Journal of Theoretical and Applied Information Technology
30th November 2019. Vol.97. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

CONTEXT-FREE GRAMMAR FOR ASPECT-ORIENTED UML DESIGN MODELING DIAGRAMS

AWS A. MAGABLEH
Faculty of Computer Science and Information Technology, Department of Information Systems
Yarmouk University, Irbid, Jordan
E-mail: aws.magableh@yu.edu.jo

ABSTRACT

It is well known that aspect orientation (AO) has the potential to support the continued smooth running of software programs. In AO, before developing a program that may need to be updated all the aspects (crosscutting concerns) contained therein must be meticulously assessed to ensure that a change to one or more of those aspects will not have an adverse effect on other parts of the program. To address this issue, in this paper, three main objectives are targeted. First, a formal representation for aspect-oriented unified modeling language (UML) design modeling diagrams is proposed in which context-free grammar (CFG) is used for the aspects. An aspect model encompasses pointcuts, advice, inter-model declarations and aspect precedence, as well as references the behaviors of other classes and aspects. To ensure that there is consistency in a system, the aspect-oriented UML design model of the system is converted into a CFG that consists of set of rules for all the strings that could be present in the formal language being assessed. Second, the extended Backus–Naur form (EBNF) is applied to represent the CFG rules for the aspect-oriented model. Third, the potential use of the proposed EBNF transformation for all aspect-oriented UML diagrams is investigated. This study is inspired by the results of existing research on object-oriented UML transformation using EBNF. As AO is an extension of object orientation, it seemed natural to extend the idea of using EBNF to AO and assess whether it would be beneficial in transforming aspect-oriented UML modeling diagrams.

Keywords: *Context-Free Grammar, CFG, Aspect Orientation, AO, Extended Backus–Naur Form, EBNF, Model Transformation.*

1. INTRODUCTION

Aspect-oriented software development (AOSD) is attracting attention because it can be used to deal with the crosscutting concerns that may affect other concerns (classes) or functionalities in a program. As the name implies these crosscutting concerns, also known as aspects, can be found throughout code, which makes it difficult if not impossible to decompose them during the analysis, design or implementation stage of the software life cycle. Furthermore, if left undealt with, their presence can result in problems of code duplication and/or significant dependencies between systems, which are also known as scattering and tangling, respectively. Both of these problems have serious implications for the smooth running of the program. However, these problems can be addressed by using AOSD to analyze, model and program the aspects.

As unified modeling language (UML) is widely used to model object-oriented designs, it seems not only natural but essential that UML is also used in the modeling of aspect-oriented designs. For successful AOSD it is necessary to conduct a precise aspect analysis and create a meticulously detailed design. Thus far, the methods that have been proposed for aspect-oriented design modeling have concentrated on producing formalisms for the specification of the various aspects. Some methods have also been introduced to address the issue of UML inter-consistency, i.e., the consistency between UML diagrams [1]. A number of methods can be used to determine UML inter-consistency. However, the method adopted in this paper is the transformational method. This method involves transforming UML diagrams into context-free grammar (CFG). When a UML diagram can be correctly transformed into a CFG, i.e., when it complies with all the specified semantic rules in the CFG, this indicates that it has consistency [1].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In this paper, a formal specification for assessing and representing aspect-oriented UML design modeling diagrams is proposed. The diagrams thus produced contain an accurate detailed representation of the main aspect orientation (AO) concepts such as advice, pointcuts, and join points, as well as a selection of other AO concepts. These diagrams are then subjected to a transformation process using a context model and the extended Backus–Naur form (EBNF) to create the production rules of the CFG. Previous experiments [2] and the experiment presented herein demonstrate that the assessment of the aspect-oriented model using CFG is highly accurate and can thus ensure that the quality of aspect-oriented UML design diagrams is maintained.

The rest of this paper is organized as follows. In Section 2, an overview of the three main concepts, namely AO, CFG and EBNF, is presented. Next, in Section 3, the relevant literature on the domain under study is briefly reviewed. Then, in Section 4 the proposed aspect-oriented UML diagram representation using CFG is described. The paper ends with Section 5 in which some conclusions are drawn and some directions for future work are suggested.

2. BACKGROUND

2.1 Aspect Orientation

Aspect-oriented programming is utilized to deal with the issue of crosscutting concerns or aspects that can be found throughout a software system. There are various types of aspects including those that concern security, logging in, and synchronization. As these concerns tend to be not only scattered, but tangled across an entire system, it is very difficult to make even minimal improvements to code without there being unforeseen negative consequences for other parts of the system. AO additionally tells how these aspects ought to be woven into the system. The concept of AOP and the first type of AOP was introduced by the Xerox Palo Alto Research Center (Xerox PARC) in the late twentieth century. Notable types of AOP that are currently in use are AspectC, AspectC++ and AspectJ [32] [33]

The AOP language specification consists of four main elements:

- Join points: Locations in the main code where crosscutting concerns exist;
- Pointcuts: Instructs AOP to coordinate up join point; to do this, AO characterizes a designator that takes the join point as a parameter;
- Advice: Code that is injected into the original code before or after a join point or around join points;
- Aspect: The modular unit that collates and encapsulates the above three elements into one unit.

In the last few years, AOP has been a key factor in researchers' attempts to try to find a way to successfully deal with AO across the entire software life cycle, not just at the initial development stage. One such approach is AOSD (Shanmughaneethi et al., 2012), which, as the name implies, considers the issue and effects of aspect orientation (AO) in all stages of software development.

2.2 Context-free Grammar

Context-free grammar is a simple mathematical mechanism that can be used to represent the parts of a sentence in a natural language as small blocks. Thus, a complete sentence is represented as a block structure. This type of grammar is a free syntax that is easy to use and facilitates the formalism of grammar in mathematical studies. It should be noted that while the agreements and references [3] found in a natural language are not considered in CFG, it does describe the fundamental recursive arrangement of sentences and the pattern of clauses within sentences accurately. Context-free grammar, which is a free syntax, can be defined as a formal grammar that has a set of rules for all the strings that may be present in a formal language. The CFG rules are used to generate patterns of strings to represent that language.

Figure 1illustrates the operands of grammar, G, in CFG, which are denoted as V, Σ, R, and S, where[4]:

- Vis a finite set of nonterminal symbols, where each element $v \in V$ is called a nonterminal variable and each of these variables denotes a different part of a given sentence.
- Σ is a finite set of terminal symbols, which are disjointed from V, and make up the content of the statement. This set of terminal symbols constitutes the alphabet of the language defined by the grammar, G.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

- Represents a finite relation from V to $(V \cup \Sigma)^*$, where the asterisk is a unary operation that represents either sets of strings or sets of symbols or characters. The members of Rare the production rules of G.
- Sis the start variable and/or start symbol and, as it is meant to characterize the whole sentence it must be an element of V (i.e., part of the finite set of nonterminal symbols).

$$G = (V, \Sigma, R, S)$$

Figure 1: The CFG equation

2.2.1. The Original and the Extended Backus–Naur Form

One way in which the production rules of CFG are written is the Backus–Naur Form (BNF). The BNF is a formal notation that is used to encode or rewrite the grammar so that humans can understand it. Many programming languages, protocols and formats have a BNF description in their specification [34]. All the BNF rules have a (*Name::=Expansion*) structure, where *::=*means "may expand into" and "may be replaced with.". The *Name* is also known as a *nonterminal symbol*. Every *Name* in the BNF is surrounded by angle brackets <> regardless of whether it appears on the left or right of the rule. The *Expansion* contains *terminal* and *nonterminal* symbols. These are linked together by sequencing and choices, where each choice is represented by a vertical bar.

The EBNF is an expansion of the BNF, which includes further expressions to represent additional operations. There is little difference between the BNF and the EBNF in terms of syntax; rather, the latter provides extra flexibility in terms of representation. The EBNF has more expressions such as: *Options (<term> ::= ["-"] <factor>),Repetition(<args> ::= <arg> { ","<arg> }), Grouping (<expr> ::= <term> ("+" | "-") <expr>)*and *Concatenation (using "the")* [5]. Given this advantage, it seemed logical to adopt the EBNF for this study. The definitions of the grammar rules for the EBNF are shown in Figure 2.

$$\begin{aligned} rule &\rightarrow name ::= expansion \\ name &\rightarrow \langle identifier \rangle \\ expansion &\rightarrow expansion\ expansion \\ expansion &\rightarrow expansion \mid expansion \\ expansion &\rightarrow name \\ expansion &\rightarrow terminal \end{aligned}$$

Figure 2: Grammar rules in Extended Backus–Naur Form

Manuscripts must be in English (all figures and text) and prepared on Letter size paper (8.5 X 11 inches) in two column-format with 1.3 margins from top and .6 from bottom, and 1.25cm from left and right, leaving a gutter width of 0.2 between columns.

3. LITERATURE REVIEW

Context-free grammar has been used in computer science for a variety of areas including diagram editors [35], parsing [6][7], formal method [36]. However, for the purpose of this paper, the discussion focuses on the use of CFG in diagram transformation generally as well as in Aspect-oriented UML diagram transformation, and also the benefits of using CFG for AO.

One of the first attempts to translate diagrams into a formal language was made by [8] in 1999, in which the authors proposed a procedure that involves scanning a diagram to convert it into a spatial relationship diagram, which is then translated into a hypergraph model and lastly into a formal representation. This procedure is illustrated in Figure 3 below.

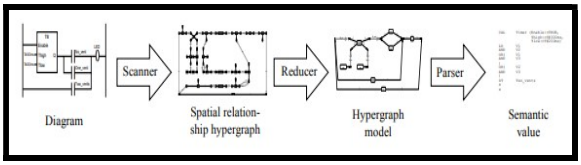

Figure 3: Example of translating a diagram into a formal language

Different studies have been proposed in the field of object oriented and aspect oriented with formal specification such as domain-specific languages. For example, in [9] the authors extended the LISA specification language through the addition of aspect-oriented features with the aim of improving the inheritance, modularity and extensibility of LISA, as well as developing a language specification that could be reused multiple

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

times. They named their approach Aspect LISA For the purpose of their proposed approach they also formally defined aspect-oriented attribute grammar (AspectAG). The way in which AspectAG differs from CFG in terms of definitions is shown in Figure 4.

AspectAG = (G, A, R, Pc, Ad):

- Pc (**pointcut productions**) = {pc_1, ..., pc_m}, where pointcut production pc_i, $1 \le i \le m$, has the following form: $pc_i < X_1, ..., X_r >$: LHS → RHS
- Ad (**advice**)= {ad_1, ..., ad_l}, where advice ad_k, $1 \le k \le l$, has the following form: $ad_k < S_1, ..., S_r >$ on pc_i {Rs_k}
- Set of attributes = A
- Set of Semantic Rule R
- Context-free grammar G

Figure 4: Definitions of main AspectAG terms

The aspect-oriented attribute grammar was soon employed in AO compilers [18]. Then, some years later, in 2005, this type of grammar was more clearly defined in [13], in which the authors describe an attribute grammar as a generalization of a CFG in which each symbol has an associated set of attributes that carry semantic information. They add that attribute values are defined by attribute evaluation rules that are linked to each production rule of the CFG. The rules are applied when computing the values of attribute occurrences as a function of some other attribute occurrences. Also, the semantic rules are localized for each CFG production. The attribute grammar can be formally represented by the following components: a CFG(G), a set of attributes (A), and a set of semantic rules ([RAG = (G, A, R]).

In another line of related prior research, it was suggested that a text-based method could be used to define the syntax of a graphic specification language such as UML [26]. The authors of that work, which was published in 2003, also defined a context-free syntax of this textual language in EBNF.

Over the years other methods have been proposed for the representation of CFG, including the grammar flow graph (GFG) [5]. The GFG is a direct graph in which each production rule of the CFG is reformulated as a node. The progression of the production rule or node is denoted by a preceding dot (.). The authors state that any CFG G can be transformed into a corresponding GFG in O(|G|) space and time. Figure 5 shows how |G| denotes the size of a CFG.

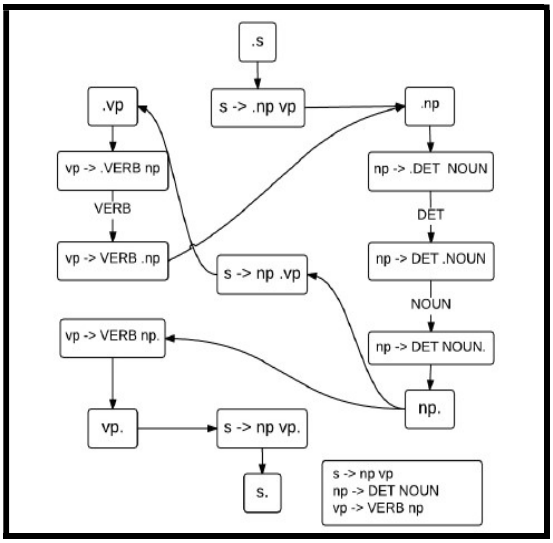

Figure 5: Example of grammar flow graph for CFG

As regards the issue of transformation, in [27], the main focus was on transforming an object orientation into a formal representation using CFG. On the other hand, in [29] and [30], the authors concentrated on trying to translate UML diagrams into a formal specification. Other works such as [31] have proposed sets of rules to convert UML modeling into formal representations by employing the concept of the CFG. Figure 6 shows an example of how the classes in UML can be transformed into CFG.

```
class Pump is
  class-sort Pump
  sort Pump-State
  attributes
    pump-id : Pump -> integer
    pump-state : Pump -> Pump-State
  operations
    attr-equal : Pump, Pump -> Boolean
  states
    pump-disabled : -> Pump-State
    pump-enabled : -> Pump-State
  events
    enable-pump :  Pump, integer -> Pump
    disable-pump :  Pump -> Pump
    new-pump :  integer -> Pump
  methods
```

Figure 6: Transformation of UML classes into CFG

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. RESEARCH QUESTIONS

This research will attempt to answer the following research questions that is well-articulated on problem statement and evaluated based on the nature and extent of information available of the parameters of the research:

RQ1: What is the current state of art of converting Aspect-Oriented and Object-Oriented design using UML into formal methods using context free grammeme?

RQ2: How possible to attempt to propose a new representation of Aspect-Oriented UML design to be converted to formal models using context free grammar (EBNF) to ensure better accurate of the models?

5. RESEARCH DESIGN

A research design has been defined as "a blueprint for conducting a study with maximum control over factors that may interfere with the validity of the findings" [17]. It has also been described as "a plan that describes how, when and where data are to be collected and analyzed" [17]. Our research design is a mixed methods approach that combines qualitative research techniques. The design consisted of few phases: 1) **Theoretical Study** where we have review of the literature (journals, books and conferences proceedings) to study all existing AO and OO related approaches that investigated the techniques of concern design to formal methods. 2) **Ideas and suggestions** where we attempted the development of a research design and model, development of a research methodology and the development of rules of transportations.

6. CONTEXT-FREE GRAMMAR FOR ASPECT-ORIENTED UML DIAGRAMS

The existing works and studies have focused more on object oriented when it comes to the field of converting object-oriented design into a formal and mathematical model. Also, existing literatus have focused mainly on building objects and aspects in term of context free grammar however this study is going to focused on capturing all parts of aspects, aspect-aspect relationships, aspects posterization and aspect relationships in the application to be converted to a formal language using Backus–Naur form (EBNF), additionally, other literatures have focused on converting some aspects design, however this study is giving a proposal to convert all UML design diagrams (behavioral and structural diagrams).

According to the principle regarding the separation of concerns [19] [20] each concern should be dealt with separately. Here, the term concern refers to something that is of specific interest to a stakeholder, and additionally in this context, that relates to the development of a system. This principle is based on human cognitive behavior, where a person is generally better able to think about and gain an understanding of one item at a time. This principle can be applied to achieve both a simplified form of project management. In fact, today, most software developers already apply this principle.

For instance, in the area of object orientation, applications are modeled and implemented by decomposing the problem and the solution space into objects, so that each object represents just one concern. Additionally, in the software development arena, the principles of encapsulation, polymorphism, inheritance, and delegation provide additional support to this process. However, in actuality, some concerns are present in a lot of diverse objects because they cannot sit within the confines of one particular object. These concerns include security, mobility, distribution, and resource management) and they necessarily crosscut other concerns.

Developers can apply AOSD to ease the identification, modularization, representation, and composition of these crosscutting concerns in a systematic way. This process is sometimes referred to as "aspectization" the end result of which is enhanced system modularity, which in turn enables developers to design systems that are easier not only easier to develop but can be maintained as well as changed with less difficulty. As mentioned above, an aspect consists of four key elements. The advice element defines the behavior of that aspect [21]. The join point element denotes where a given aspect could apply the advice in other parts of the system. Some of the most common join points can be found in the method execution, in object representations, or in the settings of attributes. Most aspect-oriented techniques aim to assist in the selection of appropriate join points by using some sort of declarative query mechanism. We call such a selection predicate a pointcut expression.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Essentially, AOSD techniques offer support to software developers in terms of abstraction, modularity, and composition which helps them to determine and deal with crosscutting concerns during the entire software life cycle. Thus, AOSD can be of benefit in the requirement engineering stage as well as in the design of the system architecture and additionally in the implementation and testing stages, and ultimately in the eventual evolution of the software. The advantage of using an AOSD technique lies in the enhanced reasoning that can be leveraged for solving a domain-specific problem. Such a technique can also reduce the size of the code needed for an application, and consequently, this has a positive impact on the cost of development and maintenance time in terms of both time and money. Importantly, the technique can also lead toa greater amount of code reuse, which is also cost-effective.

Unfortunately, due to its nature, UML [22] cannot directly support aspect-oriented modeling. Therefore, several researchers have attempted to apply the object-oriented paradigm followed by UML to address this issue. These works have, for instance focused on representing the AspectJ programming language features in UML [23] or on integrating aspects into UML 2.0 as components [24], A general summary of the work that has been conducted in this area is provided in [25].Overall, AOP seems to be a very promising as it has been utilized to good effect in a variety of language definition and implementation tools [11, 12, 13, 14, 15, 16, 10]. To generate a string of terminal symbols from a CFG, it is necessary to start with a string that contains the start symbol. The next step is to replace the start symbol on the left-hand side with the right-hand side of the production. Then, each of the nonterminal symbols in the string is replaced by the right-hand side of a corresponding production until all of them have been replaced by terminal ones.

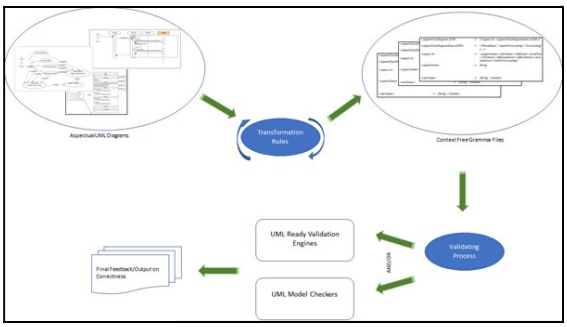

Figure 7: Proposed conceptual framework for transforming UML into AO

UML Inter-consistency defined as the concept of checking and approving the consistency between UML diagrams. There are different methods to check on UML Inter-consistency. The method adopted here to check on the intra-consistency is the 'Transformational Method'. Transformational method defined as the method which concerned checking consistency by transforming one UML diagram specification to another language such as Z-language or context free grammar. This method stated that, UML diagram is consistent when it conforms to the semantics of all context-free grammar rules. If a diagram can be correctly transformed to a context free grammar, then it means that the diagram in consistent. In this paper, for the sake of space, CFG generations for two aspectual UML structural diagrams and two for aspectual UML behavioral diagrams are provided by way of illustration.

Tables 1 and 2 contain the CFG for the aspectual UML class diagram and the aspectual object diagram, respectively.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: CFG for the Aspectual UML Class Diagram

<AspectClassDiagram ACD>	—	*(<Aspect A><AspectClassDiagramSuite ACDS>)+*
<AspectClassDiagramSuite ACDS>	—	*('Precedence' | 'AspectCrosscutting' | 'Crosscutting') |* □
<Aspect A>	—	*<AspectName><Attributes><Methods><JoinPoint><Pointcut><BeforeAdvice><AfterAdvice><AroundAdvice><StaticCrosscutting>*
<AspectName>	—	*String*
<Attributes>	—	*(String | Number)*
<Methods>	—	*String*
<JoinPoint>	—	*String*
<Pointcut>	—	*String*
<BeforeAdvice>	—	*String*
<AfterAdvice>	—	*String*
<AroundAdvice>	—	*String*
<StaticCrosscutting>	—	*String*

Table 2: CFG of the Aspectual Object Diagram

<AspecObjectDiagram AOD>	→	*(<AspectObject><AspectObjectDiagramSuite>)+*
<AspectObjectDiagramSuite>	→	*('After' | 'After Throwing' | 'After Returning' | 'Before' | 'Around' | 'method call' | 'Method Execution' | 'Method Get' | ' Method Set' | ' Constructor Call' | ' Constructor Execution' | 'Constructor Initialization' | 'Constructor Preinitialization' | ' Static Initialization' | 'Handler' | ' Advice Execution' | ' Within' | ' Method Within Code' | 'Constructor Within | 'CFlow' | ' CFlow Below' | 'This' | 'Target' | 'Argus") <Component>|* □
<AspectObject>	→	*(<AspectObjectName><JoinPoint><Pointcut>)*
<AspectObjectName>	→	*String*
<JoinPoint>	→	*String*
<Pointcut>	→	*String*

The above process can be applied to other aspectual UML structural diagrams to transform them into CFG. These UML diagrams include the aspectual package diagram, aspectual composite diagram, aspectual component diagram and aspectual deployment diagram.

Tables 3 and 4 contain the CFG of the aspectual UML use case diagram and the aspectual activity diagram, respectively. Other aspectual UML behavioral diagrams such as the aspectual state machine diagram, aspectual sequence diagram, aspectual communication diagram, aspectual interaction overview diagram and aspectual timing diagram follow the same pattern.

Table 3: CFG for Aspectual Use Case Diagram (AUCD)

<AspectUseCase Diagram AUCD>	→	*<AspectUCInstance><AspectUCInstancSuite><Relation P>+*
<AspectUCInstancSuite>	→	*→ ('After' | 'After Throwing' | 'After Returning' | 'Before' | 'Around' | 'method call' | 'Method Execution' | 'Method Get' | ' Method Set' | ' Constructor Call' | ' Constructor Execution' | 'Constructor Initialization' | 'Constructor Preinitialization' | ' Static Initialization' | 'Handler' | ' Advice Execution' | ' Within' | ' Method Within Code' | 'Constructor Within Code' | 'CFlow' | ' CFlow Below' | 'This' | 'Target' | 'Argus") |* □
<Relation P>	→	*('use.before' | 'use.after' | 'use.around')*
<AspectUCInstance>	→	*('Advice:Around | 'Advice:After | 'Advice:Before')*

Table 4: Aspectual Activity Diagram (AAD)

<AspectActivity Diagram>	→	*<AspecActivityInstance><AspectActivityInstanceSuite><Relation L>+*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

<AspectActivity InstanceSuite>	→	*('After' | 'After Throwing' | 'After Returning' | 'Before' | 'Around' | 'method call' | 'Method Execution'| 'Method Get' | ' Method Set' | ' Constructor Call' |' Constructor Execution' | 'Constructor Initialization' | 'Constructor Preinitialization' | ' Static Initialization' | 'Handler' | ' Advice Execution' | ' Within' |' Method Within Code' | 'Constructor Within Code' | 'CFlow' |' CFlow Below' | 'This' | 'Target'| 'Argus'') |*
<Relation L>	→	*('After' | 'After Throwing' | 'After Returning' | 'Before' | 'Around') <AspecActivityInstance> →('AspectActivity | 'AdviceActivity | 'JoinPointActicity')*

7. CONCLUSION AND FUTURE WORK

In this paper, the main aim was to produce a CFG representation for both structural and behavioral aspect-oriented UML diagrams. The main output was CFG files that can be utilized by a range of model checkers. Due to the fact the less attention has been paid on the importance of concerning system design that might include aspects also to formal methods. During the course of this research, we modeled all UML aspect oriented behavioral and structural diagrams, we used EBNF language.

A future research direction that would build on the results presented herein would be to investigate whether it is possible to use only CFG source files to obtain aspect-oriented diagrams and thereby enable systems to be portable. Another potential research direction would be to attempt to automate the transformation of aspect-oriented diagrams from drawings into formal specifications (CFG). The ultimate aim would to integrate diagram drawing environments with a CFG auto generation tool and then to combine the output from such research with an accurate model checking and code generation methodology.

REFERENCES

[1] Zbigniew, H., Ludwik, K., Gianna, R., Jean, L. Consistency problems in UML-based software development, Proceedings of the 2004 international conference on UML Modeling Languages and Applications, p.1-12, October 11-15, 2004, Lisbon, Portugal.

[2] DianxiangXu , Izzat Alsmadi , Weifeng Xu, Model Checking Aspect-Oriented Design Specification, Proceedings of the 31st Annual International Computer Software and Applications Conference, p.491-500, July 24-27, 2007

[3] Chen, W. Y., & Yang, H. R. (2018). A Context-free Grammar for the Ramanujan-Shor Polynomials. *arXiv preprint arXiv:1810.02732*.

[4] Minas M. and Viehstaedt G.. Diagen: A generator for diagram editors providing direct manipulation and execution of diagrams. Proc. IEEE VL'95, pp. 203-210, 1995.

[5] Berstel, J. (2013). Transductions and context-free languages. Springer-Verlag.

[6] Yonezawa, A., &Ohsawa, I. (1988, August). Object-oriented parallel parsing for context-free grammars. In Proceedings of the 12th conference on Computational linguistics-Volume 2(pp. 773-778). Association for Computational Linguistics.

[7] Bacławski, K., & DeLoach, S. A. (1998). Object-Oriented Parsing and Tran

[8] Minas, M. (1999, September). Creating semantic representations of diagrams. In International Workshop on Applications of Graph Transformations with Industrial Relevance (pp. 209-224). Springer, Berlin, Heidelberg.

[9] Rebernak, D., Mernik, M., Henriques, P. R., & Pereira, M. J. V. (2006). AspectLISA: an aspect-oriented compiler construction system based on attribute grammars. Electronic Notes in Theoretical Computer Science, 164(2), 37-53.

[10] Gray R., V.P. Heuring, S.P. Levi, A.M. Sloane, and W.M. Waite. Eli: A complete, flexible compiler construction system. Communications of the ACM, 35(2):121–131, 1992.

[11] Fors, N., & Hedin, G. (2015). A JastAdd implementation of Oberon-0. *Science of Computer Programming*, *114*, 74-84.

[12] Kalleberg, K. T., & Visser, E. (2006). Combining aspect-oriented and strategic programming. *Electronic Notes in Theoretical Computer Science*, *147*(1), 5-30.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[13] Kalleberg, K. T., & Visser, E. Combining aspect-oriented and strategic programming. In N.M.-O. HoratiuCirstea, editor, Proceedings of the 6th International Workshop of Rule-Based Programming (RULE).ENTCS, Nara, Japan, Elsevier, April 2005.

[14] Kollár, J., & Marcel, T. (2005). Temporal logic for pointcut definition in AOP. *Acta Electrotechnica et Informatica No*, *5*(1), 2.

[15] Wu, H., Gray, J., Roychoudhury, S., &Mernik, M. (2005, March). Weaving a debugging aspect into domain-specific language grammars. In *Proceedings of the 2005 ACM symposium on Applied computing* (pp. 1370-1374). ACM.

[16] Van Wyk, E. (2003). Aspects as modular language extensions. *Electronic Notes in Theoretical Computer Science*, *82*(3), 555-574.

[17] Mernik, M., &Rebernak, D. (2011). Aspect-Oriented Attribute Grammars. ElektronikairElektrotechnika, 116(10), 99-104.

[18] De Moor, O., Peyton-Jones, S., & Van Wyk, E. (1999, September). Aspect-oriented compilers. In International Symposium on Generative and Component-Based Software Engineering (pp. 121-133). Springer, Berlin, Heidelberg.

[19] Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(12), 1053-1058.

[20] Dijkstra, E. W., Dijkstra, E. W., Dijkstra, E. W., Informaticien, E. U., & Dijkstra, E. W. (1976). *A discipline of programming*(Vol. 1). Englewood Cliffs: prentice-hall.

[21] Colyer, A., & Clement, A. (2004, March). Large-scale AOSD for middleware. In *Proceedings of the 3rd international conference on Aspect-oriented software development* (pp. 56-65). ACM.

[22] Object Management Group. Unified Modeling Language Specification, Version 2.0 (Superstructure). Revised final adopted draft, OMG, 2004. http://www.omg.org

[23] Stein, D., Hanenberg, S., &Unland, R. (2002, April). A UML-based aspect-oriented design notation for AspectJ. In *Proceedings of the 1st international conference on Aspect-oriented software development* (pp. 106-112). ACM.

[24] Barra, E., Génova, G., &Llorens, J. (2004, October). An approach to Aspect Modelling with UML 2.0. In *Aspect-Oriented Modeling Workshop, AOM*.

[25] Robert E. Filman, TzillaElrad, Siobh´an Clarke, and Mehmet Aksit, editors. Aspect-Oriented Software Development. Addison-Wesley, 2004

[26] Xia, Y., &Glinz, M. (2003, December). Rigorous EBNF-based definition for a graphic modeling language. In Software Engineering Conference, 2003. Tenth Asia-Pacific (pp. 186-196). IEEE.

[27] Nakwijit, P., &Ratanaworabhan, P. (2015, November). A parser generator using the Grammar Flow Graph. In *Computer Science and Engineering Conference (ICSEC), 2015 International* (pp. 1-6). IEEE.

[28] Bacławski, K., & DeLoach, S. A. (1998). Object-Oriented Parsing and Transformation.

[29] Booch, G., Rumbaugh, J., & Jacobson, I. (1997). UML notation guide, version 1.1. *Rational Software Corporation, Santa Clara, CA*, *2*.

[30] Rumbaugh, J., Jacobson, I., &Booch, G. (2004). *Unified modeling language reference manual, the*. Pearson Higher Education.

[31] Babin, G., Lustman, F., &Shoval, P. (1991). Specification and design of transactions in information systems: A formal approach. *IEEE Transactions on Software Engineering*, *17*(8), 814-829.

[32] Laddad, R. 2003. AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications.

[33] Cherait, H., &Bounour, N. (2014). Detecting Change Patterns in Aspect OrientedSoftware Evolution: Rule-based Repository Analysis. International Journal of Software Engineering and Its Applications (IJSEIA), 8(1).

[34] Sterling, T., Anderson, M., &Brodowicz, M. (2017). *High performance computing: modern systems and practices*. Morgan Kaufmann.

[35] Costagliola, G., De Rosa, M., &Fuccella, V. (2015). Extending local context-based specifications of visual languages. *Journal of Visual Languages & Computing*, *31*, 184-195.

[36] Bastani, O., Anand, S., & Aiken, A. (2015, January). Specification inference using context-free language reachability. In *ACM SIGPLAN Notices* (Vol. 50, No. 1, pp. 553-566). ACM.