



ON ANALYSING INTERACTIONS BETWEEN ASPECTS AT REQUIREMENTS PHASE

¹AMEL BOUBENDIR, ²ALLAOUA CHAOUI

¹Asstt Prof., Department of Computer Science, University of Skikda, Algeria

²Assoc. Prof., Department of Computer Science, Faculty of Engineering, University Mentouri Constantine,
Algeria

ABSTRACT

Aspect Oriented software development (AOSD) is an emerging technology, that improve existing paradigms of development, by providing explicit mean to model crosscutting concern (aspect). However, the complexity of interactions among aspects and between aspects and base modules may reduce the value of aspect-oriented separation of cross-cutting concerns. Aspects must be identified as early possible in the life cycle. Interactions analysis, as well, is desirable to be done as early as possible in the life cycle. In this paper we propose a technique during the requirement phase that allows the user to analyse interactions between aspects, identify aspects interactions, detect and resolve conflicts between them based on the search of Hamiltonians paths. The technique is generic since it exploits the dependencies generated by the operators such as before, after, around and replace. It uses the specification of composition of aspects to analyse aspects and produce rules of composition witch may be used to compose or guide the process of composition. The technique is illustrated through examples.

Keywords: *Aspects, Aspect oriented development, Requirement phase, Aspects Interactions.*

1. INTRODUCTION

Aspects are widely accepted as proprieties that crosscut several components in a system. Aspect Oriented Software Development (AOSD) is an emerging technology that provides explicit mean to model concern that tends to crosscut multiple system components [1, 2].

It is a challenging field of research. On the one hand, the main problems have been defined and addressed, and on the other hand, these problems and theirs solutions have brought new ones.

In this context, the idea of aspects maintains the reasoning about aspects through the software development process [20]. And in order to do that, the software engineer should be equipped with techniques that provide means for systematic identification, separation, representation, composition of crosscutting concern (aspects) [20]. In addition, the software engineer must be equipped with means and methods for identification and analysis of interactions between aspects. He has need to systematic detection and resolution of potential conflicts between aspects throughout the

software development process, in order to successfully reason about aspects and successfully compose them.

From the modularity, adaptability and “evolvability” point of view, the separation of aspects in the base modules reduces the dependency between modules and improves modularity. However, understanding the behaviour of a module and verifying its correctness requires a global overview and understanding of all modules and aspects that might affect the module under construction [11].

The complexity of interactions among aspects and between aspects and base modules may reduce the value of aspect-oriented separation of cross-cutting concerns. Some interactions may lead to the expected behaviour while others are source of unexpected inconsistencies [10].

Thus, it is desirable to detect interactions and potential inconsistencies, as early as possible in the life cycle, preferably at the modelling level [10]. Generally, Aspect Oriented Requirement Engineering (AORE) approaches claim that dealing

with aspect is useful for software development [12, 14].

Identifying and managing early aspects helps to improve modularity in the requirements and architecture design and to detect conflicting concern early, when trades off can be resolved more economically" [3].

There are few works that explicitly cover the problematic of aspects interactions. A large part of this works are focused on analyse and verify programs oriented aspects such as [22, 11]. In [21] Douance et al. propose a first solution to the aspect interaction. Authors use a formal language and syntactical analysis for detection of interaction between set of aspects. They propose a framework to resolve static conflict.

In [11], Mehner et al. dealt with semantics conflicts problems of the crosscutting concerns. Through their work, they explain the difficulty to handle crosscutting concerns and their interactions. The approach oriented aspects adopted is Composition Filters. However, a number of solutions have been proposed to deal with conflicting situation during analysis phase such as [8, 15, 10, 6]. In [8], Rachid et al. propose a generic aspect oriented requirement (AORE) model based on view point and XML. In this approach the authors identify concerns and their relationships. They identify candidate aspects and define in granular level of requirement the specification of composition of each candidate aspect. The conflict are detected and resolved after composed. For resolving conflicts, the authors use a contribution Matrix and attribute weight to conflicting aspects. Also, in [15] Araujo et al. Present an approach to handle crosscutting non functional concern at requirement stage. The process passes by identify functional and no functional concerns, identify crosscutting concerns. Then, compose crosscutting concern in UML models and detect and resolve conflicts.

For dealing with conflicting situations, the authors also, suggest first study the contribution from one concern in relation to all others. If there are two or more crosscutting concerns that contribute negatively and influence the same concern, there is a conflicting case the authors, too suggest made a trade off with stakeholders and attribute priority then compose them accordingly.

In [10], Mehner et al. have proposed an approach for analysing interactions between crosscutting concerns and potential inconsistencies at requirement models. The analysis is performed with graph transformation tool. For that, activities are used to refine use case and then, the activities and

their composition are formalised by using theory of graph transformation system.

In [6], Brito et al. Propose an process to compose crosscutting concern with functional requirement; the main concepts behind this process are those of Match point, conflicting aspects, dominant aspect and composition rule [6]. A match point is where one aspect or more are applied and it is used to detect conflict. To resolve conflict we need to identify dominant crosscutting concern with higher priority. Finally the composition rule is defined for one match point and the concerns are composed accordingly.

In this paper we propose an approach, which allows the user to identify the interactions between aspects, to detect and resolve the conflicts between them at [requirement analysis phase](#). The method exploits the dependencies generated by the operators such as before, after, around and replace.

A second contribution of the paper consists of the proposition of an approach to compose aspects with the base modules using the search of Hamiltonians paths. The approach proposed in this paper is illustrated through examples.

The remainder of this paper is organised as follows. In section 2 we briefly present general concept of aspects oriented development and the main concepts of Aspect oriented requirement engineering (AORE). In Section 3 we present our contribution and explain the technique using an example. In section 4 the algorithm of the proposed technique is turn up on concrete example. Section 5 concludes the paper and presents some perspectives of the work.

2. GENERAL CONCEPTS OF ASPECT ORIENTED DEVELOPMENT

Separation of concerns is a concept that is at the core of software engineering. It refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concern (concept, goal ,purpose, etc...) [9]. Traditional approaches to software development such as object oriented and structured methods have been created with this principle. Each module (class, procedure...) encapsulates certain concerns of software system [17]. However, in a given problem decomposition, certain concerns may be not encapsulated within a modular unit (class, procedure...) [12]. They are called crosscutting concerns (Aspect) [1].

3. OVERVIEW OF THE PROPOSED TECHNIQUE

The proposed technique is generic, since, it is not depend on the way to identify aspects or compose them. It exploits the dependencies generated by the operators to reason on interaction between aspects, it uses composition specification of candidate aspects to achieve roles attribute to analyse component. And supplies an outcome: composition rules, which can be used and implemented by author languages and techniques of composition to successfully, compose aspects with component base

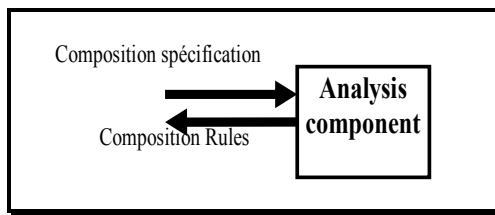


FIGURE1: ANALYSIS COMPONENT

The composition specification of aspect specifies its composition, i.e. where and how it will be attached at join points. But, this specification remains limited. Each candidate aspect encapsulates information needed for its composition. It does not know: with witch others aspects it will be attached at the same join point. it is necessary to get further specification, complete, all encompassing, that organise aspects interactions affecting the same join point: **Composition Rules** [6] . To reach this objective, it is necessary to analyse the problem in order to

- ✓ Satisfy the behaviour of any candidate aspects that will be attached at the join point
- ✓ Satisfy the base behaviour (the join point behaviour), detect potential interactions with aspects, and reason about interactions by resolving any detected conflict and satisfying dependencies between aspects.

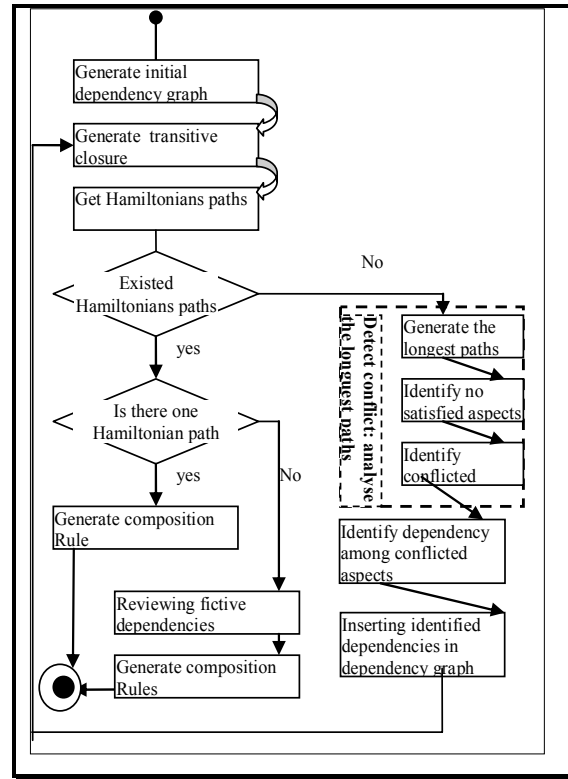


FIGURE 2: Algorithm of analysis of interactions for one join point

Similarly to [5,], this is the general strategy adopted by the proposed technique. In figure2 the general algorithm for analyse interaction in one join point is shown.

The analysis activity includes the following tasks:

- ❖ Detecting interactions between aspects
- ❖ Detecting dependencies
- ❖ Detecting potentials conflicts
- ❖ Reasoning and resolving conflicts
- ❖ Generating composition rule

3.1/- Composition specification of aspect:

It is the input of analysis component. As used in [5], we use a template (table1) to specify crosscutting concerns. The template encapsulates the crosscut specification of an aspect and the behaviour attached at composition (Advice) to a join point. It describes the composition specification for one aspect. This specification follows the general concepts adopted in AOSD. The proposed template is constructed based on the approach proposed in [20].

Aspect: (Name:	Code:...		
Advice:			
Affected use case	Operator	Affected point (optional)	Precondition (optional)	Post condition (optional)

TABLE1: Template to specify crosscutting concern (composition specification of aspect)

The template is used to specify functional crosscutting concern and non-functional concern without deference. Affected use case specifies the base concerns. And, the following operators are adopted to identify how each aspect affects the concerns (operators):

Overlap/before: the candidate aspect is applied before the base concern. The behaviour described by the candidate aspect must be satisfied before satisfaction of the base concern behaviour [15, 20].

Overlap/after: the candidate aspect is applied after the base concern. The behaviour described by the candidate aspect must be satisfied after the satisfaction of the base concern behaviour [15,20].

Override: the behaviour described by the candidate aspect substitutes the behaviour defined by the concern. This operator represents the around qualifier in Aspectj without Proceed [15, 20, 21].

Wrap: the behaviour described by the concern is enveloped by the behaviour described by the candidate aspect. This operator represents the around qualifier in Aspectj with Proceed [15, 20] These operators are generally used in AORE approaches .in follow, the notation below is adopted:

Overlap/before.....> before

Overlap/after.....>After

Override> replace

Wrap.....>around

3.2/- Detection of interactions with candidate aspects

Based on the method described in [6], we use a matrix : matching point matrix, representing the relationships between the stakeholder's requirements (actors) and the model elements (eq Use case) to identify matching points (abstraction of join point) [6], and to identify interactions between candidate aspects . The set of matching points of each candidate aspects are obtained used the composition specification (crosscut specification) of aspects and are filled in the MP-

Matrix, where each cell filled with the list of candidate aspects (denoted Cai) represents a Match point (denoted Mpi) [6].

Concern Stakeholder	Concer n1	Concer2n.....concernn
Stakeholder1	CA1,C A2 (MPa)	CA1,CA4..... (MPb)
:	
Stakeholder	CA3,C A4 (MPd)cA2((MPc)

TABLE2: Match point matrix [6]

For one matching point, it must be specified one composition rule. If there is one candidate aspect affecting the matching point, there is no problem. The dependency aspect match point (base) represented by the type of operator must be satisfied. If there are many candidate aspects affecting the same match point, there are interactions among aspects and with match point (base module).

The interaction is not always negative relationship. It may be positive or negative one, we distinguish between conflict and dependency interaction:

Conflict: captures the situation of interference, one aspect that works correct in isolation, and does not

work correctly any more, when, it is composed with other aspects. The aspect in conflict cannot take place

after satisfying anthers aspects affected the same base module. it is negative interaction [11,18].

Dependency: covers the situation where one aspect explicitly needs another aspect, and depend on it to be satisfied. A dependency is positive one [18,11]. It must be possible to reason about interactions, identify dependencies, and identify and resolve conflicts.

3.3. Identification of dependencies

To illustrate technique , lets suppose the candidates aspects A1,A2,A3,A4 A5 affected the match point (join point) P. Suppose that:

Aspect A1 overlaps before the match point (A1 before P). Aspect A2 overlaps after the match point

P (A2 after P). Aspect A3 wraps the match point with (A3 around P). Aspect A4 substitutes the match point (A4 replace P). Aspect A5 overlaps before the Match point (A5 before P). Aspect A6 overlaps after the match point (A6 after P).

There are interactions between aspects A1, A2, A3, A4, A5, A6 and also with the match point P. So, for identifying dependency we exploit the dependencies generated by the operators.

We propose tree consideration:

First consideration: Based on the type of operator applied to attach the aspect to the match point, we are convinced that there is a dependency, between aspect and the match point.

Operator before: the match point is never satisfied before the satisfaction of the aspects (A1, A5) and the satisfaction of P depends on the satisfaction of Aspects A1 and A5. So, we identify the dependencies: $P \rightarrow A1$ and $P \rightarrow A5$.

Operator After: the match point must be satisfied before satisfying the aspects A2, A6, because the behaviour of aspect A2, A6 must be attached after P. So the satisfaction of A2 and A6 depends on the satisfaction of P and we identify the dependencies $A2 \rightarrow P$ and $A6 \rightarrow P$.

The operator around: the behaviour of the aspect A3 must be satisfied in parallel with the behaviour of the join point P. It is considered like a case of synchronization (P synchronises with A3). The behaviour of the join point is satisfied after the satisfaction of the behaviour of aspect A3 (and execution of precede instruction like Aspectj). Therefore, the satisfaction of P depends on the satisfaction of A3 and we identify the dependency: $P \rightarrow A3$. This dependency is noted ($P \Rightarrow A3$) (in parallel).

The operator replace: the operator substitutes the behaviour of P by the behaviour of A4. The behaviour of P is not executed, but, unless reach P, the behaviour of A4 is not satisfied. So the satisfaction of A4 depends on P. We denote this dependency: $A4 \dashrightarrow P$ (P is not executed, A4 replace P).

Second consideration: the dependency is a transitive relationship. For aspects A_i, A_j, A_k : A_i depend on A_j and A_j depend on A_k implies A_i depends on A_k . Let's suppose candidate aspects A_i, A_j, A_k . A_i must be satisfied before A_j , and A_j must be satisfied before A_k . So it is evident that A_i must be satisfied before A_k .

Third consideration concerns: for operators around and replace, we can identify some fictive dependencies (artificial). in definite likelihood

-Operator around: the behaviour of aspect A3 must be satisfied in parallel with the behaviour of

the join point P, it permits us to deduct that exists a firm probability that the aspect A3 is dependent on all aspects of which the join point P is dependent,. Fictive dependencies $A3 \rightarrow A1$, $A3 \rightarrow A5$ are identified. We note them in red.

-The operator replace: aspect A4 modifies the behaviour of the join point P. Therefore, it permits us to conclude, that exists a concrete probability that all aspects depending on the join point P become dependent on the aspect A4. The fictive dependencies $A6 \rightarrow A4$, $A2 \rightarrow A4$ are identified.

The fictive dependencies are not real ones. They are characterized by some degree of likelihood (weak or strong), their use and identification is not mandatory but they have the advantage to help and to simplify the analysis. They allow us to generate the possible solutions on a certain degree of probability and to focus the analysis on a reduced set of dependencies.

3.4. Graph of dependency and transitive closure:

The graph of dependency $G(X, U)$ represents identified dependencies. Nodes set (X) includes join point and aspects that will be inserted. Initially, and in first stage, the set of edges (U) includes aspects-match point dependencies (with or without fictive dependencies). The transitive closure $G^+(X, U)$ of the dependency graph permits us to represent direct and indirect dependencies, while including the transitive dependencies that one can deduce.

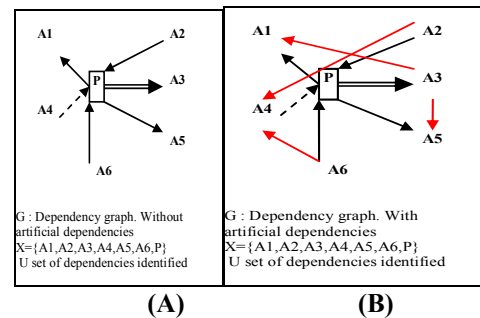


FIGURE3: Dependency graph (a): without fictive dependencies, (b) with fictive dependencies

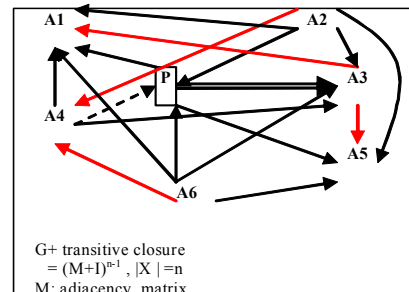


FIGURE4: TRANSITIVE CLOSURE

3.5. Detection of conflicts between Aspects

Once the initial dependency graph and its transitive closure are generated, our objective is to satisfy all aspects and the join point according to the dependencies between the aspects and the join point.

This may be done by a simple search of Hamiltonian paths in the transitive closure of dependency graph. We notice that, a Hamiltonian path is an elementary path, which passes through all nodes once only once.

So, we can consider that the Hamiltonian path in the transitive closure of dependency graph is a solution, which satisfies the behaviour of join point (bases) and aspects (that pass through all the nodes once and only once). The identification of conflicts between aspects becomes a response to the trivial question: Is there a Hamiltonian path that satisfies the bases (join point) and all the inserted aspects? If there is no Hamiltonian path, then there is a conflict. At least, one aspect is in conflict. It is not satisfied (it can not reach the join point). Notice that the conflict in this case is an *order conflict*.

In the next step, we identify which aspects are not satisfied. To this end, we generate all the longest paths in the transitive closure. We analyze generated paths to identify the non satisfied aspects for each path. Then, we identify the aspects that are satisfied in mutual exclusion. For instance, see the transitive closure shown in figure4. There are no Hamiltonian paths in the transitive closure, so there is at least one order conflict. In this case, the longest paths are shown in the following table.

Longest paths	Analyze of the longest paths
CH1= A2A4PA3A5	A6,A1: are not satisfied
CH2= A2A4PA3A1	A6,A5: are not satisfied
CH3= A6,A4PA3A5	A2,A1: are not satisfied
CH4= A6A4PA3A1	A2,A5: are not satisfied
Synthesis of conflicts analysis (mutual exclusion)	Conflict between (A1,A5) Conflict between (A6,A2)

TABLE 3: The longest paths of the example shown in figure

3.6. Conflict Resolution

Once, the aspects in conflict are detected. We must resolve them. The solution we propose consists of adding and identifying a resolution dependency between aspects in order conflict (mutual exclusion). The resolution dependencies here represent information about the order of execution of aspects in conflict. Let's say:

- ❖ The priority between aspects: A_i has a higher priority than A_j implies that aspect A_j depends on aspect A_i . The satisfaction of A_i before the satisfaction of A_j
- ❖ An aspect A_i uses an aspect A_j , this imply that aspect A_i depends on A_j (the satisfaction of A_i depend on the satisfaction of A_j)
- ❖ An aspect A_i has preconditions included in post-conditions of A_j implies that aspect A_i depends on A_j (since the precondition to execute A_i depends on the execution of A_j).

The added dependencies can be identified from the analysis of the preoccupation specifications and or making a direct trade-off with the concerned stakeholder. For illustration, see the former example. We suppose after a discussion with the stakeholder, we define a priority on concerns: A_1 has a higher priority than A_5 , A_6 has a higher priority than A_2 . We identify the dependencies $A_5 \rightarrow A_1$ and $A_2 \rightarrow A_6$. After, when conflicts are treated and resolved, the identified dependencies of resolution are added to the dependency graph.

We generate the new dependency graph which includes resolution dependency (Aspect-Aspect). Also, we generate the transitive closure of dependency graph. At last, we find again Hamiltonians paths. Two situations may occur: there is one or several Hamiltonians paths. If there are several Hamiltonians paths, we must review each solution, to verify the fictive dependencies and to only keep the strong one, the weak dependencies are removed. Therefore, Hamiltonians paths which include weak dependencies, are not considered more like a solution, and will be suppressed.

3.7. Generate composition rule

After obtaining Hamiltonians paths and verification of fictive dependencies, we can generate the composition rule specification easily. For more illustration see previous the example: (figure 3: initial transitive closure), figure 5: is Transitive closure after inserting resolution dependence ($A_5 \rightarrow A_1$), ($A_2 \rightarrow A_6$)

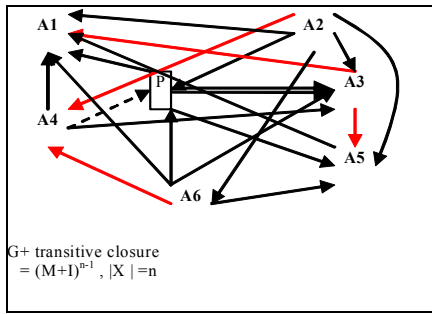


FIGURE5: Transitive closure after inserting resolution dependencies

Hamiltonians paths founded are: $Ch = A_2A_6A_4PA_3A_5A_1$.

The composition rule can be written (according to the direction of the small arrow above the path)

A1 before P A5 before P A3 around P P A replace P A6 after P A2 after P

We can write it according to LOTOS operators described in [17] as follows:

$A1 \gg A5 \gg ((P \gg A4) \parallel A3) \gg A6 \gg A2$

4. CASE STUDY

Let us consider the example borrowed from [13]. It is a simple version of the sub way. The requirements for the subway are: to use the subway a client has to own a card that must have been credited with some amount of money. A card is bought and credited in special buying machines available in any subway station. A client uses this card in an entering machine to initiate her/his trip. When she/he reaches the destination, the card is used in an exit machine that debits it with an amount that debits it with an amount that depends on the distance travelled. If the card has not enough credits the gates will not open unless the client adds more money to the card. The client can ask for a refund of the amount in the card by giving it back to a baying machine.

Let's consider the simpler situation where only the actor client is handled. The corresponding use case diagram to specify the functional concerns is illustrated in figure 8.

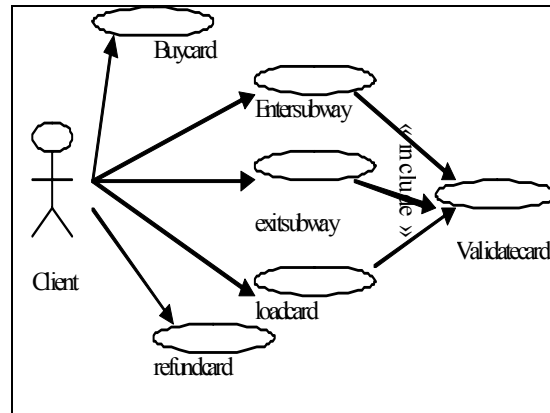


Figure 8: the use case diagram of the subway system

In this example we identify the following crosscutting concerns: validate card (functional concern) and the no functional concerns: Response time, Accuracy, Multi-access, Availability, Security.

security is composed of sub concerns: S.integrity, S.availability. The integrity is composed of sub concern: S.integrity.completeness. and S.integrity.accuracy.

Let's consider just the Enter subway and validate card use cases (match point):

- Response time (RT) concern wraps Entersubway use case : (RT **around** Entersubway)
- Availability (S.Av) overlaps before Entersubway use case : (S.AV **before** Entersubway)
- integrity (S.integrity) overlaps after the match point Entersubway: (S.integrity **after** Entersubway)
- Accuracy (S.integrity.accuracy) wraps Entersubway : (S.integrity.accuracy **around** Entersubway)
- Validatecard overlaps before Entersubway use case: (Validatecard **before** Entersubway) and accuracy (S.integrity.accuracy) wraps Validatecard use case: (S.integrity.accuracy **before** Validatecard)

step1: Identify interaction :The interaction are identified and represented in table 5.

Concern	entersubway	validatecard
Stakeholder		
client	Validate card, RT, s.integrity.AC S.AV, S.integrity.AC ,S.integrity	

Table 5. Identification of interactions

Step2: Generate initial dependency graph and transitive closure:

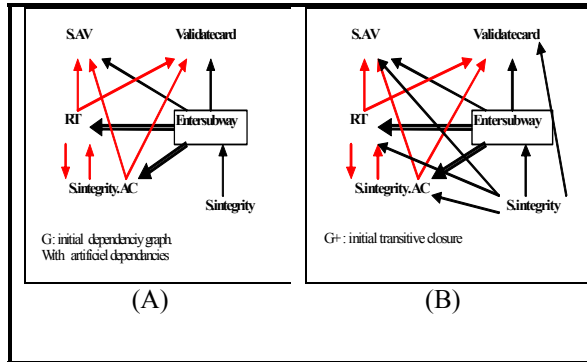


Figure 6: (A) Dependency graph, (B) Transitive closure of example.

Step3: Detection of conflicts: : No Hamiltonians paths in the transitive closure: there is conflict. Then we find the longest paths in the transitive closure and analysis of each path. See table 6.

Longest paths	Analysis of longest paths
Ch1 = S.intégrité,Entersubway,s.intégrité.AC, RT, Vaidatecard	S.AV : no satisfied
Ch2 = S.intégrité,entersubway,RT, S.intégrité.AC, sSAV	Validatecard : no satisfied
Ch3 = S.intégrité, Entersubway,RT, S.intégrité.AC, validatecard	S.AV : no Satisfied
Ch4 = S.intégrité, Entersubway,RT, s.intégrité.AC, s.AV	Validat card : no satisfied
Summary of conflict analysis (mutuel exclusion)	Conflict between (Validatecard, S.AV)

Table6: Longest paths and their analysis

Step4: Resolution of conflicts S.AV has higher priority than validatecard, (S.AV constrain all the requirement of Entersubway use case): (validatecard→s.AV) dependency is identified and inserted to the dependency graph .

Step5: Regeneration of dependency graph and transitive closure the generated dependency graph and transitive closure are shown in figure 7.

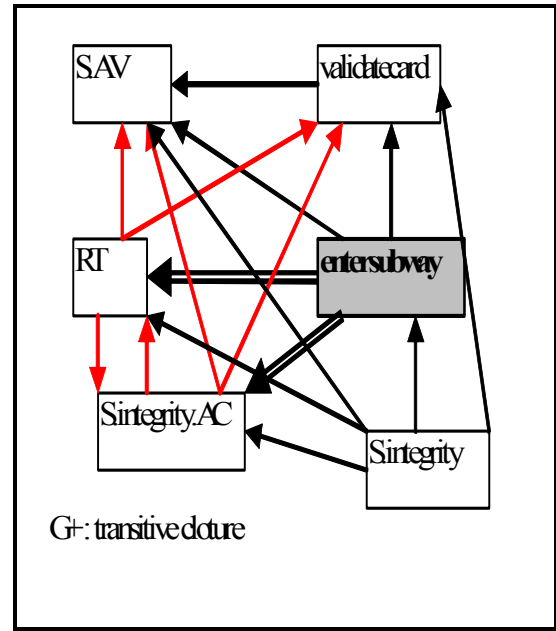


Figure 7. The generated dependency graph and transitive closure

The Hamiltonians paths are:

Ch1= S.intégrité, Entersubway,S.intégrité.AC, RT , Validatecard,S.AV

Ch2= S.intégrité,entersubway,RT,S.intégrité.AC, Validatecard, S.AVE

Step6: Reviewing fictive dependencies: the dependency (S.intégrité AC →RT)is weak dependency .So it is deleted, ch1 is not a Hamiltonian path

The solution accepted is ch2.

Step7: Generation of the composition rule:

For Entersubway use case the composition rule is :
 $S.AV \gg validatecard \gg ((entersubway \parallel RT) \parallel S.intégrité.AC) \gg S.intégrité$

for Validate card use case the composition rule is:
 $Validatecard \parallel S.intégrité.AC$

As validatecard use case is included in Entersubway use case, we can fusion the two composition rules to obtain: the synthesis composition rule for Entersubway use case:

$S.AV \gg ((Validatecard \gg (Entersubway \parallel RT)) \parallel S.intégrité.AC) \gg S.intégrité$

5. CONCLUSION

In this paper we have proposed a generic technique at requirement phase allowing the user to identify interactions between aspects. Then, detect and resolve the conflicts between these aspects. The proposed technique is generic since it is



independent on the way to identify aspects or compose them. It exploits the dependencies generated by the operators to reason on interaction between aspects and uses composition specification of candidate aspects to achieve the roles attributed to analyse component. The technique exploits the dependencies generated, by the operators such as before, after, around and replace. And also, use the search of Hamiltonians paths in transitive closure to detect potential conflicts

This work is a first step towards analysis of interactions between aspects, and there are many problems to resolve. Our future work will focus on developing a support to this method, improving and applying it on more complicated case studies.

REFERENCES:

- [1] The Aspect-oriented Software Architecture Design portal:
[Http://trese.cs.Utwente.nl/taosad/aosd.htm](http://trese.cs.Utwente.nl/taosad/aosd.htm)
- [2] AOSD homepage, [HTTP://WWW.AOSD.net](http://WWW.AOSD.net)
- [3] E.Beniassad, P.C.Clements, J.Araujo, A.Moriera, A.Rachid, B.Tekmerdogan, 2006, "Discovering early aspects" ; IEEE Software, e3[1]:61-70
- [4] J.Araujo,E.Baniassad,P.Clements,A.Moriera, A.Rachid, B.Tekinerdogan, "Early aspect: the current landscape", Technical Report , Lancaster university February, 2005 .
- [5] A.Chaoui, A.Boubendir,"An approach to analyse interactions between aspects at requirement phase", poster paer, CEE-SECR, 2009.
- [6] I. Brito, A. Moreira, "Towards a composition process for aspect-oriented requirements" , Proceeding of AOSD'03 Workshop on Early Aspects: Aspect oriented Requirements Engineering and Architecture , March 17, Boston USA. 2003.
- [7] A.Rachid, PSwer, A.Moreira and J.Araujo, "Early aspect: a model for Aspect-Oriented Requirements Engineering" in International conference on Requirements Enginnering (RE).2002,Essen, Germany : IEEE.
- [8] A. Rachid, A.Moreira and J.Araujo/ "Modulaisation and composition of Aspectual Requirements". In 2and International conference on Aspect Oriented Software Development (AOSD). 2003 Bostan, USA: ACM.
- [9] Multi-Dimensional Separation of concern: an overview:
<http://www.reseach.ibm.com/hyperspace/MDSOC.htm>
- [10] K. Mehner,Monga G. Taentzer , "Interaction Analysis in Aspect-Oriented Models," re, pp.69-78, 14th IEEE International Requirements Engineering Conference (RE'06), 2006.
- [11] Bergmans, "Towards Detection of semantic conflicts between crosscutting concerns", AAOS 2003,Darmstadt, Germany .
- [12] L.Rosenheiner,"A method for Handling Requirements- level crosscutting concern", available from URL:
http://www.pi.informatik.unisiegen.de/stt/26_1/01_Fachgruppenberichte/RE/08_rosenhainer.pdf
- [13] : I.Brito, A.Moreira, "Integrating the NFR framework in a RE Model" .In Processing of the 3rd workshop on Early Aspects, 3rd international Conference on Aspect-Oriented Software Development, March 2004.
- [14] H. Kandi, "What is an aspect in Aspect oriented Requirement Enginneering" Proceeding ofAMMASAD 2008.
- [15] J.Araujo, A.Moreira,I.Brito, A. Rachid, " Aspect oriented requirements with UML" , Workshop: Aspect-Oriented Modelling with UML ,UML 2002,Dresden,Germany . October 2002.
- [16] J. Brichau, T. D'Hondt, "Introduction to Aspect-Oriented Software Development", AOSD-Europe, 30August 2005.
- [17] I. Brito ,A.Moreira. "Advanced separation of concerns for requirements enginnering" , VIII jornadas de ingenieria de Software y bases de datos (JISBD), Alicande,Spain,12-14 november 2003.



- [18] F. sanen, E. Truyen, B. D'win, W. Joosen, N. Loughran, G. Coulson, A. Rachid, A. Nedos, A. Jackson, S. Clark, "Study on interaction issues", AOSD-Europe, 28 February 2006.
- [19] B. Tekmerdogan, A. Moreira, J. Araujo, P. Clements, Early Aspects : Aspect-Oriented Requirements Engineering and Architecture Design: Workshop Report AOSD 2004 TR-CTIT-04-44 , 119PP University of Twente Department of Computer Science , October 2004.
- [20] G. Sousa, S. Soares, P. Borda and J. Castro, Separation of crosscutting Concerns from Requirements to Design: Adapting an use case Driven Approach . In proceedings of the 3rd Workshop on Early Aspects, 3rd international conference on Aspect-Oriented Software Development, March 2004.
- [21] Xerox corporation, AspectJ Programming guide, available from: <http://eclipse.org/AspectJ> .
- [22] R. Douance, P. Frader, "Detection and resolution of aspect interactions", INRIA technical report N°RR 4435 April 2002.