Dealing with aspect interaction: Thinking for generic based specification solution

Amel Boubendir[1], Allaoua Chaoui[2]
[1]Department of Computer Science
University of Skikda
Algeria
a_boubendir@yahoo.fr

[2] MISC Laboratory, Department of Computer Science
University Mentouri Constantine
Algeria
a_chaoui2001@yahoo.com

ABSTRACT: *Despite Aspect Oriented software development (AOSD) improve existing paradigms of development, by providing explicit mean to model crosscutting concern (aspect), the complexity of interactions among aspects and between aspects and base modules still a difficult problem that may reduce the value of aspect-oriented separation of cross-cutting concerns. The need of developing efficient techniques that permits us identification and analysis of interactions between aspects, rise and grow. In this paper we propose a technique that allows the user to analyse interaction between aspects, detect and resolve conflicts between them based on the search of Hamiltonians paths. The technique is based specification it use the specification of composition of aspects to analyse aspect and produce rules of composition and it is supposed generic in fact, that exploits the dependencies generated by the operators such as before, after, around and replace . Also, our contribution will be at aspect oriented requirement engineering (AORE) since, it is desirable to early possible identify aspect and analyse interactions too. The technique is illustrated through examples.*

Keywords: Aspects, Crosscutting concern, Separation of concern, AORE, Aspects interactions

Received: 10 June 2010, Revised 7 July 2010, Acceped 13 July 2010

1. Introduction

Aspect **O**riented **S**oftware **D**evelopment (AOSD) is an emerging technology that provides explicit mean to model concern that tend to crosscut multiple system components [1,2]. It is a challenging field of research. On the one hand, the main problem have been defined and addressed, and on the other hand, these problems and theirs solutions have brought new ones [20]. From the modularity and adaptability 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 aspectoriented separation of cross-cutting concerns. Some interactions may lead to the expected behaviour while others are source of unexpected inconsistencies [10]. 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, methods and techniques for dealing with aspects interactions .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.

As, aspect must be early possible identified in the life cycle. As well, the detection of interactions and potential inconsistencies between aspects, is desirable to be as early as possible in the life cycle,[10]. Our contribution will be at aspect Oriented requirement engineering (AORE) .

However, There seems to be a strong focus on identification of aspect in all the approach of AORE[18] , a number of solutions have been proposed to deal with conflicting situation such as [8, 15, 10, 6]. In [8], Rachid et all. Propose a generic aspect oriented requirement [AORE) model based on view point and XML. In this approach the authors identify concerns and theirs relation ships. 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 all. Present an model 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 all. Propose 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 all. Propose a 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 accord ally.

In this paper we propose a technique that allows to the user analyse interaction between aspects, identify aspects interactions, detect and resolve the conflicts between them based on the search of Hamiltonians paths .The technique is supposed generic in fact, that exploits the dependencies generated by the operators such as before, after, around and replace, and based specification it uses the specification of composition of aspects to analyse aspect and produces rules of composition witch may be used to compose or in less guide de process of composition .

The remainder of this paper is organised as follows. In section2 we briefly present general concept of aspects oriented development and the main concepts of Aspect oriented requirement engineering (AORE).In section" we present a classification of aspects conflicts. In Section4,5 and 6 we present our contribution and explain the technique with abstract example. In section7 the algorithm of proposed technique is turn up on concrete example. Section 9 concludes the paper and presents some perspectives of the work.

2. General Concepts Of Aspects 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]. In consequence, the modularity of the system can be improved, leading to a reduced complexity and easier maintainability [1]. The main concepts that are introduced are: *Point cut specification* join point, Advice *and Weaving* [1, 2]. There are several aspect oriented approaches and languages. They differ in the way to specify Aspects, Point cuts, Advice and weaving [1,2,16]. Conventional aspects oriented development techniques have been fairly limited to the implementation phase. However, recent works have tried to generalise the concept of aspect and apply it to different phases of the life cycle of software [19] such as requirement analysis and design.

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]

At this phase, the crosscutting concerns are candidates aspects. The analysis of their interactions at this early stage constitutes an early understanding of their interactions. Aspect oriented requirement approaches are those that explicitly recognise the importance of identifying, treating, reasoning about crosscutting concern/requirement at the analysis phase [16,4].

These approaches improve existing requirement engineering approaches, by providing explicit representation and modularisation of concern/requirement that crosscut the requirement artefacts. The modular representation of crosscutting requirement improves the tractability of requirements, through all other artefacts of software life cycle [4]. Also, these approaches focus on the composition principle. It should be possible to compose each concern/ requirement with the rest of the concerns of the system under construction, to understand interactions, interrelations, tradeoffs (conflicts) among concerns [16, 4].

3. Categories Of Conflicts With Aspects

Aspect and base concerns can conflict with each other or themselves. There is a need to categorize the potential inconsistencies and conflicts that can occur in aspect-oriented software, in order to provide a list of issues that developers should be aware of, and, needs to be investigated. J.Hannemann, and all In [23] Define the following four categories of conflicts and inconsistencies in aspect oriented software:

• **Crosscutting specifications:** Now, aspect-oriented approaches specify crosscutting in terms of join points in the base concerns. This can lead to two problems: *accidental join points* imply accidentally matching unwanted join points thus applying the aspect's behaviour at the wrong places. And *accidental recursion that* refers to the situation when the aspect behaviour itself matches a join point specification description leading to recursion.

• **Aspect-aspect conflicts:** When multiple aspects exist in a system, they can conflict with each other which are also called *aspect interaction*. five categories of interactions are identified :

- *conditional execution* where applicability of one aspect is dependent on another aspect being applied;
- *mutual exclusion* when composing one aspect implies that another one must not be composed;
- *ordering* required when aspects influence the same point in the base concerns;
- *dynamic context dependent ordering* which differs from simple, ordering in that case depends on the context in which the aspects are being applied;
- *Tradeoffs and conflicts at requirements and architectural level* where aspects influencing the same Elements can result in having to compromise particular requirements in favour of others[23].

• **Base-aspect conflicts:** Aspects can conflict not only with each other but, also with the base concerns. Lead to *circular dependencies between aspect and base.* And the need for *base to communicate with the aspect* [23]

• **Concern-concern conflicts:** conflicts can also occur between concerns. These kinds of problems arise when concerns affect the execution or state of other concerns. The *change of functionality* made by aspects can impact other modules. *Inconsistent behaviour* can occur when an aspect destroys or manipulates the state of another aspect or the base concern. [23]

4. Overview of the Proposed Approach

The proposed technique for dealing with crosscutting concerns interactions is proposed at analysis phase. At this phase, the crosscutting concerns are candidates' aspects. The analysis of their interactions at this early stage constitutes an early understanding of their interactions, and the resolution of potential conflicts between them in first step leads to an aspect oriented development without conflicts. The proposed approach of analysing aspect is specification based approach 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 succeflly, compose aspects with component base. The proposed technique is generic one, 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,

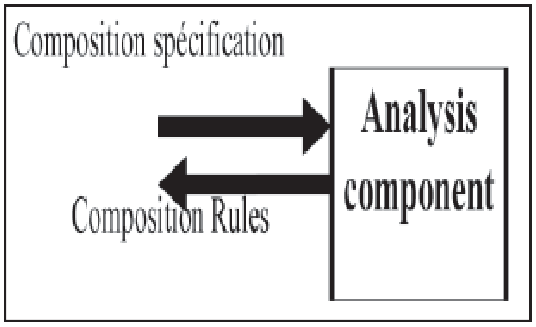

Figure 1. Analysis Component

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
- Satisfy dependencies between aspects.

Similarly to [5,24], this is the general strategy adopted by the proposed technique. The activity diagram shown in figure2 give an overview of the proposed approach

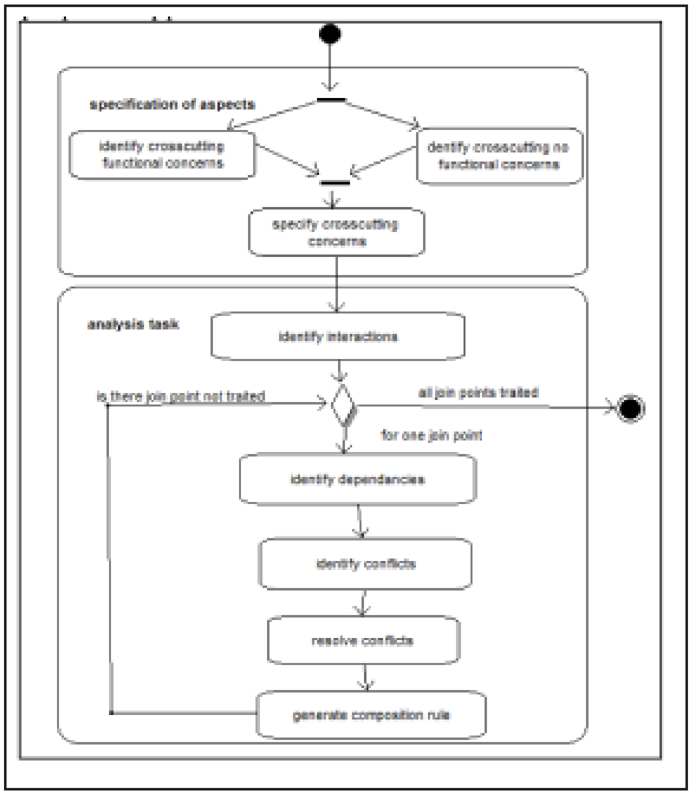
specification of aspects
identify crosscutting functional concerns
dentify crosscutting no functional concerns
specify crosscutting concerns
analysis task
identify interactions
is there join point not traited
all join points traited
for one join point
identify dependancies
identify conflicts
resolve conflicts
generate composition rule

Figure 2. Overview of the proposed approach

5. Composition Specification of Aspect: The Input of Ananlysis 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	*Operat tor*	*Affected point (optional)*	*Precondition (optional)*	*Post condition (optional)*

Table 1. 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

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

6. Analysis Activity

In figure3 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

6.1- 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

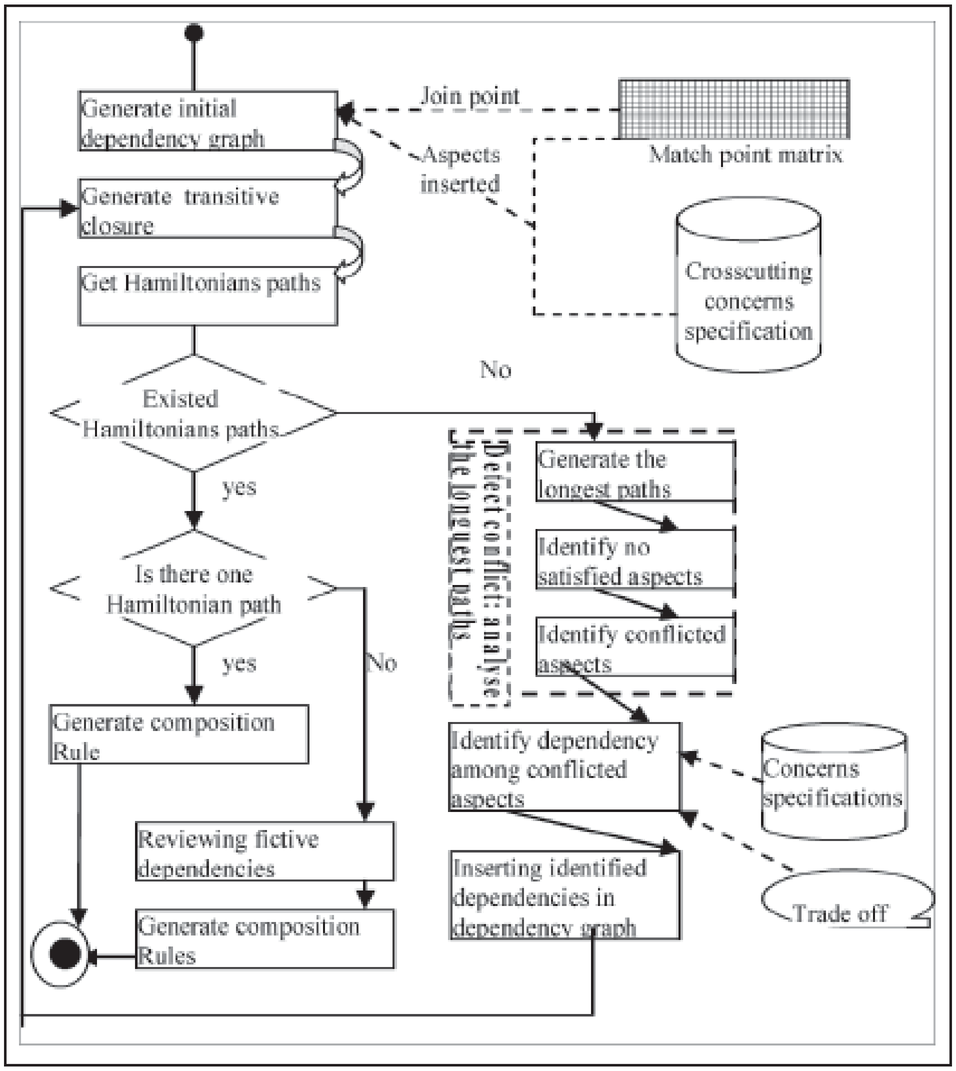
Generate initial dependency graph
Join point
Match point matrix
Aspects inserted
Generate transitive closure
Get Hamiltonians paths
Crosscutting concerns specification
No
Existed Hamiltonians paths
yes
Detect conflict: analyse the longuest paths
Generate the longest paths
Identify no satisfied aspects
Identify conflicted aspects
Is there one Hamiltonian path
yes
No
Generate composition Rule
Identify dependency among conflicted aspects
Concerns specifications
Reviewing fictive dependencies
Trade off
Generate composition Rules
Inserting identified dependencies in dependency graph

Figure 3. Algorithm of analysis interaction for one join point

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	***Concern1***	***Concern2 ……..concernn***
Stakeholder1	***CA1,CA2 (MPA)***	***CA1,CA4………………….. (MPb)***
…	**………………………….**	
Stakeholder	***CA3,CA4 (MPd)***	***…………………….….CA2 (MPc)***

Table 2. 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.

6.2. 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 afterP). 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—> A1 and P—> 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—> P and A6—> 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—> A3. This dependency is noted (P=>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—> P (P is note executed, A4 replace P).

Second consideration: the dependency is a transitive relationship. For aspects Ai,Aj,Ak: Ai depend on Aj and Aj depend on Ak implies Ai depends on AK. Let’s suppose candidate aspects Ai, Aj, Ak. Ai must be satisfied before Aj , and Aj must be satisfied before Ak. So it is evident that Ai must be satisfied before Ak .

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

-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—>A1, A3—>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—>A4, A2—>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.

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 indict dependencies, while including the transitive dependencies that one can deduce.

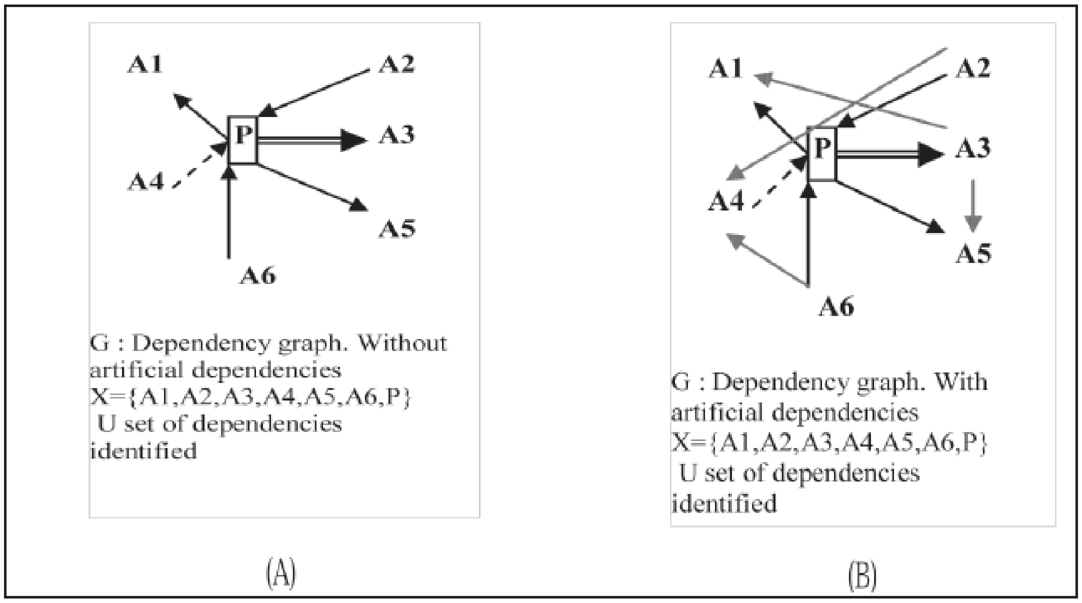
A1
A2
P
A3
A4
A5
A6
G : Dependency graph. Without artificial dependencies
X={A1,A2,A3,A4,A5,A6,P}
U set of dependencies identified
A1
A2
P
A3
A4
A5
A6
G : Dependency graph. With artificial dependencies
X={A1,A2,A3,A4,A5,A6,P}
U set of dependencies identified
(A)
(B)

Figure 4. Dependency Graph (A): Without fictive dependencies,(B):With fictive dependencies

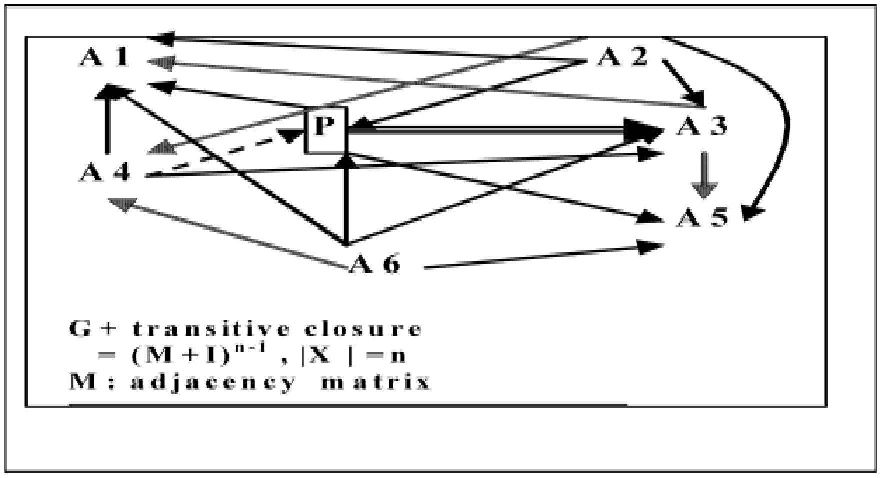

Figure 5. Transitive Closure

6.3. 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 figure5. 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.

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

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: A1 has a higher priority than A5, A6 has a higher priority than A2 . We identify the dependencies A5—>A1 and A2—>A6. After, when conflicts are treated and resolved, the identified dependencies of resolution are added to the dependency graph.

We generate the new dependency graph witch 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.

6.5. 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 6: is Transitive closure after inserting resolution dependencies (A5—>A1), (A2—>A6)

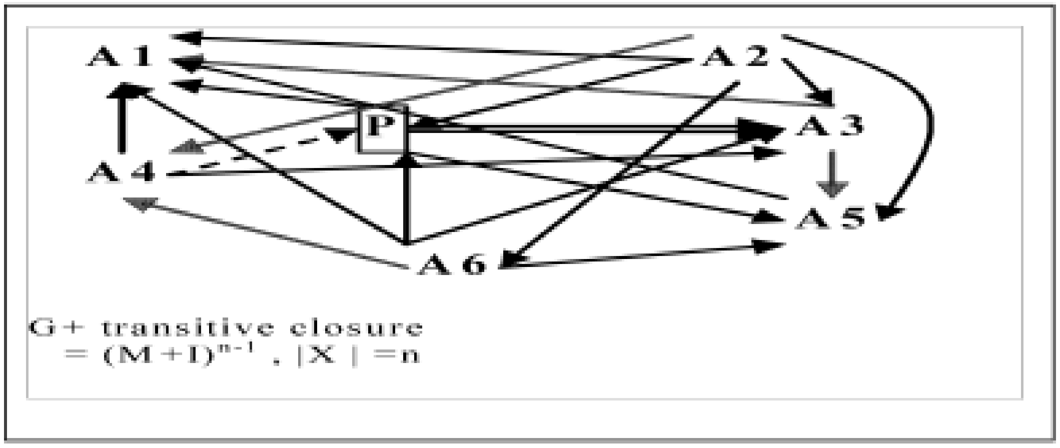

Figure 6: Transitive Closure after inserting resolution dependencies

Hamiltonians paths founded are: Ch= A2A6A4PA3A5A1<—.

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

A4 replace P

A6 after P

A2 after P

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

A1>> A5>> ((P>]A4) ||A3)>>A6>> A2

7. Cas Study

Our aim is to perform the technique on concrete case study and more explain the main ideas proposed in this paper . 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 creditedin 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 fig 7.

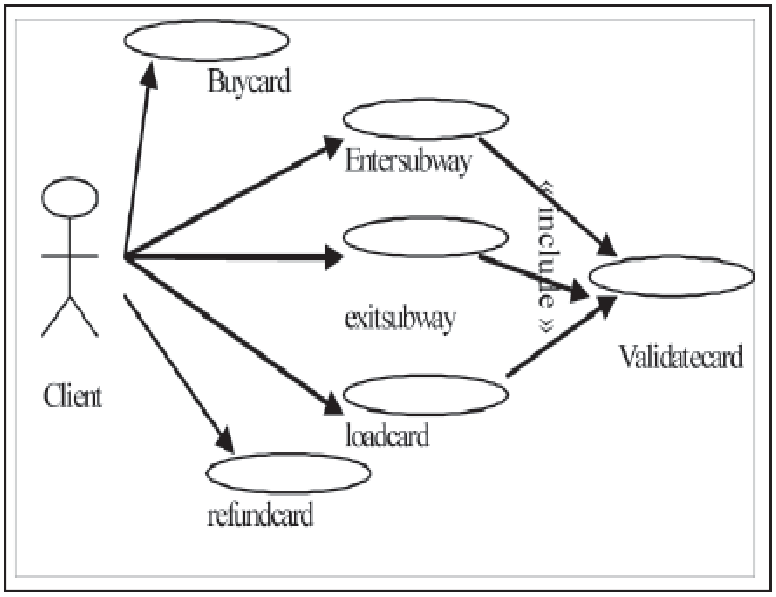

Figure 7. The use case diagram of the subway system

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

Table 5. Identification of interactions

Step 2: Generate initial dependency graph and transitive closure:

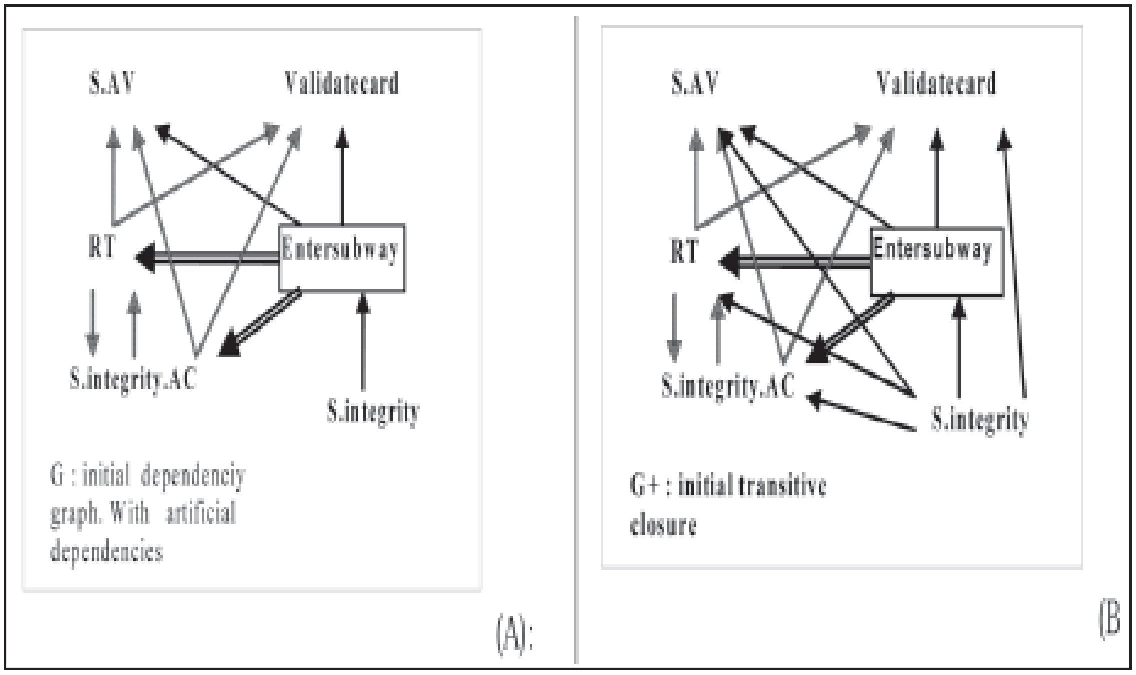

Figure 8. (A) Dependency graph, (B) Transitive closure of example.

Step 3: 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.integrité,Entersubway,s.integrity.AC, RT, Vaidatecard	S.AV : nosatisfied
Ch2 = S.integrity,entersubway,RT, S.integrity.AC, sSAV	Validatecard : no satisfied
Ch3 = Sintegrity, Entersubway,RT, S.integrity,AC, validatecard	S.AV : no Satisfied
Ch4 = S.integrity, Entersubway,RT, s.integrity,AC, s.AV	Validat card : no satisfied
synthèse d'analyse des conflits (mutuel exclusion)	Conflict between (Validatecard, S.AV)

Table 6. Longest paths and their analysis

Step 4: 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 .

Step 5: Regeneration of dependency graph and transitive closure the generated dependency graph and transitive closure are shown in figure 9.

The Hamiltonians paths: are:

Ch1= S.integrité, Entersubway,S.integrity,AC, RT , Validatecard,S.AV

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

Step 6: Reviewing fictive dependencies: the dependency (S.integrity AC—>RT)is weak dependency .So it is deleted, ch1 is not a Hamiltonian path

The solution accepted is ch2.

Step 7: Generation of the composition rule:

For Enersubway use case the composition rule is :

S.AV >>validatecard>> ((intersubway || RT) || S.integrity.AC) >>S.integrity

for Validate card use case the composition rule is:

Validatecard || S.integrity.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 Enersubway use case:

S.AV >> ((Validatecard >> (Entersubway || RT)) ||S.integrity.AC)>>S.integrity

8. Conclusion

Aspect oriented software development (AOSD) is an emerge technology, that provides explicit mean to model concern that tend to crosscut multiple system components. From the view of modularity, adaptability, the separation of aspects to the base modules reduces dependency between modules and improves modularity. However, the complexity of interactions among aspects and between aspect and base module may reduce the value of aspectoriented separation of cross-cutting concerns.

In this paper we propose a technique. That allows the user to identify interactions between aspects. Then, detects and resolves conflicts between these aspects.

The proposed technique is generic one, 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

We argue it is necessary to obtain the composition rule, indeed it is a specification that satisfy the behaviour of the base (point of junction), and aspects, as well, satisfy dependencies identified between aspects and with the base. 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 .We propose, identifies dependencies witch define an order between aspects in order conflict to solve conflicts, and imposing a certain order of execution between them .The Hamiltonians paths obtained define the solution to compose aspect, provided that we define the true dependencies. Also, if it remains several Hamiltonians paths, in this case, it is necessary to verify that composed specifications are equivalent in any case; otherwise, it is necessary to correct some possible mistakes in the specifications (concerns and/or crosscutting concerns)

This work is a first step towards analysis interaction between aspects; we are dealing with aspect-aspect conflicts we are focused on dealing with ordering conflict. And there are many problems to resolve and solution to test. Our future work will focus on developing a support to this approach, improving it by thinking for dealing with others categories of conflicts and applying it in more case studies.

References

[1] The Aspect-oriented Software Architecture Design portal http://trese.cs.Utwente.nl/taosad/aosd.htm
[2] AOSD homepage, http://www.AOSD.net
[3] Beniassad, E., Clements, P.C., Araujo, J., Moriera, A., Rachid, A., Tekmerdogan, B. (2006). Discovring early aspects, IEEE Software, e3[1) 61-70
[4] Aaraujo, J.E.,Baniassad, P., Clements, A., Moriera, A.,Rachid, B.,Tekinerdogan (2005). Early aspect: the current landscape, Technical Report , Lancaster university February.
[5] Boubendir, A. Chaoui,A. (2010). Towards a generic technique for analysing interactions between aspects at requirement phase, *In*: International Conference on Digital Information Management ICDIM 2010: 507-512.
[6] Brito, I., Moreira, A. (2003). Towards a composition process for aspect-oriented requirements, *In*: Proceeding of AOSD'03 Workshop on Early Aspects: Aspect oriented Requirements Engineering and Architecture , March 17, Boston USA. 2003. [16]
[7] Rachid, A., Swer, P., Moreira, A., Araujo, J (2002). Early aspect: a model for Aspect-Oriented Requirements Engineering in International conference on Requirements Enginnering (RE) Essen, Germany.IEEE.
[8] Rachid, A., Moreira, A., Araujo, J. (2003). Modulaisation and composition of Aspectual Requirements, *In*: 2nd International conference on Aspect Oriented Software Development (AOSD). Bostan, USA: ACM.
[9] Multi-Dime,nsional Separation of concern: an overview: http://www.reseach.ibm.com/hyperspace/MDSOC.htm
[10] Mehner, K., Monga, M., Taentzer, G. (2006).Interaction Analysis in Aspect-Oriented Models, re, pp.69-78, 14th IEEE International Requirements Engineering Conference (RE'06).
[11] Bergmans. (2003). Towards Detection of semantic conflicts between crosscutting concerns, AAOS 2003,Darmastadt, Germany.
[12] Rosenheiner, L.A method for Handling Requirements- level crosscutting concern, available from URL: http://www.pi.informatik.uni-siegen.de/stt/26_1/01_Fachgruppenberichte/RE/08_rosenhainer.pdf
[13] Brito, I., Moreira, A. (2004). 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.
[14] Kandi, H (2008). What is an aspect in Aspect oriented Requirement Enginneering, *In*: Procedding ofAMMASAD.
[15] Araujo, J., Moreira, A., Brito, I., Rachid, A (2002). Aspect oriented requirements with UML , Workshop: Aspect- Oriented Modelling with UML ,UML 2002,Dresden,Germany.
[16] Brichau, J., Hondt, T. D (2005). Introduction to Aspect-Oriented Software Development, AOSD-Europe, 30August 2005.
[17] Brito, I., Moreira, A.. (2003). Advenced separation of concerns for requirements enginnering" , VIII jornadas de ingenieria de Software y bases de datos (JISBD), Alicande,Spain,12-14 November.
[18]Sanen, F.E., Truyen, B.D'win,Joosen,W., Loughran,N.,Coulson,G., Rachid, A.,Nedos, A., Jackson, A., Clark, S.(2006). Study on interaction issus, AOSD-Europe.
[19] Tekmerdogam, B., Moreira, A.Araujo, J.(2004). Pclemnts,Early Aspects : Aspect-Oriented Requirements Engineering and Architecture Design: Workshop Report AOSD 2004 TR- CTIT-04-44 ,119PP University of twente Dep of Computer science.
[20] Sousa,G., Soares,S., Borda, P., Castro, J. (2004). 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.
[21] Xerox corporation. AspectjProgramming guide, available from: http://eclipse.org/Aspectj .
[22] Douance, R., Frader, P (2002). Detection and resolution of aspect interactions, INRIA technical report N°RR 4435.
[23] Hannemann, J.R., Chitchyan, R. (2003). Awais, « Analysis of Aspect-Oriented software, AAOS, 2003, Darmstadt, Almagne
[24] Boubendir, A., Chaoui,A. (2010).Toward a model for dealing with aspect interactions at Aspect requirement Engineering SEDE, USA.