Management and verification of the consistency among UML Models

Atsushi Ohnishi(Department of Computer Science, Ritsumeikan University, Shiga 525-8577, Japan)

Abstract

We propose a method for managing and verifying the consistency among UML (Unified Modeling Language) models with knowledge-base verification items. We have been developing a system based on the proposed method. In this paper, the outline of the method will be described.

Keywords: Object-oriented analysis, UML, Management of the consistency among UML models, Object-oriented software development

1. Introduction

In object-oriented software development, we have to spend a lot of time and the labor for the analysis, the first phase of the development, because it needs much more cost to eliminate errors generated in the analysis phase in later phases of the development, such as design or installation phase. To lessen errors of UML models, we have been researching a computer assisted object-oriented analysis method.

This research can be divided into three stages. At the first stage, a Japanese requirements description is analyzed, and the errors (for example, the omission and vagueness) will be removed. At the second stage, UML models can be derived from the Japanese requirements. In the derivation of the models, standard conditions, which the UML models should satisfy, are introduced. We check the derived UML models satisfy the conditions and guarantee the validity of the each model. At the third stage, the consistency among these models is verified, and the models are refined. These techniques have been shown in proceedings of the OO technologies, IPS Japan, 1996, 1997, 1998. In this paper, we focus on the verification method among UML models.

The term "verification of the consistency" is used in the meaning "checking that the models has adjusted mutually with a finite and cost-effective process" in this research.

There exist several related researches. [2] enables the consistency verification by giving a formal meaning to not the UML models, but the OMT models. The consistency between models is not described though data flows in the function model do not have any inconsistency.

[10] proposes a method of verifying class models from which HOL code is derived with the HOL theorem proof system where behaviors of classes are described with state transition diagrams. This verification is limited to the correctness of assertions which states attribute values before/after some behavior.

There exist techniques which enables verification by formally giving semantics to UML models, such as pUML(precise UML) [4,6] and OCL(Object Constraint Language) approaches. The related researches above are approaches of formally giving semantics to UML models. Users should formally describe the semantics of the UML models as assertions or constraints. It is not possible to verify as long as the meaning is not described and only the described matter can be verified. When these formal descriptions are wrong, the verification result becomes wrong, too. Moreover, one of the advantages of object-oriented software development is reuse of software products generated in the development, but the semantics cannot be always reused to similar software developments.

On the other hand, our verification needs not users' giving semantics to the UML models. Semantics of the models are stored into a knowledge base as shown in Figure 1. Moreover, there are no problems when the UML model is reused. In addition, our method enables to verify the consistency among six models of UML of the object-oriented analysis, and to show how to verify the corrected models again when inconsistencies are detected.

2. Management of the consistency among UML models

We verify six kinds of UML models including class diagram, state chart, sequence chart, collaboration diagram, activity diagram, and use case diagram. These six models are mainly used in the analysis phase of object-oriented software development. We manage the consistency of the two models from the six models.

We check the correspondence by comparing strings of model components' names. With such correspondences, we verify the following items shown in Figure 1. We provide them as a knowledge base. In the verification of the consistency between a state chart and a class diagram, we use an extra table, which describes states and attributes defining the states. One state definition table is provided for each of state charts.

In case of verifying the consistency between two models from six models, we have to check 15 verification steps, because the combination ${}_6C_2$ is 15. However, there exist redundant verification steps in the 15 steps. So, we can

verify the consistency of the six models within 15 steps.

	Class diagram(C)	State chart(S)	Sequence chart(Q)	Collaboration diagram(L)	Use case diagram(U)
State chart (S)	1) Class of (S) and classes in (C) 2) confirmation of classes without state charts. 3) Attributes defining states in (S) and attributes in (C) 4) Range of attribute values in (C) and (S) 5) actions, activities in (S) and methods in (C)				
Sequence chart(Q)	1) Objects in (Q) and classes in (C) 2) Messages between objects in (Q) and associations between coressponding classes in (C) 3) Messages in (Q) and methods in (C)	1) Object in (Q) and class of (S) 2) Messages in (Q) and actions, activities in (S)			
Collaboration diagram(L)	1) Objects in (L) and classes in (C) 2) Messages between objects in (L) and associations between corresponding classes in (C) 3) Messages in (L) and method in (C)	1) Objects in (L) and class of (S) 2) Messages in (L) and actions, activities in (S)	1) Objects in (L) and in (Q) 2) Messages in (L) and in (Q) for directions, sequence, source, and destination		
Use case diagram(U)	1) Actors in (U) and classes in (C) 2) Use cases in (U) and methods in (C)	1) Actors in (U) and class of (S) 2) Use cases in (U) and actions, activities in (S)	1) Actors in (U) and objects in (Q) 2)use cases in (U) and messages in (Q)	1) Actors in (U) and objects in (L) 2) Use cases in (U) and messages in (L)	
Activity diagram(A)	1) Classes in (A) and in (C) 2) Actions in (A) and methods in (C) 3) Control flows between classes in (A) and associations in (C)	1) Classes in (A) and class of (S) 2) Actions in (A) and actions, activities in (S)	1) Classes in (A) and objects in (Q) 2) Actions in (A) and messages in (Q) 3) Control flows between classes in (A) and messages in (Q)	1) Classes in (A) and objects in (Q) 2) Actions in (A) and messages in (L) 3) Control flows between classes in (A) and message in (L)	1) Classes in (A) and actores in (U) 2) Actions in (A) and use cases in (U)

Figure 1 Knowledge-based verification items among UML models

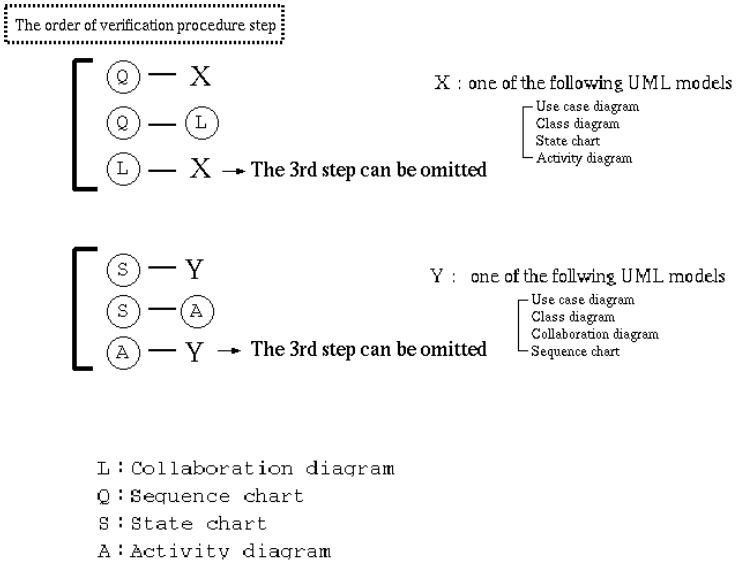

Figure 2 Redundant verification steps.

We define redundant verification steps in Figure 2. The first half part of Figure 2 shows that in three verification steps, that is, (a) verification between sequence chart(Q) and model X(either of use case diagram, class diagram, or activity diagram), (b) verification between sequence chart(Q) and collaboration diagram(L), (c) verification between collaboration diagram(L) and model X, if either two steps of the three steps have already done, the other step becomes redundant and can be omitted.

Similarly, the part of the latter half in Figure 2 shows that in three verification steps, that is, (a) verification between state chart(S) and model Y (either of use case diagram, class diagram, collaboration diagram, sequence chart), (b) verification between state chart(S) and activity diagram(A), (c) verification between activity diagram(A) and model Y, if either two step of the three steps have already done, the other step becomes redundant and can be omitted.

However, the verification between class diagram and state chart, which uses the state table, does not be omitted. For instance, the verification of sequence chart(Q) and the class diagram(C) becomes redundant and can be omitted, if in advance (i) verification between collaboration diagram(L) and class diagram(C) and (ii) verification between sequence chart(Q) and collaboration diagram(L) have already done.

When errors of the models can be detected with the verification of the consistency, model definer or describer can correct detected errors. The correction of errors of a model may influence other models and corrected model will be inconsistent with other models. Since each model of UML mutually has some relations, correction of a model needs to re-verify the consistency between the corrected model and other models. This means that the correction of a model whose errors are detected by our verification method needs re-verification among the corrected model and other models, even if other models have been already verified.

In this research, UML model definers or describers can specify the order of making models in the analysis with UML. Since some of UML models refer mutually, definers or describers can specify also the reference relations between models. We determine the order of the verification steps with both the order of making the models and the reference relations to lessen the number of the re-verifications mentioned above. Moreover, the navigation of the necessary re-verifications among a corrected model and other models becomes possible. The order of making the model in default is as follows.

(1) Use case diagram (to grasp use cases of the target system)
(2) Sequence chart (to grasp the order of messages among objects)
(3) Collaboration diagram (to grasp collaboration among objects)
(4) Activity diagram (to grasp the control flows of processes, operations)
(5) State chart (to grasp objects' states, state transitions, and events for transitions)
(6) Class diagram (to grasp static structures of classes)

If UML model definers may describe models with different order, they can define the order of making models and the reference relations between models. Next, a directed graph where a model referred chiefly to any other

models in the order of making this model is shown in figure 3.

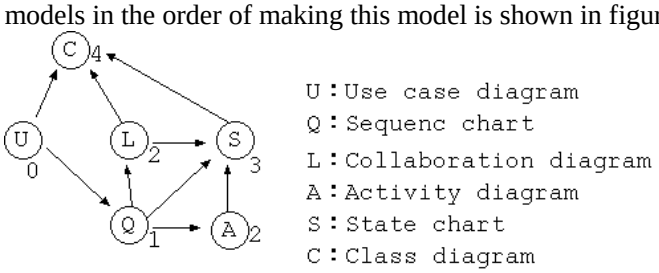

Figure 3 Model reference levels.

An edge of an arrow means the direction of referring, a node of a circle shows a model, and a number shows reference levels in figure 3. We assume that there is no closed path in the graph.

The reference level is defined as follows.

(1) Every node has its reference level.
(2) The reference level of a node, which does not have any input edges, is 0.
(3) The reference level of other nodes is the maximum value of the distance from the nodes without any input edges.

For instance, in figure 3 the class diagram is made referring to the use case diagram, the collaboration diagram, and the state chart, and the reference level becomes four, because the maximum value of the distance from the use case diagram is four. The verification order is decided by using this reference levels.

The reference level shows how much other models are referred to make the model. When a describer defines the models with a different order and/or different reference levels from that of figure 3, he can set another model reference diagram in accordance with his definition of models, and the order of the verification step is changed.

As a model with high reference level depends on other more models, the correction of the model causes re-verification or correction of other more models. Therefore, the verification should be started from the verification between the model that the reference level is as low as possible and another model.

In other words, we start from the verification between the model made without depending on other models as much as possible and another model.

The correction of a certain model can be prevented from influencing other models as much as possible when errors are detected by the verification. The decision procedure of the order of the verification step is shown below.

(1) Reference levels of the two models to be verified are sorted by bigger value
(2) Sorted result are sorted again by smaller value of the reference levels
(3) As previously mentioned in figure 2, redundant verification steps are omitted.

Figure 4 shows the decision procedure of the order of the verification step for the reference level given in figure 3. 15 kinds of verification steps are permuted in the middle of figure 4 in accordance with the above first two steps. Verification steps of models whose reference level is smaller will be set to be executed earlier, because of lessen the re-verifications.

Next, redundant verification steps are detected and omitted. The step that is redundant and can be omitted shown in figure 2 can be applied to the verification order as follows.

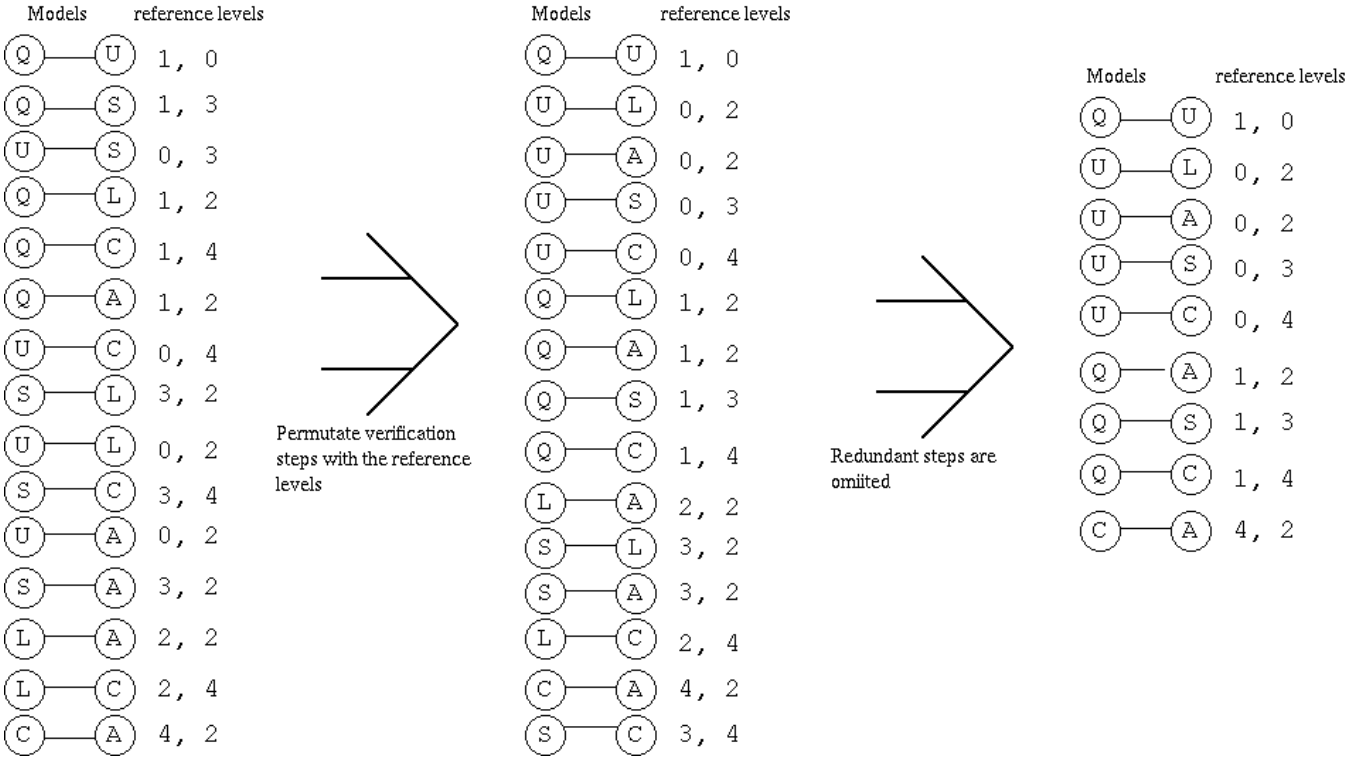

Figure 4 Permutation of verification steps and omission of redundant steps.

(1) After the verification of the consistency between sequence chart (Q) and use case diagram(U) and the consistency between use case diagram(U) and collaboration diagram(L), the verification step between sequence chart(Q) and collaboration diagram(L) can be omitted.
(2) After the verification between sequence chart(Q) and class diagram(C) and
the consistency between sequence chart(Q) and collaboration diagram(L), the verification step between collaboration diagram(L) and class diagram(C) can be omitted.
(3) After the verification of the consistency between sequence chart(Q) and state chart(S) and the consistency between sequence chart(Q) and collaboration diagram(Q), the verification step between collaboration diagram(L) and state chart(S) can be omitted.
(4) After the verification of the consistency between sequence chart(Q) and activity diagram(A) and the consistency between sequence chart(Q) and collaboration diagram(L), the verification step between collaboration diagram(L) and activity diagram(A) can be omitted.
(5) After the verification of the consistency between state chart(S) and use case diagram(U) and the consistency between activity diagram(A) and use case diagram(U), the verification step between state chart(S) and activity diagram(A) can be omitted.
(6) After the verification of the consistency between state chart(S) and activity diagram(A) and the consistency between class diagram(C) and activity diagram(A), the verification step between state chart(S) and class diagram(C) can be omitted.
(7) After the verification of the consistency between state chart(S) and collaboration diagram(L) and the consistency between activity diagram(A) and collaboration diagram(L), the verification step between state chart(S) and activity diagram(A) can be omitted.
(8) After the verification of the consistency between state chart(S) and sequence chart(Q) and the consistency between activity diagram(A) and sequence chart(Q), the verification step between state chart(S) and activity diagram(A) can be omitted.

The right part of figure 4 shows the verification order of reduced redundant steps. It shows whether there is a possibility necessary to verify which model again when a certain model is corrected. It enables also the efficient navigation of the verification and the re-verification. For example, suppose that as a result of the consistency verification between the sequence chart and the activity diagram, there are no errors in the sequence chart, and the

activity diagram needs correction. With figure 5, we can find that after the correction of the activity diagram, the consistency between the corrected activity diagram and use case diagram should be re-verified, although the consistency between the original activity diagram and the use case diagram has been already verified.

In general, when a model X is corrected, we should re-verify the consistency between the corrected model X and another model with which the model X was consistent through the past verification steps. Since we can determine the order of verification steps, we can find necessary re-verification steps and navigate the re-verification steps.

When the consistency verification between two models detects some inconsistency, it is necessary to correct either or both of two models. In general users judge which models to be corrected. In case of the correction of either or both of the two models, such corrections will influence to the past verification steps as little as possible with the order of the verification steps in accordance with the reference levels.

We applied our method to a UML-based software development and confirmed the effectiveness of the method [9].

4. Conclusions

In this paper, we propose a supporting method of verifying the consistency among UML models with knowledge-based verification items. Our method enables the efficient verification and re-verification by setting the order of the verification step derived from the model reference diagram and by omitting the redundant verification steps.

In general, when errors are found in the UML model, the correction of the errors may cause a ripple effect to other UML models. In other words, the correction of the errors may produce another errors in the other UML models. To detect and correct such ripple effect errors, we have to check UML models. Without correct checking methods, some errors may remain in UML models. Without efficient checking method, we have to do extra checks. Our method enables the efficient verification and re-verification and guarantees the consistency among the six UML models in the object-oriented analysis. These are major features of our method.

References

[1] Alhir S.S.: UML in a Nutshell, O'Reilly, 1999.

[2] Aoki T. and Katayama T.: "A Formal Model for Object-oriented Methodology" J. Japan society for software science and technology, computer software, Vol.16, No.1, pp.12-32, 1999 (in Japanese).

[3] Booch G.: Object-Oriented Analysis and Design with Applications, 2nd ed., Benjamin/Cummings Pub. 1994.

[4] Clark T., Evans A.: Foundation of the Unified Modeling Language, Proc. 2nd Northern Formal Methods Workshop, Springer-Verlag, 1998.

[5] Coad P., Yourdon, E.: Object-Oriented Analysis, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1991.

[6] Evans A., Kent S.: Core Meta-Modeling Semantics of UML: The pUML Approach, Proc. 2nd Int'l Conf. on UML,(Rumpe, B. and France, R.B. eds.) Springer-Verlag, LNCS 1723, pp.140-155, 1999.

[7] Fowler M., Scott K.: UML Distilled, Applying the Standard Object Modeling Language, Addison-Wesley, 1997.

[8] Jacobson I. et al.: Object-Oriented Software Engineering, A Use Case Driven Approach, ACM Press, 1992.

[9] Ohnishi, A.: “A supporting system for verification among models of the UML,” Trans. IEICE, pp.671-681, 2001 (in Japanese).

[10] Tateishi T., Aoki T., and Katayama T.: “Verifying the Object-Oriented Analysis Model using HOL,” Proc. Workshop on Foundation of Software Engineering (FOSE2000), Japan society for software science and technology, Kindai-kagaku, pp.117-124, 2000 (in Japanese).