Behavior Consistency Verification for Evolution of Aspectual Component-based Software

Xue-yao Zhou [a], Ning-jiang Chen[b] and Dan-dan Hu[c]

College of Computer, Electronic,and Information, Guangxi University, Nanning, P. R. China

[a]zhouxueyao@126.com , [b] chnj@gxu.edu.cn , [c]dandan_hu@yeah.net

Keywords: Aspect-oriented software architecture; Dynamic evolution; Behavior consistency

Abstract. Aspect-Oriented Software Architecture (AOA) is a high-level abstraction and integration blueprint of aspectual component-based software. A semantic model of aspectual component-based software is proposed to provide behavior description and semantic foundation for the consistency verification of software architecture dynamic evolution. By using the semantic model of Pi-calculus, a set of the consistency verification methods of dynamic evolution from multiple aspects are introduced. Finally, a case study shows the effect of these methods.

Introduction

In the open and dynamic environment of Internet, the modification of distributed architectural objects is directly reflected in the evolution of system. The software dynamic evolution helps to improve system's adaptability and flexibility, to extend the lifetime of system and to improve the expansibility of system [1]. Aspect-oriented method is unifying the decentralized public code crosscutting in other function modules to form aspect to achieve the absolute separation of system concerns .At the meantime, using weaving mechanism can weave aspects into the component system according to the need, and form aspectual component-based system [2]. The dynamic evolution of aspectual component-based system is a complicated process. Interface behavior incompatibility, the change of unobservable internal behavior and replacing, adding or deleting component may result in system's function behaviors straying from the original system. For aspectual component, the special consideration is whether the semantic of pointcuts is changed and resulting in the change of system behavior. In order to solve the above problems in the case of non-stop and confirm whether the evolution is reasonable and correct, we verify the behavior consistency of aspectual component dynamic evolution using Pi-calculus and Delta analysis.

Related Work

To guarantee system consistency during component dynamic evolution has important significance for the system running correctly. There are already a number of researches on dynamic evolution of component-based system and the protocol of component behavior. The reflection in [3] can provide support for the dynamic evolution of component type. Naming service, reflection and dynamic adaptation mechanism of middleware also allow support for the dynamic replacement and upgrade of running component. Both of PKUAS [4] and Artemis-ARC [5] system use RSA for software maintenance and evolution. PKUAS describes RSA by extending architecture language ABC/ADL. Artemis-ARC uses RSA as specific operational objects built-in software to decouple component and reinterpret the interaction between components. But the built-in RSA increases the development complexity and lacks formal foundation of component behavior and interaction. In aspect of component behavior protocol, [6] proposes formal description and compatibility verification for the component behavior of complex real-time system, which effectively improves system reliability, but does not consider the component evolution. [7] summarizes the consistency validation of component evolution and behavior protocol and puts forward an evolution behavior consistency validation algorithm. But due to the lack of analysis on aspect as well as aspectual component evolution, it can't meet the requirements of verifying consistency of evolution in aspectual software. So the above

Published by Atlantis Press, Paris, France.
© the authors, 2013

consistency verification methods of base component can't be simply applied to aspectual component. Compared with the existing work, we specially need to consider the aspectual component evolution (the evolution of pointcut or advice,etc). This paper considers of analyzing AOA and describing architecture behavior through Pi-calculus in order to validate multiple behavior consistency, especially focus on the aspectual component dynamic evolution.

Work Foundation

Pi-calculus is a computational model for the formal description and analysis of concurrent systems. Fig 1 shows the graphical representation of Pi-calculus. The process *P* sends the information along the channel through port *b*, and process *Q* receives the information along the channel.

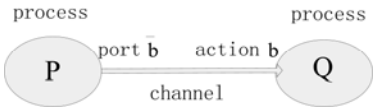


Fig 1. Representation of Pi-Calculus.

Modeling AOA With Pi-calculus. AOA includes three elements: base_component, aspect_component and configuration. The following is to model these three elements and finally to give the semantic model of the whole architecture. Firstly, the structures included in the three elements are explained in Pi-calculus—Require *r* (Eq. 1), Provide *p* (Eq. 2), JoinPoint *jp* (Eq. 3), *bind* (Eq.4) and *advice* (Eq. 5).

REQ(r,l)=($r(y).\overline{y}\,l$+0). r is the channel name, y is the name of the service provider, and l is the requested location of the service. (1)

PROV(p,s)=!($p(x).\overline{x}\,s$+0). p is the channel name, s is the service reference, and x is the requested location of s. (2)

JP(b,jp)=($b(bcp).\overline{bcp}\,jp$+0). b is the channel name, bcp is the name of joinpoint provider, and jp is the transmitted joinpoint. (3)

BIND(r,p)=$\overline{r}\,p$. declare the binding between components, r is the request service port and p is the provide service port. (4)

ADVICE(a,pcd,k,$advice$)=(a(jp).[jp=pcd](k:$advice$)). a is the channel name, pcd is the pointcut, k indicates the advice type keyword (before,after and around), advice is the executed program after capturing joinpoint and jp is the joinpoint. (5)

Secondly, the explanations of base_component (Eq.6), aspect_component (Eq.7) and configuration (Eq. 8, Eq. 9) constituted by these structures are listed as follows.

Base_Component=(v s,l,jp)(!(PROV(p,s))|!(REQ(r,l))|!(JP(b,jp))). (6)

Aspect_Component=(v pcd,k,$advice$)(!ADVICE(a,pcd,k,$advice$)). (7)

Configuration=(!BIND(r,p))|(!(Weavingrule)). (8)

Weavingrule=(*Aspect_Component*,*con*,*pcd*,*advice*). (9)

con is the constraint(the operation of dynamic evolution of architecture), the weaving semantic is when the constrain *con* is met, system will call the aspectual component interface before (after,around) calling the base component interface which is defined in pointcut *pcd*.

Finally, AOA semantic model is expressed in Eq. 10.

AOA=(!Base_Component)|(! Aspect_Component)| Configuration (10)

Consistency Verification of AOA Dynamic Evolution

Behavior consistency is behavior equivalent or the evolved system being able to finish all behaviors before evolution. Behavior consistency mentioned in this paper refers to the former. The running conditions and aspectual components of aspect-oriented system may change, and these changes may affect the behavior of system and lead to system error. The difficulties involve the range of evolution and whether the behavior state is consistent after evolution. So we must decide whether the dynamic evolution will produce inconsistent behavior in the case of non-stop. And the consistency verification of aspectual component evolution, which is the pointcut robustness test of

© the authors, 2013

system, is the most important. This kind of evolution involves the addition, deletion and change of pointcut, advice or even aspectual component. They may lead to the loss of defined join point, or the unintended capture of connection points. It can cause the base component being unable to be crosscut by aspect function, or the aspect function happens in a wrong joinpoint, which is unintended system behavior change.

The behavior of component consists of component state transition sequence, the transmitting message, interfaces between components and the operations. It can be represented as a 4-tuple: B_C=(S_C, P_C, *IP C*, *IR C*) , where *S* represents a state collection of component *C*, including the initial and final states, that is S^C={*SC init*, ,,,, *SC fina*}; P^C represents a path collection of state transition, i.e. $P^C = \{t_1, t_2, ,,,, t_n\}$, and transition path t_i can be represented by a 4-tuple $T=\{d,e,a,m\}$, $a \in (IP\ C \cup IR\ C)$, *d* means the start state, *e* means the end state, *a* means the executed operation or the function of the provider and requester interface, *m* means the transmitting message. A transition path from the initial state to the end state is $p = \{t_1, t_2, ,,,, t_m\}$, which represents the full behavior path of the component. The collection of the whole full path is represented by $P_w(C) = \{p_1, p_2,,,, p_n\}$.

Definition 1: Dynamic robustness of pointcut. It indicates the behavior interaction process between aspectual component and base component, that is, the base component meet the regulatory requirements of the aspectual component's interface and its requirements can be provided by aspectual component.

Assuming the base component *C* and aspectual component *AC* interact through interface I_C^R and I_{AC}^P, if and only if (*SC init*,*SAC init*) $\xrightarrow{P_w}$ (*SC fina*,*SAC fina*), the pointcut of *AC* is dynamic robust. (*SC init*,*SAC init*) represents the initial states of the two components, and P_w represents any existed behavior full path.

According to the above definition and the model in Section 3, If *C* evolves to C_{DE} or *AC* evolves to AC_{DE}, the semantics of interaction between C_{DE} and *AC*, expressed as: SE=(*v* $I_{C_{DE}}^R$,*IP AC*)((*v jp*)JP($I_{C_{DE}}^R$,*jp*)|(*v pcd*,*k*,*advice*)ADVICE(*IP AC*,*pcd*,*k*,*advice*)).And it can convert into Pi-calculus expressions: P(C_{DE},*AC*)=C_{DE}.$\overline{a_{DE}}$ <*jp*>|*AC*.a_{DE}(m_1)|[m_1=*pcd*].*AC*.$\overline{a_{AC}}$ <m_2>, a_{DE}, a_{AC} represent the component functions and belong to $I_{C_{DE}}^R \cup I_{AC}^P$, m_1 and m_2 are the transmitting message. Then according to the derivation expressions of Pi-calculus and the input or output in the full path, it is to be checked whether the Pi-calculus expressions eventually arrive to empty process 0. That means whether the both components move to the end state. If it arrives at the empty process 0, the pointcut is dynamic robust, otherwise it is fragile.

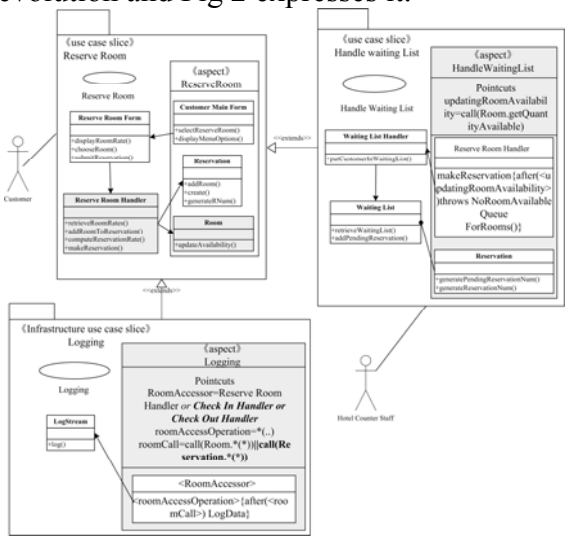
Definition 2: Static robustness of pointcut. It indicates whether the *(jp,advice)* in system are changed after evolution, which is calculating the Delta value of AOA. Assuming AOA *A* evolves to *A′* ,the Delta value of AOA is defined as: Δ pcd(*A*, *A′*)={(add(*A*, *A′*)) $\cup$ (delete(*A*, *A′*))}, add(*A*, *A′*)= $\bigcup_{j \in JP}$ (advice(*j*, *A′*)−advice(*j*,*A*)) , delete(*A*, *A′*)= $\bigcup_{j \in JP}$ (advice(*j*, *A*)−advice(*j*, *A′*)). *JP* are all joinpoints in *A* and *A′*,and JP=joinp(*A*) $\cup$ joinp(*A′*). Meanwhile, advice(*j*,*A*) expresses all matching of joinpoint and advice in *A*, and if *j* is not in joinp(*A*), the advice(*j*,*A*) is a empty set. If and only if Δ pcd(*A*, *A′*) is empty, and the evolution doesn't have an impact on the semantic of pointcut, the pointcut is static robust, otherwise it is fragile.

Case Studies

Suppose a hotel management system running on the Internet including all elements of the aspectual component-based system. It has three core functions: *Reserve Room*, *Check In* and *Check Out*. And it has a crosscutting function for logging. Also, the system needs to deal with the situation that there is no room can be reserved by using *HWaiting List*. AOSD(Aspect-Orient Software Development) methodology based on use cases is used to analyze and design the hotel management system in order to achieve the complete separation of concerns, and then get an analysis model describing base

© the authors, 2013

component and aspectual component—the use case slice model. This model makes preparations for the later evolution and Fig 2 expresses it.



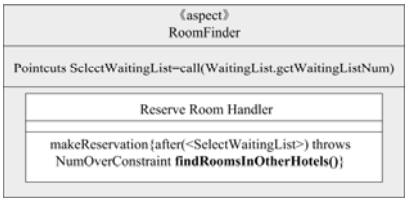


Fig 2. The use case slice model of Hotel management system. Fig 3. The new aspectual component RoomFinder.

In order to adapt to the changes of user's requirements and the context, the system begins to evolve after running for some time. When the context-aware is the update of base components *CheckInHandler* and *CheckOutHandler*,the aspectual component *Logging* will produce dynamic evolution. The bold italic in Fig 2 means deletion and the boldfaced word means addition. These changes of pointcut in aspect *Logging* are defined as evolution mode 1. When the context perceives no spare room and the number of waiting list is greater than the constraint in evolution rules, the system will perform the operations of deleting aspect *HandleWaiting* and adding a new aspect *RoomFinder* named evolution mode 2. The new aspectual component is shown in Fig 3. Towards the behavior Inconsistency caused by evolution mode 2, Pi-calculus is used to verify the dynamic robustness of pointcut. And to evolution mode 1, we can use Delta analysis [9] to verify the static robustness of pointcut.

Behavior Consistency Analysis of Dynamic Evolution. According to the AOA of hotel management system and the evolution requirements, we use the verification process of behavior consistency to verify whether the system is able to maintain behavior consistency and avoid errors occurring when the above evolutions are triggered. On the basis of the method, the specific behavior expressions of the component *ReserveRoomHandler* (*RRH*) and aspectual component *RoomFinder* (*RF*) are shown in Table 1.

Table 1. Behavior expressions and Full pathes of *RRH* and *RF*.

Component	*RRH*	*RF*
Behavior Expression	$B_{RRH}=(S^{RRH},P^{RRH},I^{P}_{RRH},I^{R}_{RRH})$, $S^{RRH}=\{A_{init},B,C_{fina}\}$ $P^{RRH}=\{t_1,t_2\}$, $I^{P}_{RRH}=\{retrieveRoomRates,makeR\}$ $I^{R}_{RRH}=\{getQuantityAvailable\}$ $t_1=(A_{init},B,makeR,\overline{Null})$ $t_2=(B,C_{fina},getQuantityAvailable,\overline{NoRoomException})$	$B_{RF}=(S^{RF},P^{RF},I^{P}_{RR},I^{R}_{RF}), S^{RF}=\{A^{'}_{init},B^{'},C^{'},D^{'}_{fina}\}$ $P^{RF}=\{t^{'}_1,t^{'}_2,t^{'}_3\}, I^{P}_{RF}=\{findRoomsInOtherHotels\}$ $I^{R}_{RF}=\{makeR,getWaitingListNum\}$, $t^{'}_1=(A^{'}_{init},B^{'},makeR,Null)$ $t^{'}_2=(B^{'},C^{'},getWaitingListNum,NumOverConstra\text{ int})$, $t^{'}_3=(C^{'},D^{'}_{fina},findRoomsInOtherHotels,\overline{Null})$
Full Path	$P_w(RRH)=\{p_1\}$, $p_1=<t_1,t_2>$	$P_w(RF)=\{p^{'}_1\}$, $p^{'}_1=<t^{'}_1,t^{'}_2,t^{'}_3>$.

Then the Pi-calculus expression of the interaction between *RRF* and *RF* and the deducing process of the full path $P_w(RF)=\{p^{'}_1\}$ are given as follows.

$$P(RRH,RF)=(RRH.\overline{makeR}<Null>|RRH.\overline{getQuantityAvailable}<NoRoomException>|RF.makeR(Null)|RF.getWaitingListNum(NumOverConstra\text{ int})|RF.\overline{findRoomsInOtherHotels}<Null>\xrightarrow{Null}RRH.\overline{getQuantityAvailable}<NoRoomException>|RF.getWaitingListNum(NumOverConstra\text{ int}|RF.\overline{findRoomsInOtherHotels}<Null>\xrightarrow{NUmOverConstra\text{ int}}RRH.\overline{getQuantityAvailable}<NoRoomException>|RF.\overline{findRoomsInOtherHotels}<Null>\xrightarrow{Null}RRH.\overline{getQuantityAvailable}<NoRoomException>\neq 0$$

So, towards evolution mode 2, the behavior is inconsistent between *RRF* and *RF*. This evolution is infeasible.

Published by Atlantis Press, Paris, France.
© the authors, 2013

Meanwhile, the pointcut definition in aspect *Logging* is significantly changed and the system clearly introduces and deletes the associated matching of joinpoint and advice. So Δ pcd(A, A') is not empty and the pointcut doesn't satisfy the static robustness.

Conclusions

This paper analyzes the inconsistent behavior problems occurred in the dynamic evolution process of aspect-oriented system and puts forward the semantic model of AOA based on Pi-calculus. Then it proposes the behavior consistency verification method with Pi-calculus and Delta analysis. Compared with [8], the method adds the relevant examine function of Aspect. Increased behavior consistency test of aspectual component evolution is the verification of pointcut robustness. Compared with [9], the robustness verification in this paper is easy to understand and adds the dynamic robustness verification based on behavioral interaction. In the scenario of hotel management system as a case study, our method has been validated the feasibility. Due to the dynamic evolution and behavior protocol involving wide knowledge, there are some issues still to be studied in the future work, including the behavior analysis of the composite operations of component in evolution, the detailed analysis of AOA aspect weaving and then explaining woven semantic.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Grant No. 61063012), the Science and Research Project of Education Department of Guangxi Zhuang Autonomous Region under Grant No. [2010] 10, the Program for Excellent Talents in Guangxi Higher Education Institutions under Grant No. [2011] 40.

References

[1] Guangquan Zhang, Mei Rong. A Framework for Dynamic Evolution Based on Reflective Aspect-Oriented Software Architecture. Computer Sciences and Convergence Information Technology, 2009:7-10.

[2] CAO Donggang, MEI Hong, ZHOU Minghui. Supporting crosscutting concern modelling in software architecture design. Frontiers of Computer Science in China 2007, 2007,1(1): 50-57.

[3] Costa-Soria,C., Hervas-Muoz,D., Perez,J., Carsi, J.A. A Reflective Approach for Supporting the Dynamic Evolution of Component Types. Engineering of Complex Computer Systems(ICECCS), 2009:301-310.

[4] Chao You, Minghui Zhou, Zan Xiao, Hong Mei. Towards a Well Structured and Dynamic Application Server. Computer Software and Applications Conference (COMPSAC'09), 2009:427-434.

[5] Ping Yu, Xiaoxing Ma, Jian Lu. Dynamic software architecture oriented service composition and evolution. Computer and Information Technology(ICCIT), 2005:1123-1129.

[6] Yangli JIA, Zhenling ZHANG, Zhoujun LI. Real-time Extension of Component Behavior Protocol and its Compatibility Verification. Computer Science. 2010,37(10):143-147.

[7] LUO Yi, XingYu LI, LianWei GUAN, HU Hao, LU Jian. Study on Behavior Consistency of System on Component Evolution. Computer Science. 2008,35(1):266-271.

[8] .Oquendo, F. Dynamic Software Architectures: Formally Modelling Structure and Behaviour with Pi-ADL. Software Engineering Advances (ICSEA'08), 2008:352-359.

[9] Maximilian Stoerzer, Juergen Graf. Using Pointcut Delta Analysis to Support Evolution of Aspect-Oriented Software. 21st IEEE International Conference on Software Maintenance, 2005:653-656.

© the authors, 2013