An Approach of Software Architectural Styles Detection Using Graph Grammar

Songpon Thongkum and Wiwat Vatanawood

***Abstract*—One of the challenging problems in software design is to evaluate the risks using the architectural styles.With the help of the architectural styles, it is able to reveal some potential design problems which do not conform to the non-functional requirements. The detection of a software architectural style is typically done by hand and it is a tedious work. However, there are still some ongoing researches on the automatic detection of the software architectural styles. In this paper, we propose an alternative scheme of the architectural styles detection using the reduction steps of a graph grammar. The definition of the context sensitive graph grammar and its derivation and reduction steps is extended and proposed to represent the typical components, interfaces and links in the software architectural model written in xADL. The xADL is one of the popular architectural description languages ever used. A case study of repository style detection scheme is demonstrated. The resulting derivation and reduction of the specific graph grammar show the valid parsability of the graph.**

***Index Terms*—Software architecture, xADL, detection architecture, graph grammar.**

I. Introduction

Today, the enterprise company is often developing a large information system based on the growth of the company. The complexity and large scale of enterprise software cause problems when the software needs to be extended. These problems can be solved by considering software architectural styles in the design model. The architectural style consists of a set of software components and their specific relationship among the components. The various performance features of the software system can be verified through design [1] choreographed by using diagrams.

After the architectural styles has been applied to systems,then the problems are more stable and easier to be solved or improved the complexity of the system. In case of the large scale and more complex software, it is difficult to investigate and detect the architectural styles. Thus, in this paper we introduce the usage of the context sensitive graph grammar to represent the software architectural model. Our approach focuses on the architectural model written in xADL, one of the well known architectural description languages (ADL). The xADL statements are analyzed and the corresponding graph grammar will be equivalently generated. The output graph grammar will be parsed and reduced into the possible target of the predefined architectural styles.

In this paper, we introduce a scheme to define the architectural style based graph grammar (ASGG) and its production rules. A sample of derivation and reduction steps is demonstrated to detect a simple repository styles. We also consider the transformation between a given software architectural model written in xADL and the corresponding graph grammar. This paper is organized as follows. The introduction is described in Section I. Section II reviews the related works and backgrounds. Section III describes our proposed architectural style based graph grammar (ASGG). Section IV describes our Architectural Style Detection Scheme and Section V is the conclusion.

II. Related Work

The graph grammars have been studied by many researchers [2], [3] and many variation of graph grammar approaches [4]-[6] have been presented. Typically, the graph grammar is the graph rewriting system, and the formal graph language is defined to enumerate all graphs from some starting graph. The context sensitive graph grammar is more effective to almost applications. The context sensitive is more expressively and flexibly formalize the graph syntax rules of visual language. The Node-label-controlled (NLC) graph grammars, defined by [4], are one of the most interesting families of graph rewriting formalisms. Moreover, there also is the context sensitive graph grammar with neighborhood controlled embedding (CS-NCE) [7] to be used to formalize syntactical rules of a wide range of visual language.

Among the mentioned researches, [5] introduced a graph grammar approach to software architectural verification and transformation. The graph grammar is made of a set of rewriting rules, called productions. A production is a rewriting rule with two equivalent graphs called left graph and right graph as shown in Fig. 1.

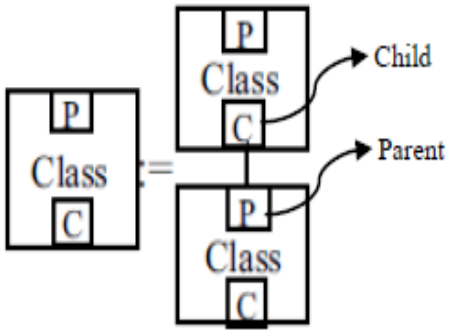


Fig. 1. Graph grammar characteristic

Manuscript received May 23, 2013; revised July 20, 2013.

Songpon Thongkum and Wiwat Vatanawood are with the Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University Bangkok, Thailand (e-mail: Songpon.T@Student.chula.ac.th; and e-mail: wiwat@chula.ac.th).

III. Our Architectural Style Based Graph Grammar

In this section, we formalize a context sensitive graph grammar for detecting architectural styles, called ASGG - Architectural Style based Graph Grammar, which is based on CS-NCE graph grammar in [7]. A number of definitions and algorithm are introduced as follows.

Definition 1: Architectural Style Name *AS* is considered as a finite nonempty set of start symbols in the proposed graph grammar. Typically, each start symbol is classified as a non-terminal node in the graph grammar. In our approach, each start symbol represents an architectural style name which will be detected. The architectural style name *AS* is formally defined as *AS = {REPOSITORY, PIPE&FILTER, ...}* where *REPOSITORY* is the target architectural style named "Repository" and *PIPE&FILTER* is the architectural style named "Pipe and Filter".

Definition 2: Graph is a 2-tuple $G = (V, E)$ where V is a finite nonempty set of nodes and E is a set of edges.

Definition 3: An Architectural Style based Graph Grammar is a 4-tuple $ASGG = (\Psi, \Sigma, \Gamma, P)$ where $\Psi \in AS$. We define $\Sigma = \Sigma N \cup \Sigma T$ and $\Sigma N \cap \Sigma T = \phi$. ΣN is a set of non-terminal nodes and ΣT is a set of terminal nodes *AE*. Γ is a set of edges on $\Sigma x \Sigma$. Each edge is a 2-tuple *(n1, n2)* where $n1 \neq n2$. P is a finite nonempty set of productions. Each production is written in term of $L ::= R$ where L is either single start symbol Ψ or graph $G = (\Sigma, \Gamma)$ and R is graph $G = (\Sigma, \Gamma)$. For a production p written as $L ::= R$, L is called the left-hand side of p and is denoted as *lhs(p)*, while R is called the right-hand side of p and is denoted as *rhs(p)*

Definition 4: Derivation of ASGG is a sequence of derivation steps to transform a single start symbol Ψ into a new graph $G = (\Sigma, \Gamma)$. A derivation step is also called the graph rewriting based on the left application of the production rule P of the ASGG graph grammar. The graph G is transformed into G' by the left application of a production p and is denoted as $G \ p \Rightarrow G'$.

Definition 5: Reduction of ASGG is a sequence of reduction steps to reversely transform any graph $G = (\Sigma, \Gamma)$ into a single start symbol Ψ. A reduction step is called the right application of the production rule P of the ASGG graph grammar. The graph G' is reversely transformed into G by the right application of a production p' and is denoted as $G' \ p' \Rightarrow G$.

Definition 6: Terminal node *AE* is a set of architectural elements in software architectural model. In our approach, the architectural elements are component, interface according to the xADL language. Thus, the terminal node *AE = { compo, intf }*.

Definition 7: The ASGG for rewriting Repository Style. According to the ASGG definition, we formally define the start symbol, non-terminal and terminal nodes as shown in Fig. 2. The production rules are written and shown in Fig. 3

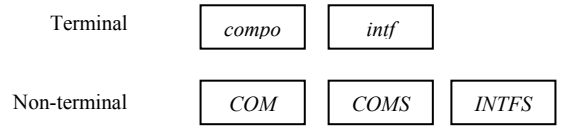


Fig. 2. The start symbol, non-terminal and terminal nodes of ASGG for repository style

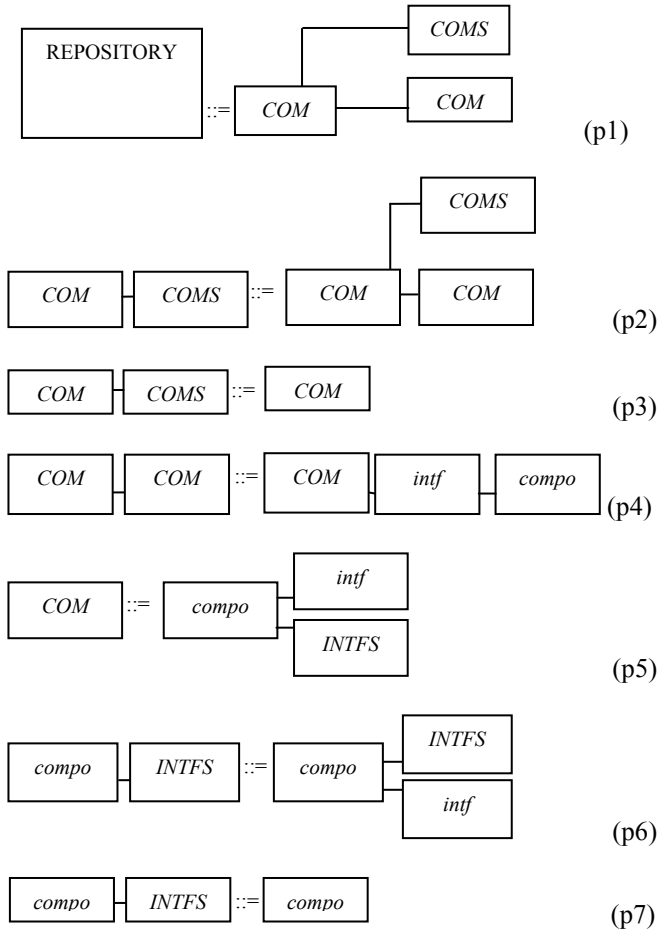


Fig. 3. The production rules of ASGG for repository style

IV. Our Architectural Style Detection Scheme

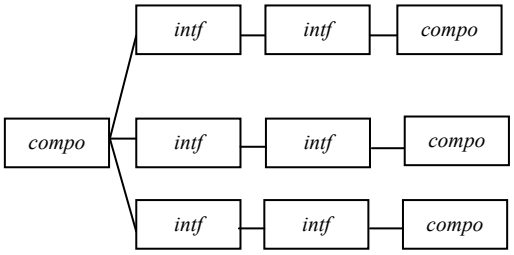
In this section, we will present how to detect software architectural style using ASGG. As we mentioned earlier, the given software architectural model is written in xADL provided by the designer. The algorithm 1 shows how to detect the expecting architectural style of Repository in the given xADL model.

Algorithm 1: The Architectural Style Detection for Repository.

Given a software architectural model *AMODEL* written in xADL as shown in Fig. 4 (a):

1) Extract a set of component elements *CE* which is specified by <component> tag in xADL.
2) For each component in *CE*, Extract a set of interface elements *IE* which is specified by <interface> tag in xADL.
3) Extract a set of link elements *LK* which is specified by <link> tag in xADL.
4) According to resulting *CE*, *IE*, and *LK* , a graph $G = (\Sigma, \Gamma)$ with only terminal nodes is generated as shown in Fig. 4 (b).
5) For each possible subgraph *SGi* of *G*, if *SGi* can be reduced into the start symbol REPOSITORY using the ASGG defined in definition 6, then a Repository style is detected and reported.
6) The Repository style is not founded if there does not exist the reducible *SGi* in step 5.

(a) Software architectural model in xADL



(b)The corresponding ASGG for xADL

Fig. 4. xADL and its corresponding ASGG

V. Demonstration of the Reduction of a Graph G = (Σ, Γ) to Detect Repository Style

As shown in Fig. 4(b), a Graph $G = (\Sigma, \Gamma)$ which is generated from a xADL model is to be reduced. The production rules in definition 6 are reversely applied to reduce the input G into a single start symbol *REPOSITORY* as shown in Fig. 5-23.

The initial graph is shown in Fig. 5 and it is considered to be reduced. The Fig. 6 showns the selected subgraph being reduced.

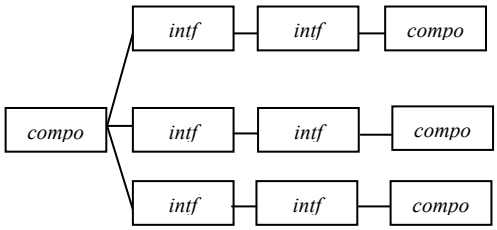


Fig. 5. The initial graph to be reduced

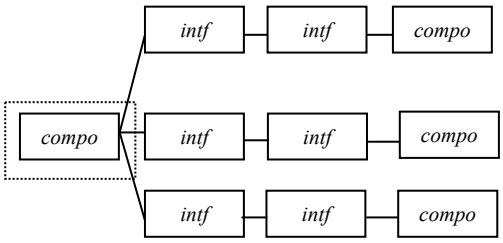


Fig. 6. The selected subgraph to be considered

Fig. 7 shows the rewritten subgraph after applying rule P7'.

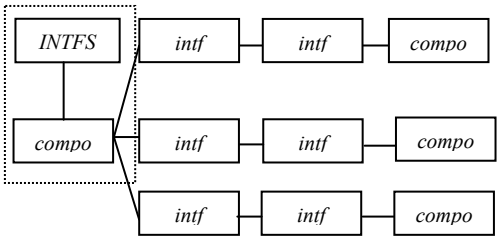


Fig. 7. The graph after applying P7'

Now we consider the subgraph shown in Fig. 8 and then applying the rule P6' yielding the result in Fig. 9.

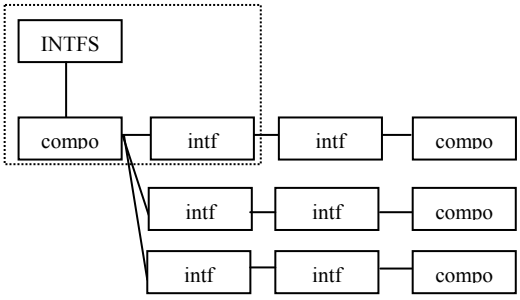


Fig. 8. The graph before applying P6'

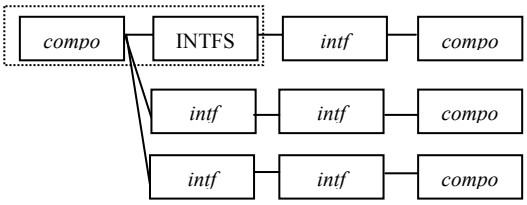


Fig. 9. The graph after applying P6'

The subgraph in Fig. 10 is then considered and the result shown in Fig. 11 is the result of applying rule P6'.

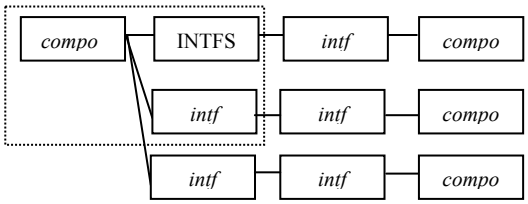


Fig. 10. The graph before applying P6'

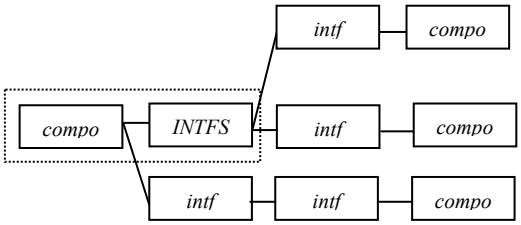


Fig. 11. The graph after applying P6'

The next subgraph to be considered is shown in Fig. 12. The rule P5' is applied as shown in Fig. 13.

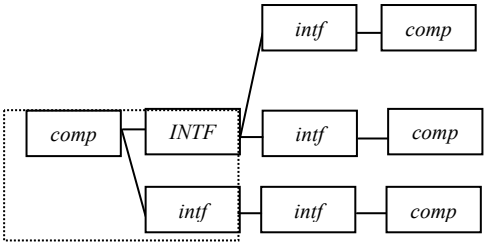


Fig. 12. The graph before applying P5'

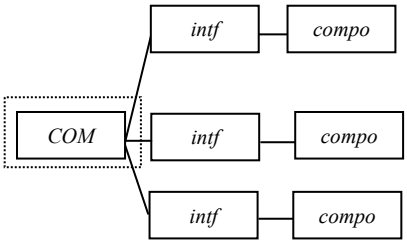


Fig. 13. The graph after applying P5'

By applying rule P4' to Fig. 14, we can change all of the non-terminals into the terminals shown in Fig. 15.

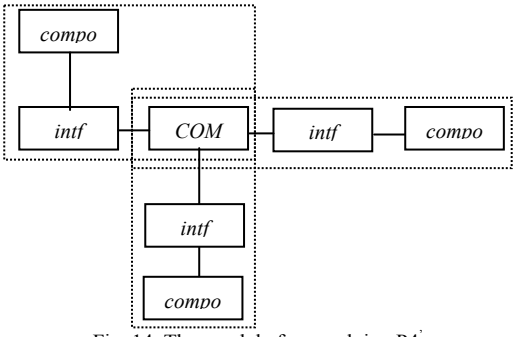


Fig. 14. The graph before applying P4'

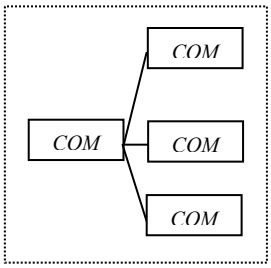


Fig. 15. The graph after applying P4' for three times

For the subgraph shown in Fig. 16, we use the rule p3' and the result is shown in Fig. 17.

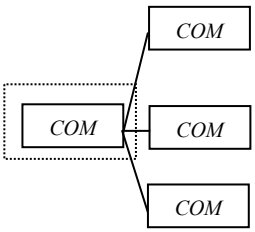


Fig. 16. The graph before applying P3

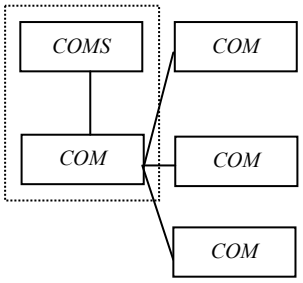


Fig. 17. The graph after applying P3'

Now we can group another COM into COMS shown in Fig. 18 by applying P2' and result after applying P2' is shown in Fig. 19.

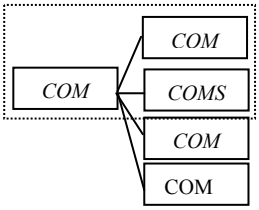


Fig. 18. The graph before applying P2'

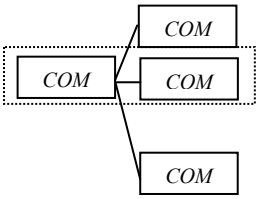


Fig. 19. The graph after applying P2'

We repeat the rule P2' again to Fig. 20 and then the result is shown in Fig. 21.

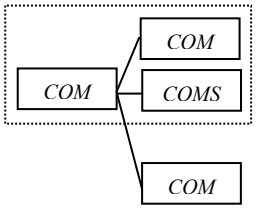


Fig. 20. The graph before applying P2'

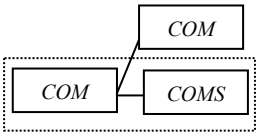


Fig. 21. The graph after applying P2'

Finally we can reduced the subgraph in Fig. 22 into single start symbol REPOSITORY by applying P1' and result after applying P1' is shown in Fig. 23

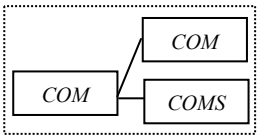


Fig. 22. The graph before applying P1'

Fig. 23. The final start symbol after applying P1'

VI. Conclusion

In this paper, we propose an alternative detection scheme of software architectural style. The context sensitive graph grammar is exploited and extended based on CS-NCE graph grammar in [7]. We formally define our Architectural Style based Graph Grammar – ASGG, to appropriately represent a software architectural model which consists of components, interfaces, and links. The detection scheme of Repository style is demonstrated. The ASGG along with derivation and reduction and the algorithm for detecting Repository style are

provided. The grammar graph is valid and parsable. However, the large graph can lead to the nontrivial problems in reduction steps.

REFERENCES

[1] E. M. Mahdieh Alemi and Hassan Rashidi,"Software architecture: A survey and classification," in *Proc. 2nd International Conference on Communication Software and Networks*, *Singapore,* 2010, pp. 454-460.

[2] J. Engelfriet and G. Rozenberg, Node Replacement Graph Grammar, in Handbook of Graph Grammars and Computing by Graph Transfonnation (Rozenberg, G., eds), World Scientific (1997), pp. 1-94

[3] M. Erwig, "Visual graphs," in *Proc. IEEE Symp*. Visual Languages, 1999, pp. 122-129.

[4] D. Q. Zhang, K. Zhang, and J. N. Cao, "A context-sensitive graph grammar formalism for the specification of visual language," *Computer journal*, *British: British Computer Society,* pp. 186-200, 2001.

[5] J. Kong, K. Zhang, J. Dong, and G. L. Song, "A graph grammar approach to software architecture verification and transformation," in. *Proc. the 27th Annual International Conference on Computer Software and Applications*, *USA: IEEE Computer Society,* 2003, pp. 492.

[6] Y. Shindo, K. Anada, and T. Yaku, " A Graph grammar model for syntaxesof financial statements," in. *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing*, *USA: IEEE Computer Society,* pp. 265-266, 2011.

[7] Y. Adachi and Y. Nakajima," A context-sensitive nce graph grammar and its parsability," in *Proceedings of 2000 IEEE International Symposium on Visual Languages, USA: IEEE Computer Society*, 2000, pp. 111 – 118.

Songpon Thongkum was born in Bangkok Thailand in 1986. He is a graduate student of Computer Engineering at Faculty of Engineering, Chulalongkorn University. His research interest is Software Engineering.

Wiwat Vatanawood is currently an associate professor of Computer Engineering at Faculty of Engineering, Chulalongkorn University. His research interests include formal specification methods, software architecture.