if and only if he owns these data."

In both cases, we may analyze its representation at different levels. Either at the code level, as these were actually implemented in [16], or consider them at an architectural level. The latter constitutes the focus of our attention.

If we look at them in this manner, we may note that once the goal is formulated in terms of behavior, it is susceptible of being translated into some formal specification or a given set of rules. The latter will

serve our purpose of model transformation with aspects and will be explored in section 5 with a different case study.

3.3 A behavioral definition of aspect

Based on the preceding examples, we propose the following definition of aspect as a set of units of execution representing a desired behavior whereas this behavior relates to the point of view of one or more stakeholders, in the context of the software development lifecycle, and affects i.e. modifies the behavior of other units of execution.

In this sense, an aspect represents a desired functionality in a software product that modifies the behavior of more than one software entity. This functionality is the semantic formulation of the desired behavior and the relations among various units of execution. In other words, an aspect is a desired functionality that involves various other units of execution.

We believe this approach helps improving the widely accepted definitions of aspects such as "concerns that cut across other concerns", it brings us a step forward in its understanding, and allows us thereon to propose a method for model transformation with AO as presented in section 5.

As an example of aspect, and the means we propose to represent them, we introduce now one of the security stereotypes defined in UMLsec.

4. An overview of UMLsec

We will now introduce UMLsec [11] as a method for giving a precise semantic body to aspects as defined in 3.3. By doing so, we allow for transformation of models, models as in UML, as explored in section 5.

We make use of the extension of the UML [11] for secure systems development called UMLsec. Recurring security requirements, such as secrecy, integrity, and authenticity are offered as specification elements by the UMLsec extension. These properties and its associated semantics are used to evaluate UML diagrams of various kinds and indicate possible security vulnerabilities. One can thus verify that the desired security requirements, if fulfilled, enforce a given security policy. One can also ensure that the requirements are actually met by the given UML specification of the system. UMLsec encapsulates knowledge on prudent security engineering and thereby makes it available to developers who may not be experts in security. The extension is given in form of a UML profile using the standard UML extension mechanisms. *Stereotypes* are used together with *tags* to formulate security requirements and assumptions on the system environment. *Constraints* give criteria that determine whether the requirements are met by the system design, by referring to a precise semantics mentioned below.

The tags defined in UMLsec represent a set of desired properties. For instance, "freshness" of a value means that an attacker can not guess what its value was. Moreover, to represent a profile of rules that formalise the security requirements, the following are some of the stereotypes that are used: «critical», «high», «integrity», «internet», «encrypted», «LAN», «secrecy», and «secure links». If relevant, their profile also contains the possible attackers associated to them as shown in Table 1.

Table 1. Attackers and threats per stereotype in the UMLsec

Stereotype	**Threats** *default*()	**Threats** *insider*()
Internet	{delete, read, insert}	{delete, read, insert}
Encrypted	{delete}	{delete, read, insert}
LAN	Ø	{delete, read, insert}

The definition of the stereotypes allows for model checking and tool support. As an example consider «secure links». This stereotype is used to ensure that security requirements on the communication are met by the physical layer. More precisely, when attached to a UML subsystem, the constraint enforces that for each dependency d with stereotype $s \in \{<< \text{secrecy} >>, << \text{integrity} >>, << \text{high} >>\}$ between subsystems or objects on different nodes, according to each of the above stereotypes, there shall be no possibilities of an attacker reading, or having any kind of access to the communication, respectively. A detailed explanation of the tags and stereotypes defined in UMLsec can be found in [11]. The extension has been developed based on experiences on the model-based development of security-critical systems in industrial projects involving German government agencies and major banks, insurance companies, smart card and car manufacturers, and other companies. There have been several applications of UMLsec in industrial development projects. There exists extensive tool-support which allows the developer to automatically analyze UMLsec models

with respect to the security requirements which are included as stereotypes against the threat scenario which is derived from the information about the physical layer of the system (see figure 3).

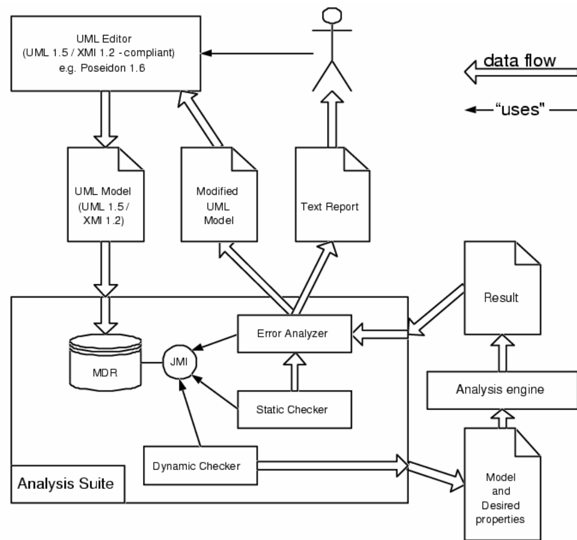

Figure 3. Overview of the proposed model verification and correction methodology

5. Transformation of models with aspects

In this section, we propose that the introduction of a desired behavior over a given model can be represented as a function Φ with the following parameters, a model S, (in this case, every subsystem instance in a UML model) and the semantic description of an aspect δ (transformation rules). In this way, Φ transforms S in S' with the introduced behavior δ.

To exemplify this, consider the package Channel in Fig. 4. as model S, the UMLsec stereotype encryption as aspect δ. The resulting model S' is shown in Figure 5.

Let us focus our attention on the Sender state machine in Figure 4 as part of the model S. The encryption behavior added to it produces a modified state machine in S' with a new State Request, added between the early state Wait with the original transition "send(d)" modified into "send(d) / request()". Moreover, state Wait in S has been added in S' with a function and a counter "entry/i:=i+1". The transition "/transmit(d)" in S has been added with the required security elements in S' according to the stated semantics of the encryption aspect. The resulting state machine is shown in Figure 5. The Receiver state machine is also transformed accordingly.

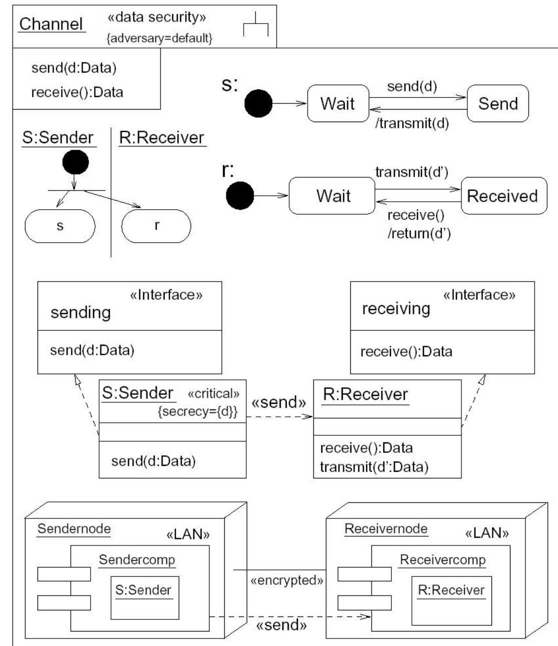

Figure 4. Security example: sender and receiver

This transformation is susceptible of being performed on every channel marked with stereotype <<encrypted>>. The original state "Send" remains unchanged, this means, that the base functionality is enhanced by the stereotype representing the new required functionality. We are not talking about "weaving" here, what the Bidirectional Object oriented Transformation Language (BOTL) [13] with the respective transformation rules performs is simply more powerful than related approaches [8, 19, 20]. Because BOTL is based on graph transformation and the representation of aspects in a formal semantic may allow us further to verify the results of the transformation against a given framework as for instance UMLsec.

As we already mentioned, we intend to base our work on the BOTL, and the related tool created at the Technical University of Munich. In this case, BOTL takes a set of transformation rules and transforms a given model. In our case, the tool plays the role of the proposed function Φ.

The above paragraphs demonstrate the significance of the definition presented in Section 3.3, because it allows us to represent aspects as units of desired behavior, provide them with a semantic representation and from there on apply it as the set of rules for BOTL. In order to transform a given UML model including in it the desired aspects.

Please note that the above transformation is not necessarily dependent on BOTL, but a similar language transformation and tool can be applied.

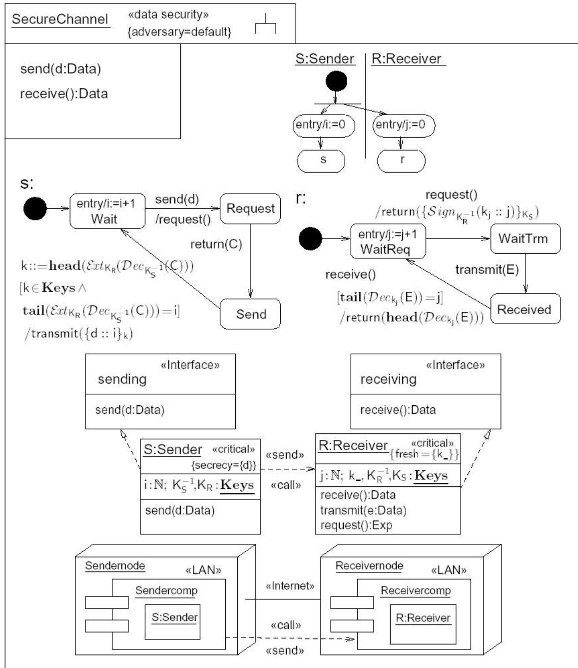

Figure 5. Security example: secure channel

As exposed in this section, in the end we aim at being able to formally verify whether the resulting model expresses the desired behavior without hindering its original characteristics.

As shown in Figure 3, we suggest that after introducing the aspect a validation of the model can be performed. The former may allow us to verify that the model *S*' be consistent with the desired characteristics.

6. Conclusion

The aim of the present paper was to examine a means of representing aspects with the aim of introducing them on a given model. We achieved this in the first place, by focusing on aspects as a set of units of execution embodying a wanted functionality. In the second place, by expressing them as specification elements with associated semantics. And finally, by regarding such semantic elements as transformation rules.

This approach allows us to achieve SoC at the modeling level. Therefore, our approach toward SoC allows for solutions that are independent of platform or programming language, hence devoid of the shortcomings of actual aspect language implementations. Indeed, viewing aspects this way, we actually need no aspect language because the model can be later implemented with existing Object Oriented techniques.

We also provided examples of aspects from other cases in the literature. From these cases and the current definitions in the field we apprehended what we believe is the core of an aspect. The definition introduced here allows for a subsequent formalization of aspects and as a result of this, we may achieve a more comprehensive transformation of models than the related AO approaches provide. The definition proposed here may also prove useful for related problems such as aspect mining in requirements and code.

As a future line of research, we aim at exploring our proposal with aspects other than security related ones. Furthermore, we aim at expressing a software product at the modeling level as a set of desired characteristics, i.e. concerns, and the relations among them. In this way, we may achieve a more abstract view of a software product and strive for expressing it with formal methods e.g. predicate calculus, which would allow to verify the consistency of the desired characteristics even before translating it to a model in UML.

7. References

[1] J. Araùjo and P. Coutinho, "Identifying Aspectual Use Cases Using a Viewpoint Oriented Requirements Method", in *Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design Workshop, 2nd International Conference on Aspect Oriented Software Development*, Boston, 2003.

[2] Czarnecki, K and Eisenecker, U., *Generative Programming: Methods Tools and Applications*, Addison-Wesley, May 2000.

[3] K. Czarnecki, and S. Helsen, "Classification of Model Transformation Approaches", In *Workshop on Generative Techniques in the Context of Model-Driven Architecture* (OOPSLA '03), 2003, pp. 1-17.

[4] Dijkstra, E.W., *A Discipline of Programming*, Prentice Hall, Englewood Cliffs, 1976.

[5] T. Elrad, M. Aksit, G. Kiczales K. Lieberherr, and H. Ossher, "Discussing Aspects of AOP", *Communications of the ACM*, vol. 44 No. 10, October 2001, pages 33-38.

[6] T. Elrad, R.E. Filman, A. Bader, "Aspect-Oriented Programming", *Communications of the ACM*, vol. 44 No. 10, Oct 2001, pp. 29-32.

[7] G. Georg, R. Reddy, and R. France , “Specifying Cross-Cutting Requirement Concerns, *7th International Conference UML 2004* (Proceedings), *Springer*, Lisbon, Portugal, October 2004, pp. 113-127.

[8] G. Georg, R. France, and I. Ray, “Composing Aspect Models”, *The 4th AOSD Modeling With UML Workshop*, UML 2003, October, 2003.

[9] J. Hannemann, Aspect Mining Tool, http://www.cs.ubc.ca/~jan/amt/, November 2003

[10] I. Jacobson, “Use Cases and Aspects - Working Seamlessly Together”. *Journal of Object Technology*, ETH Zurich, Vol. 2, No. 4, July-August 2003, pp. 7-28.

[11] Jürjens J., *Secure Systems Development with UML*, Springer-Verlag, 2004

[12] Kiselev I., *Aspect-Oriented Programming with AspectJ*, SAMS Pub., USA, 2003.

[13] F. Marschall, and P. Braun, “Model Transformations for the MDA with BOTL”, In *Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications*, CTIT Technical Report TR-CTIT-03-27, Univeristy of Twente, June 2003.

[14] T. Mens, K. Mens, T. Tourwe. „Aspect-Oriented Software Evolution”, *Special Theme: Automated Software Engineering, ERCIM News* No. 58 : 36 –37, July 2004.

[15] B. Nuseibeh, “Crosscutting Requirements”, in *Proceedings of the 3rd International Conference on Aspect Oriented Software Development (AOSD 2004)*, Lancaster, ACM, 2004, pp 3-4.

[16] F. Prilmeier, *AOP und Evolution von Software-Systemen*, Master’s Thesis, Technische Universität München, Munich, November 2004.

[17] A. Rashid, P. Sawyer, A. Moreira, and J. Araujo, “Early Aspects: A Model for Aspect-Oriented Requirements Engineering”, in *IEEE Joint International Conference on Requirements Engineering*, Essen Germany, 2002, pp. 199-202.

[18] A. Rashid, A. Moreira, and J. Araújo, “Modularisation and Composition of Aspectual Requirements”, In *Proceedings of the 2nd international conference on Aspect-oriented software development*, ACM, Boston, March 2003, pages 11-20.

[19] I. Ray, R. France, N. Li, and G. Georg, "An Aspect-Based Approach to Modeling Access Control Concerns", *Information and Software Technology*, 46(9), July 2004, pages 557-633.

[20] A. Rausch, B. Rumpe, C. Klein, L. Hoogendoorn, „Aspect Oriented Framework Modeling“,In: *Proceedings of the 4th AOSD Modeling with UML Workshop (UML Conference 2003)*, October 2003.

[21] D. Sereni and O. de Moor, “Static Analysis of Aspects”, In *Proceedings of the 2nd international conference on Aspect-oriented software development*, ACM, March 2003, pp. 30-39.

[22] S. Sutton Jr., and I. Rouvellou, “Modeling of Software Concerns in Cosmos”, *1st international conference on Aspect-oriented software development* (Proceedings), *ACM*, April 2002, pages 127-133.

[23] M. Wand, G. Kiczales, and C. Dutchyn, „A semantics for advice and dynamic join points in aspect-oriented programming“, In *ACM Transactions on Programming Languages and Systems (TOPLAS)*, ACM Press, September 2004, pp. 890-910.