Specification of Vertical Semantic Consistency Rules of UML Class Diagram Refinement Using Logical Approach

Nuraini Abdulganiyyi(1), Noraini Ibrahim(2), and Ruhaya AbdulAziz(3)

(1) Department of Mathematics and Computer Science, Federal University Kashere, Gombe State, Nigeria
E-mail: agnuraini@gmail.com
(2) Department of Software Engineering, Universiti Tun Hussein Onn Malaysia (UTHM) 86400, Parit Raja, Batu Pahat, Johor, Malaysia
E-mail: noraini@uthm.edu.my
(3) Department of Software Engineering Universiti Tun Hussein Onn Malaysia (UTHM) 86400, Parit Raja, Batu Pahat, Johor, Malaysia
E-mail: ruhaya@uthm.edu.my

ABSTRACT

Unified Modelling Language (UML) is the most popular modelling language use for software design in software development industries in which class diagram is being the most frequently used diagram. Despite the popularity, UML is being affected by inconsistency problems of its diagrams at the same or different abstraction levels. To address inconsistency in UML, this research has specified twenty-four abstraction rules of class's relation semantic among any three related classes of a refined class diagram to semantically equivalent relations of two of the classes using a logical approach. This research has also formalized three vertical semantic consistency rules of a class diagram refinement identified by previous researchers using a logical approach and the set of formalized abstraction rules. The results were successfully evaluated using hotel management system and passenger list system case studies and were found to be reliable and efficient.

Keywords- Cardinality Abstraction, Class Diagram, Inconsistency, Logical Approach, Semantic Abstraction Rule, UML.

1- INTRODUCTION

The increasing dependency on computers and software applications for saving lives, properties and time, in our contemporary world has escalated to all sectors of human endeavours. Thereby, led to an increase in the demand of efficiency and reliability of the computers and the software applications before usage, to avoid claims of what they were provided to save (that is: lives, properties and time). To ensure efficiency and reliability of software applications, software experts have agreed to define the best practice for software development, namely software engineering. The discipline of software engineering is coined to deal with poor quality of software, get projects exceeding time and budget under control. It also ensures that software is built systematically, rigorously, measurably, and within specification. In other words, software engineering is the study and application of engineering to the design, development, and maintenance of software from the start to the end of the development [1]. This research is aimed at addressing the inconsistencies of software at the design stage. Design plays a central role in the activities that leads to the development or maintenance of good software by giving an abstract representation of the system prior to development or maintenance. The consistency of the developed or maintained system with the user requirement specifications depends mostly on the consistency of the design. According to [2], software design is the process of realizing software solution to one or more set of problems.

The largest segment of design phase of software development life cycle is creating a consistent design based on a comprehensive model. These days, the infrastructures for creating this design are usually based on object-oriented modelling languages. Unified Modelling Language (UML) is the most popular object-oriented modelling language use to model a system in a way that the status of the various objects replicate the user's point of view or specification [3]. The modelling task focuses on definitions and descriptions of objects, features and actions to be operated by the user during interaction, rather than on the programming aspect [4].

2- UNIFIED MODELLING LANGUAGE

The Unified Modelling Language (UML) is a language and notation system use to specify, construct, visualize, and document models of software systems [5]. It provides sets of diagrams to model structural, behavioural, and interaction aspects of an object-oriented system. Each diagram depicts a particular design aspect of the system. UML consists of many diagrams depending on the version. For example, UML version 2.0 has 13 diagrams [5], both UML version 2.2 and 2.4 have 14 diagrams [6] and UML version 2.5 has 17 diagrams [7]. The presence of many UML diagrams, to model a system, brings a variety of views that overlap with respect to information depicted in each that can leave overall system design specification in an inconsistent state [8].

3- UML MODEL CONSISTENCY

Consistency in UML model is a state in which the structures, features and elements that appear in a model are compatible and in alignment with contents of the model and other related models with respect to requirement being modeled and UML meta-model [9]. For example, the structures, functions and relations in an initial class diagram obtained during an analysis phase of a software development must be compatible with a detailed class diagram developed during the design phase of the software development. In addition, unambiguous and consistent UML models are necessary for successful development of quality Information System (IS) [10]. However, UML model is hardly free of inconsistency problems within or with other models at the same or different abstraction levels. Inconsistency in UML model(s) usually arose due to analysts or designers viewing the same system from different points of views. Other possible causes of UML inconsistency are iterative process of an IS development, lack of UML knowledge or practice, imprecise semantic nature of the UML diagrams, difference in geographical location of developers, and multiple interpretations of user’s requirements and UML notations [11].

There are two types of consistency problems in UML; vertical and horizontal. Vertical and horizontal consistency problems are also classified into syntactic and semantic consistency problems. Despite all the challenges of consistency uncertainty of UML models, UML is also the most widely used modelling language in object-oriented software development industries. Class diagram is the most used UML diagram [12]. For this reason, this research will propose a formal specification for three vertical semantic consistency rules of class diagram refinement identified by [13] using a logical approach. The definitions of the types and classifications of consistencies are described as follows.

3-1 VERTICAL CONSISTENCY

Vertical consistency in UML is a state of semantic or syntactic compatibility of models built at different levels of abstraction such as between a model and its refinement. It is also called inter-consistency [11]. For example, an abstract class diagram developed in the analysis phase of software development must be semantically and syntactically consistent with a detailed class diagram developed in the design phase of the software development.

3-2 HORIZONTAL CONSISTENCY

Horizontal consistency is a state of semantic or syntactic compatibility of models built at the same level of modelling abstractions. It is also called intra-consistency [11]. For example, a class diagram describing the static aspects of an abstract model must be semantically and syntactically consistent with a state machine diagram describing the dynamic aspects of the classes in the model.

3-3 SEMANTIC CONSISTENCY

Semantic consistency is a state that requires models’ behaviours to be semantically compatible with one another [14]. For example, a class diagram and its refinement must be semantically compatible with each other. Unlike syntactic consistency, there is no specific method for specifying semantic consistency rules and constraints [15].

3-4 SYNTACTIC CONSISTENCY

Syntactic consistency guarantees that a model conforms to abstract syntax of the modelling language as specified by its meta-model [14]. For example, in a class diagram, the design of each class as well as the relationship between them must be syntactically correct in accordance with the class diagram meta-model. In general, syntactic consistency can be automatically checked and therefore is supported by current UML CASE tools [15].

4- PROBLEM STATEMENT

Despite the popularity of UML for object-oriented software modeling in software development industries, UML diagrams are being affected by inconsistency problems at the same and different modeling abstractions.

Inconsistency problems of UML diagrams are the major setback recorded affecting modeling with UML. Solving UML inconsistencies have gained the attention of many researchers on how to handle inconsistency in UML, though there are limited works in UML vertical semantic inconsistency management [8]. In general, syntactic consistency problems can be automatically checked and, therefore, are supported by current UML CASE tools [15]. Unlike syntactic consistency, there is no specific method for specifying semantic consistency rules and constraints [15]. Shen, Wang, & Egyed (2009) identified three vertical semantic consistency rules of a class diagram refinement and used informal approaches to manage them. The approaches used were Integrated Abstraction and Comparison (IAC), and Separated Abstraction and Comparison (SAC). These techniques require a significant amount of time and memory space in order to handle inconsistencies of a class diagram refinement. These are due to the large number of rules check and iterations involved in the algorithms. On the contrary, this research will formulate the same vertical semantic consistency rules of class diagram refinement achieved with SAC and IAC in a more effective and efficient manner using a logical approach.

5- REVIEW OF PREVIOUS WORKS

Although there are many proposals for enhancing modelling with UML, only a few works on UML semantic consistency management [8], [16]. While some of the proposals used formal methods to enhance UML modelling and software development process, others used informal methods. Lima et al., (2009) proposed a formal verification and validation (V&V) technique to check semantic consistency of a sequence diagram. The proposed technique generate PROMELA-based model from interactions expressed in a given sequence diagram. SPIN model checker is then used to simulate the execution and to confirm sequence diagram properties are written in Linear Temporal Logic (LTL). The technique was implemented as an Eclipse plug-in, with human understandable feedback to the developer. The following semantic rules of a sequence diagram were addressed; lifeline that performed the last action, the last completed action (sent or received), message used in the final action, and lifeline to/from which a message was sent/received. This technique is difficult to extend to static components of UML diagrams. According to [18], PROMELA is a process modelling language which intended use is to verify the logic of parallel systems. In other words, PROMELA can be highly suitable for modelling dynamic properties but not static features.

Shen, Wang, & Egyed (2009) presented two informal methods for checking consistency between a class diagram and its refinement at different levels of modelling abstractions. The presented techniques were Integrated Abstraction and Comparison (IAC), and Separated Abstraction and Comparison (SAC). The authors further demonstrated that SAC is highly favourable for consistency checking of software models than IAC. The techniques addressed three semantic consistency rules of class diagram refinement. The addressed rules are stated as follows; (1) every low-level class refines at most one high-level class, (2) every high-level class has at least one low-level class, which refines the high-level class, and (3) the group of relationships between any two high-level classes must be identical with the group of relationships between their corresponding low-level classes. The methods were implemented and integrated with IBM Rational Rose design tool.

He et al. (2013) proposed a method of ontology-based semantics confirmation of UML behaviour diagrams. The authors divided semantics of behaviour diagrams into static and dynamic semantics. The static semantics are defined as the notations and constraints in UML behavioural diagrams while the dynamic semantics are defined as the semantic relations among the instances of the notations while interacting. The static semantics of behavioural diagrams are transformed into ontology web language description logic (OWL DL) by converting UML behaviour diagrams and their meta-models into a DL knowledge base. While the dynamic semantics are specified in DL-Safe rules that are then expressed by SWRL (Semantic Web Rule Language) and added to the OWL DL ontology. The OWL DL is then used to check both vertical and horizontal semantic consistency of activity, sequence, and state diagrams.

Knapp, Mossakowski, & Roggenbach (2014) proposed a technique called institution based heterogeneous approach for checking semantic consistency among UML diagrams. The proposed framework can be used to verify consistency of different UML diagrams both horizontally and vertically. The vertical semantic consistency addressed in the proposal checks whether the state machine satisfies an OCL invariant or an OCL pre-/post-condition.

Abdulganiyyi, N. and Ibrahim, N., (2014) presented abstraction rules of a class diagram using a logical approach. The rules were evaluated using only one case study of hotel management system.

However, there are still issues with UML consistency checking and management, due to ambiguity of some of the proposed rules, unconformity to meta-model of the UML diagram(s), in-extensibility of some of the techniques, sometimes meaningless consistency rules proposals as well as impractical applicability of the proposed rules [8]. This research will represent the abstraction rules of [21] and formalized three vertical semantic consistency rules of a class diagram refinement identified by previous researchers using a logical approach and the set of formalized abstraction rules of [21]. The results will be evaluated with two case studies.

6- SEMANTIC ABSTRACTION OF CLASS’S RELATION USING LOGIC

Logic is the basis for stating formal proofs in all branches of mathematics [22]. This article uses logic to abstract a class diagram through breaking relationship between three classes logically to semantic equivalent relation of two of the classes. The following definitions will be used to formalize rules for abstracting a class diagram.

Definition 1: Let “CD” be a class diagram that contains a finite set of classes “CLs” and a finite set of relations “R” between classes. Thus, it can be defined as

$CD = \{<CLs>, <R>\}$ (1)

Where

$CLs = \{CL_i \mid 1 \leq i \leq n, n \in Z+\}$ is a finite set of classes (2)

And

$R = \{D, G, A, \vec{A}, S\}$ (3)

Where

$D = \{d_i \mid 0 \leq i \leq q, q \in Z+\}$ is a finite set of dependencies,
$G = \{g_i \mid 0 \leq i \leq p, p \in Z+\}$ is a finite set of generalization,
$A = \{a_i \mid 0 \leq i \leq t, t \in Z+\}$ is a finite set of bidirectional aggregation,
$\vec{A} = \{\vec{a}_i \mid 0 \leq i \leq u, u \in Z+\}$ is a finite set of unidirectional aggregation.
$S = \{s_i \mid 1 \leq i \leq m, m \in Z+\}$ is a finite set of association.

Definition 2: Let "CL" be a class in a class diagram, CL consists of a finite set of attributes and operations. Thus, it can be defined as:
$CL = \{Attr, Opr\}$ (4)
Where
$Attr = \{attr_i \mid 0 \leq i \leq k, k \in Z+\}$ is a finite set of attributes in a class (CL).
$Opr = \{opr_i \mid 0 \leq i \leq l, l \in Z+\}$ is a finite set of operations in a class (CL).

Definition 3: Let $“CL_i”, CL_j”, and\ “CL_k”$ be classes in a class diagram such that $CL_i, CL_j, CL_k \in CD$ for $i \neq j \neq k, 1 \leq i, j, k \leq n$. let $“D”, “G”, “A”, “S” \in R$ be possible relations between any two classes in a class diagram.

To abstract relationship R among any three related classes in a class diagram, the following rules are developed to convert the relationship among any three classes: CL_i, CL_j, and CL_k to the semantic equivalent relations of two of the classes (CL_i and CL_j, or CL_k and CL_i, or CL_j and CL_k). From Equation 1, 3 and 4 the following rules are developed.

Rule 1: If *CL_i aggregates CL_j* and *CL_j aggregates CL_k* then *CL_i aggregates CL_k* transitively.

$If\ (CL_i\ A\ CL_j) \wedge (CL_j\ A\ CL_k) \rightarrow (CL_i\ A\ CL_k)$ (Transitivity)

Rule 2: If *CL_i aggregates CL_j* and *CL_j* associates *CL_k* then transitively *CL_i associates CL_k*.

$If\ (CL_i\ A\ CL_j) \wedge (CL_j\ S\ CL_k) \rightarrow (CL_i\ S\ CL_k)$ (Transitivity)

Rule 3: If *CL_i aggregates CL_j* and *CL_i associate*s *CL_k* then semantically *CL_j associates CL_k*.

$If\ (CL_i\ A\ CL_j) \wedge (CL_i\ S\ CL_k) \rightarrow (CL_j\ S\ CL_k)$ (Semantically)

Rule 4: If *CL_i* aggregate *CL_j* and *CL_k* associate *CL_i* then transitively *CL_k associates CL_j*.

$If\ (CL_i\ A\ CL_j) \wedge (CL_k\ S\ CL_i) \rightarrow (CL_k\ S\ CL_j)$ (Transitivity)

Rule 5: If *CL_i aggregate CL_j* and *CL_k* associate *CL_j* then semantically *CL_i associates CL_k*.

$If\ (CL_i\ A\ CL_j) \wedge (CL_k\ S\ CL_j) \rightarrow (CL_k\ S\ CL_i)$ (Semantically)

Rule 6: If *CL_i aggregates CL_j* and *CL_j depends on CL_k* then transitively *CL_i depends on CL_k*.

$If\ (CL_i\ A\ CL_j) \wedge (CL_j\ D\ CL_k) \rightarrow (CL_i\ D\ CL_k)$ (Transitivity)

Rule 7: If *CL_i aggregates CL_j* and *CL_i depends on CL_k* then semantically *CL_j depends on CL_k*.

$If\ (CL_i\ A\ CL_j) \wedge (CL_i\ D\ CL_k) \rightarrow (CL_j\ D\ CL_k)$ (Semantically)

Rule 8: If *CL_i aggregate CL_j* and *CL_k depend on CL_i* then transitively *CL_k depend on CL_j*.

$If\ (CL_i\ A\ CL_j) \wedge (CL_k\ D\ CL_i) \rightarrow (CL_k\ D\ CL_j)$ (Transitivity)

Rule 9: If *CL_i aggregate CL_j* and *CL_k depend on CL_j* then semantically *CL_k depend on CL_i*.

$If\ (CL_i\ A\ CL_j) \wedge (CL_k\ D\ CL_j) \rightarrow (CL_k\ D\ CL_i)$ (semantically)

Rule 10: If *CL_i generalized CL_j* and *CL_j generalized CL_k* then transitively *CL_i generalized CL_k*.

$If\ (CL_i\ G\ CL_j) \wedge (CL_j\ G\ CL_k) \rightarrow (CL_i\ G\ CL_k)$ (Transitivity)

Rule 11: If *CL_i generalized CL_j* and *CL_j associates CL_k* then transitively *CL_i associates CL_k*.

$If\ (CL_i\ G\ CL_j) \wedge (CL_j\ S\ CL_k) \rightarrow (CL_i\ S\ CL_k)$ (Transitivity)

Rule 12: If *CL_i generalized CL_j* and *CL_i associates CL_k* then semantically *CL_j associates CL_k*.

$If\ (CL_i\ G\ CL_j) \wedge (CL_i\ S\ CL_k) \rightarrow (CL_j\ S\ CL_k)$ (Semantically)

Rule 13: If *CL_i generalized CL_j* and *CL_k associates CL_i* then transitively *CL_k associates CL_j*.

$If\ (CL_i\ G\ CL_j) \wedge (CL_k\ S\ CL_i) \rightarrow (CL_k\ S\ CL_j)$ (Transitivity)

Rule 14: If *CL_i generalized CL_j* and *CL_k associates CL_j* then semantically *CL_k associates CL_i*.

$If\ (CL_i\ G\ CL_j) \wedge (CL_k\ S\ CL_j) \rightarrow (CL_k\ S\ CL_i)$ (Semantically)

Rule 15: If *CL_i generalized CL_j* and *CL_j depends on CL_k* then transitively *CL_i depends on CL_k*.

$If\ (CL_i\ G\ CL_j) \wedge (CL_j\ D\ CL_k) \rightarrow (CL_i\ D\ CL_k)$ (Transitivity)

Rule 16: If *CL_i generalized CL_j* and *CL_i depends on CL_k* then semantically *CL_j depends on CL_k*.

$If\ (CL_i\ G\ CL_j) \wedge (CL_i\ D\ CL_k) \rightarrow (CL_j\ D\ CL_k)$ (Semantically)

Rule 17: If *CL_i generalized CL_j* and *CL_k depends on CL_i* then transitively *CL_k depends on CL_j*.

$If\ (CL_i\ G\ CL_j) \wedge (CL_k\ D\ CL_i) \rightarrow (CL_k\ D\ CL_j)$ (Transitivity)

Rule 18: If *CL_i generalized CL_j* and *CL_k depends on CL_j* then semantically *CL_k depends on CL_i*.

$If\ (CL_i\ G\ CL_j) \wedge (CL_k\ D\ CL_j) \rightarrow (CL_k\ D\ CL_i)$ (Semantically)

Rule 19: If *CL_i generalized CL_j* and *CL_j aggregates CL_k* then transitively *CL_i aggregates CL_k*.

$If\ (CL_i\ G\ CL_j) \wedge (CL_j\ A\ CL_k) \rightarrow (CL_i\ A\ CL_k)$ (Transitivity)

Rule 20: If *CL_i generalized CL_j* and *CL_i aggregates CL_k* then semantically *CL_j aggregates CL_k*.

$If\ (CL_i\ G\ CL_j) \wedge (CL_i\ A\ CL_k) \rightarrow (CL_j\ A\ CL_k)$ (Semantically)

Rule 21: If *CL$_i$ generalized CL$_j$* and *CL$_k$ aggregates CL$_i$* then transitively *CL$_k$ aggregates CL$_j$*.

$$If\ (CL_i\ G\ CL_j)\ \wedge\ (CL_k\ \ A\ CL_i)\ \rightarrow (CL_k\ \ A\ CL_j) \qquad \text{(Transitivity)}$$

Rule 22: If *CL$_i$ generalized CL$_j$* and *CL$_k$ aggregates CL$_j$* then semantically *CL$_k$ aggregates CL$_i$*.

$$If\ (CL_i\ \ G\ CL_j)\ \wedge\ (CL_k\ \ A\ CL_j)\ \rightarrow\ (CL_k\ \ A\ CL_i) \qquad \text{(Semantically)}$$

Rule 23: If *CL$_i$ associates CL$_j$* and *CL$_j$ associates CL$_k$* then transitively *CL$_i$ associates CL$_k$*.

$$If\ (CL_i\ S\ CL_j)\ \wedge\ (CL_j\ S\ CL_k)\ \rightarrow\ (CL_i\ S\ CL_k) \qquad \text{(Transitivity)}$$

Rule 24: If *CL$_i$ depends on CL$_j$* and *CL$_j$ depends on CL$_k$* then transitively *CL$_i$ depends on CL$_k$*.

$$If\ (CL_i\ \ D\ CL_j)\ \wedge\ (CL_j\ \ D\ CL_k)\ \rightarrow\ (CL_i\ \ D\ CL_k) \qquad \text{(Transitivity)}$$

7- ABSTRACTING CARDINALITY OF CLASS'S RELATIONS

This subsection will present rules for abstracting cardinalities of class's relation and the rules will be denoted by CRule. Abstracting cardinalities of class's relationship in a class diagram is only needed when dealing with association or aggregation. It involves component wise multiplication of the cardinalities of the left-hand sides' classes and the right-hand side's classes. For instance, in CRule 5 to get the cardinality of the first class of the abstracted relationship, multiply component-wise the cardinality of the first class on the left hand side with the cardinality of the third class on the left hand side. The same way, the cardinality of the second class of the abstracted relation, is obtained by multiplying component-wise the cardinalities of the second and fourth classes on the left-hand side.

CRule 1:

$$((CL_i)\ A\ ^{[a..b]}(CL_j)) \wedge ((CL_j)\ A\ ^{[c..d]}(CL_k) \rightarrow ((CL_i)\ A\ ^{[a*c...b*d]}(CL_k))$$

CRule 2:

$$((CL_i)\ A\ ^{[a..b]}(CL_j)) \wedge ((CL_j)^{[c..d]}\ S\ ^{[e..f]}(CL_k) \rightarrow ((CL_i)^{[c..d]}\ \ S\ ^{[a*e...b*f]}(CL_k))$$

CRule 3: $((CL_i)\ G\ (CL_j)) \wedge ((CL_j)^{[a..b]}\ S\ ^{[c..d]}(CL_k) \rightarrow ((CL_i)^{[a..b]}\ S\ ^{[c...d]}(CL_k))$

CRule 4: $((CL_i)\ G\ (CL_j)) \wedge ((CL_j)\ A\ ^{[a..b]}(CL_k) \rightarrow ((CL_i)\ \ A\ ^{[a..b]}(CL_k))$

CRule5:

$$((CL_i)^{[a..b]}\ S\ ^{[c..d]}(CL_j)) \wedge ((CL_j)^{[e..f]}\ S\ ^{[g..h]}(CL_k) \rightarrow ((CL_i)^{[a*e..b*f]}\ S\ ^{[c*g..d*h]}(CL_k))$$

Where the superscripts in the above rules represent the cardinalities of the class's relations, and other variables are as defined previously. The cardinalities of a super class, sub-class and a class with "[1]" cardinality, are replaced with "[1..1]" cardinality for easy multiplication.

8- FORMALIZATION OF VERTICAL SEMANTIC CONSISTENCY RULES OF CLASS DIAGRAM REFINEMENT

Definition 4: let "HCLD" and "LCLD" be a set of paired classes over class relations R in a high-level class diagram (HCD) and a low-level class diagram (LCD) respectively. Also, let "HCL" be a class in the high-level class diagram (HCD) and "LCL" be class in the low-level class diagram (LCD). Thus, based on Equation 1:

$$HCLD = \{< HCL_i\ Rj\ HCL_k > \mid i \neq k, 1 \leq i,k \leq n \in Z+, 0 \leq j \leq 5\} \quad (5)$$

$$LCLD = \{< LCL_i\ Rj\ LCL_k > \mid i \neq k, 1 \leq i,k \leq n \in Z+, 0 \leq j \leq 5\} \quad (6)$$

Where

$HCL_i, HCL_k \in HCD$ are classes of the high-level class diagram.

$LCL_i, LCL_k \in LCD$ are classes of the low-level class diagram.

$R_j \in$ R are possible relations between any two classes in a class diagram.

Proposition 1: A refined class diagram is vertically semantic inconsistent if it does not satisfy one of CDRR$_i$, for 0≤ i ≤3.

$$CD_{CDRR1 \wedge CDRR2 \wedge CDRR3} \longrightarrow CD_{consistent}$$

Based on Definition 1, 2, 3 and 4 established in this chapter, the three vertical semantic consistency rules CDRR1, CDRR2 and CDRR3 of a class diagram refinement to be addressed by this research are formulated as follows.

A. *Formulation of CDRR1*

The first rule CDRR1 states that: Every low-level class refines at most one high-level class. Thus, based on Definition 4:

$$HCL_i\, \mathcal{G}\ \mathrm{LCL}_j \rightarrow \nexists HCL_k\, \mathcal{G}\ \mathrm{LCL}_j \big| 1 \leq j \leq m, i \neq k, 1 \leq i,k \leq n \quad (9)$$

$$HCL_i\, \mathcal{A}\ LCL_j \rightarrow \nexists HCL_k\, \mathcal{A}\ \mathrm{LCL}_j \big| 1 \leq j \leq m, i \neq k, 1 \leq i,k \leq n \quad (10)$$

Where

$HCL = High - level\ class$

$LCL = Low - level\ class$

$\mathcal{G} = Generalized$

$\mathcal{A} = Aggregates$

B. *Formulation of CDRR2*

The second rule CDRR2 states that every high-level class has at least one low-level class, which refines the high-level class: ensures that every high-level class is refined. Based on Definition 4, class LCL$_i$ is said to be a subset of class HCL$_j$, if and only if a class HCL$_j$ generalized class LCL$_i$ or class HCL$_j$ aggregate class LCL$_i$.

Thus, If $HCL_j\, \mathcal{G}\ \mathrm{LCL}_i \rightarrow LCL_i \subseteq \mathrm{HCL}_j$ (9)

If $HCL_j\, \mathcal{A}\ \mathrm{LCL}_i \rightarrow LCL_i \subseteq HCL_j$ (10)

Also since CL$_j$ is a set as mentioned in Equation 4,

Thus, $CL_j \subseteq CL_j$ (A class "CL" is a subset of itself) (11)

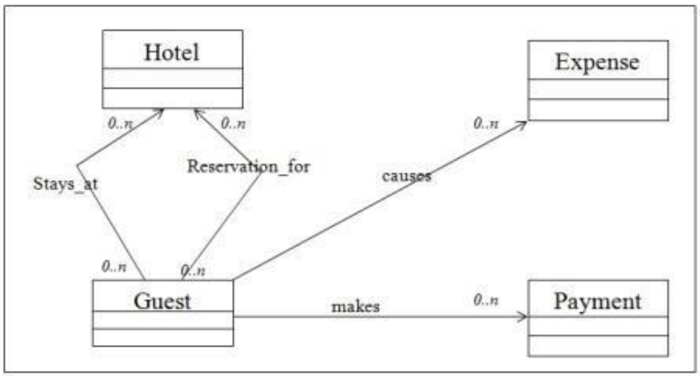

Figure 1 - High-Level Class Diagram of Hotel Management System [13]

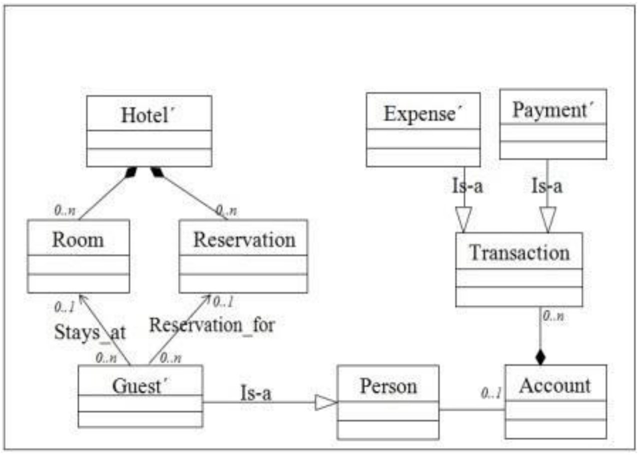

Figure 2 - Low-Level Class Diagram of Hotel Management System [13]

1) **Evaluation of CDRR1**

In order to evaluate formal specification of CDRR1, there is a need to check the consistency of low-level class diagram of Figure 2 with the high-level class diagram of Figure 1. To achieve this evaluation, the class diagrams need to be transformed to sets of paired classes over class's relations using Equation 5 and 6. Then, the two sets are used to check Equation 7 and 8. If, for all members of the two sets, the two equations are satisfied then it means the two-class diagrams are consistent with each other with respect to CDRR1. Otherwise, there is inconsistency between the two-class diagrams.

Recall that CDRR1 states that every low-level class refines at most one high-level class. From Equation 5, HCLD = {<HCL_i R_j HCL_k>} and from Equation 6, LCLD = {<LCL_i R_j LCL_k> } where HCLD is a set of paired high-level classes

class in the high-level diagram must be in the low-level class diagram and probably with further refinement. Recall that CDRR2 states that every high-level class has at least one low-level class, which refines the high-level class: ensures that every high-level class is refined. Also recall that Equation 12 is:

$$\forall HCL_i \in HCD \, \exists LCL_j \in LCD \, \big| \, (LCL_j \subseteq HCL_i), \; 1 \leq j \leq m, 1 \leq i \leq n$$

Moreover, from Equation 10, 11, 12, 13, Figure 1 and 2 it can be deduced that:

For

$$Hotel \in HCD, \exists Hotel \in LCD \, \big| \, Hotel \subseteq Hotel$$

(Based on Equation 12 and 11)

For

$$Hotel \in HCD, \exists Room \in LCD, \; Hotel \, A \; Room \; \rightarrow Room \subseteq Hotel$$

(Based on Equation 12 and 10)

For

$$Hotel \in HCD, \exists \mathrm{Reservation} \in LCD, Hotel \, A \; \mathrm{Reservation} \rightarrow \mathrm{Reservation} \subseteq Hotel$$

(Based on Equation 12 and 10)

Hence, For

$$Hotel \in HCD \, \exists \, Hotel, Room, Reservation \in LCD \mid Hotel \subseteq Hotel, \\ Room \subseteq Hotel, Reservation \subseteq Hotel$$

For

$$Guest \in HCD, \; \exists Guest \in LCD \; \big| \; Guest \subseteq Guest$$

(based on Equation 12 and 11)

For

$$Payment \in HCD \; \exists \; Payment \in LCD \mid Payment \subseteq Payment,$$

(based on Equation 12 and 11).

For

$$Expense \in HCD \; \exists \; Expense \in LCD \mid Expense \subseteq Expense$$

(based on Equation 12 and 11).

Thus conclusively, each of the classes in the high-level class diagram of the Hotel Management System of Figure 1 has at least one low-level class that refined it. Thereby, it satisfied Equation 12 and hence, Figure 2 is consistent with Figure 1 over CDRR2.

3) **Evaluation of CDRR3:**

Recall that CDRR3 states that the group of relationships between any two high-level classes must be identical with the group of relationships between their corresponding low-level classes. This ensures the existence of similar interaction of high-level classes in the low-level classes. To check this rule, there is a need to abstract Equation 15 using the needed rules from Rule 1 to Rule 24 and CRule 1 to CRule 5 of Chapter 4. The rules are applied iteratively

$$\big(\mathit{(Account)}\ A^{[0..n]}\mathit{(Expense)}\big) \wedge \big(\mathit{(Account)}^{[0..1]}\ S\ \mathit{(Guest)}\big) \\ \rightarrow \big(\mathit{(Expense)}^{[0..1]}\ S^{[0..n]}\mathit{(Guest)}\big)$$

(Applying Rule 3 and CRule 2) (23)

Combining the results of Equation 19 and 20

$$\big(\mathit{(Account)}\ A^{[0..n]}\mathit{(Payment)}\ \big) \wedge \big(\mathit{(Guest)}\ S^{[0..1]}\mathit{(Account)}\ \big) \\ \rightarrow \big(\mathit{(Guest)}\ S^{[0..n]}\mathit{(Payment)}\big)$$

(Applying Rule 4 and CRule 2) (24)

Combining the results of Equation 19 and 21

$$\big(\mathit{(Account)}\ A^{[0..n]}\mathit{(Payment)}\big) \wedge \big(\mathit{(Account)}^{[0..1]}\ S\mathit{(Guest)}\big) \\ \rightarrow \big(\mathit{(Payment)}^{[0..1]}\ S^{[0..n]}\mathit{(Guest)}\big)$$

(Applying Rule 3 and CRule 2) (25)

The final abstracted relations are the results of Equation 16, 17, 22, 23, 24, and 25. Let X be a set that contains these results as in Equation 26.

$$\therefore \quad \mathrm{X} = \left\{ \begin{array}{l} \big(\mathit{(Guest)}^{[0..n]}\ S\mathit{stay\_at}^{[0..n]}\mathit{(Hotel)}\big), \\ \big(\mathit{(Guest)}^{[0..n]}\ S\mathit{resev}^{[0..n]}\mathit{(Hotel)}\big), \\ \big(\mathit{(Guest)}\ S^{[0...n]}\mathit{(Expense)}\big), \\ \big(\mathit{(Guest)}\ S^{[0..n]}\mathit{(Payment)}\big), \\ \big(\mathit{(Expense)}^{[0..1]}\ S^{[0..n]}\mathit{(Guest)}\big), \\ \big(\mathit{(Payment)}^{[0..1]}\ S^{[0..n]}\mathit{(Guest)}\big) \end{array} \right\} \qquad (26)$$

X is the result of abstracting the low-level class diagram of the Hotel Management System (Figure 2). For the two diagrams to be consistent, Equation 13 must be satisfied by verifying that Equation 14 is a subset of Equation 26 as shown below:

$$HCLD = \left\{ \begin{array}{l} \big(\mathit{(Guest)}^{[0..n]} S\mathit{stay\_at}^{[0..1]}(\mathit{Hotel})\big), \\ \big(\mathit{(Guest)}^{[0..n]}\ S\mathit{resev}^{[0..1]}(\mathit{Hotel})\big), \\ \big(\mathit{(Guest)}\ S\mathit{causes}^{[0..n]}(\mathit{Expense})\big), \\ \big(\mathit{(Guest)}\ S\mathit{makes}^{[0..n]}(\mathit{Payment})\big) \end{array} \right\}$$

$$\subseteq$$

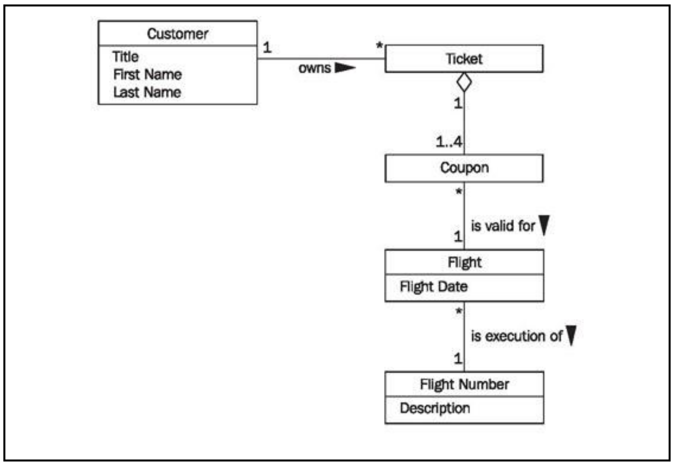

Figure 4 - Low-level class diagram of passenger's list system [23]

1) Evaluation of CDRR1

In order to evaluate formal specification of CDRR1, there is a need to check the consistency of low-level class diagram of Figure 4 with the high-level class diagram of Figure 3. To achieve this evaluation, the class diagrams need to be transformed to sets of paired classes over class's relations using Equation 5 and 6. The two sets will be used to check Equation 7 and 8. If, for all members of the two sets, the two equations are satisfied then it means the two-class diagrams are consistent with each other with respect to CDRR1. Otherwise, there is inconsistency between the two-class diagrams.

Recall that CDRR1 states that every low-level class refines at most one high-level class. From Equation 5, HCLD = {<HCL_i R_j HCL_k>} and from Equation 6, LCLD = {<LCL_i R_j LCL_k>} where HCLD is a set of paired high-level classes over class's relations, HCL_1 … HCL_n are high-level classes and R_1…..R_j are class's relations as defined in Equation 3. LCLD is a set of paired low-level classes and LCL_1 … LCL_m are classes in the low-level class diagram. Also recall that Equation 7 is

$$HCL_i \, G \, LCL_j \rightarrow \nexists HCL_k \, G \, LCL_j \,\middle|\, 1 \le j \le m, i \ne k, 1 \le i, k \le n$$

In addition, Equation 8 is

$$HCL_i \, A \, LCL_j \rightarrow \nexists HCL_k \, A \, LCL_j \,\middle|\, 1 \le j \le m, i \ne k, 1 \le i, k \le n \, .$$

From Figure 3

$$HCLD = \{((Customer)^{*} \, S_{\text{files with}} \, {}^{*}(\text{Flight})), ((Flight)^{*} \, S_{\text{is execution of}} \, {}^{1}(\text{Flight Number}))\} \quad (27)$$

From Figure 4

$$\left((\text{Ticket})^{1}\ \mathrm{A}^{\ 1..4}(\text{ Coupon})\right)\wedge\left((\mathit{Customer})^{1}\ \mathrm{S}_{\text{owns}}{}^{*}(\text{Ticket})\right)$$

$$\rightarrow\left((\mathit{Customer})^{1}\ \mathrm{S}_{\text{owns}}{}^{*}(\text{Coupon})\right)$$

(Applying Rule 4 and CRule 5)

(29)

Combining result of Equation 29 with

$$\left((\text{Coupon})^{*}\ \mathrm{S}_{\text{is valid for}}{}^{1}(\text{Flight})\right)$$

$$\left((\mathit{Customer})^{1}\ \mathrm{S}_{\text{owns}}{}^{*}(\text{Coupon})\right)\wedge\left((\text{Coupon})^{*}\ \mathrm{S}_{\text{is valid for}}{}^{1}(\text{Flight})\right)$$

$$\rightarrow\left((\text{Customer})^{*}\ \mathrm{S}^{\ *}(\text{Flight})\right)$$

(Applying Rule 23 and CRule 5) (30)

The results of abstracting Equation 28 are: $\left((\text{Flight})^{*}\ \mathrm{S}_{\text{is execution of}}{}^{1}(\text{Flight Number})\right)$, and the results of Equation 29 and 30. Thus, let X be a set that contains these results.

$$\therefore\ X=\left\{\begin{array}{l}\left((\mathit{Customer})^{1}\ \mathrm{S}_{\text{owns}}{}^{*}(\text{Coupon})\right),\\ \left((\text{Customer})^{*}\ \mathrm{S}^{\ *}(\text{Flight})\right),\\ \left((\text{Flight})^{*}\ \mathrm{S}_{\text{is execution of}}{}^{1}(\text{Flight Number})\right)\end{array}\right\} \quad (31)$$

X is the result of abstracting the low-level class diagram of the passenger list system (Figure 4). For the two diagrams (Figure 3 and 4) to be consistent Equation 13 must be satisfied by verifying that Equation 27 is a subset of Equation 31 as shown below:

$$HCLD=\left\{\begin{array}{l}\left((\mathit{Customer})^{*}\ \mathrm{S}_{\text{files with}}{}^{*}(\text{Flight})\right),\\ \left((\text{Flight})^{*}\ \mathrm{S}_{\text{is execution of}}{}^{1}(\text{Flight Number})\right)\end{array}\right\}$$

$$\subseteq$$

$$\therefore\ X=\left\{\begin{array}{l}\left((\mathit{Customer})^{1}\ \mathrm{S}_{\text{owns}}{}^{*}(\text{Coupon})\right),\\ \left((\text{Customer})^{*}\ \mathrm{S}^{\ *}(\text{Flight})\right),\\ \left((\text{Flight})^{*}\ \mathrm{S}_{\text{is execution of}}{}^{1}(\text{Flight Number})\right)\end{array}\right\}$$

Note: $\left((\mathit{Customer})^{*}\ \mathrm{S}_{\text{files with}}{}^{*}(\text{Flight})\right)\equiv\left((\text{Customer})^{*}\ \mathrm{S}^{\ *}(\text{Flight})\right)$

Hence, it can be seen that HCLD is a subset of X. This means that the group of relationships between any two high-level classes in Figure 3 is semantically identical with the group of relationships between their corresponding low-level classes in Figure 4. Based on Proposition 1 of Chapter 4, Figure 3 and Figure 4 are consistent, since they satisfied CDRR1, CDRR2 and CDRR3.

[7] S. W. Ambler, “UML 2.5: Do You Even Care?,” *Dr Dobb’s*, 2013. [Online]. Available: http://www.drdobbs.com/architecture-and-design/uml-25-do-you-even-care/240163702. [Accessed: 24-Mar-2014].

[8] F. J. Lucas, F. Molina, and A. Toval, “A systematic review of UML model consistency management,” *Inf. Softw. Technol.*, vol. 51, no. 12, pp. 1631–1645, Dec. 2009.

[9] G. Spanoudakis and A. Zisman, “Inconsistency management in software engineering: Survey and open research issues,” in *Handbook of software engineering and knowledge engineering*, vol. 1, World Science Publisher, pp. 329–380, 2001.

[10] J. Bansiya and C. G. Davis, “A Hierarchical Model for Object-Oriented Design Quality Assessment,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4–17, 2002.

[11] Z. Huzar, L. Kuzniarz, G. Reggio, and J. L. Sourrouille, “Consistency Problems in UML-Based Software Development,” *UML Model. Lang. Appl.*, pp. 1–12, 2005.

[12] B. Dobing and J. Parsons, “How UML is used,” *Commun. ACM*, vol. 49, no. 5, pp. 109–114, 2006.

[13] W. S. W. Shen, K. W. K. Wang, and A. Egyed, “An Efficient and Scalable Approach to Correct Class Model Refinement,” *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 515–533, 2009.

[14] G. Engels, J. M. Küster, R. Heckel, and L. Groenewegen, “A Methodology for Specifying and Analyzing Consistency of Object-Oriented Behavioral Models,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 26, pp. 186–195, 2001.

[15] A. Khalil and J. Dingel, “Khalil, A., & Dingel, J. (2013). Supporting the evolution of UML models in model driven software development: A Survey. Technical Report, School of Computing, Queen’s University, Canada.,” 2013.

[16] D. Torre and M. Genero, “UML Consistency Rules : A Systematic Mapping Study,” 2014.

[17] V. Lima, C. Talhi, D. Mouheb, M. Debbabi, L. Wang, and M. Pourzandi, “Formal Verification and Validation of UML 2.0 Sequence Diagrams using Source and Destination of Messages,” *Electron. Notes Theor. Comput. Sci.*, vol. 254, pp. 143–160, Oct. 2009.