SEA/Aspect: Dynamic Visualization and Composition of Concerns in Aspect-Oriented Modeling (AOM)

Pedro Ghilardi, Ricardo Pereira e Silva

Federal University of Santa Catarina
Department of Informatic and Statistics
Florianopolis, Santa Catarina - Brasil
{pghilardi,ricardo}@inf.ufsc.br

Abstract. The Aspect-Oriented Modeling (AOM) aims to raise the abstraction level from code to models in the representation of aspect-oriented systems. This paper aims at the representation of aspect-oriented software using UML, through a lightweight profile, modeling the most important features of Aspect-Oriented Programming (AOP) and enabling alternating views of the system dynamics. The developer may create different compositions of the core and the crosscutting models, visualizing only the core models, the crosscutting models, or the core intertwined with the crosscutting models. The visualization of the aspect dynamic can be enabled or disabled dynamically, by updating the compound model, with no effort demanded from the developer. The concerns are differentiated in the compound model by different colors. The proposed solution is implemented as a tool, named SEA/Aspect, that enables the automatic generation of sequence diagrams and the automatic composition of pointcuts.

Keywords: Aspect-Oriented Modeling, UML, Model Composition, Model Visualization, Model Weaving, Modeling Tool

1 Introduction

In an application, each requirement can be considered a customer's concern. The core concerns capture the core functionality and impact only a part of the system, while the crosscutting concerns capture functionality that impacts one or more parts of the system. The AOP's goal is the modularization of these concerns, so that they are kept in separate modules that implement the core concerns of an application [1].

The aspect-oriented applications usually are represented directly at the code-level. One of the limitations of this approach is that the concerns modularization makes it difficult to understand the execution flow, since the system dynamic is visualized only after the composition of concerns, or with the aid of tools for visualizing the effect of the aspects in the system core. Another limitation of the direct implementation in code is that with the low-level of abstraction, the developer may be overwhelmed with implementation details rather than on the interaction of the core and crosscutting concerns.

An aspect-oriented system demands the representation of the characteristics inherent to AOP and may have their understanding facilitated by alternating the views of the application dynamic, visualizing the crosscutting concerns composed with the core concerns. As we are working in the modeling of aspect-oriented applicatons, the model of a crosscutting concern will be referred as a crosscutting model and the model of a core concern as a core model. The Unified Modeling Language (UML), the standard language used to model object-oriented systems [2], could be used for aspect-oriented systems, but it has elements that cannot be represented with the standard meta-model. To overcome this limitation, the language could be extended by two ways: via a lightweight extension through a profile, or an extension through the modification of the language meta-model.

Some approaches have been proposed for the modeling of aspect-oriented systems using UML. A group of proposals extends the language meta-model [3] [4] [5] [6] [7], introducing non-standard language constructs to represent the structure and behavior of aspect-oriented applications. Another set of proposals extends the UML through a lightweight profile [8] [9] [10], which can be used in CASE tools that support importing profiles. Our approach specifies aspect-oriented software through a lightweight profile, using some stereotypes from the Evermann's [8] profile and proposing others to represent the behavior of the core and crosscutting concerns. The constructions are based in the AspectJ language [11]. The proposed approach allows the capture of multiple join points with wildcards in the pointcut specification and the modeling of the most important characteristics of AOP. Our approach presents a new way to establish the connection between pointcuts and advices in the modeling, using state invariants in the sequence diagram. The constructions specified by the lightweight profile allows the automatic composition of core and crosscutting models, enabling alternating views of the system dynamics (with and without the explicitation of the aspects) and, thus, facilitating the model understanding and maintenance. The SEA/Aspect, implemented in the SEA environment [12] as a tool, allows the dynamic enabling of aspects, visualizing only the behavior of the core models, the crosscutting models, or the composition of one or more crosscutting models with a core model automatically. The developer differentiates the behavior of each concern by different colors in the compound model.

This article is organized as follows: section II presents the related work. Section III presents the proposal for specification of aspect-oriented software. The section IV deals with the aspects composition and the visualization toggling tool. Section V presents a case study using the proposed tool. The conclusions are in the section VI.

2 Related Work

Some studies have been done in the modeling of aspect-oriented applications. A set of approaches extends the UML using a profile, while others create a customized meta-model. Most approaches represent the structure of aspect-oriented applications, but some also represent its behavior. The automatic composition of models is a desired feature, because it allows the toggle of views and automates part of the modeling. According to Fuentes [13], although each form of extension has its advantages and disadvantages, in most cases, the extension through a profile is better than extending the meta-model, because the meta-model extensions cannot be used in available CASE tools.

The proposals that are presented next extends the UML through profiles. Evermann's [8] proposes a profile to model the structure of aspect-oriented applications. Some stereotypes proposed by him are used in this work, as the definition of aspects and crosscutting concerns. One of the main contributions of Evermann's work is the representation of aspect-oriented applications only within UML terms, without textual specifications. However, Evermann's approach does not allow the capture of multiple join points with wildcards in the pointcut specification, because the modeler must select explicitly which model elements are captured by a pointcut. Our approach represents pointcuts with states and transitions from the UML state machine diagram, which allows the specification of wildcards. The Motorola WEAVR [9] uses the UML state machine diagram to represent pointcuts and advices, extending its semantics through a lightweight profile. The aspects are modeled using state machines focused in transitions. The specification of Motorola WEAVR is in a low-level of abstraction with constructions close to the target language code, while we represent the models in a high-level of abstraction. Our approach also differentiates from Cottenier's in the modeling of advices. While it models advices with the UML state machine diagram, we use the UML sequence diagram to represent the advices as a set of messages. In our approach, the connection between pointcuts and advices is obtained with the use of state invariants, which are added in the sequence diagrams. When the system mets a pointcut, it enters in a state, this state is referred as a state invariant in the sequence diagram and triggers the execution of the advice. The Stein's approach [10] [14] [15] allows the specification of aspect-oriented applications through a profile. However, the main objective of the approach is the automatic generation of code from aspect-oriented models, modeling the core and crosscutting concerns in a low-level of abstraction. Our approach represents the concerns in a high-level of abstraction with state machine and sequence diagrams.

The extension through meta-modeling is a characteristic of the approaches presented from now. A multi-view modeling is proposed by Kienzle [3], extending the UML meta-model. The Reusable Aspect Models (RAM) models the structure of applications using the class diagram, and the behavior using the sequence and state machine diagrams. Introductions are modeled with class diagrams. Pointcuts and advices are represented with sequence and state machine diagrams. The main contributions of RAM is that it allows a complete representation of an aspect-oriented application in the modeling phase, using class, sequence and state machine diagrams. However, Kienzle's proposal does not represent the connection between pointcuts and advices in the sequence diagram. In our approach the pointcuts are defined in state machines and the connection between pointcuts and advices is realized using state invariants added to the sequence diagram. Klein's approach [5] uses extended sequence diagrams to model the system behavior and allows the composition of multiple aspects in the same joinpoint. However, Klein's approach does not allow the representation of pointcuts with wildcards, while our profile allows the specification of wildcards in the state transitions of the state machine diagrams. Jacobson's approach [6] represents aspects in terms of use cases. A construction named use-case slice (with the stereotype *use case slice*) groups all the models of a concern, which includes introductions and collaborations. Our approach uses the Hotel Management System presented in Jacobson's work to assess the specification of aspect-oriented software. One of the limitations of the approach is the need to compose the aspect models manually, which increases the time spent in the modeling, while our approach allows the automatic composition of models. Theme/UML [4] supports the modeling of core and crosscutting concerns, automates the composition of models and represents the most important features of AOP. Theme/UML modifies the UML meta-model and is dependent on the version 1.3 of the language. Carton's proposal [16] aims to overcome this limitation, creating a marking profile to represent Theme/UML accordingly to the UML standards. However, the model composition is still limited to the constructs available on the version 1.3 of the UML. Our composition of models is not dependent on a specific meta-model, because it is realized using the standard meta-model elements. High-Level Aspects (HiLA) [17] uses state machines to model aspect-oriented applications. Pointcuts and advices are modeled using states and transitions. In comparison with graph-transformation approaches, that also uses state machines to describe aspects, the models created with HiLA are more easier to construct and understand. A tool that performs the composition (weaving) of aspects is proposed in a recent contribution [7], which automates part of the modeling process. However, HiLA changes the UML meta-model to represent aspect-oriented applications, and cannot be used in available CASE tools. We propose an extension through a UML profile, that can be used in CASE tools that allow the importing of profiles.

3 Specification of Aspect-Oriented Software

Figure 1 shows the UML profile to represent the structure and behavior of aspect-oriented applications. The profile may be used in CASE tools that support importing profiles through the XML Metadata Interchange (XMI) [18]. Relative to the structural modeling, this paper uses some definitions from the Evermann's profile [8], which allows the representation of the AOP's structural characteristics. The definitions used from Evermann's profile are the *CrosscuttingConcern* and the *Aspect* stereotypes, that are colored in beige in the profile diagram. A stereotype extends an element from the UML meta-model improving its semantics. In this paper, a meta-model element being extended will be represented inside parentheses. The *CrosscuttingConcern* stereotype extends (*Package*) and contains a set of aspects and classes. It represents a concern that impacts one or more parts of the system. The *Aspect* stereotype extends (*Class*), contains inter-type declarations and some configuration properties, such as: type of aspect instantiation, associated pointcut and a flag to indicate if this aspect is privileged. Inter-type declarations allows injection of members (method, attribute) on a class, change of inheritance hierarchy and interfaces implementation. To represent inter-type declarations, a stereotype denominated *ClassExtension* was created. This stereotype extends (*Class*) and is related to another stereotype named *Introduction*. The *Introduction* stereotype is used to mark which member is being inserted on a given class, or which inheritance relationship is being added, or which interface is being implemented. The stereotype *Introduction* extends the meta-model elements (*Attribute*), (*Operation*), (*Generalization*) and (*Realization*). Due to space limitations, we will not go into details regarding the structural modeling, which can be fully implemented using the proposed stereotypes.

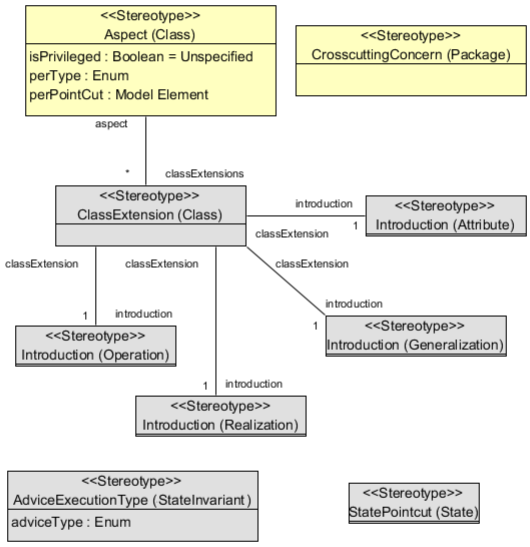

Fig. 1. UML Profile to model Aspect-Oriented Applications.

Besides inter-type declarations and aspect configurations, an aspect contains pointcuts and advices. Evermann's profile [8] represents pointcuts within the aspect definition, but without the possibility of capturing multiple join points in the pointcut specification. Our approach represents pointcuts with the UML state machine diagram. Each transition represents the capture of one or more join points, with the possibility of using wildcards. If the capture of all the join points is satisfied, that is, if the conditions for triggering a transition are satisfied, it is considered that the system has met a certain pointcut. The stereotype *StatePointcut* extends (*State*) and represents a pointcut. The composition of pointcuts is achieved by composing different state machines. The stereotype that represents pointcuts can be seen in the bottom right of figure 1. Figure 2 shows the definition of pointcuts using the proposed approach. The pointcut *AnyCall* captures calls (*call* pointcut) to any method of any class, using a wildcard to match any return type, any class and any method name with any number of parameters. The pointcut *RoomTarget* captures the occurences of a call when the target object (*target* pointcut) is of the type *Room*. Each pointcut is represented as a state in the state machine diagram. The pointcut signature is specified in the transition, which allows the use of wildcards to capture multiple joinpoints. When the pointcut state is reached, it means that the system has captured the execution points specified by the pointcut signature in the transition label.

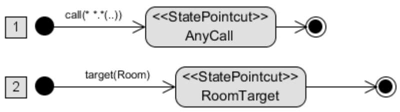

Fig. 2. The definition of two pointcuts.

The AspectJ language allows the composition of pointcuts with the logical operators *and* and *or*. The proposed approach performs the automatic composition of pointcuts using state machine digrams. Figure 3 shows the composition of the pointcuts *AnyCall* and *RoomTarget* using the *and* operator. The compound state machine contains a state with a *concurrent region*, containing sub-states that execute concurrently: *AnyCall* and *RoomTarget*. The synchronization occurs with the *fork* and *join* nodes, which means that the final state (*AnyCall AND RoomTarget*) only will be reached if both states (*AnyCall* and *RoomTarget*) are reached. This is the semantic of the *and* operator in the AspectJ language. Figure 4 shows the composition of the pointcuts using the *or* operator. Here the semantics is a bit different, because the system will reach the final state (*AnyCall OR RoomTarget*) when any of the pointcuts are reached. This is represented in the compound state machine, that shows direct transitions from both states to the final state.

A pointcut only captures the execution points of a system. The behavior to be injected in these points is represented by the advice, that is directly associated with a pointcut. Sequence diagrams are used to represent the behavior of the core concerns (core model) and the crosscutting concerns (crosscutting models). An aspect may have one or more advices and each one is represented by a crosscutting model. The behavior of an advice executes when its associated pointcuts are satisfied, i.e, all joint points are captured. As we model pointcuts with the

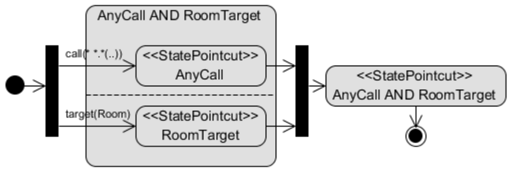

Fig. 3. The composition of two pointcuts with the AND operator.

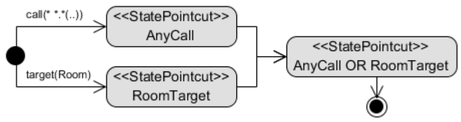

Fig. 4. The composition of two pointcuts with the OR operator.

state machine diagram and advices with the sequence diagram, the connection of the advices with pointcuts should be achieved using syntatic elements of these diagrams. We implement this connection using state invariants. A state invariant is an interaction fragment associated with a lifeline on the sequence diagram, representing a run-time constraint on the participants of the interaction. We use the reachability of a state as the constraint to trigger the advice execution.

The definition of a crosscutting model (as a sequence diagram) begins adding the aspect as the first lifeline of the diagram. A state invariant, which represents the satisfaction of a pointcut, is associated to the aspect lifeline. This means that the sequence of messages occurs only when the system reaches the state represented by the state invariant. The messages may be executed before, after or around the triggering of a pointcut. The execution time is configured using the *AdviceExecutionType* stereotype. This stereotype extends (*StateInvariant*) and can be seen in the bottom left of figure 1. The *AdviceExecutionType* has the tagged value *adviceType* of enumeration type. The valid types of the enumeration are: before, after or around.

The connection between advice and pointcuts can be better understood in the example of figure 5. This figure shows the pointcut previously specified to capture any method call on any class with any numbers of parameters (*AnyCall*). The log aspect defines a crosscutting model that logs a message using a *Logger*. The crosscutting model is described as a sequence of messages in a sequence diagram. This diagram contains a state invariant that refers to the *AnyCall* pointcut. A state invariant must have the advice execution type stereotype and the advice type tagged value to specify when the advice behavior will be executed. In this example, the advice behavior will be executed *after* the execution points captured by the *AnyCall* pointcut. The message *log()* will be executed only when the state *AnyCall* is achieved.

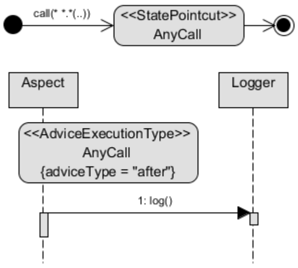

Fig. 5. The connection between pointcuts and advices using state invariants.

4 Visualization Toggling and Composition of Aspects

The profile is added to the SEA environment [12], which supports UML diagramming. The SEA/Aspect is a tool which allows the selection of which crosscutting models will be intertwined together with the core models of the system. One can automatically visualize only the core models, the crosscutting models, or core intertwined with crosscutting models. Aspects can be enabled or disabled dynamically, by updating the compound model, with no effort demanded from the developer. This automatic updating allows the toggling of views, visualizing different models compositions. The SEA/Aspect has two different phases to output a compound model: **selection** and **composition**, which are described in the next sections.

4.1 Selection

The developer selects which core and crosscutting models to compose and assign a different color to each model. These colors differentiate the elements of each model in the compound model. One or more aspects can be composed at the same time. The selected aspects are the input to the second phase, that is the composition of the core and crosscutting models.

4.2 Composition

The composition is made between a core model and one or more crosscutting models, and provides, as a result, a compound model with the structure or behavior of the crosscutting models injected into the core models. In the compound model, the croscutting behavior differs from the core behavior by different colors, as illustrated in figure 11. The composition algorithm has the **match** and **merge** activities. The first activity locates the captured concepts in the core model and the last weaves the structure or behavior of the crosscutting models in the core model, giving as result a compound model. SEA/Aspect allows the automatic composition of class and sequence diagrams. Due to space reasons, the composition of class diagrams will not be explained.

The **composition of sequence diagrams** uses wildcard matching and has the following steps:

- **Match:** With the core and crosscutting models selected by the developer, the composer finds the execution points of the core model that are impacted by the crosscutting models. This information is obtained from the pointcuts defined in the crosscutting models. The algorithm is separated in three steps:
  1. **Find the pointcut:** To obtain the pointcut from the crosscutting sequence diagram (crosscutting model), the algorithm looks for the state invariants stereotyped as *AdviceExecutionType*. The state invariant maps to the state that defines the pointcut.
  2. **Separate the pointcut:** The composer uses a regular expression to separate a pointcut in four parts: **pointcut type**, **return type pattern**, **identification pattern** and **exception pattern**. The pointcut type is mandatory and is one of the types supported by the AspectJ language, which includes *execution*, *call*, *this* and others. The return type pattern is optional and specifies the return type of the pointcut. The identification pattern is mandatory and contains the signature to be matched in the core model. Finally, the exception pattern is optional too, and is used to capture execution points that throws exceptions of a given type
  3. **Match the execution points:** The composer uses regular expressions to match which execution points are impacted by the pointcuts defined in the crosscutting models. This match supports pointcuts specified with wildcards, that is an important feature of aspect-oriented languages. It starts using the identification pattern to find context information about the impacted concepts, like package and class context of a given method, for example. When all concepts inside a given context are captured, the algorithm uses another regular expression to match the names of the captured concepts. For instance, when matching a method, the algorithm checks return type, parameters (name, type and number of parameters) and the method signature. Finally, the composer checks for exception throws, if any. As output, the concepts (classes and methods) impacted by the crosscutting models are stored to be used in the merge activity.
- **Merge:** Merge concepts of the crosscutting models with the impacted core model concepts. The merge receives as input the impacted core model concepts that should be merged with croscutting ones. The merge purpose is to inject a set of messages, lifelines, combined fragments and other sequence diagram concepts in the core sequence diagram (core model), adding new behavior defined in the crosscutting sequence diagrams (crosscutting models). To achieve this, the algorithm is separated in two steps:
  1. **Find the advice execution type:** Retrieve the advice execution type from the state invariant defined by the crosscutting model. The supported advice types are: *before*, *around* and *after* and are defined in the tagged value advice type. The advice type gives the information of when the messages should be injected in the core sequence diagram.

2. **Inject the messages:** At this time, the algorithm knows the impacted concepts, the messages to inject from the crosscutting model and when the messages should be inserted. The next step is the messages injection and reordering, because the injection of a message triggers a reordering event in the sequence diagram. With all the messages injected and ordered, the composer paints each message name with the correspondent crosscutting model color, to differentiate which message comes from which aspect. The merge produces as output a compound sequence diagram with the crosscutting concepts composed in the core sequence diagram.

5 Case Study

This section presents a case study to assess the applicability of the proposed approach in the modeling of an application. The case study is based in the Hotel Management System extracted from Jacobson's work [6]. This example allows the modeling of important aspect-oriented funcionalities and is consolidated in the literature. The use cases represented in this case study are the check-out of customers, messages logging and a program of loyalty points. As we are not going into details about the modeling of the structure of aspect-oriented software, the structural diagrams of the case study will not be shown in this section, although they can be represented using the stereotypes of the UML profile.

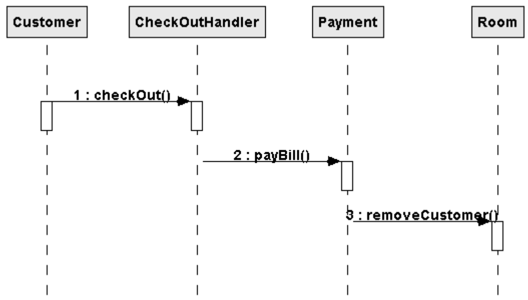

Fig. 6. Advice behavior of the core concern to check-out customers.

Sequence diagrams are used to represent the behavior of the core and crosscutting concerns, using the profile and the process proposed by this approach. The modeling of concerns using behavioral diagrams gives subsidies to achieve the toogling of views, that allows better understanding of aspect-oriented applications, visualizing the effect of the aspects in the system. We start with the sequence diagram of the check-out customers concern, that is shown in figure 6. The messages exchange starts when the object *CheckOutHandler* executes the message *checkOut()*. This message triggers the message *payBill()* to make

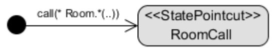

Fig. 7. Pointcut of the crosscutting concern to log messsages.

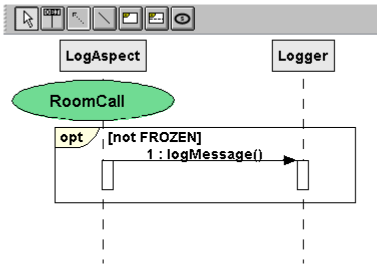

Fig. 8. Advice behavior of the crosscutting concern to log messages.

the payment of the customer account. Then, the customer is removed from the customers list using the method *removeCustomer()* of the *Room* class.

The other concerns to be modeled are the crosscutting concerns that contains pointcuts and advices. The message log concern needs to account the number of requests to a room. To achieve this, we must define a pointcut that captures the calls to the *Room* class. This pointcut is modeled using the state machine diagram and is shown in figure 7. With the pointcut defined, we create the sequence diagram to the log concern. The diagram is shown in figure 8 and contains as the first lifeline the *LogAspect*. Accordingly to the proposed approach, an aspect in the sequence diagram must have a state invariant associated, which is the trigger to execute the sequence of messages. In this case, the state invariant *RoomCall* is associated with the *LogAspect* and points to a pointcut previously defined in the state machine diagram. The semantics here is that the sequence of messages will execute only when the pointcut is satisfied. The message to be executed when the pointcut is satisfied is a call to the method *log()* of the *Logger* class. It is important to be aware of the advice type, that is defined as a tagged value in the state invariant. The tagged value is not showed in the sequence diagram to not clutter the diagram. In this case, the advice type is after, which means that the log behavior will execute after any method call to the *Room* class. The sequence diagram contains a combined fragment of type optional, that defines that the log only will be performed if the application is not frozen. An application is not frozen when it is running in the development environment.

The last concern to be modeled is the handling of the loyalty points concern, that accumulates points to a customer when it makes a payment. To model the loyalty point concern, we need a pointcut that captures when a customer makes the payment of an account. Figure 9 shows a pointcut that captures calls to the method *payBill()* of the *Payment* class. Besides the pointcut definition, we need

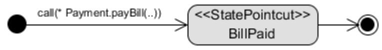

Fig. 9. Pointcut of the crosscutting concern to the loyalty points program.

to define the behavior of how a customer earn points in the loyalty program. This behavior is modeled in the sequence diagram of figure 10. The diagram has the *EarnPointsAspect* as the first lifeline and has a sequence of messages to be executed when the customer make a payment. These messages will be executed after the payment, because the advice type of the state invariant is after. Again, the tagged value is not showed in the sequence diagram to not clutter the diagram.

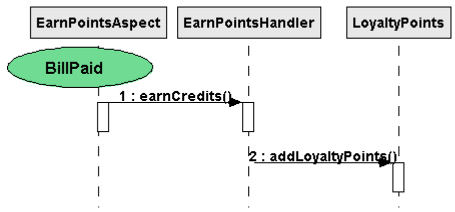

Fig. 10. Advice behavior of the crosscutting concern to the loyalty points program.

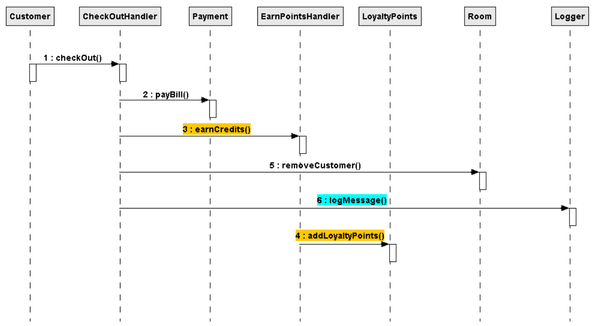

Fig. 11. Case Study: Check-out composed with log and loyalty points program.

After the modeling of all the core and crosscutting concerns, the modeler may interchange the model views, selecting which concerns he wants to see composed in the same diagram. The SEA/Aspect tool allows the selection of more than one model to be composed at the same time. Figure 11 shows a compound model, with the concerns of check-out customers, log and loyalty points composed together. The compound diagram shows the model selector in it is right bottom, that allows changing the models that are being show. The messages of different concerns are illustrated with different colors, as each concern has a color associated with it. This is useful to differentiate which messages comes from which aspect in the compound model.

6 Conclusion

The UML profile to represent aspect-orientad applications and the SEA/Aspect tool are the main contributions of this work. With the profile and the visualization tool, it is possible to model an aspect-oriented system and to visualize the core and crosscutting models in a compound model, allowing the visualization of the aspect dynamics in the system. The proposed profile may be reused in CASE tools that support the importing of profiles.

The main advantages of the proposed approach in relation to the others are: the automatic composition of pointcuts using the state machine diagram; the definition of a profile within the UML standards; and the composition and automatic visualization of the effect of the crosscutting models in the core models in the SEA environment, which eases the understanding and maintenance of aspect-oriented systems. The tool that allows the composition and visualization toggling of concerns may be added to CASE tools that allow functional extension, which is the case of the SEA environment.

References

1. Laddad, R.: AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications Co., Greenwich, CT, USA (2003)
2. (OMG), O.M.G.: Unified modeling language superstructure 2.4.1. Technical report, Object Management Group (OMG) (october)
3. Kienzle, J., Al Abed, W., Klein, J.: Aspect-oriented multi-view modeling. In: Proceedings of the 8th ACM international conference on Aspect-oriented software development. AOSD '09, New York, NY, USA, ACM (2009) 87–98
4. Baniassad, E., Clarke, S.: Theme: An approach for aspect-oriented analysis and design. In: Proc. Int'l Conf. Software Engineering (ICSE), Washington, DC, USA, IEEE Computer Society Press (2004) 158–167
5. Klein, J., Fleurey, F., Jézéquel, J.M.: Transactions on aspect-oriented software development iii. Springer-Verlag, Berlin, Heidelberg (2007) 167–199
6. Jacobson, I., Ng, P.W.: Aspect-Oriented Software Development with Use Cases (Addison-Wesley Object Technology Series). Addison-Wesley Professional (2004)
7. Zhang, G., Hölzl, M.: Weaving semantic aspects in hila. In: Proceedings of the 11th annual international conference on Aspect-oriented Software Development. AOSD '12, New York, NY, USA, ACM (2012) 263–274
8. Evermann, J.: A meta-level specification and profile for aspectj in uml. In: Proceedings of the 10th international workshop on Aspect-oriented modeling. AOM '07, New York, NY, USA, ACM (2007) 21–27

9. Cottenier, T.: The motorola weavr: Model weaving in a large industrial context. In: in Proceedings of the International Conference on AspectOriented Software Development, Industry Track. (2006)
10. Stein, D., Hanenberg, S., Unland, R.: A UML-based aspect-oriented design notation for AspectJ. In: AOSD '02: Proceedings of the 1st international conference on Aspect-oriented software development, New York, NY, USA, ACM (2002) 106–112
11. AspectJ: The aspectj project (october 2012)
12. Silva, R.P.: Support to the development and use of frameworks and components. PhD thesis, UFRGS/PPGC (march 2000)
13. Fuentes-Fernández, L., Vallecillo-Moreno, A.: An Introduction to UML Profiles. UPGRADE, European Journal for the Informatics Professional **5**(2) (April 2004) 5–13
14. Stein, D., Hanenberg, S., Unland, R.: Expressing different conceptual models of join point selections in aspect-oriented design. In: Proceedings of the 5th international conference on Aspect-oriented software development. AOSD '06, New York, NY, USA, ACM (2006) 15–26
15. Hanenberg, S., Stein, D., Unland, R.: From aspect-oriented design to aspect-oriented programs: tool-supported translation of jpdds into code. In: Proceedings of the 6th international conference on Aspect-oriented software development. AOSD '07, New York, NY, USA, ACM (2007) 49–62
16. Carton, A., Driver, C., Jackson, A., Clarke, S.: Transactions on aspect-oriented software development vi. In Katz, S., Ossher, H., France, R., Jézéquel, J.M., eds.: Transactions on Aspect-Oriented Software Development VI. Springer-Verlag, Berlin, Heidelberg (2009) 238–266
17. Zhang, G., Hlzl, M.: Hila: High-level aspects for uml state machines. In Ghosh, S., ed.: Models in Software Engineering. Volume 6002 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 104–118
18. (OMG), O.M.G.: Meta-Object Facility XMI Specification. Technical report, Object Management Group (OMG) (october 2011)