Extensions of SysML for Modeling an Aspect Oriented Software Architecture with Multiple Views

Kênia S. de Oliveira
Faculty of Computing
Uberlândia, Brazil
keniasoli@gmail.com

Joyce M. S. França
Computing Faculty
Uberlândia, Brazil
joycefranca@gmail.com

Michel S. Soares
Computing Faculty
Uberlândia, Brazil
mics.soares@gmail.com

Abstract—Aspect-oriented programming was proposed in order to handle crosscutting concerns in an efficient way. Initial proposals in this field were applied to source code. Later on, aspects were considered to be applied in other phases of software development, such as within software architecture. There are several advantages in identifying aspects at the architectural level such as reducing costs of software maintenance and preserving the notion of aspects in the development process ensuring traceability. Whereas different stakeholders need to view the system from various perspectives, the proposal of a software architecture in multiple views is essential. Therefore, this article proposes an aspect oriented software architecture model with multiple views to define the structure of aspect oriented software. An extension of the SysML modeling language is applied to model the architectural views.

Index Terms—aspects, Software architecture, SysML, aspects, Software architecture, SysMLA

I. INTRODUCTION

The design of a software architecture is a fundamental activity that help developers not only to address system complexity but also to define the structure with the aim of facilitating maintenance and evolution [1] [2]. The software architecture is also useful as a means of communication among stakeholders. Commonly a software architecture is defined in terms of multiple views, allowing a number of concerns to be described, which is useful to describe the architecture to multiple stakeholders [3]. The term view means the representation of a whole system from the perspective of a related set of interests [4].

The high complexity of software development is increased by the problem of handling crosscutting concerns that are scattered and tangled in diverse software artifacts. Aspect oriented programming [5] was proposed in order to handle crosscutting concerns in an efficient way. Initial proposals in this field were applied to source code but have evolved to other activities of software development. Aspect oriented software development (AOSD) is one proposed paradigm to manage this complexity by considering crosscutting concerns throughout the software life cycle [6]. Aspect oriented software architecture focuses on the localization and specification of crosscutting concerns in architectural designs. An architectural aspect is an interest that crosscuts architectural artifacts [7]. As normally occurs with software architectures, an aspect oriented software architecture must be capable of describing multiple views [8].

Given the importance of considering aspects in the early stages of development there are some researches presented in the field [9] [10]. Different approaches for modeling aspects at the architecture level have been proposed in recent years [11], [12], [13]. Despite the different proposed approaches, there is still much to do in this field. In many studies the mapping between aspect oriented requirements models and aspect oriented architecture has not been considered. Thus, the software architect has difficulties in identifying crosscutting concerns at the architectural level from requirements specifications [14]. Moreover, few practical applications have been proposed in the literature.

Aspect oriented software architecture is the focus of several research works since identifying aspects still in the architectural design can improve the modularity of the project, detect conflicts of interest at early phases of development, reduce cost of software maintenance and preserve the notion of aspects in the development process ensuring traceability [7]. Software architectures can separate crosscutting concerns more appropriately by using an Aspect Oriented Architecture Description Language (AO-ADL) [13]. There are a diversity of approaches of how aspects must be modeled with an Architecture Description Language (ADLs). Generally, aspects are represented by components [15]. Therefore, any adaptation of existing ADLs or the engineering of new ADLs should be based on a clear understanding of the challenges of representing aspects at the architectural level which can not be handled with the abstractions of existing ADLs.

In order to manage crosscutting concerns at the architectural level this article proposes a software architecture model considering aspects. This study proposes to apply an aspect oriented software architecture model with multiple views to define the structure of aspect oriented software. The modeling of the architectural views is performed by using SysML [16] as modeling language, with proposed extensions in order to model aspects. SysML is a UML profile with additional modeling capabilities. The language was chosen due to its high-abstraction characteristic and because of its extensions possibilities. To the best of our knowledge, SysML has not yet been applied for this purpose. As SysML does not have extensions for modeling aspects, the ones proposed in this research are presented in this article.

II. ASPECT ORIENTED SOFTWARE ARCHITECTURE WITH MULTIPLE VIEWS

The aspect oriented software architecture is based on concepts of joinpoint, pointcut, advice, type of the advice and weaver [17]. The joinpoint indicates the localization where the aspect will be inserted [5]. Poincuts are expressions that refer to the set of joinpoints [18].

An advice has types that specify how the advice will be operated. The most cited types are before, in which the advice is executed before the joinpoint, after, in which the advice is executed after the joinpoint, and around, that replace the execution of the joinpoint by execution of the advice or execute part of the advice before the joinpoint and another part after the joinpoint [19].

The insertion of the advice is performed via a special mechanism called weaver. The weaver mechanism is responsible to
compose the main programming with aspects. The aspects composition requires the identification of the relevant architectural joinpoint in which aspects and other architectural elements are naturally combined together.

Considering that a software architecture with multiple views is fundamental for software development and maintenance, the proposed aspect oriented software architecture model is described by using multiple views. The chosen views are the structural view, the development view and the scenario + requirements view as depicted in Figure 1. In the proposed model of aspect oriented architecture the views are related to each other and aspects are embedded in all views.

The views defined in the proposed aspect oriented software architecture relates to the requirements model as depicted in Figure 2. The scenarios + requirements view is related with the user requirements. User requirements describe representations at high level of abstraction and indicates what the system will do, i.e., the functionalities and restrictions from the viewpoint of the user.

The structural view is related with the system requirements. The propose of system requirements is to refine the user requirement increasing the level of detail. System requirements are used as input to the software project after their design using a modeling language.

The development view is related with implementation requirements. In the implementation level the requirements must be specified in a well-detailed manner. The level of detail is the specification of algorithms.

The requirements model is also modeled with aspects and the SysML. However, this is another topic of research work and is not addressed in this article.

### III. USING SYSML FOR MODELING THE ARCHITECTURAL VIEWS

The majority of AO-ADLs are motivated by the integration of the existing concepts of the architecture description (such as components, interfaces, connectors and configurations) with the new aspect oriented abstractions (such as, aspects, joinpoints, pointcuts and advices), with the objective of modeling crosscutting concerns in the architecture [15].

The aspect oriented software architecture can be classified into two main approaches: the symmetric approach adapts the interaction space (connectors and roles) to handle aspect orientation, which means that an explicit distinction between aspects and not aspectual components is not proposed [20]; the asymmetric approach introduces additional aspect oriented model elements such as aspectual components, aspectual interfaces and aspectual connectors, with an explicit distinction between aspects and not aspectual components [19]. In this work the asymmetric approach is chosen because there is a clear distinction between the conventional structure and the aspects structure.

The next subsections describe the aspect oriented architecture in multiples views using the SysML and extensions as an architectural description language for the design of an aspect oriented software architecture.

#### A. Scenarios View + Requirement

The scenarios + requirement view is obtained from the use cases that illustrate the interactions of an actor with the system. The actor can be a human or machine entity that interacts with the system to execute a significant work. In this view, which aspects relate to the use cases is considered. Associated with the scenario view, the requirements view is also modeled with aspects.

The objective of the structural view is to logically represent the system considering the functional requirements and how aspects relate to these requirements. Aspects can have both functional or non-functional origin.

The objective of the development view is to describe the development environment by defining the hardware/software platform that will give support to the system, and also the definition of the subsystems, packages, layers, and their interactions. The development view clearly defines technological options such as hardware, operating system, network, database and programming languages.
The scenarios view is designed by use cases. To be able to represent aspects in this view, the SysML Use Case diagram is extended, as depicted in Figure 3.

![Fig. 6. Representation of the Crosscut Relationship.](image)

In the extension, a new stereotype of type <<Aspect Use Case>> is created. Figure 4 shows how the stereotype is represented in the use case.

![Fig. 7. Block Model Extended to Represent Aspects.](image)

In order to relate an Aspect Use Case and a common Use Case, the relationship <<crosscut>> is proposed. In the use case modeling, this relationship means that an Aspect Use Case crosscuts a common Use Case.

As the <<crosscut>> relationship is used in other views besides the use case view, it is defined as a dependency model as depicted in Figure 5. Figure 6 shows the representation of the crosscut relationship. This <<crosscut>> relationship describes how aspect oriented requirements models and aspect oriented architecture can be related. It also facilitates the identification of crosscutting concerns at the architectural level from requirements specifications.

Related to the scenarios there are requirements that are also modeled using aspects. Aspects can be represented within packages, which improves the graphical display. The <<crosscut>> relationship is also applied to connect a package with aspects and common elements of the model.

### B. Structural View

The design of the structural view is performed with the SysML Block diagram with extensions. The SysML Block diagram is a stereotype of the UML Class diagram with additional restrictions and extensions defined by the SysML metamodel [16]. SysML Blocks are used in all phases of a system specification, and can be applied for many different types of systems [16].

A SysML Block describes elements related to the structure of a system. These elements can be from different abstraction levels. SysML Blocks provide information about the block itself (attributes) and can make references to other blocks (associations). Additional to the static structure, blocks also describe operations, i.e., the behavior that they can perform [21].

SysML allows blocks to have multiple compartments, each optionally identified with its own compartment name. The compartments may partition the features shown according to various criteria. Compartments may appear in any order and can be defined by each user, which brings great specificity to describe systems of a variety of domains. For the representation of aspects the SysML Block has been extended as described in Figure 7.

In order to represent an aspect the stereotype <<Aspect Block>> is created. Figure 8 describes the extensions to the SysML Block with compartments for the representation of aspects.

In the SysML Aspect Block, two compartments are defined: point and advice. The attribute Pointcut in the point compartment defines the points where the aspect is inserted, i.e., the joinpoint. In the compartment advice, the attribute Behavior is defined, which has information of the aspect behavior. As one aspect can have multiple behaviors, then each behavior must be numbered. Another defined attribute in the compartment advice is Type, which can assume three types: before, around or after. It is necessary to associate with each type the joinpoint defined in the attribute pointcut and also the behavior defined in the attribute Behavior.

In order to connect the Aspect Block with the common SysML Block the relationship <<crosscut>> defined in Figure 5 is used. This relationship indicates that an Aspect Block crosscuts a SysML Block.

### C. Development View

As mentioned in subsection III-B, SysML Blocks are applied in all phases of the specification and system design. This includes modeling also the logical or physical decomposition of a system and software specification, hardware or human elements. Therefore, in the development view, SysML Blocks and packages can be used in order to specify elements related to the implementation.

### IV. CASE STUDY

A case study was used as an instrument of research. According to [22], a case study is a research methodology appropriated for many types of research in software engineering. The Health Watcher (HW) system is chosen as case study. HW has been used as a reference for the development of aspect oriented software because of the heterogeneity of crosscutting concerns encountered in its implementation [23].

HW is a Web information system developed to improve the quality of services offered by the Department of Health of a city hall [24]. The HW system allows the citizens to register several types of complaints related to health. The full document of the original description of requirements can be obtained in [24].

The aspect oriented architecture defined in section II is described with the scenarios + requirements views, the structural view and the development view.

The scenarios view is described by use cases, as shown for example in Figure 9, referring to Employee use cases.

Figure 10 is about the structural view of the HW system. With the purpose of preventing the diagram from presenting too many relationships, all aspects except the Login aspect are represented within a package. The Login aspect relates only to Employee and Complaint block. As the Login aspect is relating directly to the joinpoints (Employee and Complaint) by means of
the relationship \texttt{<<crosscut>>}, the compartment \texttt{point} in this case is unnecessary.

The package containing the aspects is depicted in Figure 11. The \texttt{pckAspect} of Figure 10 is expanded and described in Figure 11.

The development view is described using a layered style in Figure 12.
Different stakeholders need to view the system from various perspectives. Therefore, the proposal of a software architecture in multiple views is essential. This article proposes an aspect oriented software architecture model with multiple views to define the structure of aspect oriented software. An extension of the SysML modeling language is applied to model the architectural views.

There are several models of architectural views but there are few works that deal with aspects on architectural views. In many studies the mapping between aspect oriented requirements models and aspect oriented architecture has not been considered. In this article, the mapping is proposed, which facilitates the identification of crosscutting concerns at the architectural level from requirements specifications. In this article, the presented model allows to represent different views on software considering the representation with aspects.

The main features of modeling an aspect oriented software architecture with an extension of SysML are proposed. Aspect separation is clearly obtained by extending the SysML Block and the SysML Use Cases diagrams and creating new relationships. The focus of current work is on extending ArgoUML to include the SysML metamodel with aspects as proposed in this article.

V. Conclusion

REFERENCES


