International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:12, No:7, 2018

Knowledge Representation and Inconsistency Reasoning of Class Diagram Maintenance in Big Data

Chi-Lun Liu

***Abstract*—**Requirements modeling and analysis are important in successful information systems' maintenance. Unified Modeling Language (UML) class diagrams are useful standards for modeling information systems. To our best knowledge, there is a lack of a systems development methodology described by the organism metaphor. The core concept of this metaphor is adaptation. Using the knowledge representation and reasoning approach and ontologies to adopt new requirements are emergent in recent years. This paper proposes an organic methodology which is based on constructivism theory. This methodology is a knowledge representation and reasoning approach to analyze new requirements in the class diagrams maintenance. The process and rules in the proposed methodology automatically analyze inconsistencies in the class diagram. In the big data era, developing an automatic tool based on the proposed methodology to analyze large amounts of class diagram data is an important research topic in the future.

***Keywords*—**Knowledge representation, reasoning, ontology, class diagram, software engineering.

I. Introduction

MODELING architectural design according to users' requirements is an important factor for successful software system development [1]. UML is a popular modeling standard in information systems development [2]. Class diagrams are widely used for modeling the static data aspects of software systems. The UML diagrams are used in the Rational Unified Process [3], which does not provide guidance on how to elaborate the UML diagrams.

Kendal and Kendal [4] indicate that there is a lack of a systems development methodology described by the organism metaphor. The systems development methodology, which is akin to the organism metaphor, must help an organization to survive, assimilate new ideas, and change to adapt to its environment and grow [4]. Although a lack of the organism metaphor in systems development methodologies was revealed by Kendal and Kendal [4] more than 20 years ago, the organism metaphor is more and more important today because the business environment is changing fast and this results in large amounts of data in this big data era. The core concept of the organism metaphor is adaptation which is an act of changing something. The organism metaphor implies that managing change requests in the post-development phase is important. Change requests are user requirements for system maintenance. Change requests analysis is a requirement engineering issue in the post-development phase [5]. Therefore, developing an organic methodology for analyzing change requests is an interesting research topic in information systems development.

Constructivism theory is a philosophy viewpoint which argues that assimilation and accommodation are two essential parts of the adaptation process [6]. Assimilation emphasizes on possessing of data from the external environment based on existing knowledge. Accommodation indicates that existing knowledge changes to fit new assimilated data. Constructivism theory reveals the adaptation process which is consistent with the core concept of organism metaphor. Hence, constructivism theory can be a theoretical foundation for developing an organism methodology.

Ontology is a conceptualization for representing and sharing explicit knowledge [7]. Ontology is typically offered to support a shared understanding [8]. Ontology, which is a conceptual model represented by vocabulary, comprises three elements: concepts, relationships, and constraints & axioms [9]. Knowledge representation and reasoning is of immense importance in the field of Artificial Intelligence. The knowledge representation and reasoning approach use rules and ontologies stored in a knowledge base to make inferences to solve problems. There are empirical evidences of the benefits of using the knowledge representation and reasoning approach in requirements engineering activities both in industry and academy for reducing inconsistencies in functional requirements [10].

Analyzing system models is crucial when multiple stakeholder concerns need to be addressed by information system developers [11]. Design inconsistencies are common in industries and often hard to be recognized in large systems which have big data in systems requirements [12]. Big data is a hot research topic. Using the knowledge representation and reasoning approach is emergent in the recent years [13], [14]. However, few of the related works uses the knowledge representation and reasoning approach to analyze class diagrams automatically.

This paper provides the preliminary methodology based on constructivism and ontology theories. The proposed methodology comprises a step-by-step process and a set of rules to analyze change requests to reduce inconsistencies in class diagrams maintenance. The analysis process is a four-phase circle including (1) build prior knowledge, (2) specify change requests, (3) analyze change requests, and (4) approve change requests. From constructivism theory and the knowledge representation and reasoning perspective, phase (1) and phase (2) *assimilate* and *represent knowledge* about in existing ontologies, class diagrams, and new requirements. Phase (3) and phase (4) *infer* inconsistencies from prior knowledge to *adapt* to new requirements. Phase (3) executes

Chi-Lun Liu is with the Kainan University, No.1 Kainan Road, Taoyuan City 33857, Taiwan, R.O.C. (e-mail: tonyliu@mail.knu.edu.tw).

inconsistency *reasoning*. And stakeholders decide whether change requests are consistency with existing class diagrams or not in phase (4). These rules handle 14 inconsistency situations. Scenarios in the electronic commerce context are provided to demonstrate the proposed rules.

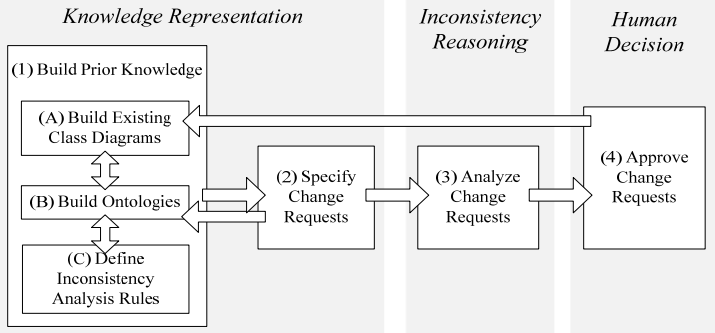

Fig. 1 The process for knowledge representation and inconsistency reasoning

II. Knowledge Representation and Inconsistency Reasoning Process

This section describes knowledge representation and the inconsistency reasoning process in the class diagram maintenance context based on constructivism and ontology theories. The proposed process is a circle and has four phases, as shown in Fig. 1: build prior knowledge, specify change requests, analyze change requests, and approve change requests. These steps are introduced as follows:

(1) Build Prior Knowledge: Users, system analysts, and knowledge engineers model the domain knowledge in ontologies, existing class diagrams, and requests analysis rules in this phase. The three steps in this phase are introduced as follows:

A. Build Existing Class Diagrams: These terms in the ontology will be used to represent class diagrams. Approved change requests will be added in existing class diagrams in this step.

B. Build Ontologies: The terms in the domain knowledge should be stored in ontologies.

C. Define Requests Analysis Rules: This paper defines 14 inconsistency analysis rules. These 14 rules may not be complete. Therefore, new rules can be added in this phase if new rules are proposed in the future.

(2) Specify Change Requests: Modeling requests to change the existing class diagrams in this phase is based on ontologies built in phase 1. Vocabularies in the ontology, which are established in phase 1, can be used to represent change requests. The vocabularies used in change requests have to be stored in the ontology. If a new vocabulary appears, add this new ontological vocabulary in step 1b. In other words, all vocabularies used in change requests have to be defined in advance.

(3) Analyze Change Requests: This phase uses ontologies and inconsistency analysis rules to analyze change requests. Implementing a knowledge representation and a reasoning tool containing ontologies and inconsistency analysis rules can analyze change requests automatically.

Approve Change Requests: Stakeholders in the management committee should negotiate a decision to approve or decline a change requests in this phase. If a change request is approved, step 1a would be performed to start these phases again.

III. Knowledge Representation Format

Two knowledge representation formats are used in inconsistency reasoning: ontology and class diagram. Knowledge representation of class diagram includes class name, attribute, method, inheritance, aggregation, and composition. Fig. 2 depicts an example about knowledge representation for class diagram. In Fig. 2, Class_A includes Attribute_I (i.e. data) and Method_X (i.e. operation). An inheritance relationship between Class_A and Class_B. A composition relationship between Class_C and Class_D. And an aggregation relationship between Class_E and Class_F.

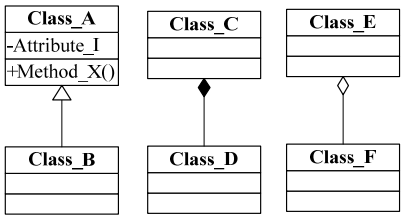

Fig. 2 Knowledge Representation Example of Class Diagram

Ontological knowledge representation includes terms and relationships. A term is a word with a specific meaning. A relationship is a semantic relation between terms. Four relationships are used in this work: synonym, antonym, kind, and part. Fig. 3 depicts an example of ontological knowledge representation. In Fig. 3, a kind of relationship exists between term i and term j. And a synonym relationship between term i and term k.

IV. Inconsistency Reasoning Rules

Inconsistency analysis rules focus on inconsistencies between a change request (CR) and an existing specification (ES) and between a change request and an ontology. Rules R1-R14 are proposed in Tables I-III. Tables I-III also show the

IF-THEN rules which are in the format of IF Condition THEN Conclusion. If all conditions in a rule are true, a conclusion would be provided. Scenarios are also provided to explain these rules.

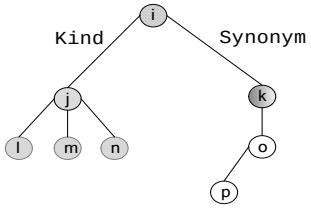

Fig. 3 Ontological Knowledge Representation Example

Generalization inconsistency analyzed by Rule1 means that a class is not only a superclass but also a subclass of another class in a wrong class diagram. Composition inconsistency detected by Rule2 and aggregation inconsistency analyzed by Rule3 mean a class may be a part or a whole of another class.

The scenario of generalization inconsistency is depicted in Fig. 4. In this scenario, Payment service (ClassM) is a superclass of Near Field Communication (ClassN) in CR in the payment system. NFC (ClassO) is a superclass of Payment service (ClassP) in ES. ClassM equals ClassP. ClassN is a synonym of ClassO because NFC is the abbreviation of Near Field Communication. According to Rule1, generalization inconsistency occurs. The structures of Rule1 to Rule3 are similar.

Some behaviors in information systems are regulated by government laws and corporation policies. Attribute and method exclusion inconsistency analyzed by Rule4 and Rule5 means an undesirable attribute and undesirable method are added.

The scenario of method exclusion inconsistency is depicted in Fig. 5. In Fig. 5, storing_credit_card_number() (i.e. MethodI()) is added in Credit_card (i.e. ClassM) in CR. A corporation policy indicates that credit card numbers cannot be stored in the database because stored credit card numbers pose a security risk related to hacking. Therefore storing_credit_card_number() (MethodJ) cannot be included in Any class (i.e. ClassN) in ES in the payment system. MethodI() equals MethodJ(). ClassM is a kind of ClassN. According to Rule5, method exclusion inconsistency occurs.

TABLE I
INCONSISTENCY ANALYSIS RULES R1-R5

	Change Request (CR)		Condition		Conclusion
	Act	Target	Ontology	Existing Specification (ES)	Analysis Suggestion
R1	Add	ClassM, ClassN, and ClassM is a superclass of ClassN	An equality or a synonym relationship exists between ClassN and ClassO. And an equality or a synonym relationship exists between ClassM and ClassP.	Class P, Class O, and ClassO is a superclass of ClassP	There is a generalization inconsistency.
R2	Add	ClassM, ClassN, and a composition relationship from ClassN to ClassM	An equality or a synonym relationship exists between ClassN and ClassO. And an equality or a synonym relationship exists between ClassM and ClassP.	ClassP, ClassO, and a composition relationship from ClassP to ClassO	There is a composition inconsistency.
R3	Add	ClassM, ClassN, and an aggregation relationship from ClassN to ClassM	An equality or synonym relationship exists between ClassN and ClassO. And an equality or a synonym relationship exists between ClassM and ClassP.	ClassP, ClassO, and an aggregation relationship from ClassP to ClassO	There is an aggregation inconsistency.
R4	Add	AttributeX in ClassM	An equality, kind, part, or synonym relationship exists between AttributeX and AttributeY. And an equality, kind, part, or synonym relationship exists between ClassM and ClassN.	Attribute Y is not allowed in ClassN	There is an attribute exclusion inconsistency.
R5	Add	MethodI() in ClassM	An equality, kind, part, or synonym relationship exists between MethodI() and MethodJ() And an equality, kind, part, or synonym relationship exists between ClassM and ClassN	MethodJ() is not allowed in ClassN	There is a method exclusion inconsistency.

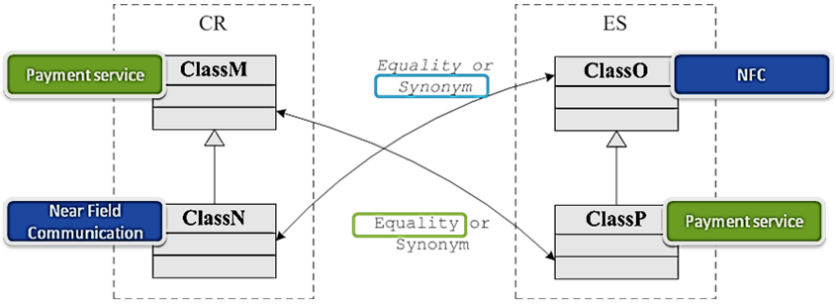

Fig. 4 Scenario of generalization inconsistency

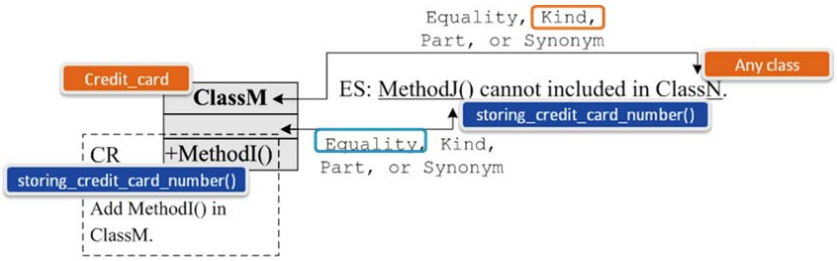

Fig. 5 Scenario of method exclusion inconsistency

TABLE II
INCONSISTENCY ANALYSIS RULES R6-R10

		Change Request (CR)	Condition Ontology	Existing Specification (ES)	Conclusion Analysis Suggestion
R6	Add	A generalization relationship from ClassO to ClassN	None	ClassM, ClassN, ClassO, and there is a generalization relationship from ClassO to ClassM	There is a multiple inheritance inhibition inconsistency.
R7	Add	A generalization relationship from ClassN to ClassM	There is an equality, part, antonym, or synonym relationship between ClassM and ClassN	ClassM and ClassN	There is a generalization and alternative inconsistency.
R8	Add	A generalization relationship from ClassN to ClassM	ClassM is a kind of ClassN	ClassM and ClassN	There is an inverse generalization inconsistency.
R9	Add	An aggregation relationship from ClassN to ClassM	There is a equality, part, antonym, or synonym relationship between ClassM and ClassN	ClassM and ClassN	There is an aggregation and alternative inconsistency.
R 10	Add	An aggregation relationship from ClassN to ClassM	ClassM is a kind of ClassN	ClassM and ClassN	There is an inverse aggregation inconsistency.

TABLE III
INCONSISTENCY ANALYSIS RULES R11-R14

		Change Request (CR)	Condition Ontology	Existing Specification (ES)	Conclusion Analysis Suggestion
R 11	Add	A composition relationship from ClassN to ClassM	There is a equality, part, antonym, or synonym relationship between ClassM and ClassN	ClassM and ClassN	There is a composition and alternative inconsistency.
R 12	Add	A composition relationship from ClassN to ClassM	ClassM is a kind of ClassN	ClassM and ClassN	There is an inverse composition inconsistency.
R 13	Delete	AttributeX	There is a equality, part, antonym, or synonym relationship between AttributeX and AttributeY. There is a equality, part, antonym, or synonym relationship between ClassM and ClassN.	AttributeY cannot be deleted in ClassN. AttributeX exists in ClassM.	There is an attribute deletion inconsistency.
R 14	Delete	MethodI()	There is a equality, part, antonym, or synonym relationship between MethodI() and MethodJ(). There is a equality, part, antonym, or synonym relationship between ClassM and ClassN	MethodJ() cannot be deleted in ClassN. MethodJ() exists in ClassM.	There is a method deletion inconsistency.

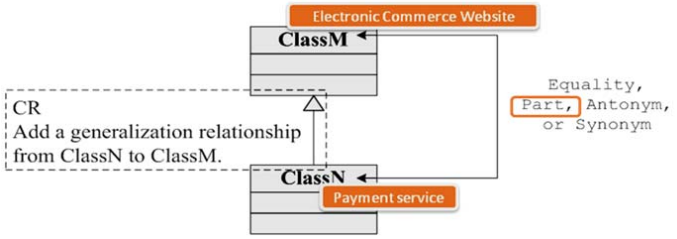

Fig. 6 Scenario of generalization and alternative inconsistency

Some programming languages, such as Java, inhibit multiple inheritance [10]. Rule6 detects the conflict about multiple inheritance inhibition inconsistency. It indicates that more than one superclass exists in a class diagram.

Generalization and alternative inconsistency in class diagrams can be detected in Rule7. Normally, if there is a generalization relationship between two classes, it means that a parent-child relationship exists between two classes. Therefore, other relationships, such as equality, part, antonym, and synonym relationships should not exist between these two classes. Besides, Rule7, Rule9 and Rule11 are similar.

The scenario of generalization and alternative inconsistency is depicted in Fig. 6. In this scenario, the ontology indicates that Payment service (ClassN) is a part of an Electronic commerce

website (ClassM). According to Rule7, adding a generalization relationship from Payment service (ClassN) to Electronic commerce website (ClassM) in CR causes a generalization and alternative inconsistency.

Rule8 detecting inverse generalization inconsistency means that the direction of generalization relationship between two classes in a class diagram is inverse comparing to the ontology. The structures of Rule8, Rule10 and Rule12 are similar.

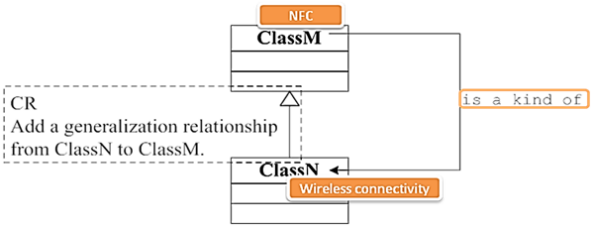

Fig. 7 Scenario of inverse generalization inconsistency

Fig. 7 depicts the scenario of inverse generalization inconsistency. In this figure, the ontology shows NFC (ClassM) is a kind of Wireless connectivity (ClassN). According to Rule8, adding a generalization relationship from Wireless connectivity (ClassN) to NFC (ClassM) in CR, which means wireless is a kind of NFC, causes inverse generalization inconsistency.

Rule13 analyzes an attribute deletion inconsistency, which means a change request intended to delete an attribute inappropriately. And Rule14 analyzes a method deletion inconsistency, which means a change request intended to delete a method inappropriately. The structure of Rule13 and Rule14 are the same.

V. Conclusion

This work proposes a methodology including a process and a set of rules for knowledge representation and inconsistency reasoning in class diagrams to automatically analyze requirements. The proposed methodology is based on ontology and constructivism theories. This methodology fills the systems development research gap in the organism metaphor. Structured domain knowledge and explicit rules can facilitate automatic conflict detection and even increase systems development productivity. In the big data and high competitive business environment context, the organism metaphor systems development methodology and the automatic system development tool are important research topics.

References

[1] J. He and W.R. King, "The Role of User Participation in Information Systems Development: Implications from a Meta-analysis," *Journal of Management Information Systems*, vol. 25, no. 1, pp. 301–331, summer 2008.

[2] N.H. Ali, Z. Shukur, and S. Idris, "A Design of an Assessment System for UML Class Diagram," In *Proc. of International Conference on Computational Science and its Application*, New York, 2007, pp. 539 - 546.

[3] P. Kruchten, *The Rational Unified Process: An Introduction*, Boston, MA: Addison-Wesley Professional, 2004.

[4] J. E. Kendal and K. E. Kendal, "Metaphors and Methodologies: Living Beyond the Systems Machine", *MIS Quarterly*, Vol. 7, No. 2, pp. 149-171, 1993.

[5] L. Liu and H. L. Yang, "Applying Ontology-based Blog to Detect Information System Post-Development Change Requests Conflicts", *Information Systems Frontiers*, Vol. 14, No. 5, pp. 1019-1032, 2012.

[6] G. M. Bodner, "Constructivism: A Theory of Knowledge", *Journal of Chemical Education*, Vol. 63, pp. 873-878, 1986.

[7] M. Gruninger, J. Lee, "Ontology: Applications and Design", *Communications of the ACM*, Vol. 45, No. 2, pp. 39–65, 2002.

[8] Richards, "A Social Software/Web 2.0 Approach to Collaborative Knowledge Engineering", *Information Sciences*, Vol. 179, No. 15, pp. 2515-2523, 2009.

[9] R. Kishore, R. Sharman, and R. Ramesh, "Computational Ontologies and Information Systems: I. Foundations", *Communications of Association for Information Systems*, Vol. 14, No. 8, pp: 158-183, 2004.

[10] D. Dermeval, J. Vilela, I. I. Bittencourt, J. Castro, S. Isotani, P. Brito, A. Silva, "Applications of Ontologies in Requirements Engineering: a Systematic Review of the Literature", *Requirements Engineering*, pp. 1–33, 2015.

[11] J. Savolainen and T. Männistö, "Conflict-Centric Software Architectural Views: Exposing Trade-Offs in Quality Requirements," *IEEE Software*, vol. 27, no. 6, pp. 33-37, Nov./Dec. 2010.

[12] Egyed, "Instant Consistency Checking for the UML," In *Proc. of 28th International Conference on Software Engineering*, New York, 2006, pp. 381–390.

[13] Y. Nomaguchi and K. Fujita, "DRIFT: A Framework for Ontology-based Design Support Systems," In: *Proc. of Semantic Web and Web 2.0 in Architectural, Product, Engineering Design Workshop*, Aachen, 2007, pp. 1-10.

[14] C. L. Liu, "CDADE: Conflict Detector in Activity Diagram Evolution Based on Speech Act and Ontology", *Knowledge-Based Systems*, vol. 23, no. 6, pp. 536-546, Aug. 2010.