Journal of Applied Sciences

ISSN 1812-5654

Journal of Applied Sciences 12 (21): 2182-2194, 2012
ISSN 1812-5654 / DOI: 10.3923/jas.2012.2182.2194
© 2012 Asian Network for Scientific Information

An Aspectual UML Modelling Tool

Aws A. Magableh, Zarina Shukur and Noorazean Mohd. Ali
School of Computer Science and Information Technology, Universiti Kebangsaan Malaysia,
43600 Bangi Selangor, Malaysia

Abstract: The aspect-orientation is a complement for object-orientation. Thus, it would be logical to investigate the adaptability of UML to aspect-orientation. This research focuses on investigating the Aspect-Oriented UML (AOUML) approaches for understanding the benefits of a comprehensive framework for AOUML. Based on this study's review, it is evident that, of late, the complexity and size of systems have grown up, which accumulatively have led to the manifestation of new concerns. Moreover, these new concerns have cut-cross other concerns and core classes in the system by its nature. Due to this fact, the concept of Advance Separation of Concerns (ASoC) has been put on the table of discussions and the need for an approach to model and represent these crosscutting concerns (Aspect), which is responsible for producing, spreading and tangling representation throughout the development life cycle, is vital. A proper databases have been searched using the suitable keywords, which match this research questions as recommended by systematic review process; this research has collected 468 studies and screened them to minimize the number of studies to 73, which are more appropriate and directly related for this present study. The general scope of this research is to model aspect (crosscutting concerns) using standard UML diagrams 2.4.1 (latest edition). UML behavioural and structural diagrams have been implemented on the top of object-orientation concepts, it has not been meant to be used to model aspect-orientation. Thus, this research has proposed a complete tailored framework that represents aspect's constructs using all UML diagrams. The objective of this position study is to investigate the aspectual UML modelling tool which is currently being designed and implemented.

Key words: AOM, aspects, aspectual UML diagrams, aspect modelling, crosscutting concerns, aspect representations, aspect-oriented UML, AspectJ

INTRODUCTION

The needs for the advance Separation of Concerns (SoC) are in demand during the software development processes (Magableh and Kasirun, 2007). Therefore, a lot of approaches have been proposed and used such as: Subject-Orientation (SO), Feature-Orientation (FO) and the most popular Object-Orientation (OO), to represent the concerns of the system. Due to the weakness of object-oriented analysis and design approach, the proposition in handling the crosscutting concerns (Aspects) (Uetanabara *et al*., 2009), Aspect-Orientation Analysis and design (AOAD) approach has been proposed to focus on the crosscutting concerns and its effect on multiple classes. In addition, the AOAD is further divided into Aspect-Oriented Requirement Engineering (AORE), Aspect-Oriented Architecture (AOA) and Aspect-Oriented Design Modelling (AODM). All these fields of AO focus on efficiently handling the Aspects throughout the software life cycle. However, the most neural field is the modelling field.

In fact, quite a good number of researches have been carried on AODM. Some are based on architectures View (Katara, 2002), aspect-oriented language (Groher and Baumgarth, 2004; Groher and Schulze, 2003), XML representation format (Suzuki and Yamamoto, 1999), some other are based on component engineering (Grundy, 2000) and component views (Muller, 2004). Furthermore, the AODM using UML Diagram is the most well-known technique, as (UML) is probably the most widely known and used modelling notation (Ali *et al*., 2007b) which has drawn the attention of a lot of researchers.

UML is considered to be the most accepted design modelling language in software engineering and the industry as it provides a powerful set of modelling tools and diagrams (Groher and Baumgarth, 2004; Asteasuain *et al*., 2008). Thus, the UML for aspect has to be investigated similar to OO. Finally, there are many propositions on Aspect-Oriented Design Modelling (AODM) using UML approaches. However, there is a lack of uniform standards (Albunni and Petridis, 2008;

Corresponding Author: Aws A. Magableh, School of Computer Science and Information Technology, Universiti Kebangsaan Malaysia, 43600 Bangi Selangor, Malaysia

Zhang, 2005). With its current state of the design level the UML is unable to properly represent aspect-orientation constructs and the crosscutting nature of the aspects (Marco *et al*., 2008).

It is very important and vital to have a uniform standard aspect-oriented modelling. Currently, there is a lack of support for a unified aspect modelling in the industry, as everyone uses their own selected approach based on their needs, however, the possibility of choosing a wrong/incompatible modelling approach is there as sometimes the selection for the proper modelling approach depends on the system size and complexity. Additionally, if a project has to be transited to some other developers/company then lack of the standard uniform modelling will be apparently a problem and will not be an easy work to handle such transition. Practically, there is a need to have a uniform standard aspect modelling approach to preserve the maintainability and consistency through out the software development.

Programming and modelling languages exist in a relation of mutual support. Thus, this study focuses on button-up technique. It starts from the well-established aspect coding languages level (AspectJ), which is a hot programming topic nowadays (Zhang *et al*., 2009) and moves backwards to design level to generate an aspect-oriented UML constructs notation (aspectual UML). As there is a lack of Aspectual design notations in AODM for designing Aspect Oriented code (AspectJ in this case) (Muley *et al*., 2010), this would mainly depend on having a look at the coding constructs, then try to implement it in a representational notation, to be used in the design modelling level of the tool. As a result, the level of encapsulation would be increased, not only at the coding level but rather at design level too. Concurrently, the level of consistency would be maintained at high level as all the coding constructs would be denoted at design phase in the first place (Kande, 2003).

BACKGROUND OF ASPECT ORIENTED DESIGN MODELLING

Here, we are explained structured Systematic Literature Review (SLR) (Ramey and Rao, 2011) to narrow down. A systematic literature review/mapping refers to the reassessment, evaluation and interpretation of the all related available research and primary studies that are relevant to clearly formulated questions, followed by extracting and analyzing information included in the review (Ramey and Rao, 2011). SLR is composed of the following steps: (1) Systematic mapping planning (2) Conduction of the search (3) Selection of the primary studies and (4) analysis and map building. The four steps and their output are depicted below in Fig. 1:

Step 1: Systematic mapping plan illustrates the plan that have used to conduct the research. It consists of three sub steps: (1) Good formulation for the RQ (2) Selection of the Databases and Resources and (3) The Inclusion Criteria (IC) and Exclusion Criteria (EC)

Step 2: Conducting the search is divided into two sub steps: (1) Choosing keywords and (2) Search strings in databases. The completion step 2, yields the final edition of the primary related studies, using the proper keywords, with the database related string searching format

Step 3: Selection of the primary studies, as a result of applying the above keywords (step 2) on the listed data sources (step 1), the research has identified that the total number of primary studies is 468. Furthermore, it has extracted the proper and directly related studies, by removing the duplication and redundancies and by reading the abstract of the articles and the full text. As a result, the research has concluded 73 primary studies related to this research's topic as shown in Table 1

Step 4: Analysis and build mapping illustrates the analysis carried out on the selected related studies. Generally, the comparison criteria should fulfill certain criteria such as standards and guidelines (Shukur and Mohamed, 2008). These criteria have been taken under the consideration

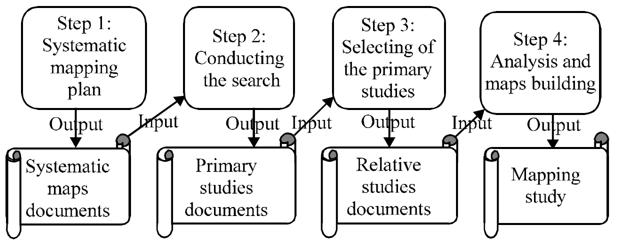

Step 1: Systematic mapping plan
Step 2: Conducting the search
Step 3: Selecting of the primary studies
Step 4: Analysis and maps building
Output
Input
Output
Input
Output
Input
Output
Systematic maps documents
Primary studies documents
Relative studies documents
Mapping study

Fig. 1: Systematic literature review processes

Table 1: Articles screening in numbers

Research question	Total references retrieved	After excluded duplicate	Total abstract screened	After abstract screened	Total full-text screened	Total included study	Final included study
RQ1	285	242	242	82	64	64	64
RQ2	160	132	102	55	45	6	6
RQ3	23	16	13	10	4	4	3

to compare the existing UML modelling approaches (Grundy, 2000; Ho *et al*., 2002; Stein *et al*., 2002; Clarke and Baniassad, 2005; Filman, 2005a, b; Jacobson and Ng, 2005; Reddy *et al*., 2006; Coelho and Murphy, 2006; Cottenier *et al*., 2007; Katara and Katz, 2007; Klein *et al*., 2007; Sharafi *et al*., 2010; Przybylek, 2010)

The selection of the criteria was not randomly done; instead it was based on critical factors such as: Language, maturity, real example, complete framework, tool support, AspectJ constructs and these criteria are shown in Fig. 2.

Figure 3 illustrates an idea on the connection between the study areas of this research, in order to come out with a new Aspectual UML modelling tool proposed by this study.

As a result of this systematic literature review, the research has finally gotten the direct related studies of aspect-oriented modelling using UML diagrams and knew where the literatures have reached on this concern. Moreover, by studying the well established AOP language (AspectJ), latest UML superstructure and infrastructure 2.4.1 and understanding the ability to integrate both to be utilized in the modelling of Aspects in the early stage of software life cycle, we have concluded that the proposed ideas to work with a complete Aspectual UML framework. This study is more concerned with the tool supports of this research proposition.

ASPECT-ORIENTED PROGRAMMING ELEMENTS (RESULTS of BOTTOM-UP APPROACH)

AOP allows programmers to implement the concept of separation of concerns. Also, it overcomes the problem of code spreading over the core concerns, which is called as code tangling and code scattering, the issue that the OO could not be able to solve efficiently, when it implements the crosscutting concerns. Additionally, AOP solves these issues by implementing new modularity unit called as Aspect. The AOP has achieved an apparent growth in the industrial context and academic researches, which has drawn to an interest in AO for all software life cycle stages (Przybylek, 2010).

Code tangling defines as a module is implemented that handles multiple concerns at the same time such as logging and security as shown in Fig. 4 (Ramnivas, 2003).

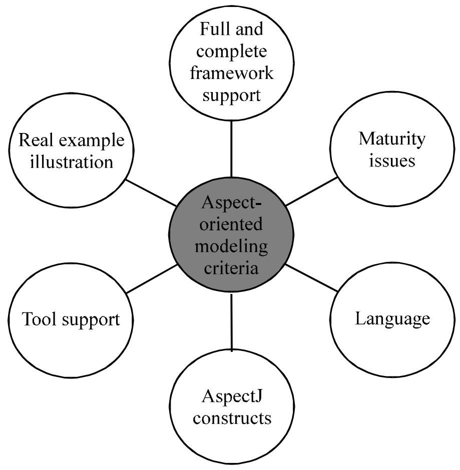

Full and complete framework support
Real example illustration
Maturity issues
Aspect-oriented modeling criteria
Tool support
Language
AspectJ constructs

Fig. 2: Criteria categories

Code Scattering means that a single concern implementation has been distributed in multiple modules as shown in Fig. 5.

AOP is not meant to patch up bad design and it is not a solution for poor design. Furthermore, it is not a new complete design process. However, it provides additional means to help the designer in addressing the potential expected future requirements, as well as solving the designer's problems (Ramnivas, 2003).

The Information System Development (ISD) has many methodologies in the field to address different software development phases (Baharom and Shukur, 2011), AOSD is one of them. Due to the realization for its importance, the AOP is intended to be applied on AOSD stages. Furthermore, AOP is considered to be as a complement for (OO). AOP has been meant to represent certain design concerns tended to be cut-cross the other core functional concern in the context of AOSD. As per to (Alam *et al*., 2009) there are many AOP such as AspectJ, AspectS and AspectML, of which the most popular is AspectJ. AspectJ is the most de facto standard language in the industry and that is why it has been selected to be the base for this study. The Table 2 shows the comparison between AspectJ and the other aspect languages. It depicted that the AspectJ is the most well-featured aspect language.

AspectJ syntactical construct: Majority of AOP has invented a common constructs for crosscutting concerns

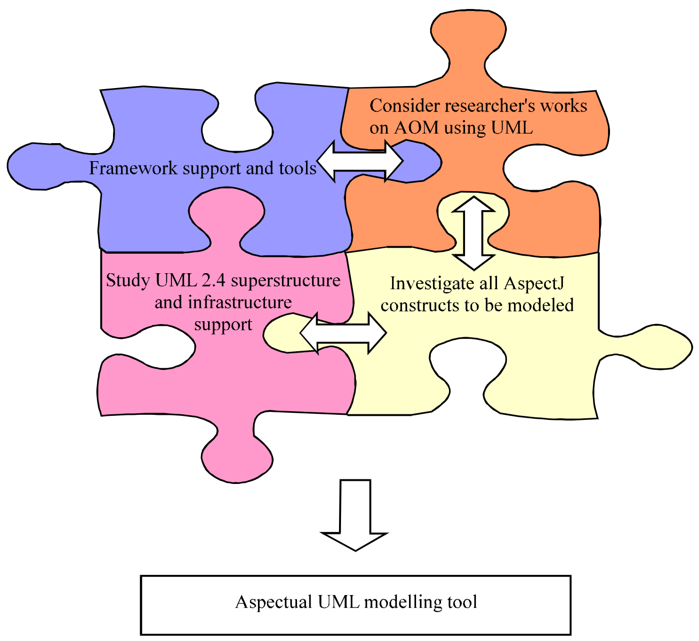

Consider researcher's works on AOM using UML
Framework support and tools
Study UML 2.4 superstructure and infrastructure support
Investigate all AspectJ constructs to be modeled
Aspectual UML modelling tool

Fig. 3: Connection between the study's areas

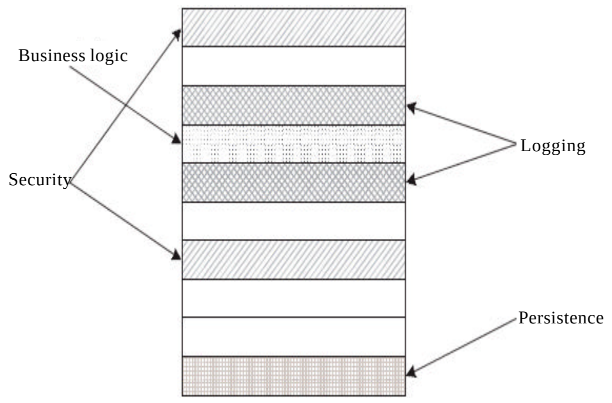


Fig. 4: Code tangling representation, Ramnivas (2003)

Table 2: Aspect programming feature comparison

	Aspect	AspectS	AspectML
Aspects can be instantiated	×	√	AspectML does not have an aspects construct
Aspect inheritance	×	√	
Nested aspects	√	×	
Privileged aspects	√	×	
Polymorphic pointcuts	×	×	√
Polymorphic advice	×	×	√
Advice on field access	√	×	NA

representation and capturing. These constructs are called as join point, pointcut, advice, introduction and aspects.

Join point is a well defined point in the execution code of the program; it would be a constructor call, normal method call, data member assignments but not limited to these only. Additionally, it has been categorized into: Method call, method execution, constructor call, constructor execution, static initializer execution, object pre-initialization, object initialization, field reference, field assignment, handler execution and advice execution.

Pointcut is a program construct that captures join points, select join points and collect context at these join points (target object), it is categorised as method call, method execution, get, set, constructor call, constructor execution, constructor initialization, constructor pre-initialization, static initialization, handler, advice execution, within pointcut, within code, cflow pointcut, cflow below pointcut, this, target and args pointcut.

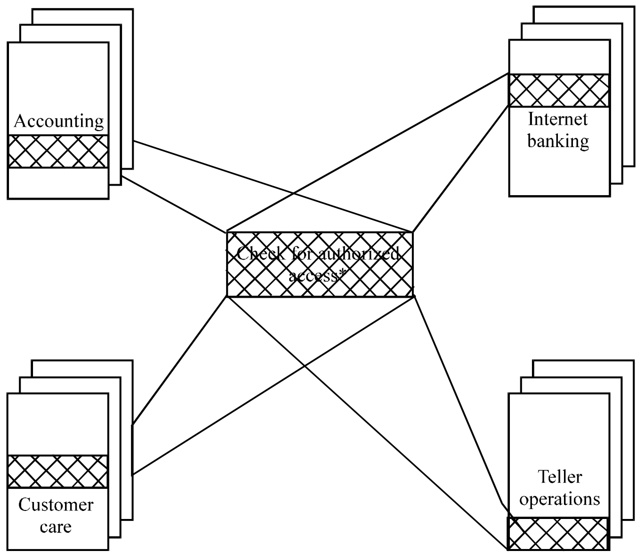
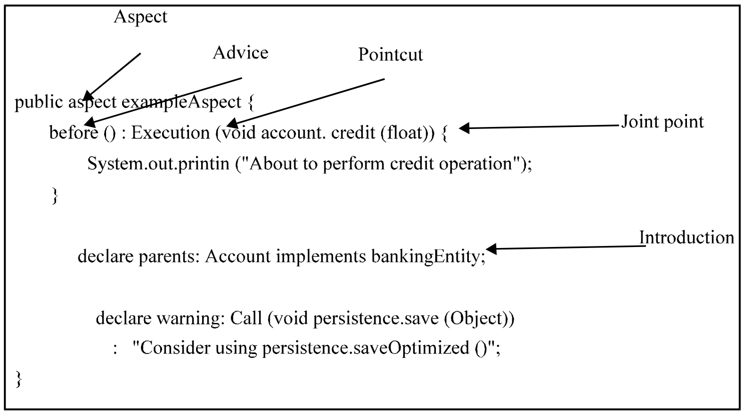


Fig. 5: Code scattering representation Ramnivas (2003)



```
public aspect exampleAspect {
     before () : Execution (void account. credit (float)) {
          System.out.printin ("About to perform credit operation");
     }

          declare parents: Account implements bankingEntity;

               declare warning: Call (void persistence.save (Object))
                  :   "Consider using persistence.saveOptimized ()";
}
```

Aspect
Advice
Pointcut
Joint point
Introduction

Fig. 6: Aspect coding sample

Advice is the code to be ejected and executed at a join point that has been selected by a pointcut. This advice might be a before-advice which runs prior to the execution of the new code, after-advice which runs after the new code is ejected and executed, around- advice, which alters the core code execution.

Introduction is the static crosscutting instruction that introduces some changes to the static hierarchy of the core code, classes, interfaces and aspects of the system; it directly helps the behaviour of the system and helps to achieve the dynamic crosscutting as well. For instance, it adds new data member to class, adds new member function or changes the inheritance hierarchy.

Aspect is the unit of modularization of AspectJ that captures all these constructs. It contains the code that expresses the waiving rules of pointcut, join points, advices and introductions as shown in Fig. 6.

Benefits of AspectJ programming: Critics of AOP often converse about the difficulty of understanding it. Indeed it takes some time, patience and practice to master AOP. However, the main reason behind the difficulty is simply the individuality of the methodology. The benefits of AOP actually far outweigh the perceived costs. Some of these benefits are:

- Cleaner responsibilities of the individual module
- Higher modularization
- Additional code reuse
- Reduced costs of feature implementation

In reality the flow of AOP (AspectJ in this case) program is hard to follow, as a lot of concerns have been weaved with the core system functionality. Moreover, AOP is not about providing solution to the unsolved problems; it is about providing better way of solving a specific problem with less effort and improved maintainability. Finally, AOP is not considered as a replacement of the Object-Oriented Programming (OOP) or other procedural programming; AOP adds new additional concepts to represent the crosscutting concerns.

CURRENT UML NOTATION ON ASPECT-ORIENTATION

Modelling is the process of providing structure for the problem, experiments to explore the solution; decrease the development cost and reduces the mistakes that might take place. Specifically, aspect-oriented modelling demonstrates aspect and its constructs form the early stage of development life cycle. The benefits of modelling aspect are making the design more reusable, makes it easier to enable the code generation, helps learning and documenting aspects and aspects specification and the most important point is modelling the aspect from the requirement stage, design phase then implementation makes the process and transitions consistent and maintainable (Omar *et al*., 2002). AOM includes modelling tasks starting from requirements engineering via analysis to design, this consider to be as an essential part of AOSD.

Literature (Iqbal and Allen, 2007) investigates the AOM issues and misunderstood concepts. The literature clarifies and identifies some misconception takes place when AOM is being used. It has pointed out that the there are many unsolved AOM issues that is due to the nature of the aspect itself and its constructs. Aspects have been represented as class-like formation, however, the class is a construct for object-orientation and a lot of principles such as, encapsulation, inheritance and instantiation have been tied with it. Based on the above factors, it has been found out that Aspect can not be represented as a class as it can not be instantiated when the user needs; however, it is instantiated when the system demands. Moreover, Aspects do not maintain the concept of encapsulation all the time as it accesses the private data of its base class to perform some actions. Additionally, Aspect's children can not override the piece of advices of the parent Aspect as its signature designator is not unique. Finally, AOM has been investigated and a lot of propositions have been proposed by the researchers, some are based on Theme/UML (Clarke and Barriassad, 2005), SUP (Omar *et al*., 2001), CoCompose (Dennis and Bergmans, 2002), Aspect at Design Time (ADT) (Jose *et al*., 2000) and Aspect-Oriented UML modelling which is the concern of this research.

Aspect-oriented UML modelling: Unified Modelling Language (UML) is an object-oriented analysis and design language from the Object Management Group (OMG) that standardizes several diagraming methods, including Grady Booch's, Rumbaugh's and Ivar Jacobson's. In addition, UML has become a standard modelling language in the industry, as well as the most well-established and commonly used by designers and analysts (Ali *et al*., 2007b).

UML modelling extension mechanisms have been categorized into two types. The first one is the UML Meta Object Facility Metamodel (Heavy-weight) and the second one is the Constructing UML Profile (light-weight).

UML Meta Object Facility Metamodel (MOF metamodel) is referred to as heavy-weight extension. The metamodel constructed can be as communicative as needed. It is harder than constructing a UML profile and does not have a lot of supportive tools as compared with UML profile.

Constructing UML profile (light-weight) is usually called as light-weight extension, because all the current existing constructing UML profile extension techniques, do not implement any new UML Meta-model elements. Further, constructing UML profile extension techniques are usually considered as predefined set of constraints, tagged values, graphical representations and stereotypes. Moreover, constructing UML profile extension method supplements aspect based on the flexibility and extendibility nature of the standard UML domain modelling.

Grundy (2000), the Aspect-Oriented Component Engineering (AOCE) focuses on recognizing a mixture of portion or Aspects of an overall system. A component offers services to other partner components or requires from other components. Aspects are horizontal cut cross through a system, which apparently would affect many other components such as persistency and distribution. Developers use components to represent Aspects with different components capabilities in the software development such as requirements engineering and design. The AOCE proposes representation for Aspect and its nature by providing a new framework for describing and reasoning about component capabilities from multiple perspectives.

Based on literature (Ho *et al*., 2002), it proposed a UML all purpose transformer (UMLAUT) toolkit which used the MOF UML mechanism extension to model Aspects. It is an Aspect-Oriented UML models used to build Aspect weavers for constructing detailed design model from high level of abstraction. UMLAUT gives the developer the ability to program the weavers at the level of UML Meta model. Additionally, The UMLAUT provides the user with a general purpose representation which can be reused for different function with specific demands weaver that optimizes the weaving process demonstrated by UMLAUT.

Stein *et al.* (2002), considers reviews one of the light-weight UML extensions. It has been developed as a design notation for AspectJ; it extends the existing UML standard notations. It comes with a new production AspectJ weaving process. However, it does not represent all AspectJ constructs to be modelled as well as not all UML diagrams have been used in the proposition, they have focused on the class diagram.

Clarke and Baniassad (2005), Theme/UML is used to produce separate design models for each “theme” elicited from the Theme/Doc requirements phase and then it does encapsulate the concern representing some kind of functionality in a system. The Theme/UML is considered to be a heavy-weight extension of the UML metamodel version, as it adds some new elements to the standard representation. Basically, the Theme/UML creates no restrictions on the UML diagrams that might be used for modelling. Nevertheless, package and class diagrams are specifically used for structure modelling and sequence diagrams are used for behaviour modelling.

Filman (2005b) aims to model aspect independently from the existing types of the aspect-oriented programming languages. The Class diagrams are used to express the structural dependencies and state the machines model and the behavioural dependencies of concerns and Aspects. The approach provides a guideline on how to refine the modelling continually from class diagram to the state of model machine. The approach focuses on these UML diagrams only and did not assume the modelling might take place for other diagrams.

Jacobson and Ng (2005), Use case software development method to represent aspects. It has been realized by extending the UML 2.0 metamodel. AOSD with Use Cases (AOSD/UC) implemented with an organized processes that concentrate on the SoC throughout the software development life cycle. Starting from requirements engineering (RE) with use cases to the implementation and design phase with component diagrams and class diagrams. While sequence diagrams are used to model behavioural structure of the system. Concerns are modelled using a use case stereotype and the approach does not come with any support tools.

Filman (2005b), proposes a mix mode mechanism (heavyweight and lightweight), where it makes use of the UML profile extension to model different domain as well as UML meta object facility model mechanism, by proposing new Meta object/notation. It proposes a Java Aspect Component (JAC), which does not depend on any platform. This JAC comes with new UML notations to represent aspects and its implementation. It supports all the steps of Aspect-Orientation development from its design, to its implementation ending with the deployment. This approach depends on adding stereotypes to classes to implement aspects and non functional concerns.

Reddy *et al.* (2006), aspect-oriented class design model includes of different Aspect models. Each one of them explains an attribute that crosscuts the other models including the primary one. The Aspect model and the main models are combined to get an integrated view. Composition approaches have been described that utilizes two compositions; the first one is composition algorithm and the second one is composition directives. Its prototype tool supports default class diagram composition.

Coelho and Murphy (2006) presents crosscutting structure, which is usually done using two ways: (1) tree views, which involve developers in combining information across multiple views manually and (2) static structure diagrams, which will probably suffer from extreme graphical complexity. An active model is an approach that attends to these problems, by presenting the right crosscutting structure at the proper time. To control the diagram complexity, the right information is determined through automatic projection and abstraction procedures that select representation elements. The model is presented at the right time using two combinations. The first one is a user-driven expansion operation that adds more detailed info to the model and the second one is through the interaction features that added by the end user.

Cottenier *et al.* (2007) proposed a state diagram extension specification to represent a new join point. The interfaces included in a function invocations on the state of the module instance. These specifications are not identified for a specific Aspect, However, unclearly describes the behavioural observation of the module. Cottenier *et al.* (2007) have shown additionally how a join point selection mechanism is able to infer points smartly which might be placed somewhere very deep inside the implementation of a component. It refines the class diagram and the composite structure to capture the static structure of the system. It uses the state machine extension to represent the behaviours of the system.

Katara and Katz (2007), is using the architecture view to group aspect designs. The architecture model provides an aspect-oriented representation on software design using UML. The model gives you the ability to analysis the aspects as a viewpoint to observe the impact of adding/deleting Aspects in the model. It adds some new stereotypes to model aspects such as <<Aspects>>, <<Concerns>>, <<Bind>>, <<replace>> and <<Unify>>.

Klein *et al.* (2007) proposes an Aspect-Oriented UML approach using the standard UML. It has not proposed any new notation and it did not use the UML extension ability, just to maintain the standardization with no changes. It is originally based on Message Sequence Charts (MSC) a standardized scenario language. It uses UML 2.0 sequence diagram. Indeed, no extensions to the UML Sequence diagram have been made; relatively a simplified meta model for sequence diagram has been designed, where conformity with the original UML Sequence diagram is accomplished through model transformation in the supplementary tool support.

Przybylek (2010) proposed aspect-oriented UML modelling Which is an extension establishing a new package called AoUML, which consists of elements to represent the primary AO concepts. It also proposes to reuse elements from the UML 2.1.2 infrastructure and superstructure specifications.

RESEARCH PROPOSITION: ASPECTUAL UML TOOL

Aspectual UML modelling tool suite consists of several integrated sub-tools and each consecutive one works on the output received from the previously applied sub-tool. Figure 7 shows AOP profile domain model of this research, which illustrates all aspect constructs that has been considered in this research modelling tool.

Figure 8 depicts the architecture of aspectual modelling tool; it consists of several drawings modules for 14 well know UML diagrams as shown in Fig. 9.

Then these drawings will be saved in Scalable Vector Graphics (SVG) file format, which is an XML based file format for describing two-dimensional vector graphics. SVG can describe three types of graphics-vector graphics (paths consisting of straight lines and curves), images and texts. These graphics can be grouped, styled, or transformed into different shapes using template objects, clipping paths, nested transformations, alpha masks and filter effects. SVG drawings can also be dynamic, interactive, or animated. Animations may be defined either by using embedded SVG animation elements, or through scripting. Other features of SVG include hyperlink support, scripting based on events being based on XML and susceptibility to compression. Some other similar works have been done by converting the drawings into a PETAL file which is the extension used by rational rose (Ali *et al.*, 2007a) proposes a proper extraction notation with a accurate and reliable extraction process.

Based on this research approach, this SVG file will be an input to any other tools that read this extension and generate a drawing out of it; moreover, this file will be an input to a model checker to check the consistency and validity of the domain modelling.

This study modelling approach uses latest UML edition 2.4.1, which have not been used to model Aspects, as per the infrastructure and super structure of UML 2.4.1, some notations are valid to be used to model aspects. However, some other aspects constructs would not be

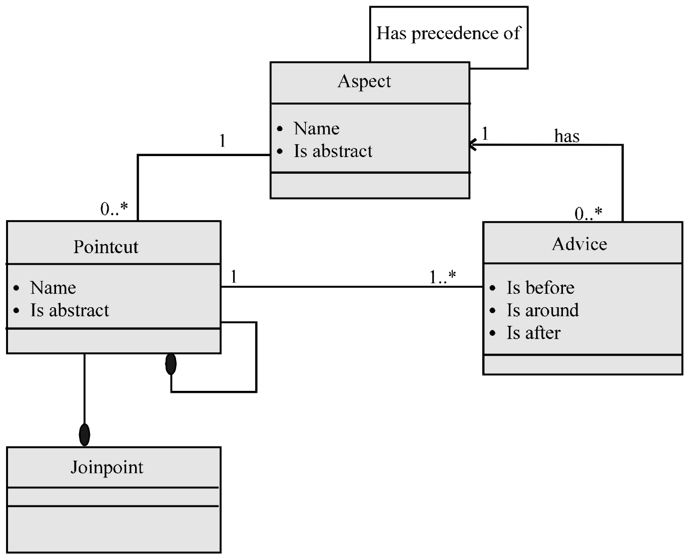


Fig. 7: AOP profile domain model

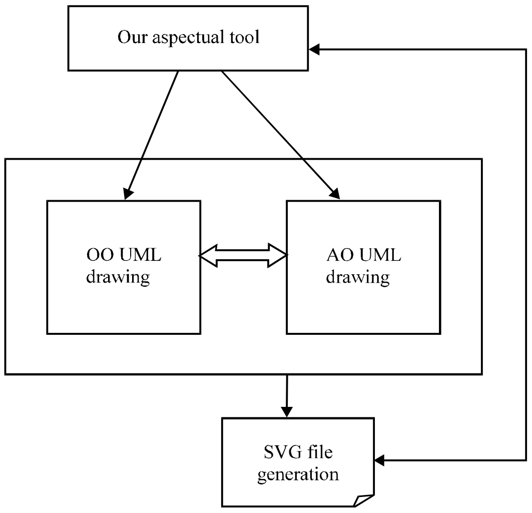


Fig. 8: Tool architecture

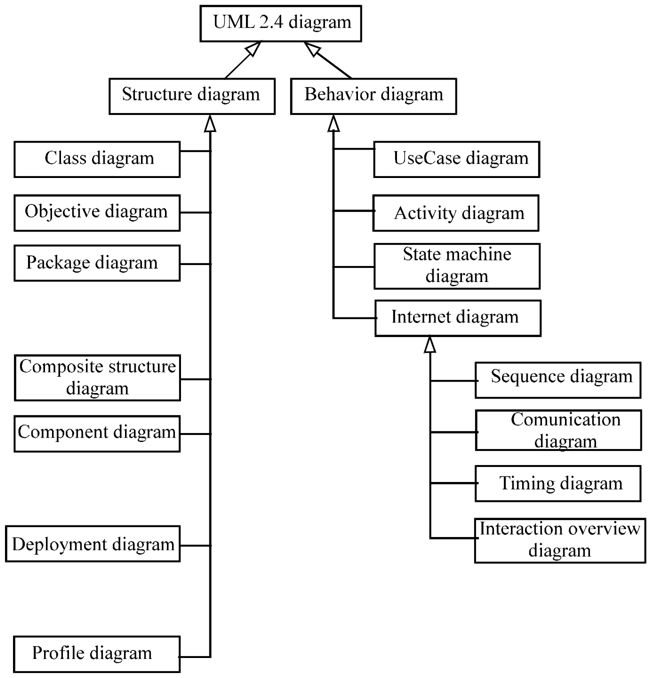


Fig. 9: Aspectual UML diagrams

modelled efficiently; hence, this research has proposed new notations to model aspects constructs.

This research model proposes new notations to represent all kind of AspectJ constructs, to be modelled from the early stage of software life cycle. Due to the fact that AspectJ is the widely used aspect programming language in the industry and due to the lack of modelling support for AspectJ, this research has proposed a new heavy-weight UML extensions, to model all detailed constructs of AspectJ, moreover, a lot of researches have been carried on modelling AspectJ constructs, however, majority of them have focused on one or more UML diagrams and none of them have studied the applicability of modelling the AspectJ constructs in all UML diagrams, as well as the details of AspectJ. Figure 10 shows basic print-screen of the tool.

By looking at the evaluation aspect of the proposed tool and approach Foxley *et al*. (1997) suggest that the

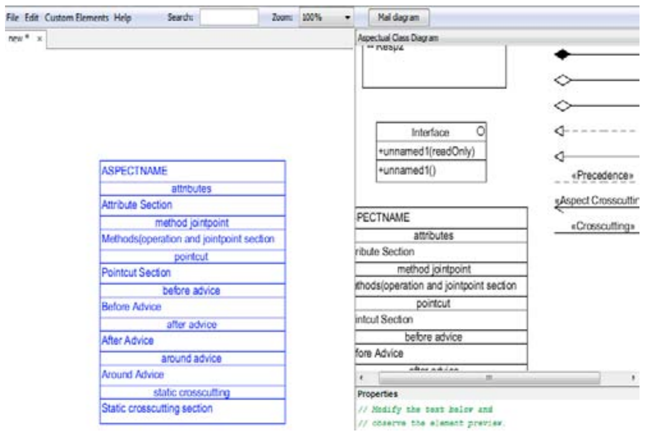


Fig. 10: Print-screen of the tool

most common methods of measure the quality is to look at different kind of factors that affect the quality and measure them separately. By applying that on this research; correctness, usability and tractability will be measured toward the end of the research.

ASPECTUAL UML TOOL CONTRIBUTION

Here, we highlighted the significance and contributions of this research. In addition, it explains the observation on weaknesses and strengths of this research, there are some significance and contributions of this research proposed tool:

- The tool makes use of the latest UML edition 2.4.1
- The tool gives the option to draw an OO notation as well as AO modelling notations
- The tool represents all AspectJ detailed constructs
- The tool represents all AspectJ constructs in all UML diagrams
- The tool has the ability to transform the drawings into SVG extensions and makes use of this powerful extension
- The tool provides a complete framework for modelling general domain systems that has aspects impeded in it

It is very normal that any tool has strengths and weaknesses. Different observers have different opinions and ways for looking at and evaluating systems. However, there are certain strengths and weaknesses that could be mentioned regarding the Aspectual tool. The strengths are:

- The tool combines two different stages together, the first one is the design and modelling stage the other one is the implementation stage
- This tool will open rooms for more researchers and students' projects, since students will be able to refer to this project and its report for their own benefits in developing systems
- This tool makes use of the power of UML modelling to come out with Aspectual UML drawings
- This tool overcomes the issues of the current exciting UML modelling tools, as majority of these tools such as Rational Rose (Baharom and Shukur, 2011) and MagicDraw do not provide the option to represent crosscutting concerns (aspects)

The weaknesses of the tool are:

- It is not easy to use these kinds of tools, because understanding the concept of aspect orientation modelling requires extensive understanding
- Currently the tool supports English language only
- It does not support all the processes of software life cycle such as requirement engineering

CONCLUSION

In conclusion, this research has identified that almost all primary studies have similarities in their objectives. Additionally, all of them are heading towards addressing and modelling the crosscutting concerns (Aspects) using UML which is the focus of RQ1. Furthermore, majority of the studies were focused on extending the current UML model and make use of the extendibility feature of UML. Only few researches had proposed their own notations and extensions. The majority of the researches were focusing on one or two diagrams of the UML; none of them were addressing a complete framework. Moreover, majority of them were not providing tools for their propositions, they depend on the existing tools. Furthermore, all the researchers had focused on the older versions UML rather than the current UML edition, which is UML 2.4. Moreover, none of the studies have focused on the new Infrastructure specification and the Superstructure specification of the adaptability and compatibility of UML 2.4 with the proposed model. This research believes that Aspect-Oriented programming should be extended to the entire software development life cycle. Each aspect of the implementation should be declared during the design phase, so that there will be a clear traceability from requirements through source code. Finally, as a future works the tool will be tested using module documentation-based testing or MD-test.

ACKNOWLEDGMENTS

This study has been funded by the faculty of computer science and information technology in University National Malaysia (UKM). The Aspect-Oriented systematic review has inspired us and opened some new avenues for research. This research is currently carrying on and has started on creating Aspectual UML 2.4 Modelling Approach, using the bottom up technique, to capture and represent all the crosscutting concerns (Aspect) constructs and elements including AspectJ as a base for the study.

REFERENCES

Alam, F.E., J. Evermann and A. Fiech, 2009. Modeling for dynamic aspect-oriented development. Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering, May 16-24, 2009, Vancouver, Canada, pp: 143-147.

Albunni, N. and M. Petridis, 2008. Using UML for modeling cross-cutting concerns in aspect oriented software engineering. Proceedings of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications, April 7-11, 2008, Damascus, Syria, pp: 1-6.

Ali, N.H., Z. Shukur and S. Idris, 2007a. A design of an assessment system for UML class diagram. Proceedings of the International Conference on Computational Science and Applications, August 26-29, 2007, Kuala Lampur, pp: 539-546.

Ali, N.H., Z. Shukur and S. Idris, 2007b. Assessment system for UML class diagram using notation extraction. Int. J. Comput. Sci. Network Secur., 7: 181-187.

Asteasuain, F., B. Contreras, E. Estevez and P.R. Fillottrani, 2008. Evaluation of UML extensions for aspect oriented design. http://grise.upm.es/rearviewmirror/conferencias/jiisic04/Papers/7.pdf

Baharom, S. and Z. Shukur, 2011. An experimental assessment of module documentation-based testing. Inform. Software Technol., 53: 747-760.

Clarke, S. and E. Baniassad, 2005. Aspect-Oriented Analysis and Design the Theme Approach. Addison Wesley, Boston, USA.

Coelho, W. and G. Murphy, 2006. Presenting crosscutting structure with active models. Proceedings of the 5th International Conference on Aspect-Oriented Software Development, March 20-24, 2006, New York, pp: 158-168.

Cottenier, T., A. van den Berg and T. Elrad, 2007. Join point inference from behavioral specification to implementation. Proceedings of the 21st European Conference on Object-Oriented Programming, July 30-August 3, 2007, Berlin, Germany, pp: 476-500.

Dennis, W. and L. Bergmans, 2002. Using a concept-based approach to aspect oriented software design. Proceedings of the 3rd International Workshop on Aspect Oriented Software Development, April 22-26, 2002, Enschede, The Netherlands.

Filman, R.E., 2005a. Aspect Oriented Software Development with Java Aspect Components. In: Aspect Oriented Software Development, Filman, R.E. (Ed.). Addison-Wesley, NY., USA.

Filman, R.E., 2005b. Expressing Aspects Using UML Behavioral and Structural Diagrams. In: Aspect-Oriented Software Development, Filman, R.E., T. Elrad, S. Clarke and M. Aksit (Eds.). Addison-Wesley, New York, USA., ISBN-13: 9780321219763, pp: 459-478.

Foxley, E., O. Salman and Z. Shukur, 1997. The automatic assessment of Z specifications. Proceedings of the Conference on Integrating Technology into Computer Science Education: Working Group Reports and Supplemental, Uppsala, Sweden, June 1-5, 1997, ACM, New York, USA., pp: 129-131,.

Groher, I. and S. Schulze, 2003. Generating aspect code from UML models. Proceedings of 4th International Workshop on AOSD Modelling with UML, October 2003, San Francisco, CA., USA .

Groher, I. and T. Baumgarth, 2004. Aspect-orientation from design to code. Proceedings of the Workshop on Aspect-Oriented Requirements Engineering and Architecture Design, March 21, 2004, Lancaster, UK.

Grundy, J., 2000. Multi-perspective specification, design and implementation of software components using aspects. Int. J. Software Eng. Knowledge Eng., Vol. 10,

Ho, W., J. Jezequel, F. Pennaneac and N. Plouzeau, 2002. A toolkit for weaving aspect oriented UML designs. Proceedings of the 1st International Conference on Aspect-Oriented Software Development, April 22-26, 2002, Enschede, The Netherlands, pp: 99-105.

Iqbal, S. and G. Allen, 2007. Aspect-oriented modeling: Issues and misconceptions. Proceedings of the 5th International Conference Software Engineering Advances, August 25-31, 2007, French Riviera, France, pp: 337-340.

Jacobson, I. and P.W. Ng, 2005. Aspect-Oriented Software Development with use Cases. Addison-Wesley, New York, ISBN: 9780321268884, Pages: 418.

Jose, H., F. Sanchez, F. Lucio and M. Toro, 2000. Introducing separation of aspects at design time. Proceedings of the 14th European Conference on Object-Oriented Programming, June 11-12, 2000, NY., USA.

Kande, M., 2003. A concern-oriented approach to software architecture. Ph.D. Thesis, Swiss Federal Institute of Technology (EPFL). Lausanne Switzerland.

Katara, M., 2002. Superposing UML class diagram. Proceedings of the Workshop on Aspect-Oriented Modeling with UML Model-Driven Development, September 30, 2002, Germany.

Katara, M. and S. Katz, 2007. A concern architecture view for aspect-oriented software design. Software Syst. Model., 6: 247-265.

Klein, J., F. Fleurey and J. Jezequel, 2007. Weaving Multiple Aspects in Sequence Diagrams. In: Transactions on Aspect-Oriented Software Development III, Rashid, A. and M. Aksit (Eds.). Springer-Verlag, Berlin, Heidelberg, pp: 167-199.

Magableh, A.A. and Z.M. Kasirun, 2007. Collaborative aspect-oriented requirements tool. Proceedings of the 3rd Malaysian Software Engineering Conference: Striving for High Quality Software, December 3-4, 2007, Selangor, pp: 12-17.

Marco, M., C. Anis, S. Jaroslav and W. Jan, 2008. Applying and evaluating AOM for platform independent behavioral UML models. Proceedings of the AOSD Workshop on Aspect-Oriented Modelling, March 31-April 04, 2008, Belgium, pp: 19-24.

Muley, K., U. Suman and M. Ingle, 2010. Representing join point in UML using pointcut. Proceedings of the International Conference on Computer and Communication Technology, September 2010, Kerala, India, pp: 557-561.

Muller, A., 2004. Reusing functional aspects: From composition to parameterization. Proceedings of 5th Aspect-Oriented Modeling Workshop (AOM) in Conjunction with the UML, October 11, 2004, Lisbon, Portugal.

Omar, A., T. Elrad and A. Bader, 2001. A UML profile for aspect oriented modelling. Proceedings of OOPSLA Workshop on Aspect Oriented Programming, (AOP'01), USA.

Omar A., A. Bader and T. Elrad, 2002. Weaving with statecharts. Proceedings of the Workshop on Aspect-Oriented Modeling with UML, April 22-26, 2002, Ensehede, The Netherlands.

Przybylek, A., 2010. Separation of crosscutting concerns at the design level: An extension to the UML metamodel. Proceedings of the International Multiconference on Computer Science and Information Technology, October 18-20, 2010, Wisla, Poland, pp: 551-557.

Ramey, J. and P. Rao, 2011. The systematic literature review as a research genre. Proceedings of the Conference Seminar on Professional Communication, April 27-29, 2011, Portsmouth, UK.

Ramnivas, L., 2003. AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications Co., Greenwich, Greater London, UK., ISBN-13: 9781930110939, pages: 481.

Reddy, R., S. Ghosh, R. France, G. Straw and J.M. Bieman *et al*., 2006. Directives for Composing Aspect-Oriented Design Class Models. In: Transactions on Aspect-Oriented Software Development I, Rashid, A. and M. Aksit (Eds.). Springer, New York, USA., ISBN-13: 9783540329725, pp: 75-105.

Sharafi, Z., P. Mirshams, A. Hamou-Lhadj and C. Constantinides, 2010. Extending the UML metamodel to provide support for crosscutting concerns. Proceedings of the 8th ACIS International Conference on Software Engineering Research Management and Applications, May 24-26, 2010, Montreal, QC., Canada, pp: 149-157.

Shukur, Z. and N.F. Mohamed, 2008. The design of ADAT: A tool for assessing automata-based assignments. J. Comput. Sci., 4: 415-420.

Stein, D., S. Hanenberg and R. Unland, 2002. An UML based aspect-oriented design notation. Proceedings of the 1st International Conference on Aspect-Oriented Software Development, April 23-26, 2002, Enschede, The Netherlands.

Suzuki, J. and Y. Yamamoto, 1999. Extending UML with aspects: Aspect support in the design phase. Proceedings of the Workshop on Object-Oriented Technology, June 14-18, 1999, Lisbon, Portugal, pp: 299-300.

Uetanabara J., P. Parreira, A. Lazanha, R. Camargo and R. Penteado, 2009. A preliminary comparative study using UML-AOF-A UML Profile for aspect-oriented frameworks. Proceedings of the 8th ACM on Aspect-Oriented Software Development, March 2-6, 2009, Charlottesville, Virginia.

Zhang, G., 2005. Towards aspect-oriented class diagrams. Proceedings of the 12th Asia-Pacific Software Conference on IEEE Computer Society Engineering, December 15, 2005, Washington, DC., USA., pp: 763-768.

Zhang, J., Y. Chen, G. Liu and H. Li, 2009. An aspectual state model and its realization based on AOP. Proceedings of the WRI World Congress on Software Engineering, vol 3, May 19-21, 2009, Xiamen, Fujian, China, pp: 163-166.