ABSTRACT

From the point of view of the software life cycle in Software Engineering, software architecture is the core of the structure and behavior of software. Thus software architecture design is bound to be the core of the software design, and the basis for the subsequent code development as well. The significance of software architecture design is self-evident. As software architecture design itself is a kind of modeling activity, one problem is raised: how to verify the correctness of the standard modeling language of software architecture design, UML. In addition, the drawbacks of the traditional software verification methods were discussed, like less accuracy and non-automation. To verify the correctness of UML, the formal semantics is needed in advance, while as a graphic notation, UML hardly has formal semantics. And then one research method was finally provided: using formal methods to describe UML formally and evaluate it an equivalent formal semantics in a formal language, and then verify the UML model formally according to its semantics. To further present the semantics of UML precisely, providing UML some OCL constraints is one major method. Generally, OCL constraints need to be written manually, and also, the correctness and overhead on personnel are necessary to be considered. So automatically generating OCL constraints template for UML models is badly required, which may be referred by software designers. Eventually, the overall efficiency of Software Engineering could be improved.

UML is the de facto standard of modeling language for software design process. From 4 aspects of the historical development, sub-graph types, modeling tools and the UML in an XMI notation, UML was briefly introduced, and the two UML sub-graphs discussed in this paper: UML class diagram and state diagram were specified in details. Next, the basic concepts of formal methods and its main branches were introduced, and the current research home and abroad on the formalizing of UML was summarized. Finally, OCL, the UML standard sub-language, situation calculus, the formal language used, and its realization – Prolog, a logic programming language, were introduced. The feasibility of transformation from UML to situation calculus was further analysed. Hence the correctness of the solution was guaranteed theoretically.

Therefore, the situation calculus based formal description of UML was put forward. First, the meaning of choice of UML class diagram and state diagram as research objects was given. And UML class diagram and state diagram were formally described respectively. At first the formal semantic structures of the two sub-graphs were given, and then the mapping mechanism between the elements of two sub-graphs and mathematical logic, and situation calculus was shown. The transformation algorithms of two sub-graphs to mathematical logic statements and Prolog code were also proposed and shown in terms of pseudo code. And a definition of two basic error types in UML was stressed: domain-independent grammatical errors and domain-dependent semantic errors. Specific examples and automatically generated Prolog code of these errors are given as well.

Furthermore, we discussed how to automatically generate OCL constraints templates for UML models. First of all, the meaning of automatic generation of OCL constraints was emphasized, and then the application domain of the OCL constraints in this paper was shown, followed by an analysis of how to extract the OCL constraints target application object from UML models and an efficient algorithm. Eventually, example code given, the extraction algorithm was implemented by Perl.

As a complement of the theory shown as above and a proof of feasibility, details of the realizing the proposed algorithm of UML-situation calculus transformation and the OCL constraint template automatic generation, that is, a prototype tool, USCVSC was designed and implemented. Firstly, the system implementation framework and code framework of this prototype tool were established; secondly, the user interface of the prototype tool was illustrated, and a more detailed description was given to the four basic sub-functional interfaces. Finally, a statement was made that through this prototype tool, the clients could do the checking of syntax errors and the verification of semantic errors of UML model, as well as OCL constraints template automatical generatation. All the features were integrated in this prototype tool.

Finally, the use of the USCVSC prototype tool was highlighted. Taking a university education system and university application system, a real case for examples, we demonstrated basic functions of the prototype tool. First the characteristics of the examples were described, and then UML class diagram and state diagram of the

examples were designed by UML modeling tool. At last the prototype tool, USCVSC is employed to verify the errors. As for the UML syntax error checking, it could be completed in USCVSC itself, while the UML semantic error verification required USCVSC and Prolog parser in combination. In the end, how to use USCVSC to automatically generate OCL constraint template for UML class diagram was demonstrated and some examples of OCL constraint statements were given.

Based on situation calculus as a formal language, the UML model is formally verified and OCL constraint is automatically generated. Thus, the Multiple Initial State syntax error and the No Logical Operator in Guard syntax error, etc in UML model can be found, and UML model semantic errors such as the Requirements Not Complete error and the Requirements Logic error are verified as well. In this way, it is convenient for software designers to correct the original UML model design, avoiding unnecessary system overhead in the following phases of Software Engineering. Ultimately overall efficiency of the various stages in Software Engineering is enhanced and contributions are made for the automation of Software Engineering.

KEY WORDS: UML; situation calculus; Prolog; OCL; formal verification; automatic generation