Aspect Oriented Programming Methodology to Support the Design of Specific Domain Framework

Xavier Medianero[1], Sérgio Crespo C.S. Pinto[2] and Clifton Clunie[3]

**[1,3] Technological University of Panama,
Panama City, Panama**

**[2] Universidad do Vale Do Rio dos Sinos
São Leopoldo, Brazil**

Abstract

The aspect-oriented programming has valuable advantages over other programming paradigms, but in turn it presents difficulties when applying the concepts within the stages of analysis and development to reduce the drawbacks of this paradigm. This paper proposes a methodology to reduce the drawbacks of the paradigm, at the same time provides steps that involve elements of common analysis in the Requirements Engineering with Aspects (basic unit of paradigm) in order to create the framework for a specific domain. The proposed methodology brings together some benefits methodologies, but it emphasizes the treatment of the first disadvantages of the programming aspects and the location and identification of aspects and elements; in addition, this article provides a tool that supports some methodology steps by generating part of the framework code base. In the process of treatment issues, the analysis is oriented to the specification of aspects using AspectJ, with rules to locate and determine aspects within its four cyclical stages. Finally, it includes a case study which evaluates the steps in this methodology

Keywords: *Aspect Oriented Programming, Methodology, Software Engineering, Requirements Engineering.*

1. Introduction

The Aspect Oriented Programming (AOP) is a paradigm that provides a high level of benefits in the process of development and maintenance of Software Engineering. It promotes and encourages the separation of business concepts in cross-sectional characteristics providing advantages over other modern paradigms [1].

The purpose of the AOP is the separation of functionality with cross sections in blocks called "Aspects" [2]. Aspects are units of abstraction and composition which collect instructions that are difficult to encapsulate because of its presence in different functions [3]. Aspects are not identifiable and independent units such as classes, as a matter of fact; they are abstract elements that generally provide added features and functionality to other elements due to their spread by the same logic.

The aspects encapsulate the crosscutting features, as well as classes and methods that only have information relevant to its functionality, making these tasks more efficient and easier; likewise, they capture external features and procedures independent of the method allowing an easier and more complete analysis. Tangled and dispersed code will be reduced because unnecessary code will not be present at all functions, but only in aspects.

The AOP paradigm also has some inconveniences that hinder its use; these disadvantages are minor glitches due to transversal crosscutting nature and behavior in the base language. Among the disadvantages [4] we have:

- Conflict between aspects.
- Conflict between the base language and aspects
- Complexity in Aspect-Oriented Analysis.

The methodology emerges as a solution to some problems related to last point, this is split up in structural conflicts [5, 6], behavioral [5, 7] and dynamic characteristics [8], which will be treated in this paper.

Software engineering is not only for the end product design software, but also for the generation of domain-specific framework containing the essential elements for a macro product. The framework are oriented application generators to a specific domain [9], these allow the creation of structured software based on features that are present in the core and hot spots.

The proposed methodology is based on the way AspectJ [3] handles and determines the components of the aspects, while integrating characteristics of Aspect Oriented Requirements Engineering (AORE) [10, 11], Aspect Oriented based on Component Requirements Engineering (AOCRE) [12], Structured Lexicon to Aspect Identification (SLAI) [13] Methodologies and View Point

Model [14]. This methodology creates structure of analysis to minimize the AOP problems found at conceptual model [5] and Join Point model [7] frameworks, via AspectJ.

This paper is divided into the following sections: related works in Section II, AOP conflicts in section III, the proposed methodology and support tool in Section IV and V respectively. And finally an evaluation of both in a case study in Section VI.

2. Related Works

There is a need for a systematic methodology for the analysis of the aspects in the early phases of engineering process [13] due to emerging technologies, the time of appearance of the aspects and their integration into Software Engineering. In addition, existing methodologies focus on attacking a particular problem but not in a generalized way.

2.1. AORE Methodology

The AORE methodology is based on the classification of cross functional concerns in aspects from the functional requirements (RF) and nonfunctional ones (RNF) provided by users [10]. Most models are developed for AORE approach based on Theme / Doc [15].

The AORE methodology uses a procedure based on the treatment of issues as a coherent set of requirements, wherein the cross-cutting aspects are obtained from recurrence matrices created by the influences of the issues, indicating how many and which elements work in a positive or negative way on others [16]. Said methodology uses the rule of decomposition and it allows you to draw a projection of dependencies.

2.2. AOCRE Methodology

The AOCRE methodology separates the functional and nonfunctional requirements of a system by relating the keys (aspects) with the supplied components or the missing system [12].

This methodology is based on the specialization process (decomposition of aspects) in smaller aspects, maintaining the integrity of the components. One aspect that controls the issue of user interface can be decomposed into views, quality over user actions, feedback mechanism, scalability and extensibility [12].

2.3. SLAI

The SLAI Methodology (Structured Lexicon for Identifying Aspects) is based on the identification of potential aspects in the design phase; these aspects are identified and specified in conjunction with the treatment of functional and non functional requirements of software [13]. SLAI works with segmentation and replication of use cases by using lexical identification.

2.4. View Point Model

The Point of View Model is the integration of approach to views and aspects, getting a more solid structure for the management of requirements [14]. The views are created from the possible scenarios and system features, solving conflicts that present themselves in the detailed requirements.

2.5 Others Models

Some models can minimize the disadvantages of the AOP. The Theme/Docs Model uses orientation to topics; this separates system requirements according to the themes that represent issues of concern. This model is based on lexical analysis procedures for the separation in matters under the concept of the AOP [11]. The Use Case model to non-functional requirements using use cases that represent the smallest unit of the system, while non-functional requirements which are seen as infrastructure use cases that analyze the behavior and identify the crossing points of base use case [17] and the viewpoints model based on the separation of interest using a multi-view approach by rules of decomposition, definition and conflict management [17].

3. Aspects Oriented Programming Complications

3.1. Structural Conflicts:

This inconvenience is the difficulty of knowing the objectives, components and characteristics of aspects after prolonged maintenance periods [5, 6]. This conflict should not be confused with one of the advantages of the AOP, which facilitates the maintenance by eliminating the scattered code. These two features are different because the structural conflict lies in the semantics loss of information about functionality while maintaining orderly and understandable code, which is the advantage of the AOP on OOP, because in the end, the code loses order and becomes dispersed and tangled, making it difficult to understand.

3.2. Behavior Conflicts

The difficulty lies in the natural complexity of the aspects concerning the correct and logical location of its elements in the system [5, 7], since due to its characteristic of transversality, it is easy to fall into the error of location, causing ambiguities at the time of execution, therefore causing the aspect to perform incorrectly, which will eventually lead to conflict within the framework.

3.3. Multifunctional Conflicts

This problem comes about because of the nature of the AOP. As the AOP intends encapsulation concerns (features) in aspects, this task is blurred if this issue is multifunctional [3, 8], i.e. if a case presents several targets and is used in completely different processes, the task of encapsulation is made more difficult due to the lack of rules for deciding in which aspect this functionality is located. In addition, this conflict refers to the problem in reference [4] which is about grouping of aspects in cross cutting section.

3.4. Dynamic Facilities:

One advantage of the AOP is to dispense, activate or modify aspects at runtime [8], in order to change features without having to shut down, which is very beneficial, but the task of deciding what will be considered as dynamic aspects and measuring the impact of same is a complex task.

4. Methodology: MEDFOAR

The proposed methodology will be called "MEDFOAR" (Aspect Oriented Methodology to Design Frameworks in Requirements) and it proposes four stages for the identification and treatment of aspects. This process is cyclical to avoid redundancy while aspects are identified and specified (however, in some cases the results can be obtained in a single iteration). These four phases are:

4.1. Detail of Requirements

The first stage consists of identifying the elements that has the framework; these elements include actors or entities involved in some way with the software.

4.1.1 Approach to Views

At this stage approach is used to view orientation by objectives, depending on the functional requirements and stakeholders of the system, the views are defined based on the scenarios of the framework.

The elements of the schemes aimed at determining views at this stage are as follows: Name of the view (it is an identifier naming schema), Stakeholder (the entities that interact with the system within the selected view, semi-automatic processes) Associated functions (functions that appear in the view), and influences.

4.1.2 Use Case Development

The use case diagram of the system involves actors in conjunction with a flow of activities performed to achieve a given process.

4.1.3 Accounting of determining identifiers of functionalities

Process: Identifiers (significant nouns and verbs in the name of the use cases) are stored in a repository, so it is possible to determine the frequency of each within the procedure. Also included is an influence of IDs on the cardinality of the features and views, so it can be associated with few cross-cutting which are also possible to make a case

The SLAI methodology specifies use cases for cross-relating each one with its influences, obtaining identifiers of phrases, verbs and phrases needed to be segmented by replicating the different use cases through the diagrams the identification of aspects [13]; MEDFOAR uses identifiers in order to account for the presence of functionality throughout the system.

4.1.4 Identification of multifunctional modules

Through the needs and software requirements, it is possible to determine the modules to be multifunctional, in other words, the ones that are used in several procedures.
For this task, the methodology uses the use case diagrams, scenarios and views. A module is said to be multifunctional when it is present in a use case and it is employing a function which is already absorbed in another. This methodology step is a response to the difficulty in treating of abstracting functional modules within a particular concern. At this stage, identify functional modules and low abstraction segment meet.

The elements obtained after the application of this stage are used case diagrams, charts, views, actors, lexical identifiers database and multi-prone modules.

4.2. Aspects Identification

At this stage, candidate aspects are determinate by using the last elements of the previous stage together with the application of these rules.

4.2.1 Identification of influences and dependencies

It uses the use case modeling because these diagrams are expressed in the functions to be performed before and after the addition to the requirements and rules for certain processes. The relationship observed in use case diagrams and their meaning can be observed in Table 1.

Table 1 Dependencies through the use case

Link	***Induces***
A → B	Basic influence of A on B
A extends B	Probable influence of A on B
A includes B	Forced influence of B on A
A,B specialize C	A, B apply same as C

Justification: using the use case diagrams, MEDFOAR induces influences, dependencies and cross cutting of some features, so that there is an analytic view of the interrelationships of the different software modules.

4.2.2 Application of the Rule of Decomposition of Functionalities

This phase is based on AORE; nevertheless it differs because AORE treats the elements of relationships as crosscutting issues and analyzes their influences by identifying the type of aspects, while MEDFOAR applies the rule for functions instead of issues, and it analyzes their influences and dependencies by focusing on location and identification of aspects and their elements.

The rule of decomposition is a process that identifies potential candidate aspects through a system of influences and dependencies of functionalities [10]. This rule assigns a value to each feature, which depends on the number of influences and dependencies present. When one has a large number of influences or dependencies, should designate an aspect that cuts this function and the associated class.

Furthermore, if a functionality associated with an aspect has an <include> dependency that only affects this function, it is determined that the influential function will also have an associated aspect.

4.2.3 Aspects through non-functional requirements

Using a list of the most common non-functional requirements determine the most applicable to the system depending on their adaptability ones. This procedure includes the identification of NFR (Non-functional requirements) tenders obtained from the users of the system.

Recalling that some NFR are scattered in different areas of the framework, the methodology proposes to take into consideration the requirements like: availability of service, security, system performance, response time, reliability of processes, performance, multi-user capability, legal cases and adaptability to the network.

Justification: non-functional requirements should be viewed as elements of analysis within the project because their presence can change the focus of the system.

4.2.4 Selecting Candidate Aspects

Candidate aspects will be obtained by the union / interception of the following sets. A set consists of the elements most frequently achieved in the selection process IDs Determinants. The other set consists of those obtained by the decomposition rule.

4.2.5 Selection of Classic and Dynamic Aspects

The candidates are selected aspects of the binding of cross-functional set. Then, there are measures to determine when an aspect can be considered dynamic or not; consequently that does not involve the stability or its performance.

a. If the aspect candidate has any influence of small branching.
In determining the functionality within the use case diagrams, is possible to find an aspect whose intercepted functionality has few influences; any aspects satisfying this condition can be treated as dynamic.

b. If the aspect candidate has any influence of long branching
If the branches are long, it is necessary to evaluate the possibility that if cutting or modifying any functionality at runtime in turn adversely affects some important functionality or if the design is very difficult to evaluate, if so then the aspects intercepting these functions cannot be considered dynamic. However, this decision is under review and impact as designed.

4.2.6 Integration Candidate Aspects:

In this section, candidate aspects are unified depending on their group within the group of lexical determiners. Identified aspects that are under the same set of identifiers are joined, forming a new aspect that intersects the union of all classes and functions that intercepted the previous ones.

4.2.7 Application of Metadata

Identifiers are established semantic aspects, classes and framework components through a layer of metadata. The information stored is as follows: Item Name (ID), element type (aspects, class, method, etc.), functions to which it is associated and its relevance within the flow and functionality description element.

As a result at this stage, we have: decomposition matrix, candidate aspects, dynamic aspects and metadata.

4.3. Candidate Aspects Specification

At this stage, we analyze the components of the aspects by specifying and locating them within the framework of analysis while controlling redundancy.

4.3.1. Aspect's Range Control

At this stage we determine the range to every aspect candidate by specifying dependencies and influences of the aspect. Unlike the procedure of the decomposition rule, it specifies when they run and whether there are any conditions for execution by determining access points as cross-cutting functions and the actions that are executed before and after. This point refers to the cut points and advice.

Justification: this step attacks difficulty that exists when setting matters in determining cross-section, after the whole process of global analysis on the aspect, each one is associated with macro functionality.

4.3.2. Treatments of Elements of Aspects

At this stage, the methodology identifies and analyzes the cut points (CP), join points (JP) and advices (AP). The instructions (IS) will not be evaluated with rules at any stage because the operations are not predictable as other features, but you can refer to classes or objects. The analysis of each element is done through the following rules:

Cross-cuttings which are displayed on the direct relationship between the issues and use cases that intersect, i.e. depending on the functionality of the use case, they are conceptualized within a transversal feature, it represents the transversal functionality in each one of the use cases associated with the element that generates it.

Cut-Points are elements which allow access to the aspect at a certain code stage. As it directly affects classes, objects and methods, the location may be submitted within the calls or execution of methods, constructors, initializing objects, assigning an attribute and many other circumstances. The methodology establishes points of location to the use cases that encapsulate each of these elements, referring to the name of the objects involved.

Join-Points which are elements associated with a particular crosscutting. Join-Points represent the grouping of cut points, which is related to the class of AspectJ, their existence lies in the possibility of grouping many links to a cut; therefore its benefit is present in multiple accesses to a function. In this methodology, the JP will be located within the Use Case diagram, only if global access is required for all the cross sections.

Advices are elements which should be placed in the use case to which the aspect intercepts. The advice will be placed next to parent aspects so as to be recognized according dependences and influences of the functionality that cuts the associated CP. This representation includes time: after (), before () and around ().

The symbols used were obtained from the work of Losavio et al [18].

4.3.3. Redundancy Control

Due to the process specification of aspects, these can be found with very similar functions and elements, resulting in redundancy of definitions, which must be analyzed. Aspects with more than 85% similarity must be unified in one aspect; the comparison criterion is based on the classes, methods that intercepts, and the similarity of its elements (cuts and advice). The candidate aspects can be converted into one or maintained separately.

4.3.4. Component Integration

As applied in AOCRE and the process of the analyzing of Software Engineering, aspects and associated classes are integrated into components. This task should define if they are associated under the same cross-section and if the elements have a strong relationship when running the routines of framework.

4.3.5. Metadata of Sub Elements

This process applies metadata to integrate the elements of the aspects which are CP, JP, advice, instruction and cross-cutting.

Justification: this step reduces the lack of semantic knowledge of each aspect element when the system tends to be very large or after many changed processes, undermining the structural conflict.

The results of this phase are the following elements: specification of the aspects, metadata aspects' elements, location of the elements of the aspects, aspects specialization (reducing duplication) and detailed aspects.

4.4 Aspects in Conflicts

4.4.1 Cataloging Aspects

This stage is similar to AORE in execution, but it differs in the objective; AORE classifies concern in aspects, decisions and functions, as its analysis elements are concerns, proposal from methodology classify candidate aspects separately of the functions and decisions; this process applies only to a certain group of issues.

This step is only applied when the aspects identified have weak persistence, i.e. they may not be real aspects. It is applied if the aspects identified lack cut-points, join-points and / or advice, which is caused when an aspect is only obtained through the method of Lexical Identifiers.

4.4.2 Weighing Aspects

At this stage, the aspects are weighted depending on several factors. This process is conducted 1 to 1. The selection of the actors in the conflict is displayed according to the scheme of the system and the point of view of stakeholders.

The evaluation in Table 2 gives the range of 0 to 7 for the Stakeholders and Users (if applicable), where 7 is the highest possible value for each item intercepted. For every influence that gets an intercepted function, there is 1 of importance and there are 3 for each class intercepted. This evaluation is given by the relevance of each element within the design. The aspect with more value will be called the impact between the two selected.

Table 2 Weighing Aspects by interceptions

Element to evaluate	Value	Justification
User / Stakeholders	0 – 7	The Influence evaluated by users and stakeholders about an aspect. This value applies to each element that intercepts: Aspects, Class or Functions.
Intercepted Feature	1x	The functions influenced by the methods the aspects intercepted.
Intercepted Classes	3x	Intercepts Classes
	1x	By each Aspect's element that intercept a class.

5. Support Tool

There is a support tool for this methodology. It is made in Java by using the Eclipse platform in conjunction with AspectJ.

The tool works in three (3) phases:
Phase 1: Following the stage 2.1 Identification of influence and dependency, the information is stored from the form of the tool that contains all the functionality of the system.

Phase 2: The relations between elements are added in the model of the tool, so that takes influences and dependencies. Once this task is performed, the tool by provides candidate aspects cutting associated classes and functionality according to the rules of the methodology (cut-points, join-points and advice are created in this process). The tool checks the total number of influences and dependencies; using this number determines the functions that are crosscutting, and creating an aspect with the same name; then it creates its elements linked to the features to corresponding according the structure of AspectJ.

Phase 3: In addition, the tool generates a graph that displays the list of aspects, classes and features, conducts resolution of conflicts, controls redundancy and generates the case of the system logically added.

Image 1 show relationship between Methodology's Phases, Support Tool's functions and Final Products, Image 2 shows a screen with the options in this phase.

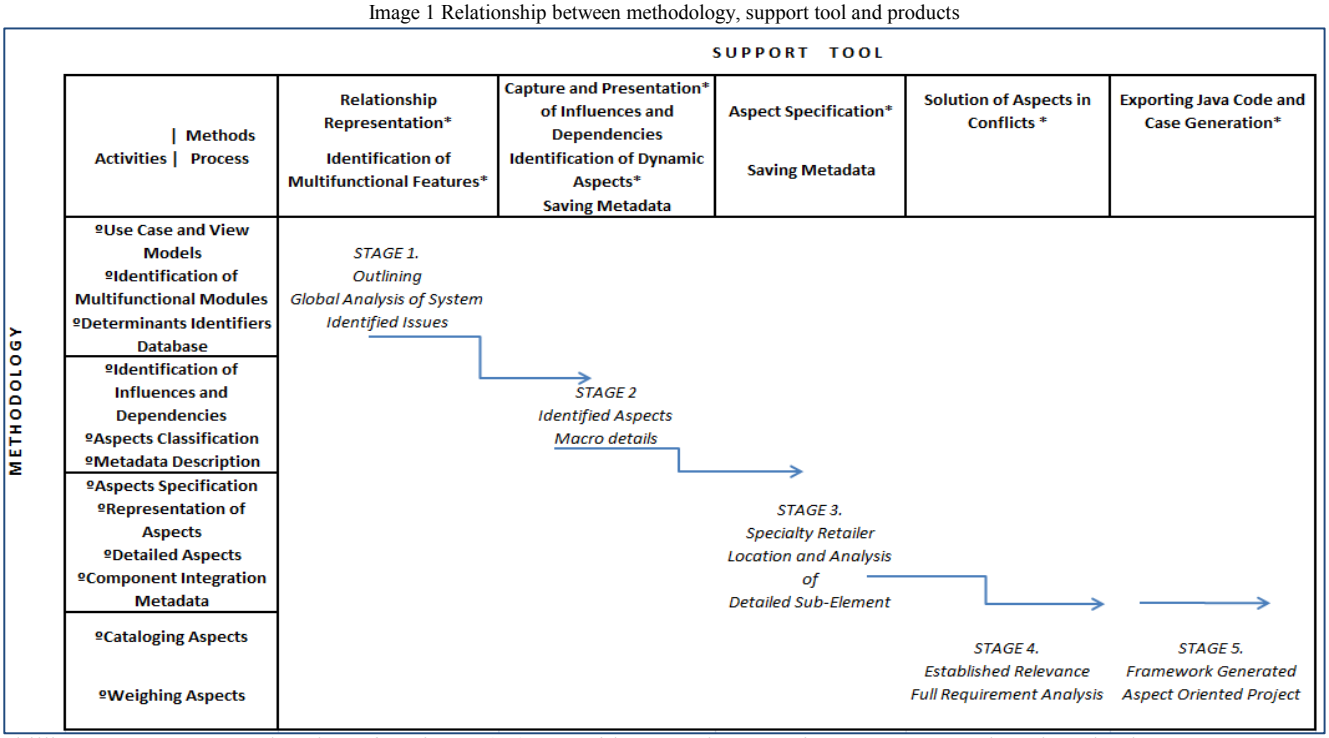
6. Case Study

The methodology was applied in a case study based on an ATM machine (Terminal Service for Banks) [14] which needs a software to manage the hardware and user support, likewise it communicates with the bank's database.

A software is required for an ATM machine that allows the performance of the following operations: (1) Accepts the client requests, (2) Allow cash withdrawals, (3) Provide account information, (4) Allow balance transfer (5) Provide recognition of bank users and foreign users, (6) Provide availability by 24 hours a day.

Additional extras operations were created for the full implementation of this methodology: (7) Allow the purchase of Phone Cards and Transportation Tickets, (8) Allow the payment of utilities, for example, water bills, telephone, (9) Allow the user to print transaction proofs,

Image 1 Relationship between methodology, support tool and products



billing statements, and other functions (10) Provide support for display of bank movements of account, (11) Allow retrieving information from user accounts and passwords, (12) Update bank booking.

The operations 9, 10, 12 apply only to bank customer. Image 3 show the use case diagram related to the ATM machine. It is one of the first tasks (Stage 1.2).

Image 2 View of Support Tool (1 Elements, 2 Methodology steps 3. Relationship and redundancy controls)

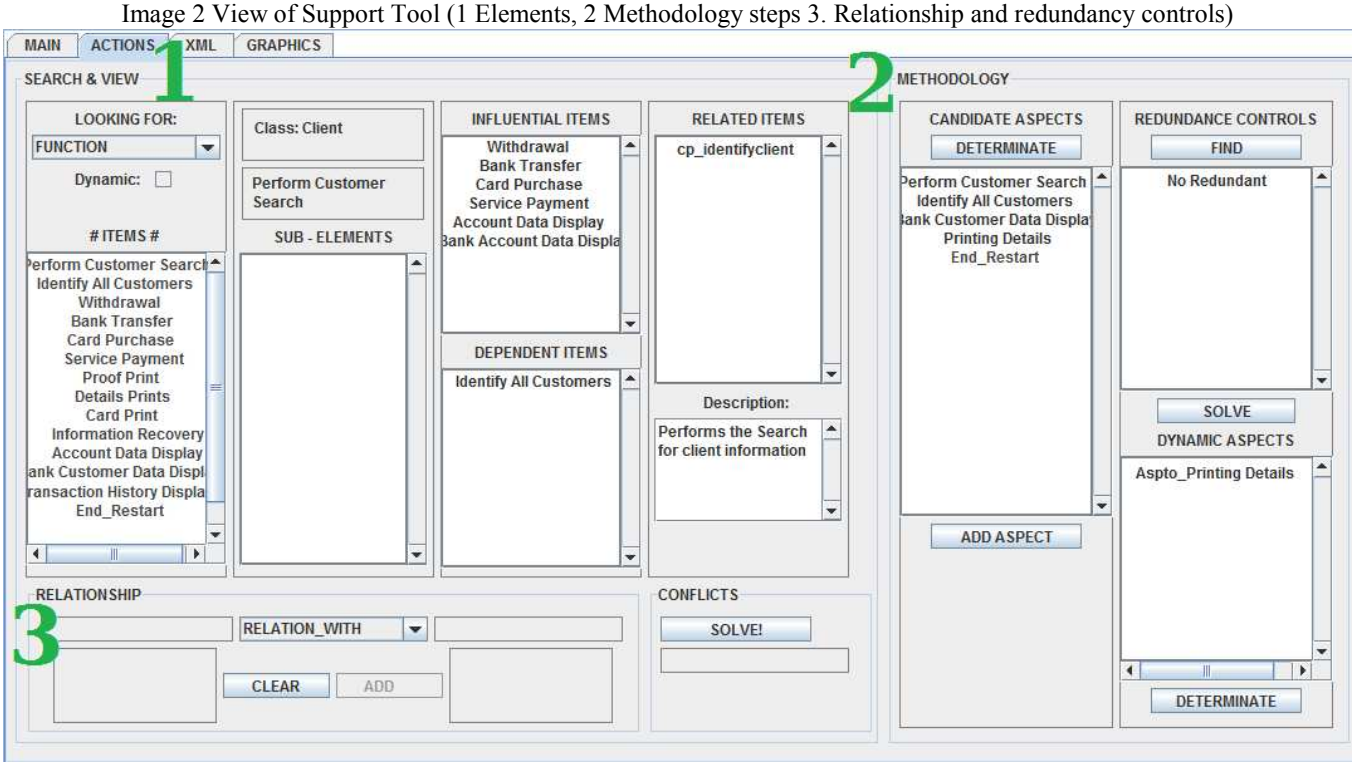
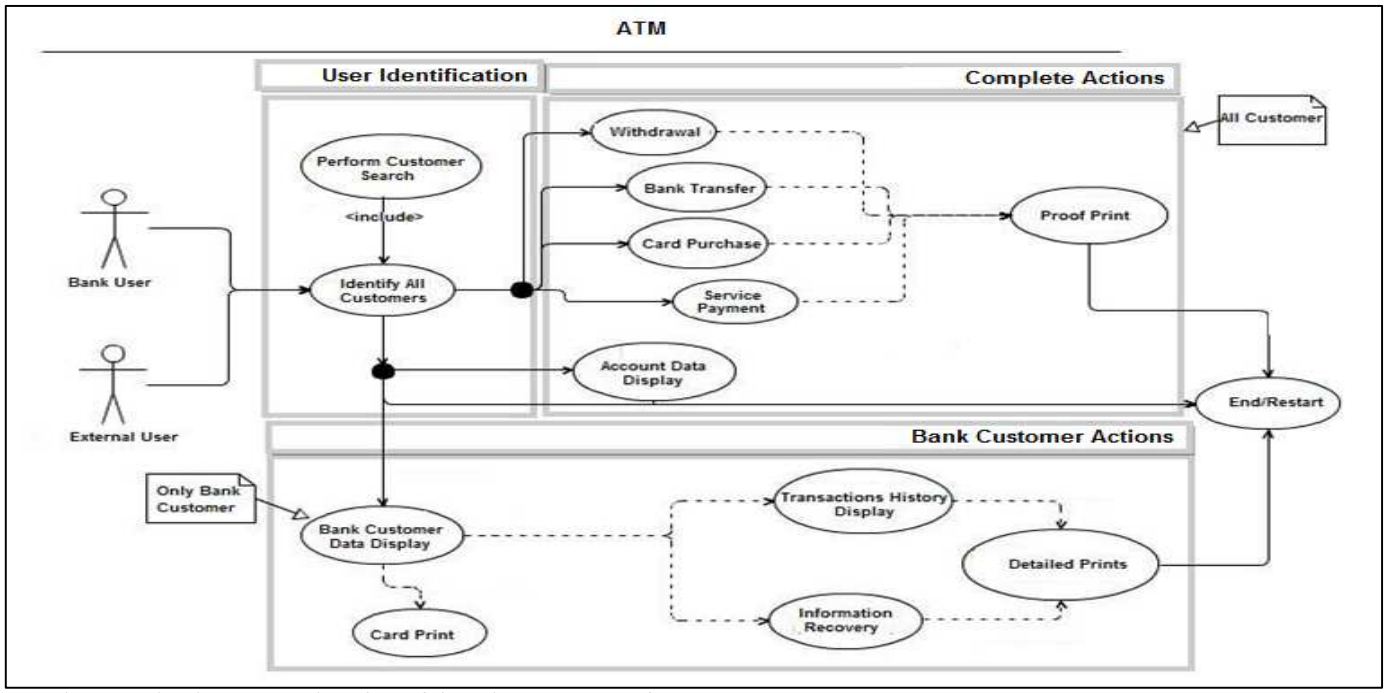


Image 3 Used case diagram.



Subsequently, there an explanation of the relevant stages of the process:

Stage 1.3: Accounting of determining identifiers of functionalities. The repository of feature lexical identifiers is created by using use case diagram. That is shown in Table 3.

Stage 2.1: Identification of Influences and Dependencies. The array of influences and dependencies of this system is presented in Table 4.

Stage 2.2: Application of the Rule of Decomposition of Functionalities. The features with larger number of influences and dependencies are: Identify All Customer (6 influences), Bank Customer Data Display (3 influences), Proof Prints (4 units), Detailed Prints (2 units, few dependencies), and End/Restart (3 dependencies). They became candidate aspects.

Applying the included clause (direct relation by <include>) Perform Customer Search became candidate aspect too.

Stage 2.4 Selecting Candidate Aspects
Combining the two processes of identification, we have to the aspect crossing "Details Impression" with low dependency but it becomes part of the group of aspects for its high number of repetitions in the lexicon identifier repository.

Hence, candidate aspects in this system are as follow: (1) Aspect cutting Identify All Customers function, (2) Aspect to Perform Customer Search, (3) Aspect cutting Bank Customer Data Display, (4) Aspect cutting Detailed Prints, (5) Aspect to Proof Print, (6) Aspect cutting End / Restart function.

Table 3 Lexicon Identifiers Repository

Lexical Identifiers	Retrieved from	Freq
Search	Perform Customer Search	01
Identify	Identify All Customers	01
Withdrawal	Withdrawal	01
Transfers	Bank Transfer	01
Purchase	Card Purchase	01
	Service Payment	01
Print (Impression)	Proof Print	
	Detailed Prints	03
	Card Print	
Recover	Information Recovery	01
Display	Account Data Display	03
	Bank Customer Data Display	
	Transactions History Display	

Stage 2.6 Integration Candidate Aspects
Two of the candidate aspects are associated under the name "Print" According to Table 3. Therefore, applying the methodology joins those on a single aspect intercepting all functions individually crossed. This union will be called "Aspect Printing Details Plus"

Stage 3.2 Treatments of Elements of Aspects
Elements were located within the designed methodology; their representation within the use case diagram is shown in Image 4. These obtain the following result:

Aspect to Identify All Customers
Cross-Cutting: ATM System – User Identification.
1Cut-Point: it is located by intercepting the Identify All Customer function.
Join-Point: it is the point of union of all elements that refers to the intercepted class. It is linked to the cut-point.
Advice: that is linked to the cut-point, due to the presence of large number of influences, its intervention is after ().

Table 4 Matrix of Influences and Dependencies

FUNCTIONS	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1 Identify Customer	*		→	→	→	→		E	E					
2 Perform Search Customer	I	*												
3 Withdrawal			*				E							X
4 Bank Transfer				*			E							X
5 Card Purchase					*		E							X
6 Service Payment						*	E							X
7 Proof Print							*							→
8 Account Data Display								*						→
9 Own Customer Data Display									*	E	E	E		
10 Card Print										*				X
11 Information Recovery											*		E	X
12 Account Movements												*	E	X
13 Details Print													*	→
14 End / Restart														*

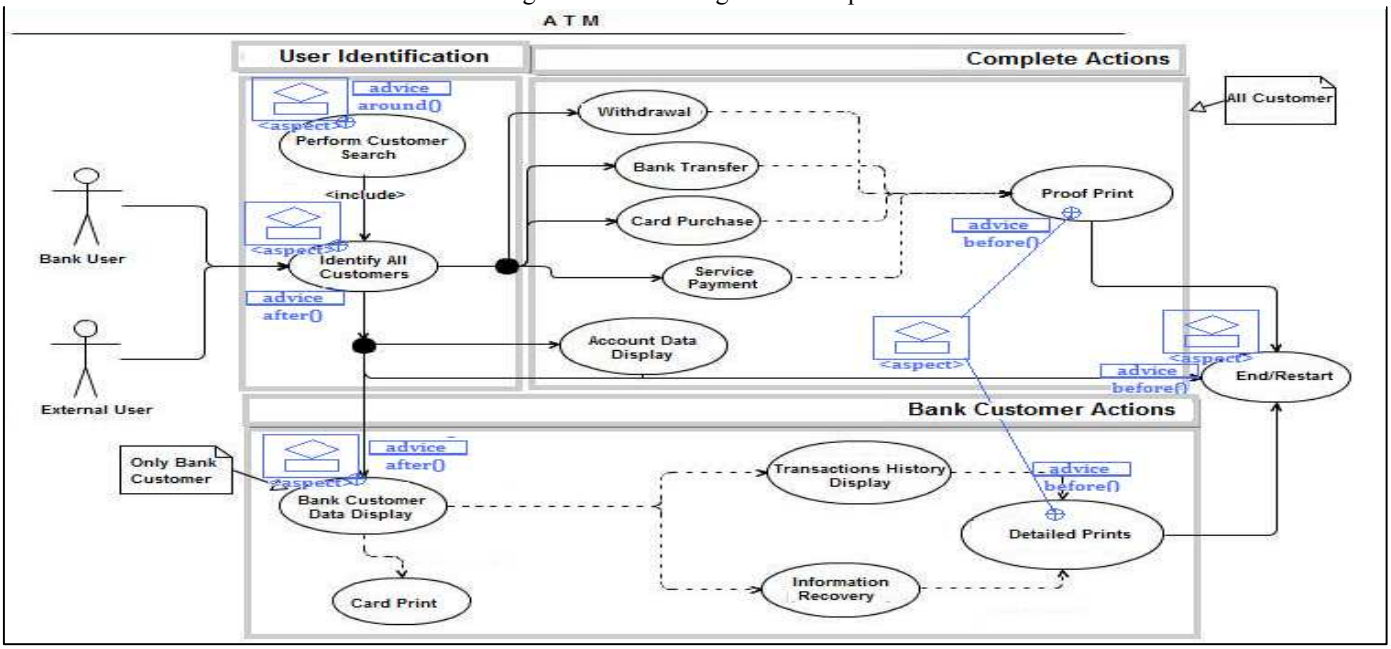
Aspect to Perform Customer Search
Cross-Cutting: ATM System - User Identification.
1Cut-Point: it is located by intercepting the Perform Customer Search function.
1Join-Point: it is the point of union of all elements that refers to the intercepted class. It is linked to the cut-point.
Advice: it is linked to the cut-point and its intervention time is around () because this aspect was identified by included clause.

Aspect to Bank Customer Data Display
Cross-Cutting: ATM System - Complete Actions.
1Cut-Point: it is located by intercepting to Bank Customer Data Display function.
1Join-Point: same as above JP.
Advice: it is linked to the cut-point, due to the large number of influences its time is after ().

Aspect Printing Details Plus
Cross-Cutting: ATM System – Global Actions.
2Cut-Points: one of them is located by intercepting the Proof Print function and one by intercepting the Detailed Print function.
2Join-Point: they are intercepting both classes related to the above function.
2Advices: There are two definite advices; the first one is related to the first cut point and its execution is before () due the number of dependencies. The second advice as it relates to the second cut point and its execution time is before ().

Aspect to Restart
Cross-Cutting: ATM System – Global Actions.
1Cut-Point: it is located by intercepting Restart/Stop function.
1Join-Point: same as in the previous case with single

Image 4 Used case diagram with aspects.



cutting function.
1Advice: it is linked to the cut-point, due to the large number of influences that time is after ().

The case study concluded with the generation of the framework case, the basic structure of the classes, methods and aspects from the support tools such as an Eclipse Aspect Project with AspectJ.

7. Conclusions and Future Works

Incorporating the AOP paradigm in the design stages of the Engineering Requirements allows you to add a more solid structure to the model because you can have all its benefits from the early stages.

The aspects in MEDFOAR are extracted from the used cases, the functional and non- functional requirements plus the system objectives. This methodology uses rules based on influences and dependencies of functions to perform all the tasks of identification and determination of issues.

The methodology in concurrence with the support tool provides the following advantages: The treatment of aspects is performed during the routine tasks of object-oriented paradigm; this method allows the identification, specification and locations of areas that impact the system. The structured approach attacks the problems and maximizes the benefits in the early stages of the software development. The tool allows searching and comparison of elements by minimizing redundancies, while it permits the generation of the basic framework of the analyzed system.

Future works will extend on the phases of the methodology to other stages of the engineering software; insofar as the support tool is concerned, they will allow the integration of classes already defined in code form, and they will test the methodology in different settings and environments for its refinement.

Acknowledgment

Work financed by the National Secretary of Science, Technology and Innovation (SENACYT) of the Republic of Panama through the proposal APY-GC10-061B.

References

[1] Z. Hua, et al., *"The Framework of Agent-Oriented Programming,"* Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on, 2005.
[2] A. Solberg, et al., *"Using Aspect Oriented Techniques to Support Separation of Concerns in Model Driven Development,"* in 29th Annual International Computer Software and Applications Conference, 2005.
[3] Fernando Asteasuain and B. Contreras, *"Programación Orientada a Aspectos. Análisis del Paradigma,"* Licenciatura, Ciencias y Computación, Universidad Nacional del Sur, Argentina, 2002.
[4] G. Cugola, et al., *"Language Support for Evolvable Software: An Initial Assessment of Aspect-Oriented Programming,"* in International Workshop on the Principles of Software Evolution, IWPSE99, 1999.
[5] H. Hu, et al., *"An AOP Framework and Its Implementation Based on Conceptual Model,"* ISECS International Colloquium on Computing, Communication, Control and Management, p. 4, 2009 2009.
[6] A. Nusayr, *"AOP as Formal Framework for Runtime Monitoring,"* in FSE-16 Doctoral Symposium, Atlanta, Georigia, USA, 2008.
[7] W. Cazzola, et al., *"Semantic Join Point models: Motivations, Notions and Requirements,"* presented at the In SPLAT 2006 (Software Engineering Properties of Languages and Aspect Technologies), 2006.
[8] G. Jun-Wei, et al., *"A MDA based Aspect-Oriented Model Dynamic Weaving Framework,"* presented at the International Conference on Computer Science and Software Engineering, Wuhan, Hubei 2008.
[9] M. E. Markiewicz and C. J. P. d. Lucena, *"El Desarrollo del Framework Orientado al Objeto."*
[10] M. Aoyama and A. Yoshino, *"AORE (Aspect-Oriented Requirements Engineering) Methodology for Automotive Software Product Lines,"* presented at the Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific, Beijing, 2008.
[11] A. Rashid, *"Aspect-Oriented Requirements Engineering: An Introduction,"* presented at the 16th IEEE International Requirements Engineering, 2008. RE '08., Catalunya 2008.
[12] J. Grundy, *"Aspect-oriented requirements engineering for component-based software systems,"* in IEEE International Symposium on Requirements Engineering, 1999., Limerick, 1999, pp. 84 - 91.
[13] C. C. Budwell and F. J. Mitropoulos, *"The SLAI Methodology: An Aspect-Oriented Requirement Identification Process,"* in 2008 International Conference on Computer Science and Software Engineering, Wuhan, Hubei, 2008, pp. 296 - 301
[14] P. Yu-Ning and L. Qiang, *"A Viewpoint-Oriented Requirements Elicitation Integrated with Aspects,"* presented at the World Congress on Computer Science and Information Engineering, 2009 WRI., Los Angeles, CA 2009.
[15] Z. Jingjun, et al., *"Aspect-Oriented Requirements Modeling,"* presented at the 31st IEEE Software Engineering Workshop, 2007. SEW 2007., Columbia, MD 2007.
[16] M. Tabares, et al., *"Aspect Oriented Software Engineering: An Experience of Application in Help Desk Systems,"* Dyna rev.fac.nac.minas, vol. 147, 2007.
[17] L. Londoño, et al. (2008) "*Análisis de la Ingeniería de Requisitos Orientada por Aspectos según la Industria del Software"*. Revista EIA. 43-52.
[18] F. Losavio, et al., *"UML Extensions for Aspect Oriented Software Development,"* Object Technology, vol. 8, 2009.

Authors:

Xavier Medianero-Pasco is a student of the Master of Science in Information and Communication Technology at the Technological University of Panama. He was awarded a Bachelor's degree in Engineering and Computer Science from the Technological University of Panama. His research interests include Aspect Oriented Programming, Grid Computing and other topics.

Sergio Crespo S.C. Pinto is a professor at the Universidade do Vale do Rio dos Sinos, Brazil. PhD awarded by the Pontificía Universidade Católica do Rio de Janeiro. His research interests include Frameworks, Design Patterns and Software Engineering among others.

Clifton Clunie is a professor at the Technological University of Panama, PhD awarded by the Universidade Federal do Rio de Janeiro. His research interests include Software Engineering, Frameworks and Quality Assurance.