UML-AOF: A Profile for Modeling Aspect-Oriented Frameworks

José Uetanabara Júnior[1]
Centro Universitário Eurípedes de Marília - Univem
Marília, São Paulo, Brasil
CEP 17.525-901
miklotovx@gmail.com

Valter Vieira de Camargo[2]
Universidade Federal de São Carlos – UFSCar
São Carlos, São Paulo, Brasil
CEP 13.565-905
valtercamargo@hotmail.com

Christina Von Flach
Universidade Federal da Bahia – UFBA
Salvador, Bahia, Brasil
CEP 40.170-110
christina.flach@gmail.com

ABSTRACT

The design model of an application that was developed with support of frameworks involves both the framework and the application design. This results in complex architectures represented by design models that are difficult to understand; because there are many framework characteristics which are not evident when plain UML is used. The same problem occurs with Aspect-Oriented Frameworks (AOF). In AOF-based development there are units which deserve attention from different developers – application engineers and framework engineers. Besides, there are a number of architectural characteristics in AOFs which do not appear in Object-Oriented Frameworks. So, in order to make these specific characteristics clearer in the models we propose UML-AOF, an UML profile for designing AOFs. UML-AOF was created based on an existing UML profile for aspect-oriented programming and takes into consideration some AspectJ idioms, patterns and also stereotypes from a profile for object-oriented frameworks called UML-F. UML-AOF was evaluated by means of its application in the design of a persistence and security AOF. We observe that UML-AOF makes some specific AOF architectural characteristics clearer in design models, improving the understandability of the architecture as well as the behavior.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design – *Representation*; D.2.11 [**Software Engineering**]: Software Architectures – *Patterns*.

General Terms

Design, Management.

Keywords

Aspect-Oriented Frameworks, Aspect-Oriented Modeling.

[1] This research is supported by CAPES.

[2] This research is partially supported by CNPq: Projeto Latin-AOSD - PROSUL-Proc. no. 490478/2006-9.

1. INTRODUCTION

After the emerging of Aspect-Oriented Programming (AOP) [9], several frameworks begin to use it to modularize crosscutting concerns in their architecture [3][5][10] leading to the appearing of Aspect-Oriented Frameworks (AOF). In this paper, we use the term "Crosscutting Framework" (CF) as a kind of AOF which encapsulates just one crosscutting concern. CFs are used to support the development of non-functional parts of an application [3].

Despite all of the well-known reuse benefits promoted by framework-based development, applications that are developed with their support have a three-layer complex architecture, which results in complex design models. The first layer encapsulates abstract and concrete modular units of the framework and it is of interest to the framework engineers; the second layer is responsible for putting together the first and the third layers by creating concrete modular units extending the abstract ones available in the framework (first layer) and it is of interest to the application engineers; and the third layer is the application itself and also it is the application engineer that is in charge of it. In the context of CFs, the problem is worse than with conventional frameworks, as the development of an application may be based on several CFs, and each of them addresses just one crosscutting concern. The final design models are complex and make comprehension, reuse, and maintenance very difficult tasks [3].

Another problem is that CF design models have a lot of architectural and design details not present in standard frameworks, such as: hook methods, hook pointcuts, join points, variabilities, patterns/idioms to abstract behaviors, features, etc. Some authors have presented some idioms [8] and patterns [3] to modularize these frameworks in order to reach good levels of maintainability. These idioms/patterns have been extensively used in the "implementation" of these frameworks [3][5][10] , but they are not present at the design level, as the plain UML does not provide support to make these characteristics evident.

In this paper we present UML-AOF, an UML profile for modeling CFs. The proposed profile uses the Evermann´s profile [4] as base AOP profile and also some concepts already defined in UML-F [6]. Besides, four idioms proposed by Hanenberg [8] are represented as stereotypes. The Data Catcher Pattern proposed by Camargo and Masiero [3] is also used in the profile as stereotypes and tagged values. The aim is to bring to the level of detailed design the main CFs characteristics. As a case study, we apply our proposed profile in a Security and Persistence CF [3]. We have noticed an explicit separation among layers that compose the detailed design, enhancing comprehension, reuse and maintenance tasks. Another contribution is that the profile can be used as a guide to develop good architectural designs, as it incorporates idioms and patterns.

This paper is structured as follows: basic concepts are presented in Section 2; the UML profile for design CFs is presented in Section 3; the persistence and security CFs used as case study are presented in Section 4; the related works are presented in Section 5; and the conclusion and future work are presented in Section 6.

2. BASIC CONCEPTS

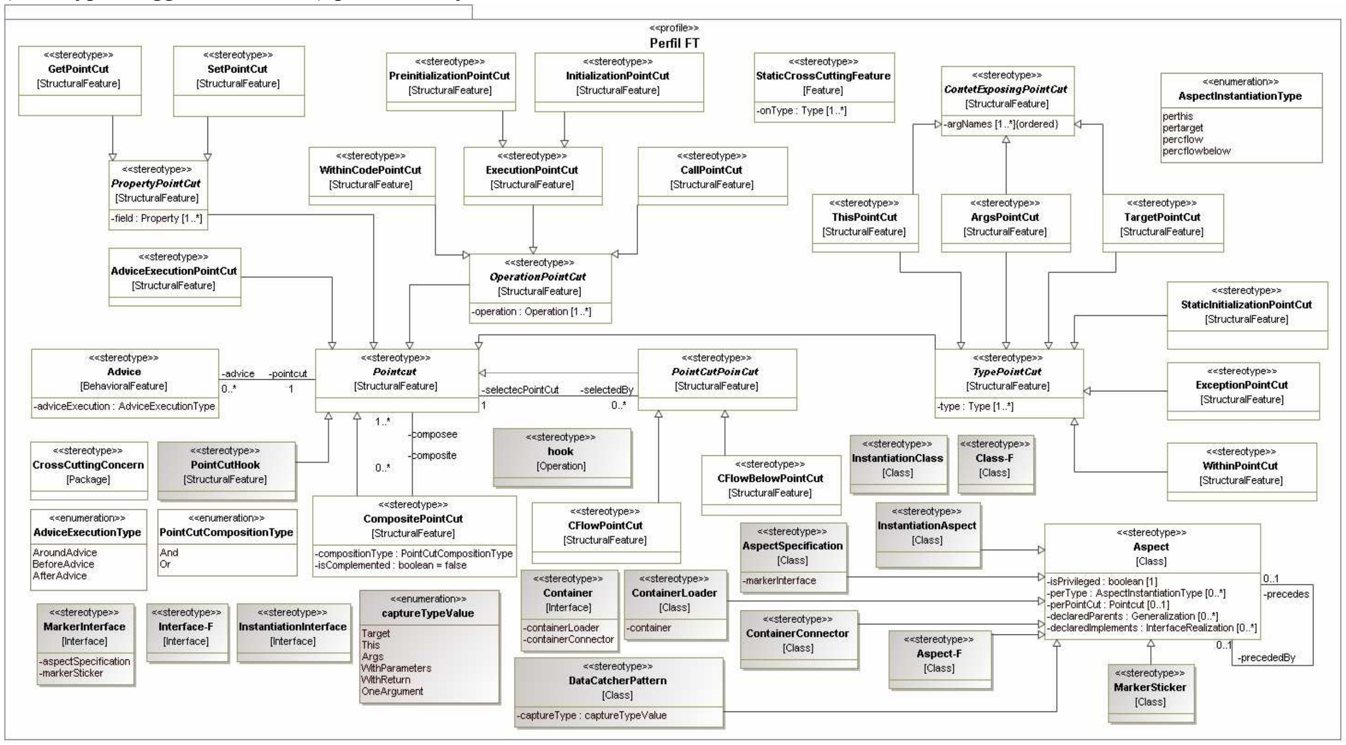
2.1 Evermann´s Profile

Evermann [4] developed a profile for AOP adherent to AspectJ concepts. The authors have chosen this language because of its maturity to implement aspects. The gray meta-classes in Figure 1 present Evermann´s profile along with the additions (stereotypes, tagged values, etc.) performed by us. Rather than specializing meta-classes, Evermann extended the existing UML meta-classes through the creation of stereotypes [4]. So, each class in Figure 1 is a stereotype that may be applied in model level. The extension relationship is depicted by brackets ([]). For example, the Aspect stereotype has the word `Class` between brackets, indicating that this stereotype extends the `Class` meta-class.

We have chosen this profile because of the following reasons: it is adherent to AspectJ language - which is the same language used to implement the CFs and it is a light-weight profile, which guarantees that many available tools can be used to implement it.

2.2 Hanenberg´s Idioms

Hanenberg *et al*. [8], proposed a set of eight idioms for AspectJ aiming at structuring the code to reach good levels of reuse and abstraction. Although the idioms can be used for developing conventional software, their main application is in framework development [8], since they aim to abstract the application details, in order to make the code more generic and independent.



<<profile>> Perfil FT
<<stereotype>> GetPointCut [StructuralFeature]
<<stereotype>> SetPointCut [StructuralFeature]
<<stereotype>> PreinitializationPointCut [StructuralFeature]
<<stereotype>> InitializationPointCut [StructuralFeature]
<<stereotype>> StaticCrossCuttingFeature [Feature] -onType : Type [1..*]
<<stereotype>> ContetExposingPointCut [StructuralFeature] -argNames [1..*]{ordered}
<<enumeration>> AspectInstantiationType perthis pertarget percflow percflowbelow
<<stereotype>> PropertyPointCut [StructuralFeature] -field : Property [1..*]
<<stereotype>> WithinCodePointCut [StructuralFeature]
<<stereotype>> ExecutionPointCut [StructuralFeature]
<<stereotype>> CallPointCut [StructuralFeature]
<<stereotype>> ThisPointCut [StructuralFeature]
<<stereotype>> ArgsPointCut [StructuralFeature]
<<stereotype>> TargetPointCut [StructuralFeature]
<<stereotype>> AdviceExecutionPointCut [StructuralFeature]
<<stereotype>> OperationPointCut [StructuralFeature] -operation : Operation [1..*]
<<stereotype>> StaticInitializationPointCut [StructuralFeature]
<<stereotype>> Advice [BehavioralFeature] -adviceExecution : AdviceExecutionType
-advice 0..* -pointcut 1
<<stereotype>> Pointcut [StructuralFeature]
-selectecPointCut 1 -selectedBy 0..*
<<stereotype>> PointCutPoinCut [StructuralFeature]
<<stereotype>> TypePointCut [StructuralFeature] -type : Type [1..*]
<<stereotype>> ExceptionPointCut [StructuralFeature]
1..* -composee 0..* -composite
<<stereotype>> hook [Operation]
<<stereotype>> CrossCuttingConcern [Package]
<<stereotype>> PointCutHook [StructuralFeature]
<<stereotype>> CFlowBelowPointCut [StructuralFeature]
<<stereotype>> InstantiationClass [Class]
<<stereotype>> Class-F [Class]
<<stereotype>> WithinPointCut [StructuralFeature]
<<enumeration>> AdviceExecutionType AroundAdvice BeforeAdvice AfterAdvice
<<enumeration>> PointCutCompositionType And Or
<<stereotype>> CompositePointCut [StructuralFeature] -compositionType : PointCutCompositionType -isComplemented : boolean = false
<<stereotype>> CFlowPointCut [StructuralFeature]
<<stereotype>> AspectSpecification [Class] -markerInterface
<<stereotype>> InstantiationAspect [Class]
<<stereotype>> Aspect [Class] -isPrivileged : boolean [1] -perType : AspectInstantiationType [0..*] -perPointCut : Pointcut [0..1] -declaredParents : Generalization [0..*] -declaredImplements : InterfaceRealization [0..*]
0..1 -precedes 0..1 -precededBy
<<stereotype>> MarkerInterface [Interface] -aspectSpecification -markerSticker
<<stereotype>> Interface-F [Interface]
<<stereotype>> InstantiationInterface [Interface]
<<enumeration>> captureTypeValue Target This Args WithParameters WithReturn OneArgument
<<stereotype>> Container [Interface] -containerLoader -containerConnector
<<stereotype>> ContainerLoader [Class] -container
<<stereotype>> ContainerConnector [Class]
<<stereotype>> Aspect-F [Class]
<<stereotype>> DataCatcherPattern [Class] -captureType : captureTypeValue
<<stereotype>> MarkerSticker [Class]

Figure 1 – Adaptation of Evermann´s meta-model.

Each of these idioms provides roles that must be played by classes/aspects in order to structure the code, as each role provides a well defined responsibility/behavior. Four of eight idioms have been used in this work: Container Introduction, Marker Interface, Composite Pointcut and Abstract Pointcut. They were chosen because they are recurring implementation strategies in CFs found in the literature [3][5][10]. Due to this, we argue that they must be represented in design models.

2.3 UML-F Profile

UML-F [6] provides notational elements describing basic characteristics of OOF (Object-Oriented Frameworks). In this profile both stereotypes and tagged values have been grouped into an element called "UML-F tag". The use of this tag is the same as a UML stereotype, however it can have a value, like tagged values.

There are two categories of tags: tags for modeling frameworks (`<<application>>` and `<<framework>>`), and tags for essential principles of OOF development (`<<template>>` and `<<hook>>`). The first category has tags that give support for the framework documentation, mostly for the application designer/developer. These tags allow differentiating between modular units belonging to the framework from modular units belonging to the application. The second category has tags representing two roles imposed by the template method design pattern, template method, and the hook method. These tags make the identification of the framework instantiation points easier; and they are used in the profile proposed by the authors.

2.4 Data Catcher Pattern

Camargo and Masiero [3] proposed a pattern for modeling CFs, called Data Catcher, aiming at providing composition alternatives to compose a CF with a base code (application). Each composition alternative is a different manner of catching data from the base code. There are six composition alternatives: 1) With Return; 2) This; 3) Target; 4) Args; 5) With Parameters; and 6) One Argument. Each of these composition alternatives is represented by one abstract aspect which may be specialized through a concrete aspect in order to weave the CF with a base code. In the concrete aspect, the application engineer must provide a concrete pointcut with join points of a specific base code. Depending on the aspect chosen, the framework will catch the object/value using different mechanisms.

The use of such pattern increases the reuse levels of the framework and splits its structure into two distinct parts. The first part deals with variabilities of the crosscutting concern, and the second deals with the composition part. This separation enables the framework and application engineers to concentrate on tasks and problems inherent to each part separately.

3. A PROFILE FOR DESIGNING CFs

Figure 1 presents the proposed profile. We had the concern of extending the Evermann's profile and UML meta-model without inserting inconsistencies; so all modifications were validated in the Magic Draw modeling tool. This tool complies with the UML 2.0 meta-model.

The `<<instantiationAspect>>` and `<<aspect-F>>` stereotypes have been created to represent if the aspect belongs to instantiation layer or to the framework layer. The framework layer is the portion of the final design model that framework engineers are interested in; mainly because this part is where the framework maintenance and evolution tasks occur. The instantiation layer is the portion of the final design model that is in charge of application engineers, as they must create concrete units to reuse the framework. These kind of stereotypes allow an easier identification of the modular units in charge of both developers, allowing each one to concentrate on his part.

The `<<instantiationAspect>>` and `<<aspect-F>>` stereotypes extend the `<<Aspect>>` stereotype from Everman's profile. The `<<interface-F>>` and `<<instantiation Interface>>` stereotypes are similar, but they extend the `<<Interface>>` meta-class. The same reasoning is applied for the `<<instantiationClass>>` and `<<class-F>>` stereotypes, but they extend the `Class` meta-class. The `<<pointCutHook>>` stereotype was created to indicate which pointcut is a variability that can be concretized by the application engineer. The last stereotype, `<<hook>>`, has been inherited from UML-F, and denotes that the method is a hook method, i.e., a method that can be concretized by the application engineer. These stereotypes are not related to implementation details and code organization; they are just representative and must be used for documentation.

The enumerations and stereotypes shown below are related to implementation details and code organization as they represent idioms and patterns found in the literature. These stereotypes have a richer semantic than those shown above because they need a better definition from which class they extend. The stereotypes that are based on idioms and patterns can aid developers in structuring the design reaching good reuse and maintenance levels.

The set of stereotypes `<<markerInterface>>`, `<<markerSticker>>` and `<<aspectSpecification>>` represent the Marker Interface idiom [8]. The stereotype `<<markerInterface>>` is used to specify which interface is being used as base type for some application classes so that an aspect, playing the role of "aspect specification", can be previously designed to crosscut all of the interface subtypes. The aspect that connects the marker interface with the subtypes is another aspect playing the role of "marker sticker".

The set of stereotypes `<<container>>`, `<<container Connector>>` and `<<containerLoader>>` represent the Container Loader idiom. The stereotype `<<container>>` is used to mark which interface is being used as a base type for some application classes; as is in the Marker Interface idiom. However, the aim is to introduce a number of extrinsic characteristics in this interface to be inherited by its subtypes. This is done thought inter-type declarations by an aspect playing the role of "container loader". The aspect responsible for connecting the "container" to "container loader" plays the role of "container connector".

The enumerations `Target`, `This`, `Args`, `WithParameters`, `WithReturn`, and `OneArgument` specify respectively the composition types of Data Catcher Pattern [3].

4. CASE STUDY

In this section we present the case study that we have developed to analyze the applicability of our proposed profile. The application is a web personnel system and the system is a framework-based system, as it was developed based on two CFs; Persistence and Security. The whole system has twenty seven aspects, twenty one classes and two interfaces, but only pieces of the final design model are shown in Figure 2 because of space limitations.

Each class, aspect, and interface in Figure 2 has below its name, in parentheses, the name of the package that it belongs to. For example, `PersistentRoot` interface has the word (Persistence). So, this interface belongs to the Persistence framework (notice that each framework is encapsulated in a package). The abstract aspect `LogingLog` has the word (Security), so it belongs to the Security Framework. The same reasoning is valid for the other units that have the word (Instantiation) and (Application). This division aids the engineers to concentrate on what they are interested in.

The `PersistentEntities` abstract aspect (letter "a") aims at introducing a set of persistence methods into the `PersistentRoot` interface (letter "d"). These methods can be seen through the `<<staticCrosscuttingFeature>>` stereotype proposed by Evermann. This stereotype has a tag called `onType` whose value is the unit which will receive the methods. The `PersistentRoot` interface must be used as a base type for all application classes which need to have their objects stored in a database. The `MyPersistentEntities` concrete aspect (letter "f") aims at declaring which are the application classes which need to have their objects persisted. The PersistentEntities abstract aspect plays the role of "container loader" and therefore it is stereotyped as <<containerLoader>>. The PersistentRoot interface plays the role of "container", so it is stereotyped as <<container>> and the MyPersistentEntities concrete aspect has the role of "container connector", so it is stereotyped as <<containerConnector>>. These stereotypes makes the use of the Container Introduction idiom evident.

The OORelationalMapping (letter "b") aspect must crosscut all of the sub-types of PersistentRoot interface, characterizing these units as the application of the Marker Interface idiom. So, the OORelationalMapping aspect is stereotyped as <<aspectSpecification>>, the PersistentRoot interface is stereotyped as <<markerInterface>> and SecurityPersistentEntities concrete aspect (letter "g") is stereotyped as <<markerSticker>>, since the "marker sticker" role connects the interface PersistentRoot with the application classes. The usage of these stereotypes makes evident the use of the idiom Marker Interface in the framework project.

As the Container Introduction and Marker Interface idioms are composed by three roles each of them, sometimes it is difficult to found in the model the units playing the other roles of the idiom. To improve this search process, we have created some tags, for example, the aspects stereotyped with <<containerLoader>> and <<aspectSpecification>> have the tags container and markerInterface. The performed extension can also be seen in Figure 1. We have incorporated container and markerInterface attributes into the <<containerLoader>> and <<aspectSpeficiation>> stereotypes, the aspectSpecification, markerSticker attributes into <<markerInterface>> stereotype and the containerConnector and containerLoader into the <<container>> stereotype. These tags allow identifying the interface each aspect affects and vice versa. For example, the tag container in the PersistentEntities abstract aspect (letter "e"), receives the value PersistentRoot, as PersistentRoot is the interface that will encapsulate all the methods insert by PersistentEntities. Also, the tags containerLoader and containerConnector in the PersistentRoot interface (letter "h"), allows identifying the aspects that plays these roles: PersistentEntities and MyPersistentEntities. These tags are optional and must be used when the same idiom is applied more than once.

The <<LoginLog>> and <<LoginLogWithParameters>> aspects correspond to the Data Catcher Pattern [3]. The former aspect <<LoginLogWithParameters>> (letter "c") is one of the composition alternatives provided by this pattern, so it is stereotyped as <<DataCatcherPattern>>. This means that the Data Catcher Pattern [3] is being used in the design. The {compositionPattern = WithParameters} tagged value indicates the composition type provided by this aspect (in this case, WithParameters). The same reasoning is valid for the other ways of composition (Target, This, Args, etc).

The getAttributeToBeFoundInTheSession() abstract method is an example of a method that must be concretized by the application engineer. For these kind of methods the stereotype <<hook>> must used in order to make clear the extension points of the framework. The LoginLog() abstract pointcut is an example of an abstract pointcut that must be concretized by the application engineer in order to define in which execution points (join points) the Log concern must be applied In that case the <<PointCutHook>> stereotype is applied. An example of both stereotypes can be seen in letter (i) and (c) on Figure 2.

After CFs have been modeled with our proposed profile we have noticed the following advantages: 1) The model becomes more organized, as the three layers can be easily identified, propitiating software engineers to concentrate themselves on their specific parts; 2) The communication about design decisions and maintenance tasks is improved when the idioms and patterns are known among developers; and 3) The CF reuse process is facilitated because the variabilities and hooks are much clearer than the design model without our stereotypes.

5. RELATED WORK

Many researchers have proposed profiles for AOP [1][2][4][7]. However, only one of them [11] has proposed a profile specific for CFs, like the one presented in this work. Rausch [11] proposed models to be used in requirements and design levels for AOF. Besides, a technique for gluing models was developed. This technique uses bidirectional arrows and notes with OCL (*Object Constraint Language*). Each hook is specified inside the notes. Through the OCL, the application elements are linked with the hooks in the framework. Rausch´s work has some deficiencies that must be improved. It does not have a meta-model. It has large and complex OCL statements because there is not a specific language for the aspect binding [11]. Therefore, a complete profile of the UML needs to be developed [11].

6. CONCLUSION

The notational elements created in this work aim at providing support for framework and application engineers evidencing the most important characteristics of the framework development and reuse process. For framework engineers, the proposed profile presents stereotypes/tagged values that represent idioms and patterns usually applied during AOF development. Besides, some stereotypes allow distinguishing clearly the modular units of the framework layer, application layer, and instantiation layer. With this separation, each engineer can easily find the elements that need to be modified. Also, the stereotypes for the idioms and the Data Catcher Pattern improve the communication among framework developers by providing a unique vocabulary.

It is important to notice that it is perfectly possible to build a design model without these elements. However, the reuse process and changes required in the framework architecture can be much more difficult. So, their application brings maintainability, reusability, and productivity benefits for the experts involved. The final design model of an application that has been developed with the support of several CFs is complex. The existence of stereotypes that aids identifying the important components of this complex architecture is important.

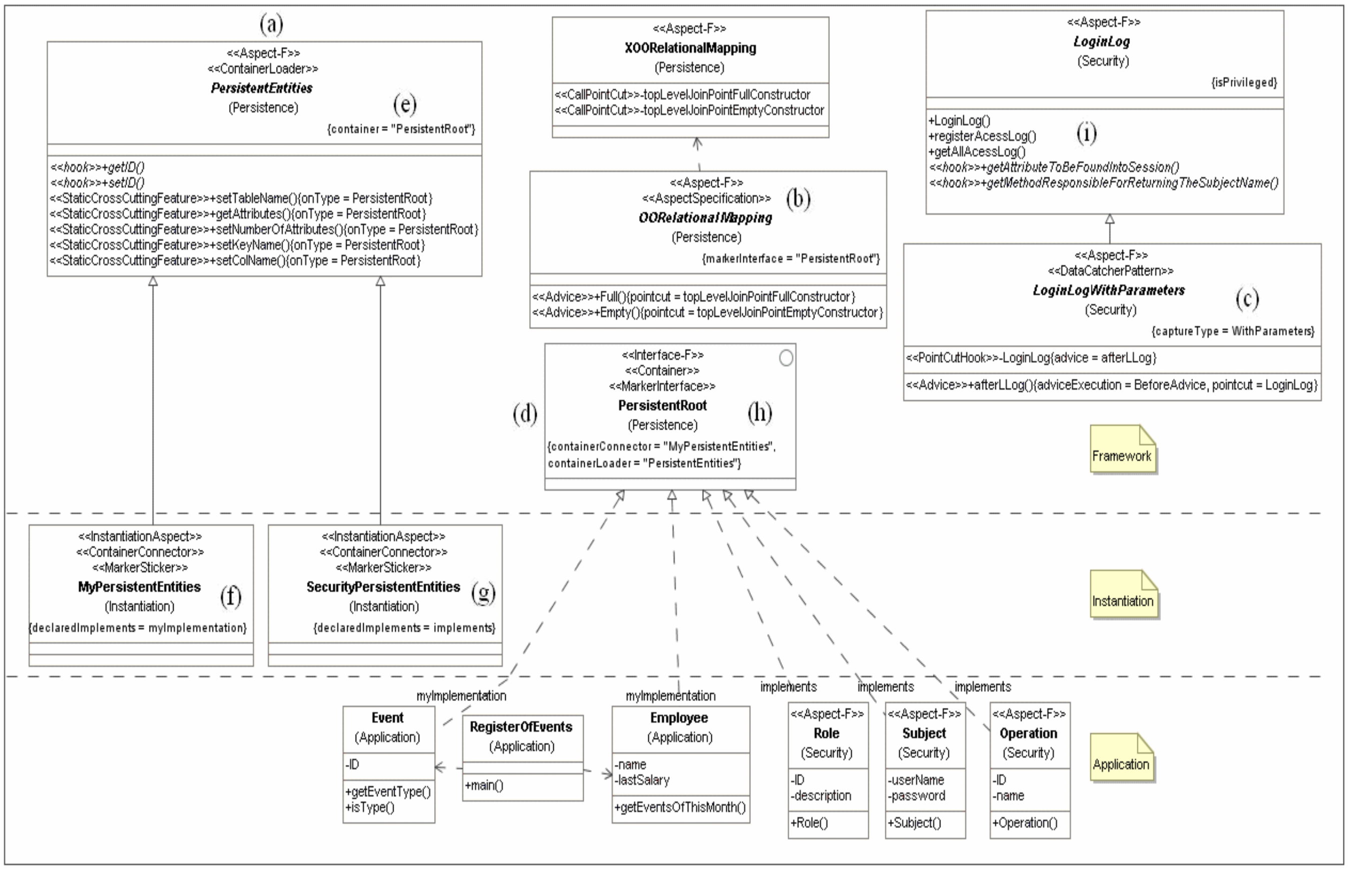

(a)
<<Aspect-F>>
<<ContainerLoader>>
PersistentEntities
(Persistence)
(e)
{container = "PersistentRoot"}
<<hook>>+getID()
<<hook>>+setID()
<<StaticCrossCuttingFeature>>+setTableName(){onType = PersistentRoot}
<<StaticCrossCuttingFeature>>+getAttributes(){onType = PersistentRoot}
<<StaticCrossCuttingFeature>>+setNumberOfAttributes(){onType = PersistentRoot}
<<StaticCrossCuttingFeature>>+setKeyName(){onType = PersistentRoot}
<<StaticCrossCuttingFeature>>+setColName(){onType = PersistentRoot}
<<Aspect-F>>
XOORelationalMapping
(Persistence)
<<CallPointCut>>-topLevelJoinPointFullConstructor
<<CallPointCut>>-topLevelJoinPointEmptyConstructor
<<Aspect-F>>
<<AspectSpecification>>
OORelationalMapping
(b)
(Persistence)
{markerInterface = "PersistentRoot"}
<<Advice>>+Full(){pointcut = topLevelJoinPointFullConstructor}
<<Advice>>+Empty(){pointcut = topLevelJoinPointEmptyConstructor}
<<Aspect-F>>
LoginLog
(Security)
{isPrivileged}
+LoginLog()
+registerAcessLog()
(i)
+getAllAcessLog()
<<hook>>+getAttributeToBeFoundIntoSession()
<<hook>>+getMethodResponsibleForReturningTheSubjectName()
<<Aspect-F>>
<<DataCatcherPattern>>
LoginLogWithParameters
(c)
(Security)
{captureType = WithParameters}
<<PointCutHook>>-LoginLog{advice = afterLLog}
<<Advice>>+afterLLog(){adviceExecution = BeforeAdvice, pointcut = LoginLog}
<<Interface-F>>
<<Container>>
<<MarkerInterface>>
PersistentRoot
(d)
(h)
(Persistence)
{containerConnector = "MyPersistentEntities",
containerLoader = "PersistentEntities"}
Framework
<<InstantiationAspect>>
<<ContainerConnector>>
<<MarkerSticker>>
MyPersistentEntities
(f)
(Instantiation)
{declaredImplements = myImplementation}
<<InstantiationAspect>>
<<ContainerConnector>>
<<MarkerSticker>>
SecurityPersistentEntities
(g)
(Instantiation)
{declaredImplements = implements}
Instantiation
myImplementation
myImplementation
implements
implements
implements
Event
(Application)
-ID
+getEventType()
+isType()
RegisterOfEvents
(Application)
+main()
Employee
(Application)
-name
-lastSalary
+getEventsOfThisMonth()
<<Aspect-F>>
Role
(Security)
-ID
-description
+Role()
<<Aspect-F>>
Subject
(Security)
-userName
-password
+Subject()
<<Aspect-F>>
Operation
(Security)
-ID
-name
+Operation()
Application

Figure 2 – Case Study Modeled with the Profile.

The presented profile must be enhanced and extensively evaluated. As future work, we plan to perform maintainability assessments of the proposed profile based on different case studies, comparing frameworks developed in a traditional manner with frameworks that use the stereotypes and idioms packaged into the profile. The possible conflicts that may occur between stereotypes also deserve further investigation. Another future work we intend to conduct is change the base profile for another; therefore, we can evaluate if the proposed set of extensions (stereotypes, tagged values and enumerations) can be applied in other profiles as well.

7. REFERENCES

[1] Aldawud, O., Elrad, T., Bader. A. UML Profile for Aspect-Oriented Software Development. In: Proceedings of Workshop of Aspect Oriented Modeling with UML of Aspect Oriented Software Development Conference (AOSD), 2003.

[2] Barra, E., Génova, G. and Llorens, J. 2004. An approach to aspect modeling with UML 2.0. In Proc. 5th Int. Workshop on Aspect-Oriented Modeling, October 2004.

[3] Camargo, V.V.; Masiero, P.C. A Pattern to Design Crosscutting Frameworks. In: Proceedings of the 23rd Annual ACM Symposium on Applied Computing (ACM-SAC'08), Fortaleza, Brasil, 2008.

[4] Evermann, Joerge. 2007. A Meta-Level Specification and Profile for AspectJ in UML. Victoria University Wellington, Wellington, New Zealand. AOSD 2007.

[5] Fayad, M. E.; Johson, R. E. (eds) (2000). Domain-Specific Application Frameworks: Frameworks Experience by Industry, John Wiley & Sons. 2000.

[6] Fontoura, M., Pree, W., Rumpe, B. The UML Profile for Framework Architectures. Addison Wesley, 2002.

[7] Groher, Iris, Baumgarth, Thomas. 2004. Aspect-Orientation from Design to Code. Munich, Alemanha. 2004.

[8] Hanenberg, S., Unland, R., Schmidmeier, A. AspectJ Idioms for Aspect-Oriented Software Construction. In: Proceedings of 8th European Conference on Pattern Languages of Programs (EuroPLoP), Irsee, Germany, 25th–29th June, 2003.

[9] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irving, J. Aspect Oriented Programming. In: Proceedings of ECOOP. pp. 220-242, 1997.

[10] Rashid, A., Chitchyan, R. Persistence as an Aspect. In: Proceedings of the 2nd International Conference on Aspect Oriented Software Development – AOSD. Boston – USA, 2003.

[11] Rausch, A., Rumpe, B., Hoogendoorn, L. Aspect-Oriented Framework Modeling. In: The 4th AOSD Modeling With UML Workshop. 2004.