RESEARCH ARTICLE OPEN ACCESS

Detecting Aspect Intertype Declaration Interference at Aspect Oriented Design Models: A Database Approach

Ahmed Sharaf Eldin*, Maha Hana**, Shady Mohammed Elsaid***
*(Department of Information Systems, Faculty of Computers and Information, Helwan University, Egypt)
**(CIC Cairo Deputed from Helwan University, Egypt)
***(Department of Information Systems, Faculty of Computers and Information, Helwan University, Egypt, Department of Computer Science, Faculty of Computers and Information Systems, Um Al-Qura University, Makkah Al-Mukaramah, Saudi Arabia)

ABSTRACT
Implementing crosscutting concerns requires aspect oriented developers to be enabled to introduce some members to core concerns modules along with other. This may lead to a problem of interference among modules, either between classes and aspects or among aspects themselves. Such conflicts may cause program to crash at runtime. Interference problem is addressed but with complex solutions that become more complicated proportionally with the project size. In this work a relational database approach and relational algebra is used to detect intertype declaration interferences in aspect oriented design models in order to capture conflicts in an early stage before having it as runtime error. Detection is done in an approach not that complex as the previous ones.

Keywords – Aspect Oriented Programming, Databases, Interference Detection, Intertype Declaration, Relational Algebra.

I. INTRODUCTION

Aspect oriented programming – AOP – appears to enable software developers to address crosscutting concerns. If a behavior being developed is needed across a range of software, then it is named a crosscutting concern and implemented as an Aspect. For example authentication task that is needed in several modules in banking system that can be viewed as being developed horizontally over these classes [1] [2].

Crosscutting concerns introduce code scattering problem in conventional development environment, which occurs when a single functionality is spanned over several units, or it is just duplicated chunk of code [3] [4]. Code tangling represents another problem when two or more concerns are at the same class and dependent on each other and cannot be disassociated [3]. Therefore a development paradigm is needed to address those problems, which is aspect oriented programming [5].

AOP resolves the previously mentioned limitations by developing the crosscutting concerns as independent modules therefore the overall modularity of the developed software is enhanced [6].

AOP targets crosscutting concerns via concept of obliviousness that means programmer doesn't have to care about where the coded aspect will be used inside the code of the entire system [7]. Thus, completely independent modules – aspects – will be developed that increase the overall system productivity and reducing code complexity by resolving crosscutting concerns problem.

Aspects may interfere with each other because of weaving – injecting – code at the compile time into another code. Research done in [6] [8] [9] shows the interference among aspect and its causes. The targeted interference type here is that one resulted from introducing new members to base classes or other aspects at the runtime. The work presented here detects introduction conflicts at design level, instead of being detected at compile time or even runtime as an error.

The rest of the paper is structured as the following: section two illustrated AOP interference problem. Section three demonstrated the related work in AOP interference detection problem. Section four explains the detection approach using relational database model and formal query language. Section five includes an example for the proposed solution. Finally, the conclusions and future work are in section six.

II. Aspect Interference

Aspects may interfere with each other or with their base classes in several forms. First, several aspects addressing the same join point may interfere together as there is no fixed rule for aspect execution order. This is called crosscutting specifications interference that is mainly caused by the usage of the wildcard operator (*) that matches any return value from a method, which causes accidental joinpoints.

Second, join points may change due to weaving of an aspect that may add new join points to the code or removes existing join points. Other aspects could be affected due to these changes. This is called aspect-aspect conflicts [6]. Third, aspect may change some variables those are required for other aspect behaviors. This may cause circular dependencies between aspect and base class; so it is called base-aspect conflicts [6].
Forth, interference occurs when aspects have introductions – declarations – that are contradicting with each other due to use of inter-type declaration or structural superimposition [9]. Such a conflict makes contradiction among concerns are required to be achieved, which gives the name concern-concern conflicts [6]. This can occur in several ways, an aspect introduces new members – variables or methods – to another aspect or class. If the aspect or class has a member with the same name as the introduced one then interference will occur.

III. Related Work

A. Code Level Detection

A graph-based model used in [10] to detect intertype declaration conflicts by converting the aspect oriented source code, written in Java and AspectJ, structure to a graph model and the intertype declarations, part of code written in AspectJ, to a graph transformation rules that is to be applied to the graph model. In this model each program element is mapped to a graph node with a unique label to this element. The element node has primarily two edges, the first one called isa to represent its type of the element, whether it is a class, aspect, method, interface, etc. The second edge called named that holds the actual name of the element. Transformation rules are applied as edges to this graph model and then conflicts can be detected. Figure 1 shows an example of graph representation to a sample program. Using this approach is getting complicated along with the program size. Thus, a simple program of 200 elements to be represented in graph adjacency matrix, complexity of O(N^2), will cost 40000 comparisons. What if 1000 elements or more?

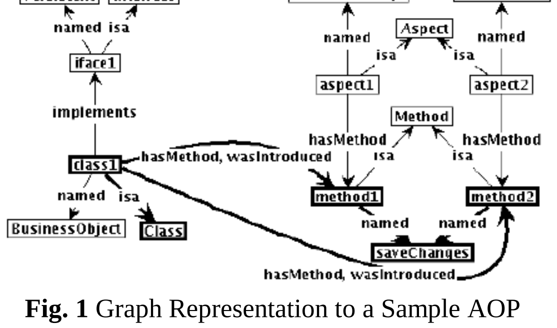


Fig. 1 Graph Representation to a Sample AOP Program with Intertype Declarations [10]

Program slicing is used in [11] to investigate the woven code – byte code – as it is dependent on AspectJ programming language. A slice is representing a module which can be an aspect or a class. Slices are investigated against intersection; if two slices intersects then they interact.

Unit testing is used in [12] [13] to detect interference in aspect oriented program. When aspect <A> requires a unit test that is added by aspect <B>, and vice versa between <B> and <A> it is kind of circular dependency between <A> and <B>. Conflict between aspects can be determined when an aspect suppress a unit test that is required for another aspect, for example an aspect <C> changes a field that is required for aspect <D>. Dependency conflict detected when aspect <E> issues a unit test that is required for aspect <F>. In other words, aspect <E> existence is required for aspect <F> functionality. Figure 2 shows those types of interference detection respectively. Pentagon represents a unit test, octagon represents an aspect, arrow represents <<issue>> direction, dashed circular-end line represents <<require>> direction, and the lightning bolt represents <<suppress>> direction. The main drawback of this approach is lacking of base/aspect interference detection.

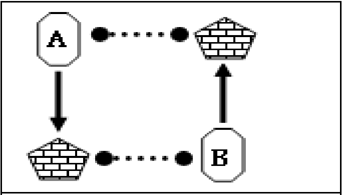
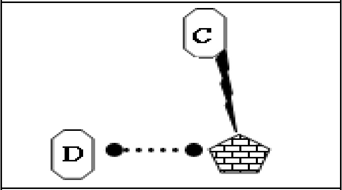
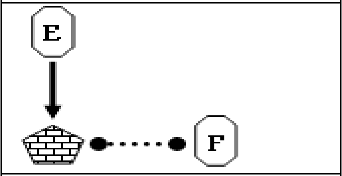

Circular Dependency

Conflict Interference

Dependency Interference

Fig. 2 Aspect Interference Detection using Unit Testing

B. Model Level Detection

Modeling aspect oriented software provides a good way to check software before actual coding phase. Theme/UML and UML extension mentioned in [14] and [15] respectively needs a formal way to transform them into a computerized form in order to facilitate the model checking process. In [16] a technique named MATA – Modeling Aspects using a Transformation Approach – was introduced to specify and compose aspects based on graph transformation. Graphs are then used in [17] to analyze aspect interference and in [18] to detect interference in UML-based aspect oriented models. Those techniques suffer from complexity of the algorithm being developed and limited project size.

IV. Intertype Declaration Interference Detection

Aspects and classes in aspect oriented modeling have relationships like those in object oriented modeling. Using the relational modeling technique, a database is resulted to resemble a UML-based aspect oriented model. Thus, for any creation to an aspect, class, or any of their members a record is added to the corresponding table. Figure 3 shows this database schema in order to understand the relational algebra written to extract intertype declaration interference.
In the following there are relational algebraic expressions to extract the intertype declaration interference types. First, aspect introduces a variable to another aspect has a variable with the same name. Second, aspect introduces a variable to a class has a variable with the same name. Third, aspect introduces a method to another aspect has a method with the same name. Forth, aspect introduces a method to a class has a method with the same name. Fifth, an aspect introduces a member – variable or method – to a base class or another aspect, while another aspect or more introduces members with the same name.

The last one is considered to be the trickiest one as due to the obliviousness nature of aspect oriented paradigm. For example, a software engineer may design an aspect (X) to handle a specific crosscutting concern with a member (M) introduced to another concern (A), at the same time anther software engineer may design an aspect (Y) that implements another crosscutting concern deals with concern (A) and introduces a member (M) to it. This type of error cannot be detected at design time and if those aspects are not compiled together it's a runtime error.

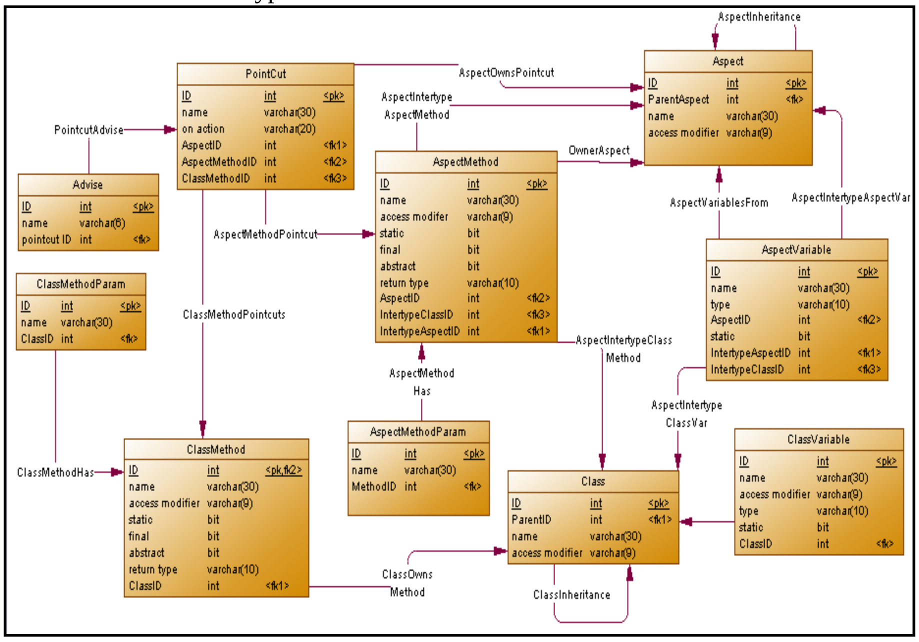


Fig. 3 Aspect Oriented UML-based Correspondent Database

a. Aspect/ Aspect Variable Interference

The concept behind the following is that a variable is intertype interfered if it's declared as intertype to another aspect and the affected aspect has a variable with the same name regardless to its data type. By performing the last join the relation type InterferingVariables contains those aspect variables interfering with other aspects' variables telling the causer and the affected aspects.

- $\mathbf{R_1}$ (OwnerAspectID, OwnerAspect) $\leftarrow \pi_{\text{ID, Name}}(\mathbf{Aspect})$
- $\mathbf{R_2}$(VarID, VarName, IntertypeAspectID, AspectID) $\leftarrow \pi_{\text{ID, Name, IntertypeAspectID, AspectID}}$ (**AspectVariable**)
- $\mathbf{R_3} \leftarrow \mathbf{R_1} \bowtie_{\text{OwnerAspectID=AspectID}} \mathbf{R_2}$
- $\mathbf{R_4} \leftarrow \sigma_{\text{IntertypeAspectID} \neq \text{Null}} (\mathbf{R_2})$
- $\mathbf{R_5} \leftarrow \mathbf{R_1} \bowtie_{\text{OwnerAspectID=IntertypeAspectID}} \mathbf{R_4}$
- $\mathbf{R_6} \leftarrow \mathbf{R_1} \bowtie_{\text{OwnerAspectID=AspectID}} \mathbf{R_5}$
- **InterferingVariables** $\leftarrow$
  $\mathbf{R_3} \bowtie_{\text{R3.OwnerAspectID = R6.IntertypeAspectID AND R3.VarName=R6.VarName}} \mathbf{R_6}$

b. Aspect/ Class Variable Interference

The concept behind the following is that an aspect variable is intertype interfered with class variable if it's declared as intertype to this class and the affected class has a variable with the same name regardless to its data type. By performing the last join the relation type InterferingVariables contains those aspect variables interfering with other classes' variables telling the causer aspect and the affected classes.

- $\mathbf{R_1}$ (OwnerAspectID, OwnerAspect) $\leftarrow \pi_{\text{ID, Name}}(\mathbf{Aspect})$
- $\mathbf{R_2}$(AsVarID, AsVar, AspectID, IntertypeClassID) $\leftarrow \pi_{\text{ID, Name, AspectID, IntertypeClassID}}(\mathbf{AspectVariable})$
- $\mathbf{R_3} \leftarrow \sigma_{\text{IntertypeClassID} \neq \text{Null}}(\mathbf{R_2})$
- $\mathbf{R_4} \leftarrow \mathbf{R_1} \bowtie_{\text{OwnerAspectID=AspectID}} \mathbf{R_3}$
- $\mathbf{R_5}$(CVarID, CVar,OwnerClassID) $\leftarrow \pi_{\text{ID, Name, ClassID}}(\mathbf{ClassVariable})$
- $\mathbf{R_6}$ (CID, OwnerClass) $\leftarrow \pi_{\text{ID, Name}}(\mathbf{Class})$
- $\mathbf{R_7} \leftarrow \mathbf{R_5} \bowtie_{\text{OwnerClassID=CID}} \mathbf{R_6}$
- **InterferingVariables** $\leftarrow \mathbf{R_4} \bowtie_{\text{AsVar= CVar AND IntertypeClassID=CID}} \mathbf{R_7}$

c. Aspect/ Aspect Method Interference

The concept behind the following is that an aspect method that is declared as intertype one to another aspect interferes if this aspect has another method with the same name regardless to the return type or the parameters. By performing the last join the relation type InterferingMethods contains those aspect methods interfering with other aspects' methods telling the causer aspect and the affected aspects.

- $\mathbf{R_1}$ (OriginalAspectID, OriginalAspect) $\leftarrow \pi_{\text{ID, Name}}(\mathbf{Aspect})$
- $\mathbf{R_2}$ (MethodID, Method, AspectID, IntertypeAspectID) $\leftarrow \pi_{\text{ID, Name, AspectID, IntertypeAspectID}}(\mathbf{AspectMethod})$
- $\mathbf{R_3} \leftarrow \mathbf{R_1} \bowtie_{\text{OriginalAspectID=AspectID}} \mathbf{R_2}$
- $\mathbf{R_4} \leftarrow \sigma_{\text{IntertypeAspectID} \neq \text{Null}}(\mathbf{R_2})$
- $\mathbf{R_5} \leftarrow \mathbf{R_1} \bowtie_{\text{OriginalAspectID=AspectID}} \mathbf{R_4}$
- $\mathbf{R_6} \leftarrow \mathbf{R_1} \bowtie_{\text{OriginalAspectID=IntertypeAspectID}} \mathbf{R_5}$
- **InterferingMethods** $\leftarrow \mathbf{R_3} \bowtie_{\text{OriginalAspectID=IntertypeAspectID AND R3.Method = R6.Method}} \mathbf{R_6}$

d. Aspect/ Class Method Interference

The concept behind the following is that an aspect method is intertype interfered with class one if it's declared as intertype to this class and the affected class has another method with the same name regardless to its return data type or parameters. By performing the last join the relation type Interfering Methods contains those aspect methods interfering with other classes' methods telling the causer aspect and the affected classes.

- $\mathbf{R_1}$ (CID, CName) $\leftarrow \pi_{\text{ID, Name}}(\mathbf{Class})$
- $\mathbf{R_2}$ (CMID, CMethod, ClassID) $\leftarrow \pi_{\text{ID, Name, ClassID}}(\mathbf{ClassMethod})$
- $\mathbf{R_3} \leftarrow \mathbf{R_1} \bowtie_{\text{CID=ClassID}} \mathbf{R_2}$
- $\mathbf{R_4}$ (AsMethodID, AsMethod, IntertypeClassID, AspectID) $\leftarrow \pi_{\text{ID, Name, IntertypeClassID, AspectID}}(\mathbf{AspectMethod})$
- $\mathbf{R_5} \leftarrow \sigma_{\text{IntertypeClassID} \neq \text{Null}}(\mathbf{R_4})$
- $\mathbf{R_6} \leftarrow \mathbf{R_5} \bowtie_{\text{AspectID=ID}} \mathbf{Aspect}$
- **InterferingMethods** $\leftarrow \mathbf{R_3} \bowtie_{\text{CID=IntertypeClassID AND CMethod = AsMethod}} \mathbf{R_6}$

e. Aspect-Aspect/ Aspect-Class Member Interference

The concept behind the following is that this interference type occurs if intertype members from different aspects target the same concern either core or crosscutting and those members have the same name. the following expressions extracts the interference in case of a member is a variable.

- $\mathbf{R_1} \leftarrow \sigma_{\text{IntertypeClassID} \neq \text{Null}}(\mathbf{AspectVariable})$
- $\mathbf{R_2} \leftarrow \mathbf{R_1} \bowtie_{\text{AspectID=ID}} \mathbf{Aspect}$
- $\mathbf{R_3} \leftarrow \mathbf{R_2} \bowtie_{\text{IntertypeClassID=ID}} \mathbf{Class}$
- $\mathbf{R_4} \leftarrow \sigma\,(\mathbf{R3})$
- **Result** $\leftarrow \mathbf{R_4} \bowtie_{\mathbf{R4}\text{.AspectVariableName = }\mathbf{R3}\text{.AspectVariableName AND }\mathbf{R4}\text{.IntertypeClassID = }\mathbf{R3}\text{. IntertypeClassID AND }\mathbf{R4}\text{.AspectID <> }\mathbf{R3}\text{.AspectID}} \mathbf{R3}$

V. Case Study: Customer Account

The following case, shown in figure 3, demonstrates an example based on [19] for aspect oriented UML-based class diagram models a part of bank customer account management. First, there is a class represents three of the core concerns of any account: balance inquiry, deposits, and withdrawals. Second, there are two aspects represent the crosscutting concerns for any account transaction: Authentication and Authorization. In this model there are five typical interference of intertype declaration kind. Table 1 contains a snapshot of the actual data in the database design in figure 2 concordant with the sample UML-based aspect model in figure 4 is used to facilitate the queries detects the interference within this model.

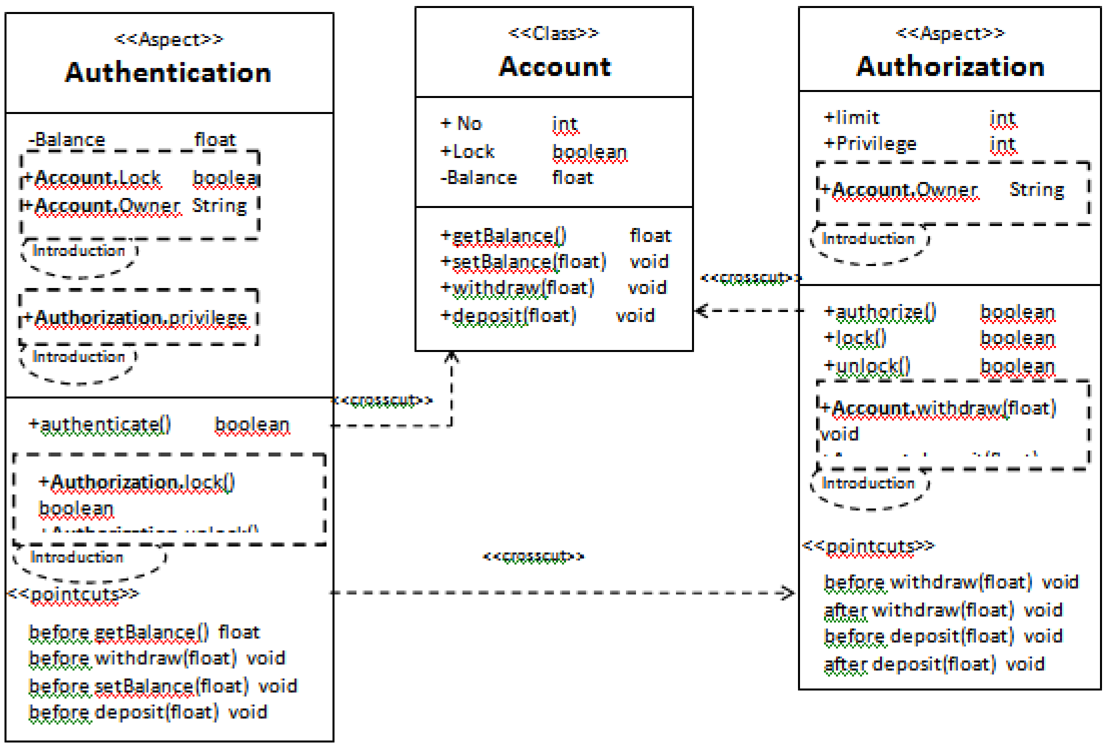


Fig. 4 UML-based Aspect Oriented Model

Class

ID	NAME	ACCESS MODIFIER	PARENTID
12	Account	public	null

ClassMethod

ID	NAME	ACCESS MODIFIER	STATIC	FINAL	ABSTRACT	RETURN TYPE	CLASS ID
16	getBalance	public	0	0	0	float	12
17	setBalance	public	0	0	0	void	12
18	withdraw	public	0	0	0	void	12
19	deposit	public	0	0	0	void	12

Class Method Param

ID	TYPE	METHOD ID
17	float	17
18	float	18
20	float	19

Class Variable

ID	NAME	TYPE	ACCESS MODIFIER	STATIC	CLASS ID
6	NO	int	public	No	12
7	Lock	boolean	public	No	12
8	Balance	float	private	No	12

Aspect

ID	NAME	ACCESS MODIFIER	PARENTASPECT
10	Authentication	public	null
11	Authorization	public	null

Aspect Variable

ID	NAME	TYPE	ACCESS MODIFIE	ASPECT ID	STATIC	INTERTYPE CLASSID	INTERTYPE ASPECTID

			R				
20	Balance	float	private	10	No	null	null
21	Lock	boolean	public	10	No	12	null
22	Owner	string	public	10	No	12	null
23	limit	int	public	11	No	null	null
24	privilege	int	public	11	No	null	null
25	Owner	string	public	11	No	12	null
26	privilege	int	public	10	No	null	11

Aspect Method

ID	NAME	ACCESS MODIFIER	STATIC	FINAL	ABSTRACT	RETURN TYPE	ASPECT ID	INTERTYPE CLASS ID	INTERTYPE ASPECT ID
7	authenticate	public	0	0	0	boolean	10	null	null
8	authorize	public	0	0	0	boolean	11	null	null
9	lock	public	0	0	0	boolean	11	null	null
10	unlock	public	0	0	0	boolean	11	null	null
11	withdraw	public	0	0	0	void	11	12	null
12	deposit	public	0	0	0	void	11	12	null
13	lock	public	0	0	0	boolean	10	null	11
14	unlock	public	0	0	0	boolean	10	null	11

Aspect Method Param

ID	TYPE	METHOD ID
12	float	11
13	float	12

Pointcut

ID	NAME	ON ACTION	ASPECTID	CLASS METHOD ID	ASPECT METHOD ID
5	pc1	call	10	16	null
6	pc2	call	10	17	null
7	pc3	call	10	18	null
8	pc4	call	10	19	null
9	pc1	call	11	18	null
10	pc2	call	11	19	null

PointcutAdvise

ID	NAME	POINTCUTID
4	Before	5
6	Before	6
8	Before	7
10	Before	8
11	Before	9
12	After	9
13	Before	10
14	After	10

Table 1. Aspect Oriented UML-based Model Corresspodant Data

VI. Results

By applying SQL queries based on the relational algebraic expressions mentioned in the previous section the following results shown in table 2 come out. First, Privilege variable is detected as an interfering member declared in Authentication aspect "OwnerAspect" and the aspect cause this interference is the Authorization aspect "IntertypeAspect" field. Second, Lock variable is detected as an interfering member declared in the Account class "OwnerClass" as a normal member and Authentication aspect has it as intertype member declared to Account class, and interfering with its member Lock.

Third, the two methods lock() and unlock() are detected as interfering methods between aspects as they are declared basically in the aspect Authentication "OriginalAspect", while they are declared in as intertype members to this aspect at Authorization aspect "IntertypeAspect". Forth, the methods withdraw() and deposit() declared in the aspect Authorization "Owner Aspect" are intertype members interfere with those original members at class Account and its methods withdraw and deposit. Fifth, the members that can be declared in two or more aspects and intertype declared to another unit class or aspect are detected such as the example of the variable owner that is declared within two aspects Authorization and Authentication "Aspect1 and Aspect 2" as intertype member to class Account "AffectedClass" that now has two members with the same name. This type of conflicts cannot be detected till runtime, and then crashes the software being run.

Aspect/Aspect Variable Interference

OWNER ASPECTID	**OWNER ASPECT**	**VARNAME**	**INTERTYPE ASPECT ID**	**INTERTYPE ASPECT**
10	Authentication	privilege	11	Authorization

Aspect/Class Variable Interference

AS VARID	**ASPECT VAR**	**OWNER ASPECT**	**INTERTYPE CLASS ID**	**CLASS ID**	**OWNER CLASS**
21	Lock	Authentication	12	12	Account

Aspect/Aspect Method Interference

ORIGINAL ASPECTID	**ORIGINAL ASPECT**	**INTER-MID**	**OR-MID**	**INTER METHOD NAME**	**ORIGINAL METHOD NAME**	**INTERTYPED ASPECT ID**	**INTERTYPED ASPECT**
10	Authentication	13	9	lock	lock	11	Authorization
10	Authentication	14	10	unlock	unlock	11	Authorization

Aspect/Class Method Interference

As Method ID	**As Method**	**Owner Aspect**	**Intertype ClassID**	**CID**	**Class**	**CMID**	**Class Method**
11	withdraw	Authorization	12	12	Account	18	withdraw
12	deposit	Authorization	12	12	Account	19	deposit

Aspect-Aspect/ Aspect-Class Member Interference

Aspect1 ID	**Aspect1**	**Aspect2 ID**	**Aspect2**	**Variable**	**Intertype ClassID**	**AffectedClass**
10	Authentication	11	Authorization	Owner	12	Account
11	Authorization	10	Authentication	Owner	12	Account

Table 2. Aspect Oriented UML-based Model Intertype Interferences Detected

VII. Conclusion and Future Work

Aspect interference problem is one of the most complicated problems in AOP. Despite of AOP importance but the probability of unexpected behavior of software being run and complexity of resolving this problem, many developers prefer OOP and avoiding AOP with all its advantages. Using relational model to represent the corresponding UML-based aspect model and relational algebra provided a simple way to detect one of the interference types – intertype declaration interference – at design time regardless to the size of the software to be designed. The work presented here can be done using XML and X-Query instead of traditional DBMS and SQL thus it can be used within CASE tools easily for local development or can be implemented over a database server for distributed teams targeting the same project.

References

[1] Berg, Klaas van den, Conejero, Jose Maria and Chitchyan, Ruzanna. *AOSD ontology 1.0 public ontology of aspect orientation.* s.l. : Common Foundation for AOSD, 2005. p. 90, Report.

[2] J.D. Meier, David Hill, Alex Homer, Jason Taylor, Prashant Bansode, Lonnie Wall, Rob Boucher Jr., Akshay Bogawat. *.NET Application Architecture Guide,.* 2nd Edtion. s.l. : Microsoft Corporation., 2009.

[3] Harbulot, Bruno. *SEPARATING CONCERNS IN SCIENTIFIC SOFTWARE USING ASPECT-ORIENTED PROGRAMMING.* Computer Science, Manchester University. Manchester : Center of Novel Computing, 2006. p. 194, PhD Thesis.

[4] Dessì, Massimiliano. *Spring 2.5 Aspect-Oriented Programming.* [ed.] Sneha Kulkarni. Olton : Packt Publishing Ltd., 2009. pp. 13-16. 978-1-847194-02-2.

[5] *Aspect-Oriented Programming.* Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes,. Finland : Springer-Verlag, 1997. European Conference on Object-Oriented Programming (ECOOP). pp. 220-242.

[6] *Aspects: Conflicts and Interferences (A Survey).* André Restivo, Ademar Aguiar. 2007. Actas da 2^a Conferência de Metodologias de Investigação Científica. pp. 145-153.

[7] Robert E. Filman, Daniel P. Friedman. *Aspect-Oriented Programming is Quantification and Obliviousness.* Research Inistitution for Advanced Computer Science. s.l. : Workshop on Advanced Separation of Concerns, 2001.

[8] Katz, Emillia, et al. *Detecting Interference Among Aspects.* Computing Department, Lancaster University. Lancaster : European Network of Excellence on Aspect-Oriented Software Development, 2007. p. 38, Report.

[9] Durr, Pascal, Bergmans, Lodewijik and Aksit, Mehmet. *Reasoning about Behavioral Conflicts between Aspects.* Enschede : University of Twente, 2007. Technical Report . TR-CTIT-07-15.

[10] *A Graph-based Approach to Modeling and Detecting Composition Conflicts Related to Introductions.* Havinga, Wilke, et al. Vancouver : ACM International Conference Proceedings Series, 2007. International Conference on Aspect Oriented Software Development, AOSD 2007. pp. 85-95. 1-59593-615-7.

[11] Balzarotti, Davide and Monga, Mattia. Using Program Slicing to Analyze Aspect Oriented Composition. 2004.

[12] *Towards detecting and solving aspect conflicts and interferences using unit tests.* Restivo, André and Aguiar, Ademar. Vancouver, British Columbia, Canada : ACM, 2007. Proceedings of the 5th workshop on Software engineering properties of languages and aspect technologies.

[13] Team, MSDN. Unit Testing. *MSDN.* [Online] Microsoft, 2012. [Cited: 1 21, 2013.] http://msdn.microsoft.com/en-us/library/aa292197%28v=vs.71%29.aspx.

[14] Clarke, Siobhán and Baniassad, Elisa. *Aspect-Oriented Analysis and Design: The Theme Approach.* Crawfordsville : Addison Wesley Professional, 2005. 0-321-24674-8.

[15] *A UML Extension to Graphically Represent Aspect Oriented Systems Perspectives.* Ferreira, Helivelton Oliveira and Dias, Luiz Alberto Vieira. Las Vegas : IEEE, 2010. Seventh International Conference on Information Technology. pp. 1312-1313.

[16] Whittle, Jon and Jayaraman, Praveen. MATA: A Tool for Aspect-Oriented Modeling Based on. Graph Transformation. [ed.] Holger Giese. *Models in Software Engineering.* Berlin : Springer Berlin Heidelberg, 2008, Vol. 5002, pp. 16-27.

[17] Staijen, Tom and Rensink, Arend. *A graph-transformation-based semantics for analysing aspect.* Natal, Brazil : Workshop on Graph Computation Models,, 2006.

[18] Ciraci, Selim, et al. A graph-based aspect interference detection approach for UML-based aspect-oriented models. [ed.] Shmuel Katz and Mira Mezini. *Transactions on aspect-oriented software development.* Heidelberg : Springer-Verlag, 2010, Vol. VII, pp. 321- 374.

[19] Stein, Dominik. *An Aspect-Oriented Design Model Based on AspectJ and UML.* Management Information Systems. Alfter, Germany : University of Essen, 2002. p. 203, MSc Thesis.