
Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

**Technical Report No. NJU-SEG- 2009-IC-003**

# Modeling and Integrating Aspects with UML Activity Diagrams

Zhanqi Cui, Linzhang Wang, Xuandong Li, Dianxiang Xu

Postprint Version. Originally Published in: Proceedings of the 24th Annual ACM Symposium on Applied Computing (ACM SAC2009), USA, ACM Press, 2009, pp.430-437.

# Modeling and Integrating Aspects with UML Activity Diagrams

Zhanqi Cui, Linzhang Wang and Xuandong Li
State Key Laboratory of Novel Software Technology
Department of Computer Science and Technology, Nanjing University
Nanjing, Jiangsu, P.R. China 210093
zqcui@seg.nju.edu.cn;{lzwang,lxd}@nju.edu.cn

Dianxiang Xu
Department of Computer Science,North Dakota State University,Fargo, ND 58105, USA
dianxiang.xu@ndsu.edu



## ABSTRACT

Dealing with crosscutting concerns has been a critical problem in software development processes. Aspect-Oriented Programming (AOP) provides a viable programming-level solution by separating crosscutting concerns from primary concerns. To facilitate handling crosscutting concerns at earlier software development phases, this paper proposes an aspect-oriented modeling and integration approach at the design level. In our approach, primary concerns are depicted with UML activity diagrams as primary models, whereas crosscutting concerns are described with aspectual extended activity diagrams as aspect models. Each aspect model consists of pairs of pointcut and advice model. Aspect models can be integrated into primary models automatically. To this end, a prototype tool called Jasmine-AOI has been implemented as an Eclipse plug-in. With the tool support, we have conducted two case studies, including 15 primary models and 8 aspect models. The case studies demonstrate that our approach can greatly facilitate reasoning about crosscutting concerns when a system is modeled with activity diagrams.

## Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*Methodologies*

## General Terms

Design

## Keywords

crosscutting concern, aspect-oriented modeling, integration, UML activity diagram

## 1. INTRODUCTION

Dealing with crosscutting concerns has been a critical problem in software development processes. Aspect-oriented programming (AOP) [1] provides a viable programming-level solution by modularizing crosscutting concerns into aspects. Aspect-oriented modeling (AOM) handles crosscutting concerns by providing a higher level of abstraction to alleviate software complexity in the design phase. In particular, earlier awareness of crosscutting concerns in the model-centric design can guide the subsequent implementation and validation activities. UML activity diagrams [2] can be used for various purposes, such as validating existing requirements, guiding system implementation, and generating test cases for an implementation. They are usually drawn to depict control flows, behaviors of software, and scenarios performed by different stakeholders. Accordingly, crosscutting concerns, such as authentication and logging, cut across multiple activity diagrams that model different scenarios. Therefore, a rigorous approach to aspect-oriented modeling with activity diagrams is highly desirable. This approach is also expected to integrate aspect models with primary models automatically in order to reason about the impacts of crosscutting concerns on the primary concerns.

***Proposed solution.*** In this paper, we propose a rigorous approach of aspect-oriented modeling with activity diagrams. In our approach, activity diagrams are not only used to describe primary concerns, but also extended to represent crosscutting concerns. Crosscutting concerns are either sequential or parallel aspects that are running sequentially or in parallel with primary concerns. For example, the authorization concern in a banking system is a sequential aspect because the failure of authorization will terminate the subsequent process of the primary concern. An informing concern that notifies users in a mailing list when subscribed events happen is a parallel aspect, because failures of informing should not affect the subsequent primary processes. We integrate aspect models with primary models by three steps: identifying join points in the primary models which match the pointcut models, initializing the advice models based on the identified join points, and weaving the initialized advice models into the primary models.

***Contributions.*** The main contribution of this paper is a lightweight aspect-oriented extension to activity diagrams for modeling crosscutting concerns as sequential and parallel aspects with pointcut and advice models. The elements of activity diagrams that can be defined as join points are


*SAC'09* March 8-12, 2009, Honolulu, Hawaii, U.S.A.

systematically categorized and appropriate advice types for them are summarized. A set of rules is developed to automatically integrate aspects with primary models. In addition, we report two case studies that demonstrate the feasibility and effectiveness of our approach.

The rest of the paper is organized as follows. We start by introducing the aspectual extension to activity diagrams and aspect-oriented modeling with activity diagrams in section 2. Section 3 discusses our approaches for integrating aspects with primary models. Section 4 presents the prototype tool and the two case studies. Section 5 reviews related work. Finally we conclude this paper and discuss the future work in section 6.

## 2. ASPECT-ORIENTED MODELING WITH ACTIVITY DIAGRAMS

UML activity diagrams can describe control flow based program logic at different levels of abstraction and express concurrency more naturally. The meaning of activity diagrams is explained in terms of petri-net notations (tokens, flows etc.) instead of state machines. There are three types of elements in activity diagrams: *nodes*, *edges*, and *groups*. Each node is *ActionNode*, *ControlNode*, or *ObjectNode*. *Nodes* in activity diagram are connected by *edges* which include *ControlFlow* and *ObjectFlow*. *Groups* are general grouping constructs for nodes, edges, and groups. A group may contain other groups, and elements can belong to more than one group. Groups can put related nodes and edges together and highlight the relationship between them. We formally define activity diagrams as follows.

*Definition 1.* **(Activity Diagram)**. An activity diagram AD is a 4-tuple (N, E, G, F), where:

- N = $\{n_1, n_2, \cdots, n_i\}$ is a finite set of nodes.
- E = $\{e_1, e_2, \cdots, e_j\}$ is a finite set of edges.
- G = $\{g_1, g_2, \cdots, g_k\}$ is a finite set of groups.
- $F \subset (N \times E) \cup (E \times N)$ is the flow relation between nodes and edges. Let *Predecessor* and *Successor* be two flow relation functions. For each element $e \in$ (N $\cup$ E), *Predecessor(e)*=$\{e_i | e_i \in (N \cup E) \wedge (e_i, e) \in F\}$, *Successor(e)*= $\{e_i | e_i \in (N \cup E) \wedge (e, e_i) \in F\}$;

### 2.1 A Running Example

Let us consider a banking system adapted from [10]. Fig. 1 and Fig. 2 show the traditional models of withdraw and transfer scenarios. In the activity diagrams the primary features are accompanied by five crosscutting security concerns: *Authentication*, *Authorization*, *Logging*, *Informing*, and *Monitoring*. They are indicated by the gray areas. It is easy to tell that the primary features and crosscutting concerns are tangled together. Also, most of the crosscutting concerns are involved in both models. This makes them redundant and difficult to maintain. Our approach will separate these crosscutting concerns into aspect models.

Two of the crosscutting concerns that appear in the withdraw scenario are chosen for the running example. One concern is the requirement of authorization which needs to be performed before transactions, in order to ensure that the authorized user has been validated. The other concern is the

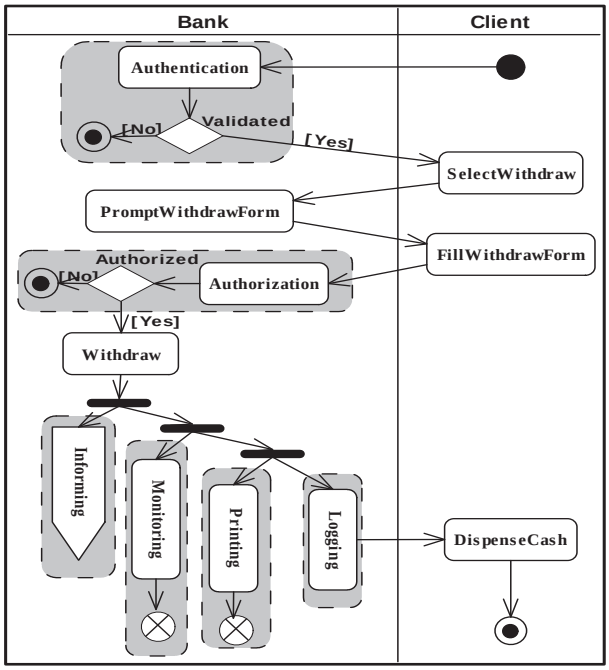


**Figure 1: The traditional model of withdraw**

requirement of informing users that the balance has been changed after transactions are completed in order to detect potential embezzlements. Obviously, these two security concerns also cut across other scenarios that modify account balance, such as deposit and transfer. Fig. 3 describes the primary model of the withdraw scenario without considering the crosscutting concerns. This model is thus much simpler than Fig. 1. In the following sections, we will illustrate how to modularize crosscutting concerns into aspect models and integrate the aspects with the primary model.

### 2.2 Aspectual Extension to Activity Diagrams

For modeling crosscutting concerns, we follow such terms as "join point", "pointcut", and "advice" from the AspectJ[3] terminology. However, we introduce them into activity diagrams with similar meanings.

As listed in Table 1, we introduce seven stereotypes and three tagged values into activity diagrams. The *Target* column specifies where the extensions could be used. ≪Pointcut≫ is used to stereotype an activity diagram to indicate that it is a pointcut model. A tagged value "advice" is added to ≪Pointcut≫ for denoting the name of the corresponding advice model. In order to denote the position of the join point element in a pointcut model, mark the join point element with the stereotype ≪Joinpoint≫. An element in a pointcut model stereotyped with ≪Argument≫ indicates that elements in primary models matched to it are actual arguments of the related formal parameter in the corresponding advice model. A tagged value "parameter" is added to ≪Argument≫ for denoting names of the related formal parameter. The formal parameters in the corresponding advice model can map to actual arguments by this tagged value. ≪Advice≫ is used to stereotype an activity diagram to indicate that it is an advice model. A tagged value "type" is added to ≪Advice≫ for denoting the type of the ad-

**Table 1: Extensions for modeling aspects**

| Extension | Type | Applies To | Description |
|---|---|---|---|
| ≪Pointcut≫ | Stereotype | Diagram | Indicate that an activity diagram is a pointcut model. |
| advice | Tagged Value | ≪Pointcut≫ | Indicate the corresponding advice model of the pointcut model. |
| ≪Joinpoint≫ | Stereotype | Element | Denote the position of the join point element in a pointcut model. |
| ≪Argument≫ | Stereotype | Element | Indicate elements that serve as actual arguments for related formal parameters in the corresponding advice model. |
| parameter | Tagged Value | ≪Argument≫ | Denote the name of the element in the advice model which are related the ≪Argument≫ element. |
| ≪Advice≫ | Stereotype | Diagram | Indicate that an activity diagram is an advice model. |
| type | Tagged Value | ≪Advice≫ | Indicate the type of the advice model. "type" is either "Before" or "After". |
| ≪Entry≫ | Stereotype | Node | Denote where tokens flow in an advice model from primary models. |
| ≪Exit≫ | Stereotype | Node | Denote where tokens flow out an advice model to primary models. |
| ≪Parameter≫ | Stereotype | Element | Indicate elements that serve as formal parameters in an advice model. |

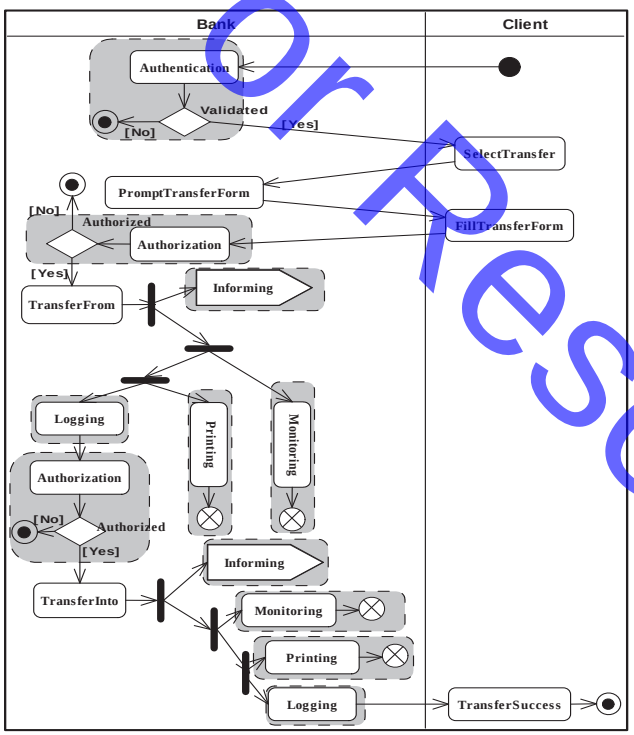


**Figure 2: The traditional model of transfer**

vice model. ≪Entry≫ and ≪Exit≫ are used to stereotype two nodes in an advice model to denote where tokens will flow in from and flow out to primary models, respectively. ≪Parameter≫ is applied to indicate elements which are formal parameters in an advice model. A formal parameter's name equals to the related ≪Argument≫ element's tagged value "parameter" in the corresponding pointcut model.

## 2.3 Modeling Aspects

Crosscutting concerns are depicted by *aspect models*, which consist of pointcut models and corresponding advice models, both are specified by extended activity diagrams. An *pointcut model* serves as a predicate to select join points from primary models and specifies advice model to be applied to these join points picked out from the primary models. An *advice model* specifies additional enhancements or constraints with respect to the crosscutting concern under study.

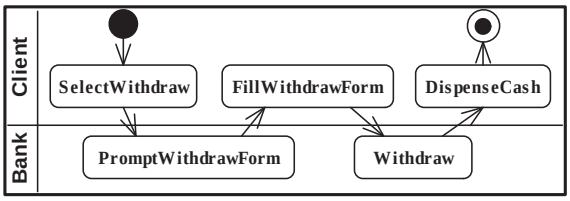


**Figure 3: The primary model of withdraw**

### 2.3.1 *Join Points in the Activity Diagram*

In order to design and integrate aspect models, positions in primary models that are appropriate to apply advices should be defined firstly. These positions are called *join points*. In AspectJ, join points are defined as "Principled points in the dynamic execution of a program" [3] such as: method calls, method executions, and object initializations, whereas join points at the model level are lack of systematical analysis and categorization. In [4], join points were tagged on primary models which would retangles crosscutting concerns with primary features again. McNeile et al. [5] considered join points as events and states. Fuentes et al. [6] defined join points as observable behaviors in the activity diagram, such as object creation and destruction, the sending and receiving of a message/method, the throwing of an event, and the raising of an exception.

In our approach, join points are standard elements of activity diagrams where crosscutting enhancements or constraints could be added. More specifically, for an element to be a join point, it should be a standard element and has an explicit name to distinguish itself from other elements in an activity diagram, and it can be applied to additional enhancements or constraints caused by crosscutting concerns. For example, an *Action* can be a join point, however, a *ControlNode* cannot. Table 2 lists all elements of activity diagrams that can be defined as join points.

### 2.3.2 *Pointcut*

A pointcut model is used to select join points by specifying the positions in primary models to which the corresponding crosscutting concerns should be applied. Stein et al. presented a graphical way, which is a new diagram named Join Point Designation Diagram(JPDD) [7], to represent the selection of join points in a programming language independent manner. In [8] and [9], pointcuts were specified by explicit textual directives. In [6], a pointcut model is expressed by means of a sequence diagram. In our approach, pointcuts are modeled with extended activity diagrams. We

**Table 2: Join points in the activity diagram**

| | Name | Join Point | Advice Type | |
|---|---|---|---|---|
| | | | Before | After |
| Nodes | Action | √ | √ | √ |
| | AcceptEventAction | √ | × | √ |
| | SendSignalAction | √ | √ | × |
| | DecisionNode/MergeNode | × | - | - |
| | ForkNode/JoinNode | × | - | - |
| | InitialNode/ActivityFinal | × | - | - |
| | FlowFinal | × | - | - |
| | DataStore | √ | √ | √ |
| | ObjectNode | √ | √ | √ |
| Edges | ControlFlow | √ | √ | √ |
| | ObjectFlow | √ | √ | √ |
| Groups | Activity | √ | √ | √ |
| | ExceptionHandler | √ | √ | × |
| | ExpansionRegion | √ | √ | √ |
| | ParameterSet | × | - | - |
| | InterruptibleActivityRegion | × | - | - |
| | ActivityPartition | × | - | - |

choose activity diagrams to represent pointcut because we intend to implicitly filter join points based on their behavioral properties rather than explicit directives. The pointcut model is defined as follows.

*Definition 2.* **(Pointcut Model)**. A pointcut model PM is an extended activity diagram with a 7-tuple (N, E, G, F, AM, A, j), where:

- N, E, G, F are the nodes, edges, goups, and flow relations of an activity diagram for the pointcut model.
- AM denotes the name of the corresponding advice model.
- A = $\{a_1, a_2, \cdots, a_u\} \subseteq (N \cup E \cup G)$ is a set of actual arguments. Set A establishes a bridge which connects formal parameters in the corresponding advice model to primary models' elements matched A.
- j $\in (N \cup E \cup G)$ is a special element that specifies the location constraints of candidate join points in activity diagrams of primary concerns.

A pointcut model is stereotyped with ≪Pointcut≫. In order to indicate the corresponding advice model of the pointcut model, a tagged value "advice" is added to ≪Pointcut≫. The pointcut model is used to select a set of join points that satisfy certain constraints from primary models. In the activity diagram, join points can be identified by self information, structural scope, and running context. Based on the semantics of the activity diagram, a pointcut model specifies the constraints of candidate join points by the following three facets:

- Signature pattern of the join points. Such as the names of elements, the types of elements. Wildcards (such as "*" and "?") and logic operators (such as "||", "&&", and "!") are introduced to improve the expressibility of pointcut model. Such constraints can be mapped to the *call* or *execution* pointcut in AspectJ.
- Scope constraints. For example, join points can be specified by indicating which groups they belong to. Such constraints can be mapped to the *within* or *withcode* pointcut in AspectJ.

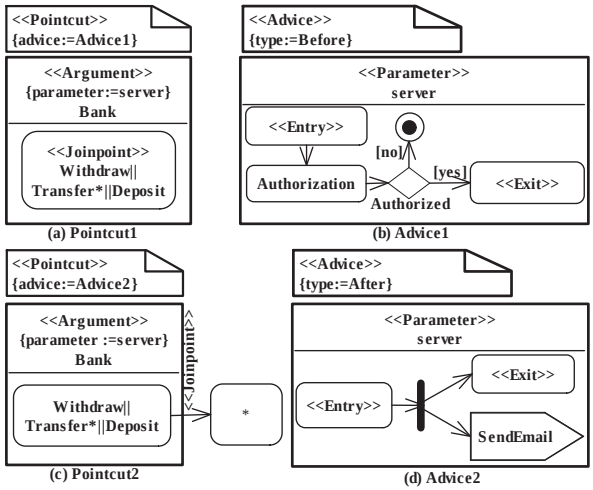


**Figure 4: Pointcut and advice models of authorization and informing**

- Running context. For example, join points can be specified by indicating their predecessor and successor elements. Such constraints can be mapped to *cflow* or *cflowbelow* pointcut in AspectJ.

In the running example, we construct two pointcut models to select join points from the primary models where the authorization and parallel informing aspects emerge, respectively. The pointcut model Pointcut1 depicted in Fig.4 (a) is constructed to select the elements in primary models to which the authorization advice will be applied. The pointcut model is stereotyped with ≪Pointcut≫. A tagged value "advice" indicates the corresponding advice model is "Advice1". This pointcut model describes the constraints of target join points from the three facets that we defined in section 2.3.1: the join point is an ActionNode, the name of node is "Withdraw", "TransferInto", "TransferFrom", or "Deposit"; the node belongs to an ActivityPartition named "Bank"; the predecessor and successor elements of the node are arbitrary. There is an argument element "Bank" in the pointcut model with a tagged value "parameter:=server". This tagged value maps this argument to the formal parameter element "server" in Advice1. This pointcut model will capture the node "Withdraw" in Fig.3. The pointcut model Pointcut2 depicted in Fig.4 (c) is constructed to select join points in the primary models to which the parallel informing advice will be applied. The join points should meet the following constraints defined in this pointcut model: the join point is an edge of ControlFlow; the edge has a predecessor node and arbitrary successor nodes (The predecessor node is in an ActivityPartition named "Bank", the name of the predecessor node should be "Withdraw", "Transfer", or "Deposit"). This pointcut model will capture the outgoing edge of "Withdraw" in Fig.3.

### 2.3.3 Advice

Advices are also modeled by extended activity diagrams. An advice model is defined as follows.

*Definition 3.* **(Advice model)**. An advice model AM is an extended activity diagram, which can be defined as an 7-tuple (N, E, G, F, P, entry, exit), where:

- N, E, G, F are the nodes, edges, groups, and flow relations of an activity diagram for the advice model.
- P = $\{p_1, p_2, \cdots, p_v\} \subseteq (N \cup E \cup G)$ is a set of formal parameters.
- entry $\in$ N and exit $\in$ N are two nodes that indicate where the tokens will flow in from and flow out to primary models, respectively. Predecessor(entry) $\notin$ E $\bigwedge$ Successor(exit) $\notin$ E.

An advice model is stereotyped with ≪Advice≫. In order to indicate the type of the advice model, a tagged value "type" is added to ≪Advice≫. "type" would be either "Before" or "After". Similar to AspectJ, an advice can be extra operations before or after matched join points in primary models. A "*Before*" advice is enhancements that should be finished before the join points, while an "*After*" advice is enhancements running after the join points. We impose a constraint that advice cannot change the flow sequence of the primary models other than adding additional process sequence into the primary models or terminate the progress. For this reason, the "*Around*" advice is not supported in the current approach. The advice types for potential joint points are listed in Table 1.

In this paper, crosscutting concerns are modeled as sequential and parallel aspects. *Sequential aspects* are critical features that their running results determine the residual processes in primary models. *Parallel aspects* are uncritical and time consuming features that their running results should not influence the residual processes in primary models. In the activity diagram, nodes and groups represent some operations or behaviors, whereas edges represent transformation of tokens. This semantic diversity leads to different effects of advice for nodes and edges. Based on this difference, the advice for nodes and groups is typically sequential, and the advice for edges is parallel.

In the running example, the authorization concern is to check whether or not the user is granted with permissions before transactions are performed. The informing concern is to send the user an email after the transactions are successfully performed, notifying that the balance has changed. Fig. 4 (b) models the authorization concern as sequential advice which means that the authorization action needs to be performed before the join point nodes. As the advice model presents, if the user has been granted with permissions, he can continue the process to finish the transaction. Otherwise, if the user is unauthorized, the transaction will be terminated. The advice model is stereotyped with ≪Advice≫. The tagged value "type", which is tagged on ≪Advice≫, indicates the type of the advice is "Before". In the advice model, there is an element named "server" stereotyped ≪Parameter≫ that serves as a formal parameter. The two nodes stereotyped ≪Entry≫ and ≪Exit≫ denote where the tokens will flow in from and flow out to the primary models respectively. Fig. 4 (d) models the informing concern as parallel advice. In the advice model the "SendEmail" action is fired at the join point and running in parallel with the residual flow of the primary model. Informing through email is a time consuming process and is not a critical feature. Therefore, the result of the informing advice should not influence the main flow of the primary model. As the advice model presents, the action "SendEmail" is fired after the synchronizion bar.

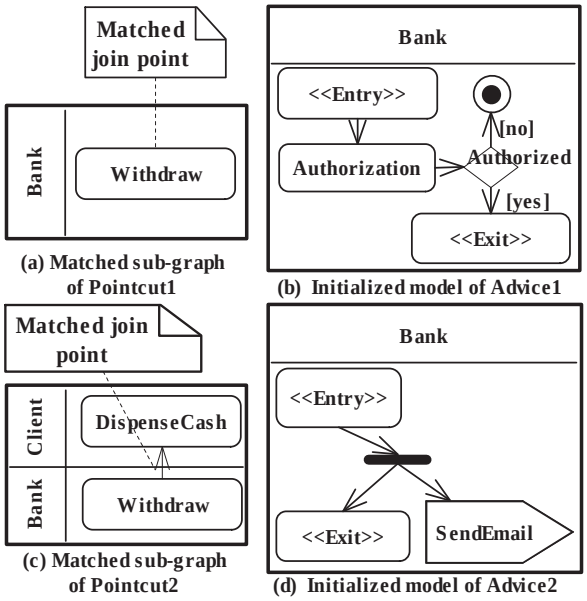


**Figure 5: Matched sub-graphs and initialized advice models of authorization and informing**

## 3. INTEGRATING ASPECT MODELS

To understand how crosscutting concerns affect primary features, we can integrate the aspect models into the primary models to create a system model. The integration is done by finding join points in primary models, initializing advice models, and weaving advices into primary models.

### 3.1 Finding Join Points

The first step of integration is to find all join points in a primary model that match a given pointcut. This is to find the primary model's sub-graphs that match the behaviors defined in the pointcut model. The nodes, edges, and groups of the primary model are possible join points when they satisfy the constraints defined in the pointcut model.

*Definition 4.* **(Matching Sub-Graph)**. A matching sub-graph between a primary model AD ($N_1$, $E_1$, $G_1$, $F_1$) and a pointcut model PM ($N_2$, $E_2$, $G_2$, $F_2$, AM, A, j) is a sub-graph MSD ($N_3$, $E_3$, $G_3$, $F_3$) of AD which satisfies the following conditions.

- $N_3 \subseteq N_1$, $E_3 \subseteq E_1$, $G_3 \subseteq G_1$, and $F_3 \subseteq F_1$.
- There is an element $e_i \in (N_3 \cup E_3 \cup G_3)$ conforms[1] to PM.j.
- For each element $e_j \in ((N_2 \cup E_2 \cup G_2)$-PM.j), there is an element $e_k \in (N_3 \cup E_3 \cup G_3)$ conforms to $e_j$.

The algorithm for finding out all matching sub-graphs between a primary model and a pointcut model is a bidirectional breadth first traversal of a directed diagram. The starting points of the traversal are the elements in the primary model that conform to the join point element of the

[1]Two elements are said to be conformed to each other if all attributes of the two elements are equal. Two attributes are equal if their values are equal with consideration of logical operations and wildcards or at least one of them is null.

pointcut model. The traversal is a recursive process that compares the predecessor and successor elements of the initial element and the corresponding element in the pointcut model, respectively. For each element in the pointcut model, if a conforming element is found in a sub-graph of the primary model, the sub-graph is a matching sub-graph, i.e., a join point is identified in the primary model. Algorithm 1 details how matching sub-graphs are identified.

**Algorithm 1.** Identifying matching sub-graphs

**INPUT**: primary model AD ($N_1$, $E_1$, $G_1$, $F_1$ ),
pointcut model PM ($N_2$, $E_2$, $G_2$, $F_2$, advice, A, j)
**OUTPUT**: Set MSGs *//each $MSG_i$ in MSGs is a matching //sub-graph between AD and PM*
**for each** element $e_i$ **in** AD {
  **if**($e_i$ conforms to PM.j) {
    Sub-graph temp := Compare({PM.j}, {$e_i$}, **null**)
    **if**(temp != **null**) MSGs := MSGs + temp
    *// found a matching sub-graph*
  }
} **return** MSGs

PROCEDURE: Compare(Set $Ele_1$, Set $Ele_2$, Sub-graph temp)
*/*If for all elements in $Ele_1$, there is a corresponding element in $Ele_2$, then return the subset of $Ele_2$ which match to $Ele_1$, else return null.*/*
**for each** element $ele_1$ **in** Set $Ele_1$ {
  **if** there is an $ele_2$ in $Ele_2$ equal to $ele_1$ {
    **if** ($ele_2$ is a node) temp.N := temp.N + $ele_2$
    **else if** ($ele_2$ is an edge) {
      temp.E := temp.E $\bigcup$ $ele_2$
      temp.F := temp.F $\bigcup$ ($ele_2$, Successor($ele_2$))
      temp.F := temp.F $\bigcup$ (Predecessor($ele_2$), $ele_2$)
    } **else** temp.G := temp.G $\bigcup$ {$ele_2$}
    $Ele_2$ := $Ele_2$ - $ele_2$
    Set $Pre_1$ := Predecessor($ele_1$), $Pre_2$ := Predecessor($ele_2$)
    **if** ($Pre_1$ != **null**) Before := Compare($Pre_1$, $Pre_2$, temp)
    **if** (Before == **null**) **return null**
    *// predecessor elements of $ele_1$ and $ele_2$ not match*
    Set $Succ_1$ := Successor($ele_1$), $Succ_2$ :=Successor($ele_2$)
    **if** ($Succ_1$ != **null**) After := Compare($Succ_1$, $Succ_2$, temp)
    **if** (After == **null**) **return null**
    *// successor elements of $ele_1$ are not match to $ele_2$*
  } **else return null** *// no element in $Ele_2$ matches to $ele_1$*
} **return** temp

Consider Fig. 3. It is the primary model of the withdraw scenario in the banking system. The node "Withdraw", matches to the node stereotyped ≪Joinpoint≫ in the pointcut model Pointcut1 defined in Fig. 4 (a) with the consideration of wildcards and logic operations. So, the sub-graph in Fig. 5 (a), which contains the ActivityPartition "Bank" and the node "Withdraw", matches Pointcut1. Similarly, the sub-graph in Fig. 5 (c), which contains the ActivityPartition "Bank", the node "Withdraw", the outgoing edge of "Withdraw", and the node "DispenseCash", matches the pointcut model Pointcut2 defined in Fig. 4 (c).

## 3.2 Initializing Advice Models

For the purposes of reuse, advice models are parameterized as templates. Formal parameters in an advice model serve as interfaces. The elements captured by the corresponding pointcut model from primary models bear similarity to actual parameters in a function call. An advice model needs to be initialized before being woven into primary models if it contains any formal parameters. All the formal parameters in it are replaced with the elements from the primary models which conform to the related actual arguments of the corresponding pointcut model. Thus, we establish a mechanism for parameter-passing between primary and aspect models. This mechanism makes an advice model reusable as a template upon various environments of join points in primary models. Algorithm 2 describes how an advice model is initialized.

**Algorithm 2.** Initializing an advice model

**INPUT**: pointcut model PM ($N_1$, $E_1$, $G_1$, $F_1$, AM, A, j),
advice model AM($N_2$, $E_2$, $G_2$, $F_2$, $P$, *entry*, *exit*),
$MSG_i$ which is a matching sub-graph
**OUTPUT**: $AM_i$ ($N_i$, $E_i$, $G_i$, $F_i$, $P_i$, $entry_i$, $exit_i$)
*//Initialized advice model of AM using $MSG_i$*
$N_i$ := $N_2$, $E_i$ := $E_2$, $G_i$ := $G_2$, $F_i$ := $F_2$, $P_i$ := $P$,
$entry_i$ := *entry*, $exit_i$ := *exit*
**for each** $p_v$ **in** $P_i$ {
  element $a_u$ := the actual argument in PM.A related to $p_v$
  element e := the element in $MSG_i$ which conforms to $a_u$
  replace $p_v$ with e
} **return** $AM_i$

In the two advice models of the running example in Fig. 4, there is a formal parameter of an ActivityPartition 'server" in both Advice1 and Advice2. This formal parameter intends to catch the ActivityPartition in primary models which represents the server side. Pointcut1 and Pointcut2 both capture ActivityPartition "Bank" for the formal parameter 'server". The ActivityPartition "Bank" is the actual value of the formal parameter 'server". The effect of initializing Advice1 and Advice2 is that the ActivityPartition 'server" in both of the two advice models are replaced with "Bank" in the primary model. The initialized advice modes are depicted in Fig. 5 (b) and (d).

## 3.3 Weaving Advices into Primary Models

In this subsection, we discuss the weaving of initialized advices and primary models based on the types of advice and join points. Our approach allows to explicitly specify the precedence in which aspect models are integrated.

If the join point of a pointcut model is a node or a group, the effect of weaving an advice model into a primary model is that the additional processes designed in the advice model are inserted into the primary model based on the identified join points. The advice will process before or after the join points, depending on the advice type, the actions defined in the advice model, and the sequential primary model. If the join point of a pointcut model is an edge, the effect of weaving an advice model into a primary model is to synchronize running flows of the advice and the primary model at the identified join points. If the advice type is "Before", the processes designed in the advice model and the primary model run in parallel and then they are synchronized at the join point. Otherwise, if the advice type is "After", the join point forks two tokens to fire the advice model and the residual of the primary model simultaneously. The detail of weaving an advice model with a primary model is given in algorithm 3.

Fig. 6 is the integrated model of the withdraw scenario after weaving the advice models of *authorization* and *informing* with the primary model. In this model, the authorization advice was inserted before the "Withdraw" node and the informing advice was inserted after outgoing edge of the "Withdraw" node. With the petri-net like semantics of the activity diagram, it is obvious to find that the "Withdraw" can only be fired after successfully authorized. Its firing will enable both "DispenseCash" and "SendEmail". And the result of informing is disrelated with the main action flow of the withdraw scenario. This running semantic fulfills the requirements of authorization and informing.

In fact, the two security concerns in the withdraw sce-

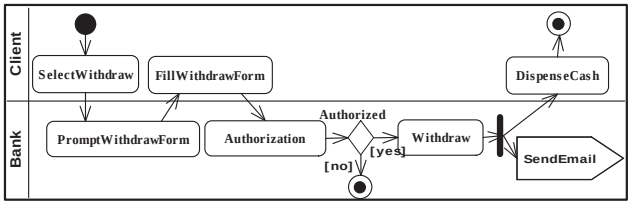


**Figure 6: The integrated model of withdraw**

nario also cut across other scenarios that modify account balance (e.g., deposit and transfer). The aspect models reflect the authorization and informing concerns of all balance changing operations in the banking system. So the aspect-oriented modeling that encapsulates crosscutting concerns can be easily reused in other applications.

**Algorithm 3.** Weaving an advice model

**INPUT**: primary model AD ($N_1$, $E_1$, $G_1$, $F_1$), advice model AM, pointcut model PM ($N_2$, $E_2$, $G_2$, $F_2$, AM, A, j), Set MSGs
*//MSGs is a set matched sub-graphs between AD and PM*
**OUTPUT**: WM ($N_3$, $E_3$, $G_3$, $F_3$)
**for each** sub-graph $MSG_i$ **in** MSGs {
  $AM_i$ ($N_i$, $E_i$, $G_i$, $F_i$, $P_i$, $entry_i$, $exit_i$) :=
    the initialized advice model of AM based on $MSG_i$
  Element $j_i$ := the element in $MSG_i$ which conforms to PM.j
  **if** $j_i$ is a node and AM is a "Before" advice{
    Edge e := Predecessor($j_i$)
    Node $n_1$ := Predecessor(e), $n_2$ := $j_i$
  } **else if** $j_i$ is a node and AM is an "After" advice {
    Edge e := Successor($j_i$),
    Node $n_1$ := $j_i$, n2 := Successor(e)
  } **else if** $j_i$ is an edge {
    Edge e := $j_i$
    Node $n_1$ := Predecessor($j_i$), Node $n_2$ := Successor($j_i$)
  }
  Edge se := Successor($AM_i$.entry), pe := Predecessor($AM_i$.exit)
  WM.$N_3$ := AD.$N_1$ $\bigcup$ $AM_i.N_i$, WM.$E_3$ := AD.$E_1$ $\bigcup$ $AM_i.E_i$
  WM.$G_3$ := AD.$G_1$ $\bigcup$ $AM_i.G_i$, WM.$F_3$ := AD.$F_1$ $\bigcup$ $AM_i.F_i$
  WM.$F_3$ := WM.$F_3$ - ($AM_i$.entry, se) + ($n_1$, se)
  WM.$F_3$ := WM.$F_3$ - (pe, $AM_i$.exit) + (pe, $n_2$)
  WM.$N_3$ := WM.$N_3$ - $AM_i$.entry - $AM_i$.exit
  WM.$E_3$ := WM.$E_3$ - e
} **return** WM

## 4. PROTOTYPE TOOL AND CASE STUDY

In order to perform the integration automatically, we have implemented a prototype tool named *Jasmine-AOI*[2] as an Eclipse plug-in. With the support of *Jasmine-AOI*, we have conducted two case studies: a banking system adapted from [10] and a point-of-sale terminal (POST) system adapted from CoCoME[3]. In the banking system, there are eight activity diagrams that describe the primary functional features. Five of the scenarios "login", "logout", "add account", "delete account", and "update account" are performed by clerks and the other three scenarios "withdraw", "transfer", and "deposit" are performed by clients. In the POST system, there are seven activity diagrams that describe the primary functional features. Scenarios "login" and "logout" are performed by cashiers, scenarios "sale" and "refund" by customers, scenarios "start up" and "shut down" by managers, and scenario "add new user" by system administrators.

Based on the functional scenarios we distilled out eight crosscutting concerns. Five of them are sequential aspects and three are parallel aspects. Table 3 shows the details of all crosscutting aspects. The primary, pointcut, and advice models are created in Enterprise Architect (EA)[4], and are imported to *Jasmine-AOI*, respectively. After assigning the precedence of the aspects, the tool initializes them based on the identified join points and weaves them into the primary models to generate integrated system models. We manually verified these integrated models and validated that they conform to both the primary requirements and the crosscutting concerns in Table 3. The models can then guide the subsequent aspect-oriented programming and testing activities. Through the case studies, we found that modeling crosscutting concerns separately can reduce the complexity of primary models. Furthermore, the eight aspect models are reused fifty eight times in the primary models of the two systems. This indicates the feasibility of reusing aspect models through our aspect-oriented modeling approach.

## 5. RELATED WORK

Aspect-oriented modeling can be done with different notations. Jacobson and NG [11] leveraged the use case diagram with ≪extend≫ relationship for modeling aspects as extension use cases for extension points defined in base use case. Xu et al. developed an aspect-oriented extension to petri-nets for modeling and verifying security concerns [12]. Zhang et al. [13] presented aspect-oriented state machines for modeling state-crosscutting concerns.

Several approaches have been proposed to integrate aspect-orientation with UML diagrams. France et al. presented directives for composing class diagrams based on its syntax [14, 8]. They also proposed a technique to compose sequence diagrams using tags on primary models [4]. Xu et al. provided an approach to weave state charts with aspect models for testable specification and test generation [9, 15]. Whittle et al. presented an aspect composition language SDMATA for state diagrams to support rich composition forms [16]. In order to solve the problem of multiple weaving of sequence diagrams, Jézéquel [17] defined the correspondence between a base model and a pointcut as three isomorphisms. In their approach, join points can be detected even some events occur between the events specified in the pointcut. Klein et al. [18] proposed a semantic based composing technique for sequence and interaction overall diagrams. Fuentes et al. [6] described how to construct and execute aspect-oriented activity diagram models and illustrate with an online book store example. Table 4 compares our approach to most related works from various perspectives.

In this paper, the target models are general design models instead of other types of models. While in [14], Reddy provided an aspect-oriented modeling method for the UML class diagram which is commonly used to design the statical structural of software. The semantic based composing technique in [18] is proposed for sequence charts, which are used to design interaction of software. In this paper, we focus on aspect-oriented modeling with UML activity diagram, which is a powerful tool to design behaviors of software at different levels of abstraction.

## 6. CONCLUSIONS AND FUTURE WORK

In this paper, we present an aspect-oriented approach for modeling and integrating crosscutting concerns as sequen-

[2]Jasmine-AOI, http://cs.nju.edu.cn/lzwang/Jasmine-AOI/
[3]rCOS CoCoME, http://www.iist.unu.edu/cocome/
[4]Enterprise Architect, http://www.sparxsystems.com.au/

**Table 3: Crosscutting concerns in the Bank and POST system**

| Concerns | Advice Type | Join Point Type | Description | Occurrence in Bank | Occurrence in POST |
|---|---|---|---|---|---|
| Authenticating | Before | Node | Validate legal user. | 7 | 6 |
| Authorizing | Before | Node | Guarantee critical operations are empowered. | 6 | 5 |
| Integrity | Before | Node | Avoid adding duplicate items to the database. | 2 | 1 |
| Auditing | After | Node | Record operations performed by employees. | 5 | 5 |
| Logging | After | Node | Log operations performed by clients. | 3 | 2 |
| Printing | After | Edge | Printing receipts after transactions finished. | 6 | 2 |
| Monitoring | After | Edge | Monitor operations performed by clients for finding potential abnormity. | 3 | 2 |
| Informing | After | Edge | Inform clients when subscribed events happened. | 3 | 0 |

**Table 4: Comparison with most related works**

| | Xu et al. [12] | Reddy et al. [14] | Xu et al. [9, 15] | Klein et al. [18] | Fuentes et al. [6] | Jasmine-AOI |
|---|---|---|---|---|---|---|
| Primary Model | Petri-net | Class Diagram | State Model | Sequence Chart | Activity Diagram | Activity Diagram |
| Pointcut Model | Textual | Textual | Textual | Sequence Chart | Sequence Chart | Activity Diagram |
| Join point Declaration | Explicit | Explicit | Explicit | Implicit | Implicit | Implicit |
| Advice Model | Petri-net | Class Diagram | State Model | Sequence Chart | Activity Diagram | Activity Diagram |
| Selection Method | Direct | Direct | Direct | Semantics | Structure | Syntax |
| Tool Support | - | Kompose | - | - | - | Jasmine-AOI |
| Target Model | Verification | Design | Testing,Verification | Design | Executable | Design |



tial and parallel aspect models based on UML activity diagrams. We add lightweight extensions to standard activity diagrams with stereotypes and tag values. An aspect model is designed as pairs of pointcut model and advice model with extended activity diagrams. Advice models are woven into primary models according to corresponding pointcut models. Two case studies have been conducted to demonstrate the feasibility of our approach. Concerning the future work, we will focus on verifying integrated models against system requirements and testing system implementation against the verified models. We also intend to build an aspect model repository of typical crosscutting concerns. The aspect models can then be reused in different applications.

## 7. ACKNOWLEDGEMENT

The authors at Nanjing University are supported by the National 863 High-Tech Programme of China (No.2007AA01 0302), the National Natural Science Foundation of China (No.60721002 and No.60603036), the Jiangsu Province Research Foundation (BK2007139), and Scientific Research Foundation of Graduate School of Nanjing University (No.2008CL07).

## 8. REFERENCES

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *ECOOP*, pp. 220–242, 1997.

[2] OMG, UML Superstructure v2.1:, http://www.omg.org/technology/documents/formal/uml.htm.

[3] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An Overview of AspectJ. In *ECOOP*, pp. 327–353, 2001.

[4] R. Reddy, A. Solberg, R. France, and S. Ghosh. Composing Sequence Model Using Tags. In *MoDELS workshop on Aspect-Oriented Modeling*, 2006.

[5] M. McNeile and E. Roubtsova. Csp Parallel Composition of Aspect Models. In *AOSD workshop on Aspect-Oriented Modeling*, pp. 13–18, 2008.

[6] L. Fuentes and P. Sánchez. Towards Executable Aspect-Oriented UML Models. In *AOSD workshop on Aspect-Oriented Modeling*, pp. 28–34, 2007.

[7] D. Stein, S. Hanenberg, and R. Unland. Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design. In *AOSD*, pp. 15–26, 2006.

[8] R. B. France, I. Ray, G. Georg, and S. Ghosh. Aspect-Oriented Approach to Early Design Modelling. *IEE Proceedings - Software*, 151(4):173–186, 2004.

[9] D. Xu and W. Xu. State-Based Incremental Testing of Aspect-Oriented Programs. In *AOSD*, pp. 180–189, 2006.

[10] R. C. Bjork. *ATM*. http://www.math-cs.gordon.edu/courses/cs211/ATMExample/.

[11] I. Jacobson and P.-W. Ng. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley Professional, 2004.

[12] D. Xu and K. E. Nygard. Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets. *IEEE Transaction on Software Engineering*, 32(4):265–278, April 2006.

[13] G. Zhang, M. M. Hölzl, and A. Knapp. Enhancing UML State Machines with Aspects. In *MoDELS*, pp. 529–543, 2007.

[14] Y. R. Reddy, S. Ghosh, R. B. France, G. Straw, J. M. Bieman, N. McEachen, E. Song, and G. Georg. Directives for Composing Aspect-Oriented Design Class Models. *T. Aspect-Oriented Software Development I*, pp. 75–105, 2006.

[15] D. Xu, I. Alsmadi, and W. Xu. Model Checking Aspect-Oriented Design Specification. In *COMPSAC*, pp. 491–500, 2007.

[16] J. Whittle, A. Moreira, J. Araújo, P. K. Jayaraman, A. M. Elkhodary, and R. Rabbi. An Expressive Aspect Composition Language for UML State Diagrams. In *MoDELS*, pp. 514–528, 2007.

[17] J.-M. Jézéquel. Model Driven Design and Aspect Weaving. *Journal of Software and Systems Modeling (SoSyM)*, 7(2):209–218, May 2008.

[18] J. Klein, L. Hélouët, and J.-M. Jézéquel. Semantic-Based Weaving of Scenarios. In *AOSD*, pp. 27–38, 2006.