International Journal of Hybrid Information Technology
Vol.7, No.4 (2014), pp.365-376
http://dx.doi.org/10.14257/ijhit.2014.7.4.31

An Aspect-oriented Software Architecture Description Language AO-ADL Based on XYZ

Yanting Cao[1] , Mei Rong[2*] and Guangquan Zhang[3,4]

[1]Suzhou Polytechnic Institute of Agriculture, Suzhou 215008, China
[2]Shenzhen Tourism College, Jinan University, Shenzhen, 518053, China
[3]School of Computer Science and Technology, Soochow University, Suzhou 215006, China
[4]State Key Laboratory of Computer Science, Chinese Academy of Science, Beijing 100190, China
**rongmei@sz.jnu.edu.cn*

Abstract

Aspect-Oriented Programming (AOP) can resolve the code tangling problem in Object-Oriented Programming (OOP) via using the technology of separation of concerns. Software architecture is becoming an important part in the phase of software design, it has the ability of helping designer to handle the structure and the complexity of large software systems, and Aspect-Oriented Software Development (AOSD) is a new paradigm proposed to manage the complexity by crosscutting concerns in the whole software life-cycle. In order to adequately specify aspect-oriented design, Aspect-Oriented Architecture Description Languages (AO-ADL) are needed. XYZ/ADL is an architecture description language which is based on temporal logic language XYZ/E. XYZ/ADL separates computation and communication into two different architecture elements – component and connector, but lacks some appropriate support to represent these crosscutting behaviors. So, XYZ/ADL must be extended to resolve the problem above by adding a kind of new elements – Aspect and modifying the former component and connector. At last, we illustrate them on an example of the Hotel Management System (HMS) via using AO- ADL.

Keywords: *Aspect-oriented programming; Software architecture; XYZ/ADL; Aspect-oriented architecture description language*

1. Introduction

OOP performs a functional decomposition but has the limitation that some concerns which cross several structural pieces can not be located in a single module. The concerns will appear continually in many modules of the software system, which may results in bad modules and the high cost of the maintenance and evolution of the codes. This is called scattering. From the perspective of reuse, a module will have a low possibility of being reused if it contains lots of concerns (a concern is a problem needed to be resolved by the software system). The situation that many concerns mix together is called code tangling.

AOP is proposed by Kiczales [1] as a way to apply separation of concerns in software development to deal with the code tangling problems of OOP mentioned above. AOP introduces a new notion of aspect, which encapsulates the crosscutting concerns. The core concept of AOP includes two main steps. One is the implementation of aspect in a loosely coupled way by describing different concerns of the system and the relationships among them. The other is to weave the aspects into an executable program which depends on the support mechanism of the AOP environment.

From the perspective of software development, it will be very helpful to generate codes from early designs if the final structure of implementation can be considered in each phase of the software development. Therefore, the separation of aspects of the system at the software architecture level can be done in order to achieve the traceability between architectural design and coding phase seamlessly. Recently, numerous studies, whose results show the beneficial effects of an architectural approach, have been presented (AOSD [2] sites).

Architecture description languages (ADLs) are a sound and convenient approach to represent software architecture. Traditional ADLs do not normally provide appropriate support for separating any kinds of crosscutting concerns that frequently result in poor architectures descriptions with AO. In order to adequately specify the aspect-oriented design, AO-ADL is introduced. In this paper, we present the AO-ADL, which is an evolution of our pervious works XYZ/ADL [3] via adding some new elements and mechanisms. XYZ/ADL is an architecture description language which is based on temporal logic language XYZ/E [4].

The remainder of this paper is organized as follows: Section 2 XYZ/ADL is illustrated in details. AO-ADL based on XYZ/ADL is introduced in Section 3; In Section 4, we illustrate AO-ADL on a simple HMS example; Section 5 presents the related work on AO-ADL and Section 6 ends this paper with conclusions and future works.

2. XYZ/E and XYZ/ADL

XYZ [5] system is a CASE environment based on temporal logic and conforming to various ways of programming such as programming with HLL, hierarchical specification or production rules, sequential or concurrent, textual or graphical, *etc*. All these programming paradigms can be unified with a uniform framework of a temporal logic language, XYZ/E [4] is an executable temporal logic language based on Manna-Pnueli’s Linear Time Temporal Logic, which combines both static and dynamic semantics in a unified framework and supports the whole procedure of stepwise refinement, *i.e.*, from the abstract specifications to executable codes.

XYZ/ADL [3] is a software architecture description language which Based on the executable temporal logical language XYZ/E. XYZ/ADL can not only represent the system description at different abstract levels from formal specification to executable program which under a unified logical framework, but also can represent both dynamic semantics and static semantics of software architecture. XYZ/ADL, suitable for the formal description and helpful for the refinement to the software architecture at different abstract level, can verify the semantic consistency of the process of refinement with tools in the XYZ system. The following are architectural units of XYZ/ADL.

(a)Simple component

Simple component contains interface and computation. Interface contains port description and functional specification. The former shows the interactive behaviors between components and external environment, and the later shows the function of components. A port, through which a request can be sent from component to external environment, is represented by channel. The syntax of port is listed as follow:

%PORT PortName = = ChannelType Declaration; [PortBchavior]

Explaining what a component can do, functional specification is a temporal logic formula, whose syntax is in the following:

%FUNCTION = = [Function Specification]

By attaching each port into a whole, computation specification is a complete behavior description of component, whose syntax is listed as follows:
%COMPUTATION= = [Computation Specification]

(b) Connector

Connector describes the common character of some interactive behavior, whose instance can be used to connect different components which can satisfy the requirement. A connector consists of two parts: interface and interactive protocol. Defined by a group of roles, interface indicates all the external behaviors of components joining the interaction. Interactive protocol describes how to join roles together and how to compose computation specification of components joining the interaction. Role is represented by channel, whose syntax is in the following:

%ROLE RoleName==DataType Declaration; [RoleBehavior]
Interactive protocol is represented by a unit of XYZ/E, whose syntax as follows:
%GLUE= = [Interact Protoco1]

(c) Compound component

The description of compound component's port is the same with simple component, while its behaviors are represented by connection of several components. The composing declaration of compound components presents that which components and connector instance it contains, whose syntax as follows:
%COMPOSITION = = [
ComInsName: ComponentName;
…
ConInsName: ConnectorName;]
The attachment definition of compound components indicates inner configuration of architecture. The syntax is as follows:
%ATTACHMENTS = = [
ComInsName.PortName # ConInsName.RoleName;

ComInsName.PortName ## PortName;]

3. An AO-ADL Based on XYZ/ADL

For the failure of the current ADLs to separate and modularize the crosscutting function, in this paper, the concept of aspect in AOP will be abstracted and introduced into the software architecture. Introducing the new concept aspect into XYZ/ADL will be helpful to modularize the crosscutting function at the software architecture level. Meanwhile, the connector in ADL must be modified for interacting with aspect and other components.

3.1. Aspect

A crosscutting concern, which is represented by the service aspect provided can influence many related components. To describe the crosscutting concern of ADL, we introduce an new concept Aspect which consists of pointcut and advice. Being used to capture and recognize the connecting point, pointcut can decide which components the aspect will be crosscut, while advice shows the crosscutting function the aspect has.

With "and", "or" and "not"("&&", "||" and "!"), represented as "∧ ", "$V" and "~", pointcut is composed of several joinpoints. Joinpoint is a clearly defined position where components are crosscutted by aspect when the program is executed. The syntax of pointcut is as follows:

%POINTCUT PointcutName = Joinpoint Declaration; [∧ ; $V; ~]

Advice is a complete crosscutting behavior, and it is a specific description of crosscutting function existed in the software architecture. It was described as follows:

%ADVICE AdviceName = [Advice Specification]

The situations when crosscutting behaviors of aspect crosscut into all the connected components are: before the joinpoint, around the joinpoint and after the joinpoint. The following are the semantic explanations of aspects and the relationship between them.

Definition 1 An AO-ADL description of aspect consists of pointcut and advice. Pointcut contains a group of joinpoints, while advice is the complete crosscutting functional description of aspect. The specification of aspect is a XYZ/E unit:

(POINTCUT1||POINTCUT2||…||POINTCUTn)|| (ADVICE1|| ADVICE2||…|| ADVICEn)

POINTCUT is the set of {Joinpoint1, Joinpoint2, … , Joinpointn}; The time sequence of Aspect crosscutting into components is represented by temporal operator.

3.2. Aspectual Connector

Aspect encapsulated the crosscutting concerns, but, it needs to crosscut to the appropriate components. Therefore, the connector in XYZ/ADL should be modified in order to build connection among components and between aspect and components which have crosscutting relationship with the aspect. The connector consists of interface and interactive protocol. Interface is defined by a group of roles which contain: (1) BaseRole, which connects the ports of components, (2) AspectRole, which connects Aspect. Interactive protocol also needs to be modified The sentence with the word "glue" represents the interactive protocol not only among components but also between aspect and component. The corresponding syntax is as follows:

%ASPECTUALCONNECTOR AspectualConnectorName= [
%BASEROLE BaseRoleName=Data Type Declaration; [BaseRole Behavior]
%ASPECTROLE AspectRoleName = Data Type Declaration; [AspectRoleBehavior]
%GLUE= = [Interact Protocol]]

The specification semantic explanation of Aspectual Connector is listed as follow.

Definition 2 Aspectual Connector is defined by a group of BaseRole BR1BR2…BRn, AspectRole AR1AR2…ARm and the definition of interactive protocol GLUE, in which, the behaviors of each BaseRole BRi are described as BRBehaviori(i=1,2,…,n), while the behaviors of each AspectRole ARi are described as ARBehaviori (i=0,1,…,m). The specification of AC is a unit of XYZ/E as follows:

GLUE || (BRBehavior1|| BRBehavior2|| … ||BRBehaviorn)||(ARBehavior0|| ARBehavior1||…||ARBehaviorm)

The channel set appeared in the GLUE is {R1, R2,…, Rn}.

3.3. Component

Because some components have a crosscutting relationship with aspects, a special port (Joinpoint) should be defined in order to represent the position into where aspect crosscut. Components are divided into two kinds: one requests services, and the other provides services. The components which have crosscutting relationships with aspects request the functions of aspect, and were considered as requesting services component. The joinpoint of component is represented by a group of channels, the syntax is listed as follows:

%JOINPOINT JoinpointName= =FunctionName Declaration;

The normative semantic explanation of component is listed as follows:

Definition 3 A component DC consists of two parts: interface description and computing specification, Interface description contains a group of ports, a qualitative description and a group of Joinpoints. The jointpoint showing the position where aspect crosscut into component, the number of jointpoint is zero indicates that this component has no crosscutting relationship with other aspects. Computing specification is the full abstract behavioral description of a component. If a DC has k ports, with behavioral description being PBehavior1, Pbehavior2, …, PBehaviork, functional section being f and computing specification being ComSpec, then the relationship among them is that PBehaviori (i=1,2,…,k) is the refinement of variable set which does not contain control variable appeared in each port of ComSpec, and that ComSpec logically entails f, namely ComSpec⇒f.

3.4. The Composition between Aspect and Component

The composition between aspect and component means that the crosscutting function of aspect will take effect in a certain position of a component with the help of Aspectual Connector. The compound component makes the function of aspect embedded into the corresponding component. The compound component has new entire functions by compositing original functions and crosscutting functions of aspect; its ports do not change but the number of joinpoints reduces by one. A compound component can be treated as a new simple component. The syntax of compound component which consists of component and aspect is listed as follows:

```
%COMPOSITION= = [
ComponentInstanceName: ComponentName;
…
        AspectInstanceName: AspectName;
…
        AspectualConnectorInstanceName:
AspectualConnectorName;
…]
```

4. Modeling and Analysis of HMS

A HMS is mainly aimed at business data processing and provides a platform for managing all the businesses of the hotel. The main function of HSM consists of Booking rooms, Registering rooms and Settling accounts.

(a) Booking rooms

If there are rooms available for meeting the customer's demands, we create the reservation for customer. Otherwise, if customer wants to wait, we record the room type and add the customer to the waiting queue of booking rooms. If the customer does not agree to wait, then ending the booking schedule.

(b) Registering rooms

After recording the detail information of customer, then we allocate a room and create a blank accountant bill for customer. If the customer has booked the room, then the booking record must be deleted.

(c) Settling accounts

Firstly, we settle a bill for the customer. If there are other customers waiting for the rooms, assign the room to the first customer of the waiting queue. If there are no waiting customers, add the room into idle queue.

In order to model the Aspect-Oriented architecture of HMS, crosscutting concerns of system should be extracted from the main function module. In this paper, we extract the crosscutting concerns as the Table 1 shown.

Table 1. The Crosscutting Concerns of HMS

Crosscutting concerns	The description of crosscutting concerns
Verification of Identity	Before the input of the customer's waiting record into the waiting queue, we should verify the customer's identity.
Time crosscutting	Crosscutting time after successful deal with the information of booking rooms, waiting rooms, successful waiting room, customer check in and customer check out, which record the time message of customer's activities
Database anomalies	Database processing without taking into account the issue of handling exception. Database anomalies are added to the database when abnormal operation appeared.

The core concerns of HMS are as shown as Table 2 below:

Table 2. The Core Concerns of HMS

Core concerns	Function description
Manage room information	Manage the serial number, price, type, interior equipment information of room, etc. Provide the information of room for customers and managers.
Apply for booking	Receive the booking application from customers; customers choose room type and apply it when the room available. Otherwise, waiting for booking.
Save the booking message	Deal with the successful booking information for customers
Deal with booking waiting	Give the available room to the first customer in the waiting queue.
Check in	Deal with the information of customer. Delete the booking record if the customer has booked the room, build blank bill for the customer.
Check out bill	Deal with the check out bill procedure for customer.
Manage customer information	Mange the customer information such as: the basic information of customer, the room number, reserve time, check in time, check out time, accommodation status and the consume bill, and so on.
Manage room status	There are three status of a room: idle, booked and reside. Deal with the status of the room, inquire and set the status of the rooms.
Manage database	Query, insert, delete and maintain database.

We mapped the core concerns of the HMS to the components of architecture, the crosscutting concerns to Aspects via separating concerns of the HMS. The behaviors of port and role consist of inputting data and outputting data in this system. So we define two types of port in this paper.

%PORTTYPE IN (DT, vn) = = DT;
□[LB=Start⇒\$OIN?vn∧\$OLB = L1;

```
LB=L1∧~(vn = EOF)⇒$OIN?vn∧$OLB = L1;
LB=L1∧(vn = EOF)⇒$OLB = STOP]
/* the description of IN port */
%PORTTYPE  OUT (DT, vn) = = DT;
□[LB = Start⇒$OLB=L1;
LB = L1∧~(vn=EOF)⇒$OOUT!vn∧$OLB = L1;
LB = L1∧(vn=EOF)⇒$OOUT!EOF∧$OLB = STOP]
/* the description of OUT port */
```

DT is the parameter of the data type, vn is the parameter of variable name. EOF is close signal.

4.1. Aspects of HMS

There are some Aspects in HMS: TAspect deals with time crosscutting, VAspect hands verification of customer's identity. DBAspect deals with database anomalies operation. Because of the space limited, we just take TAspect as an example. The description of the TAspect with AO-ADL is as follows:

TAspect: it crosscut into Reservation, Waiting, Check in, Check out component, the components execute Advice after the corresponding Joinpoint executed. So we use After Advice. The description of TAspect is blow:

```
%ASPECT TAspect= =[
%POINTCUT getTime= =JpName;
□[LB1 = START ⇒ $OLB1 = DetectJp;
LB1=DetectJp ⇒ $OJpName = CkJp∧$OLB1= DetectJp;
LB1 = DetectJp ⇒ $OJpName = RoomWtJp3∧$OLB1
= DetectJp;
LB1=DetectJp⇒$OJpName=RoomRvJp2∧$OLB1=DetectJp;
LB1 =DetectJp⇒$OJpName=RoomWtJp1∧$OLB1
=DetectJp;
LB1=DetectJp ⇒$OJpName=CoJp∧$OLB1= DetectJp;
LB1= DetectJp ⇒$OJpName=EOF∧$OLB1= End;
LB1=End⇒$OLB=STOP]
/*Aspect executed by component when the component execute the Joinpoint */
%ADVICE getTime= =After;
□[LB1=START⇒$OLB1=StartAdvice;
LB1= StartAdvice ⇒!![
JpName =CkJp |> Checkin. setCkininfo()∧$OLB1=End,
JpName = RoomWtJp3|>Waiting.setWaitingsucc()∧$OLB1
= Get_Time,
JpName=RoomRvJp2|>Reservation.setReservesucc()∧
$OLB1= Get_Time,
JpName= RoomWtJp1|> Waiting.setCustomerer()∧$OLB1= Get_Time,
JpName=CoJp|>Checkout.setCkoutinfo()∧$OLB1
=Get_Time];
LB1= Get_Time ∧setTime()⇒$OLB1 = End;
LB1= End⇒$OLB1=STOP]]
```

4.2. Connectors of HMS

Aspectual Connector is used to describe the common features of certain interactive behaviors in AO-ADL. Instants of it can be used to connect the different components and Aspects. There are two main types of Aspectual Connector in HMS.

(a) **C_CCon** Aspectual Connector, which connects components with other components, the description is listed as follows:

```
%ASPECTUAL CONNECTOR C_CCon = = [
%BASEROLE Source = = OUT(DT, vn);
%BASEROLE Sink = = IN(DT, vn);
%GLUE = = []]
```

(b) C_ACon Aspectual Connector, which connects components with Aspects, the description is listed as follows:

```
%ASPECTUAL CONNECTOR C_ACon = =[
%ASPECTROLE ASource = = OUT(DT, vn);
%BASEROLE BSink = = IN(DT, vn);
%GLUE= =[]]
```

4.3. Components of HMS

We mapped the core concerns into the components of the HMS, which consists of **Room**, **RoomReserve**, **Check in**, **Check out, State**, **Customers**, **DBoper**. Due to the space limited, we just take **RoomReserve** as an example. **RoomReserve** is a compound component, which consists of three child components: **Reserve** makes a request for room booking, **Reservation** saves the booking information, **Waiting** waits for booking.

The compound component **RoomReserve** described with AO-ADL is listed as follows:

```
%COMPONENT RoomReserve  = = [
%PORT RoomIn1 = = IN( ROOM, rooms );
%PORT RoomIn2 = = IN( ROOM, allStates );
%PORT RoomIn3 = = IN ( ROOM, room );
%PORT RoomOut1 = = OUT( CUSTOMERER, baseInfo );
%PORT RoomOut2 = = OUT( ROOM, roomRsv );
%PORT RoomOut3 = = OUT( CUSTOMERER, reserveSucc );
%PORT RoomOut4 = = OUT( ROOM, roomWt );
%PORT RoomOut5 = = OUT( CUSTOMERER, waitingSucc );
%JOINPOINT RoomRvJp1 = = setReservesucc();
%JOINPOINT RoomWtJp1 = = setCustomerer() ;
%JOINPOINT RoomWtJp3 = = setWaitingsucc();
%COMPOSITION = = [
         Reserve: reserve;
        Reservation: reservation;
        Waiting: waiting;
        VAspect: vAspect;
%ATTACHMENTS = [
reserve.RsOut1#AC21.Source;
reservation.RvIn#AC21.Sink;
reserve.RsOut1#AC17.Source;
waiting.WtIn1#AC17.Sink;
reservation.RvJp1#AC19.BSink;
vAspect.Pointcut#AC19.Asource;
waiting.RvJp1#AC20.BSink;
vAspect.Pointcut#AC20.Asource;
reserve.RsIn1## RoomIn1;
reserve.RsIn2## RoomIn2;
waiting.WtIn2## RoomIn3;
reserve.RsOut3## RoomOut1;
reservation.RvOut1## RoomOut2;
reservation.RvOut2## RoomOut3;
waiting.WtOut1## RoomOut4;
waiting.WtOut2## RoomOut5;
reservation.RvJp1## RoomRvJp1;
waiting.WtJp1## RoomWtJp1;
waiting.WtJp3## RoomWtJp3]]
```

5. Related Works

Recently, there are many aspect-oriented architecture description languages proposed by researchers. Some of them are extensions of traditional ADLs, but others are new languages. All the proposals introduce aspects into ADLs in different ways: component views, connector views, or aspect views, either a symmetric or an asymmetric approach.

PRISMA ADL [6] is a highly evolved language created to describe systems developed in the framework of the PRISMA architectural model [7], in which aspects are a new abstraction used to define the internal structure of both components and connectors. It takes the symmetric approach to provide PRISMA with a very natural way of dealing with crosscutting.

FuseJ [8] is an asymmetric approach that combines components and aspects and includes the concept of XML-based configurations to specify the weaving information. AO-Rapide [9] is an extension of Rapide ADL, taking an asymmetric approach with aspects as components. An XML-based aspect-oriented architecture description language [10] is taking an asymmetric approach, which defines the component as the architectural block to model both functionality and aspectual behavior and extends semantics of connectors to specify aspectual composition information. Aspect LEDA [11] is taking an asymmetric approach with aspects being components, which based on the formal ADL - LEDA .It allows the architecture obtained to be evaluated and checked.

Aspectual Acme [12] is an AO-ADL extension of the Acme language in which the concept of aspectual connector is defined to connect aspects with components, instead of using components to play the role of aspects.

In this paper, we introduce a symmetric AO-ADL, which based on XYZ/ADL. We consider Aspect to be a first-class entity which differs from Component and Connector, and separate the crosscutting concerns which are modularized of system from software architecture in order to avoid scattering and tangling of codes.

6. Conclusions and Future Works

This paper promotes the concept of AOP at coding phase up to software architecture by adding Aspect into XYZ/ADL and adding aspect role in connector for dealing with the interaction between aspect and component, and proposes the related composition mechanism so as to form the Aspect-Oriented Architecture Description Language (AO-ADL). When we model the software architecture, the crosscutting concerns can be separated, which is helpful for the smooth transition from software architecture to code and reinforcing the modular degree of software system. Our future work is to verify the correctness of the architecture described by extended XYZ/ADL via tools provided in XYZ systems.

Acknowledgements

This research was sponsored by the National Natural Science Foundation of China (61100074), the Natural Science Foundation of Jiangsu Province (BK2011281), Applied Basic Research Program of Suzhou (SYG201241).

References

[1] G. Kiczales, "Aspect-Oriented Programming", ACM Computing, Surveys, vol. 28, (4es), **(1996)**, pp. 154.
[2] J. Fabry, A. Kellens, S. Denier and S. Ducasse, "Aspect Maps: Extending Moose to Visualize AOP Software", Science of Computer Programming, vol. 79, **(2014)**, pp. 6-22.
[3] G. Q. Zhang, H. Shi, M. Rong and H. J. Di, " Model Checking for Asynchronous Web Service Composition Based on XYZ/ADL", Lecture Notes in Computer Science, LNCS, vol. 6988, **(2011)**,pp. 428-435.
[4] M. Rong, C. Liu and G.Q. Zhang, "Modeling Aspect-Oriented Software Architecture Based on ACME", ICCSE, **(2011)**, pp. 1159-1164.
[5] Z. S. Tang, "The Design Philosophy of XYZ System", Journal of Software, vol. 1, no. 1, **(1990)**, pp. 47-55.
[6] J. Perez, I. Ramos, J. Jaen, P. Letelier and E. Navarro, "PRISMA: Towards Quality, Aspect-Oriented and Dynamics of Software Architectures", Proceedings of 3rd IEEE International Conference on Quality Software (QSIC03), Dallas, USA, **(2003)** November 6-7.
[7] J. Perez, E. Navarro, P. Letelier and I. Ramos, "A Modeling Proposal for Aspect-Oriented Software Architectures", Proceedings of 13th IEEE Conference on the ECBS, IEEE Computer Society Press, Postdam, Germany, **(2006)** March.
[8] D. Suvee, B. DeFraine and W. Vanderperren, "A Symmetric and Unified Approach Towards Combining Aspect-Oriented and Component-Based Software Development", CBSE2006, LNCS, Springer, Heidelberg, vol. 4063, **(2006)**, pp. 114-122.
[9] K. Palma, Y. Eterovic and J. M. Murillo, "Extending the Rapid ADL to Specify Aspect-Oriented Software Architectures", TR, University of Extremadura, Spain, **(2005)**.
[10] M. Pinto, and L. Fuentes, "AO-ADL: An ADL for Describing Aspect-Oriented Architectures", Early Aspects 2007 Workshop, LNCS4765, Springer-Verlag, Berlin Heidelberg, **(2007)**, pp. 94-114.
[11] A. Navasa, M. A. Prez-Toledano and M. Murillo, "An ADL Dealing with Aspects at Software Architecture Stage", Information and Software Technology, vol. 51, **(2009)**, pp. 306-324.
[12] T. atista, C. havez, A. arcia, U. ulesza, C. SantAnna and C. Lucena, "Aspectual Connectors: Supporting the Seamless Integration of Aspects and ADLs", Proceedings of the XX Brazilian Symposium on Software Engineering (ABES'06), **(2006)** October.

Authors

Yanting Cao, she obtained her MS in Computer Software Engineering from the School of Computer Science and Technology, Soochow University in 2007.At the same time, she is serving as a full time faculty in the School of Computer & Software, Suzhou Polytechnic Institute of Agriculture. Her research interest includes software engineering, high performance computing.

Mei Rong (corresponding author), she received the PhD degrees in Computer Science from Chongqing University, in 1998, respectively. She is currently an associate professor in the Shenzhen Tourism College, Jinan University, China, and is the member of CCF. Her research interests include software engineering, formal methods, cloud computing and Cyber Physical Systems.