AN ALGEBRAIC APPROACH TO CONSISTENCY CHECKING BETWEEN CLASS DIAGRAMS

HIDEKAZU ENJO, MOTONARI TANABU, JUNICHI IIJIMA

NTT DATA Corporation, Yokohama National University, Tokyo Institute of Technology
enjouh@nttdata.co.jp, tanabu@ynu.ac.jp, iijima.j.aa@m.titech.ac.jp
3-3-9 Koto-ku, Tokyo 135-8671, Japan, 79-4 Tokiwadai, Hodogaya, Yokohama 240-8501, Japan,
2-12-1 Ookayama, Meguro-ku, Tokyo 152-8550, Japan
+81-50-5546-2297, +81-45-339-3744, +81-3-5734-3942

ABSTRACT
There are several diagram methods for data modeling like a class diagram. It is very hard to describe a big data model of a large enterprise system into one diagram. A large set of partial data models are used during designing an information system for a large enterprise. The skill of modelers makes fluctuation and discrepancy among data models. It is necessary how to keep consistency among data models. There are two kinds of inconsistency among data models described in class diagrams. One is inconsistency between two data models like differences between attributes, data types, or multiplicities of same name classes. The other is inconsistency depending on order of divides and mergers for data models. We present syntax and semantics of a class diagram describing a data model for foundation for class diagram algebra. Then we introduce algebraic structure for syntactical and semantical merger operations on class diagrams and the consolidation condition. Satisfying the consolidation condition keeps consistency of class diagrams syntactically and semantically. We also show associative law and commutative law of the syntactical and semantical merger operations for the class diagram algebra that keeps avoiding inconsistency depending on order of divide and merging.

Keywords: class diagram algebra, data modeling, consistency checking
INTRODUCTION
In order to design a large enterprise system to handle huge complex information, it is important to analyze a lot of requirements and make a model of big complicated information. There are several diagram methods to support data modeling like class diagrams. However, it is very hard to describe a big data model of a large enterprise system into one diagram. A large set of partial data models are used during designing an information system for a large enterprise. This situation raises a risk of embedding inconsistency because the skill of modelers makes fluctuation and discrepancy among data models. And inconsistency within the set of partial data models decreases the quality of the data model. It is necessary how to keep consistency among data models.

The Unified Modeling Language (UML) (UML, 2005) is a standard modeling language, especially for object oriented design. Class diagrams, which are a type of UML diagrams, are describing static views of data models. So it is necessary to clear how to keep consistency among class diagrams. There are two kinds of inconsistency among data models described in class diagrams. One is inconsistency between two data models like differences between attributes, data types, or multiplicities of same name classes. The other is inconsistency depending on order of divides and mergers for data models.

Inconsistency between two parts of a model
It is easy to merge two parts of model like fig.1 and fig.2 and the result is fig.3 because same classes are merged into one class in a result diagram and same associations are merged into one association of that.

![Figure 1 – Example (1) of a Partial Data Model](image1)

![Figure 2 – Example (2) of a Partial Data Model](image2)
However, it is not easy to merge two parts of a model if there are two classes of same name but different set of attributes. For example, fig.1 and fig.4 have a same named class – “Order Details” – but a set of attributes are different. A class named Order Details in fig.1 has three attributes named “Goods,” “Num” and “Remarks” but that of fig.4 has only “Goods” and “Num.”

There are several ways for merging. For example, one is a merge of two classes when same name and same set of attributes. The result is fig.5, when merging fig.1 and fig.4 in the strict way. However, the class diagram of fig.5 is awkward because two classes with same name but attributes are different in one class diagram.
Another way for merging two class diagrams is to select the class in the first diagram when there is an inconsistent class. Fig.6 is the result of merging fig.1 and fig.4.

Also associations named “PO” in fig.1 and fig.7 are different because an association named “PO” in fig.1 associates among classes named “Buyer,” “Order Details” and “Order” but that of fig.7 associates among classes named “Order,” “Order Details” and “Seller.”

Figure 5 – Result of Merging Example (1) and Example of an Inconsistent Class

Figure 6 – Another Result of Merging Example (1) and Example of an Inconsistent Class

Figure 7 – Example of an Inconsistent Association
There are also several ways for merging two associations. One is a merger of two associations when same name and same set of related classes. The result is fig.8, when merging fig.1 and fig.7 in the strict way. However, the class diagram of fig.8 is also awkward because two associations with same name but set of related classes are different in one class diagram.

Modification of associations is also other merging way. The diagram of Fig.9 is the result of association modification during merging from a triple association of “Buyer,” “Order Details” and “Order” and that of “Order,” “Order Details” and “Seller” to a quadruple association of “Buyer,” “Order Details,” “Order,” “Seller.”

There are many ways of merging but in this paper we propose a simple operation of merging and constraints to class diagrams for clearing the algebraic structure of class diagrams.
Inconsistency depending on a sequence of merging and division

There are many sequences of merging if many class diagrams are merged. For example, one sequence is merging a diagram A and a diagram B then merging a diagram C like \((A \bullet B) \bullet C\). Another sequence is merging a diagram B and a diagram C then merging a diagram A like \((B \bullet C) \bullet A\). If merging operation is associative and commutative, results of all merging sequences are same. However, a merging way selecting the class in the first diagram violates commutative law. For example, fig.6 is the result of fig.1 and fig.4 with selecting the class from a first diagram and fig.10 is the result of fig.4 and fig.1. Two diagrams are different because merging operation with selecting the class from a first diagram isn’t commutative. So merging operation is preferred commutative and associative. Additionally, we propose a simple associative and commutative operation of merging in this paper.

![Diagram](image)

Figure 10 – Result of Example Merging an Inconsistent Class and Example (1)

RELATED WORK

There are several studies related consistency analyses for class diagrams. Tsiolakis and Ehrig (Tsiolakis, 2000) analyzed the consistency of class and sequence diagrams by using attributed graph grammars. Xia and Kane (Xia, 2003) analyzed the consistency of class and sequence diagrams. Kösters, Six and Winter (Kösters, 2001) analyzed the consistency of use case and class diagrams. Yeung (Yeung, 2004) analyzed the consistency of class and state diagrams. They focused between different types of UML diagrams. Berardi, Calvanese, and Giacomo (Berardi, 2005) presented the correspondence between class diagrams and Description Logics, which enable us to utilize Description Logics based systems for reasoning on class diagrams. Szlenk (Szlenk, 2006) presented the mathematical definition of class diagrams and studied the consistency within a single class diagram. Kaneiwa and Satoh (Kaneiwa, 2006) propose optimized algorithms that compute respective consistencies for class diagrams based on first order predicate logic. Sabetzadeh and Easterbrook (Sabetzadeh, 2005) studied the merging of class diagrams to gain a unified perspective. However, they focused how to merge inconsistent and incomplete class diagrams for requirement engineering and they haven’t been clear detail of algebraic structure for a merging operation.
NOTATION
A set \textit{Name} is a set of name of all elements consist of class diagram including a class name, an association name, an attribute name and a role name. A set \textit{DataType} is a set of data type, where type of all data including integer and character represented by \{\textit{Int, Bool, Char, String, Date, ...}\}. A multiplicity \((m, n)\) is a pair of lower bound \(m\) and upper bound \(n\), where \(0 \leq m \leq n, m \in N_0, n \in N_1\). A set \(N_0\) is a set of integer grater than 0. A set \(N_1\) is a set of integer grater than 1. \(\prod A\) denotes the set of all subsets of \(A\).

THE SYNTAX OF A CLASS DIAGRAM
A class diagram is a type of UML and employed to model concepts in static views for an information structure of enterprise system consisting of classes and their interrelationships. A class diagram has been represented by graphical and abstract syntax. In this section, we focus abstract syntax for simplifying discussions. We define the abstract syntax of the class diagram as followed:

**Syntactical Definition of a class**
A class \(c\) is a pair \((\text{Name}(c), \text{Attrs}(c))\), where \text{Name}(c) \in \text{Name} is a name of the class \(c\) and \text{Attrs}(c) is a list of attributes in the class \(c\). An attribute list \text{Attrs}(c) is a finite list \((\text{Attr}(c,1),...,\text{Attr}(c,n))\) of finite number \(n\). An attribute \text{Attr}(c,i) is a triple of \((\text{Name}(c,i), \text{Type}(c,i), \text{Multi}(c,i))\), where \text{Name}(c,i) \in \text{Name} is a name of \(i\) th attribute and a member of a set \(\text{Name}\). \text{Type}(c,i) \in \text{DataType} is a type of \(i\) th attribute and a member of a set \(\text{DataType}\). \text{Multi}(c,i) \in \text{Multiplicity} is a multiplicity of \(i\) th attribute and a member of a set \(\text{Multiplicity}\). A set \text{Class} is a set of all classes.

<table>
<thead>
<tr>
<th>Order</th>
<th>Contact: String[1..*]</th>
<th>Tel: String[1..1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\text{Buyer}, (\text{Contact, String, (1,*))}, (\text{Tel, String, (1,1}))),</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 11 – Example of a Class**

**Syntactical Definition of an association**
An association \(a\) is a pair \((\text{Name}(a), \text{Assocs}(a))\), where \text{Name}(a) \in \text{Name} is a name of the association \(a\) and \text{Assocs}(a) is a list of associated classes in the association \(a\). An list \text{Assocs}(a) of associated classes is a finite list \((\text{Assoc}(a,1),...,\text{Assoc}(a,n))\) of finite number \(n\). An associated class \text{Assoc}(a,i) is a triple of \((\text{Role}(a,i), \text{Multi}(a,i), \text{AC}(a,i))\), where \text{Role}(a,i) \in \text{Name} is a name of \(i\) th association, \text{Multi}(a,i) \in \text{Multiplicity} is a multiplicity of \(i\) th association, \text{AC}(a,i) \in \text{Class} is a related class of \(i\) th association. A length of the list \text{Assocs}(a) must be at least 2. A set \text{Association} is a set of all associations.
Syntactical Definition of a class diagram

A class diagram is a pair \((C, A)\), where \(C \in \text{Class}\) is a subset of \(\text{Class}\) and \(A \in \text{Association}\) is a subset of \(\text{Association}\). A class diagram \((C, A)\) suffers following conditions.

Condition 1: if names of two classes are different, two classes are different.

\[
\forall c_i \in C, \forall c_j \in C, \text{Name}(c_i) = \text{Name}(c_j) \Rightarrow c_i = c_j \quad [1]
\]

Condition 2: if names of two associations are different, two associations are different.

\[
\forall a_i \in A, \forall a_j \in A, \text{Name}(a_i) = \text{Name}(a_j) \Rightarrow a_i = a_j \quad [2]
\]

Condition 3: all associated classes of an association are included in the set of classes in the class diagram.

\[
\forall a \in A, \forall (r_i, m_i, ac_i) \in \text{Assocs}(a) \Rightarrow ac_i \in C \quad [3]
\]

In order to reduce the complexity, we consider eliminating some components as operation and generalization. Because we discuss only behavior of data model rather than object model.

Syntactical algebra for class diagrams

A merger operation is defined in an analogous to union of set because same classes are merged into one class and same associations are merged into one association. We define a merger operation as followed:

Definition of a Syntactical Merger Operation

Given two class diagrams \((C_X, A_X)\) and \((C_Y, A_Y)\), a syntactical merger operation \(\triangledown: (C, A) \times (C, A) \rightarrow (C, A)\) is defined as \((C_X \cup C_Y, A_X \cup A_Y)\).

\[
(C_X, A_X) \triangledown (C_Y, A_Y) \Leftrightarrow (C_X \cup C_Y, A_X \cup A_Y) \quad [4]
\]
However, if this syntactical merger operation is applied class diagrams of fig.1 and fig.4, the result of fig.5 violates condition 1 of class diagram definition. The result of this merger operation on class diagrams of fig.1 and fig.7 violates condition 2 of class diagram definition. Although we omit the proof because of space limitations, following conditions are formed.

**Proposition 1**

The syntactical merger operation \((\nu)\) on class diagrams is closed if and only if following a consolidation condition is satisfied.

Consolidation condition: Let \((C_X, A_X)\) and \((C_Y, A_Y)\) be class diagrams. If names of any classes in \(C_X\) and \(C_Y\) are same, those classes are same. If name of any associations in \(A_X\) and \(A_Y\) are same, those classes are same.

\[
\forall c_1 \in C_1, \forall c_2 \in C_2, Name(c_1) = Name(c_2) \Rightarrow c_1 = c_2 \\
\forall a_1 \in A_1, \forall a_2 \in A_2, Name(a_1) = Name(a_2) \Rightarrow a_1 = a_2 \tag{5}
\]

The syntactical merger operation \((\nu)\) is closed on the class diagrams which satisfy the consolidation condition. This operation and conditions are simple enough for merging.

Although we omit the proof because of space limitations, the syntactical merger operation \((\nu)\) is associative and commutative like set operation union. It is a commutative semi-group that the algebraic structure of class diagrams which satisfy the consolidation condition and the syntactical merger operation \((\nu)\). All results are same where any sequence of merging with the syntactical merger operation \((\nu)\).
THE SEMANTICS OF A CLASS DIAGRAM
An instance of a class is called an object. An instance of an association is called a link. A link is a connection between two or more objects of the classes. We introduce complex data as a domain and interpretation mapping for the semantics of class diagram as followed:

A Domain for the Semantics of a Class Diagram
A object is a pair $(x,(v_{1,1},...,v_{1,j_h}),..., (v_{i,1},...,v_{i,j_i})) \in (ID \times Value^*)$, where $x \in ID$ is a name of identifier like $O1$, $(v_{i,1},...,v_{i,j_i})$ is a value list like $(Yamada,Suzuki)$ and $((v_{i,1},...,v_{1,j_h}),..., (v_{i,1},...,v_{i,j_i}))$ is a list of value list like $((Yamada,Suzuki),(030009999))$. A set $Objects$ is a set of all objects.

$O1$

(Yamada,Suzuki)

030009999

$(O1,((Yamada,Suzuki),(030009999)))$

Figure 14 – Example of an Object

A link is a pair $(y,(o_1,...,o_k)) \in (ID \times Objects^*)$, where $y \in ID$ is a name of identifier like $L1$, $(o_1,...,o_k)$ is a list of objects and $k \in N$. A set $Links$ is a set of all links.

$L1$

$O1$

(Yamada,Suzuki)

030009999

$L2$

2008-08-26

$O4$

Linux

$O4$

$(L1,(O1,((Yamada,Suzuki),(030009999))), (O2,((2008-08-26))), (O4,((Linux),(1))))$

Figure 15 – Example of a Link

A complex data is a pair $(O,L) \in (\prod Objects, \prod Links)$, where $O \in Objects$ is a set of objects, $L \in Links$ is a set of links, satisfying $(\forall(y,(x_1,...,x_n)) \in L, \forall k, 1 \leq k \leq n, x_k \in O)$ that means all objects related all links $L$ are included in objects $O$. A set $ComplexData$ is a set of all complex data.
An Interpretation Mapping for the Semantics of a Class Diagram

An interpretation mapping \( I(t) \mapsto P_t(v) \) of a data type name \( t \) is a map to a predicate \( P_t(v) \iff v \in D_t \), where \( D_t \) is all values of data type named \( t \). For example, data type \texttt{Date} is mapped to \( P_{\text{Date}}(v) \), where \( P_{\text{Date}}("2008-08-26") \) is true.

An interpretation mapping \( I(c) \mapsto P_c(x) \) of a class \( c \) is a map to a predicate \( P_c(x) \), where number of attribute within lower bound and upper bound and a value of an attribute is a member of the data type related an attribute.

\[
\text{Buyer} \mapsto P_{\text{Buyer}}(z) \\
\begin{align*}
P_{\text{Buyer}}((w, ((v_{1,1}, \ldots, v_{1,j_1}), \ldots, (v_{2,1})))) & \iff \\
&w \in ID \land (1 \leq j_1 \land \forall k, 1 \leq k \leq j_1, P_{\text{String}}(v_{1,k}) \land P_{\text{String}}(v_{2,k}))
\end{align*}
\]

Figure 17 – Example of an interpretation mapping for a Class

An interpretation mapping \( I(a) \mapsto P_a(z) \) of an association \( a \) is a map to \( P_a(z) \), where all objects related links are associated to classes defined as followed:

\[
(P_a((y, (x_1, \ldots, x_n))) \iff y \in ID \land P_{\text{AC}(a,0)}(x_1) \land \ldots \land P_{\text{AC}(a,n)}(x_n)) \]  

[6]
An interpretation mapping \( I(\text{Multi}(a)) \mapsto P_\text{Multi}(a)(x, z) \) of a multiplicity \( \text{Multi}(a) \) is a map to a predicate \( P_{\text{Multi}(a)}(x, z) \), where a number of links related each object is within lower bound and upper bound as followed:

\[
P_{\text{Multi}(a)}(o, L) \iff \\
\forall k, 1 \leq k \leq n_a, \\
(P_{\text{AC}(a,k)}(o) \Rightarrow \exists_{x_1, x_{k-1}, x_k} x_k \exists_{x_{k+1}, x_{k+2}, \ldots, x_{n_a}, y}, \\
((y, (x_1, \ldots, x_{k-1}, o, x_{k+1}, \ldots, x_{n_a})) \in L \land y \in \text{ID} \land P_o((y, (x_1, \ldots, x_{k-1}, o, x_{k+1}, \ldots, x_{n_a}))))),
\]

\[
(P_{\text{AC}(a,k)}(o) \Rightarrow \exists_{x_1, x_{k-1}, x_k \ldots, x_{n_a}, y}, \\
((y, (x_1, \ldots, x_{k-1}, o, x_{k+1}, \ldots, x_{n_a})) \in L \land y \in \text{ID} \land P_o((y, (x_1, \ldots, x_{k-1}, o, x_{k+1}, \ldots, x_{n_a}))))).
\]

Figure 18 – Example of a interpretation mapping for an Association

An interpretation mapping \( I((\text{C}, A)) \mapsto P_{(\text{C}, A)}((O, L)) \) of a class diagram \((\text{C}, A)\) is a map to a predicate \( P_{(\text{C}, A)}((O, L)) \), where condition 1 for class is that all predicates \( P_{C}(z) \) mapped from classes are satisfied, condition 2 for association is that all predicates \( P_{A}(z) \) mapped from associations are satisfied and condition 3 for multiplicity is that all predicates \( P_{\text{Multi}(a)} \) mapped from multiplicity are satisfied.

Conditions are following.

\[
P_{(\text{C}, A)}((O, L)) \iff \\
((O, L) \in \text{ComplexData}), (\forall o \in O, \exists c \in \text{C}, P_{c}(o)),
\]

\[
(\forall I \in L, \exists a \in \text{A}, P_{a}(l)), (\forall o \in O, \forall a \in \text{A}, P_{\text{Multi}(a)}(o, L)).
\]

Figure 19 – Example of a interpretation mapping for a Multiplicity of an Association
\[(C, A) \rightarrow P_{(C,A)}((O, L)) \]
\[P_{(C,A)}((O, L)) \iff ((O, L) \in \text{ComplexData}) \land \\
(\forall o \in O, (P_{\text{Buyer}}(o) \lor P_{\text{Order}}(o) \lor P_{\text{Order Details}}(o))) \land \\
(\forall l \in L, P_{\text{PO}}(l)) \land (\forall o \in O, P_{\text{MultiPO}}(o, L)) \]

**Figure 20 – Example of an interpretation mapping for a Class Diagram**

**Semantical algebra for class diagrams**

A structure \( \mathfrak{S} \) of a class diagram is a pair \((D_M, I_M)\), where \( D_M \) is a domain which all elements consist of complex data and \( I_M \) is an interpretation mapping for data types, classes, associations, multiplicities of associations and class diagrams defined above. On this structure \( \mathfrak{S} \), we can evaluate whether a complex data is a instance of a class diagram through the complex data is assigned into the class diagram.

\[P_{(C,A)}([P_{\text{Buyer}}, ((O_1, ((Yamada, Suzuki), (030009999))), P_{\text{Order}}, ((O_2, ((2008 – 08 – 26)))]], \]
\[P_{\text{Order Details}}([O_3, ((PC, Laptops), (1))]), P_{\text{Order Details}}([O_4, ((Linux), (1))])], \]
\([P_{\text{PO}}(L_2, (O_1, ((Yamada, Suzuki), (030009999))), (O_2, ((2008 – 08 – 26)), (O_3, ((PC, Laptops), (1)))]), \]
\[P_{\text{PO}}(L_2, (O_1, ((Yamada, Suzuki), (030009999))), (O_2, ((2008 – 08 – 26)), (O_4, ((Linux), (1))))]) \]

**Figure 21 – Example of an Assignment to a Class Diagram**

Although we omit the proof because of space limitations, it is a model that substructure \( \mathfrak{S}_{(C,A)} \), which complex data in the domain of structure \( \mathfrak{S} \) are replaced with a set of all instances of the class diagram \((C,A)\).

**Definition of a Semantical Merger Operation**

Given two predicates \( P_{(C_x, A_x)} \) and \( P_{(C_y, A_y)} \) mapped from class diagrams \((C_x, A_x)\) and \((C_y, A_y)\), a semantical merger operation \( \triangledown \) is defined as followed:
\[ (P_{G_{X,A_{X}}}) \hat{\vee} P_{(G_{Y,A_{Y}})}((O,L)) \Leftrightarrow (O,L) \in \text{ComplexData}, \]
\[ \exists(O_X,L_X) \in \text{ComplexData}, \exists(O_Y,L_Y) \in \text{ComplexData}, \]
\[ (O_X \cup O_Y, L_X \cup L_Y) = (O,L), \]
\[ P_{(G_{X,A_{X}})}((O_X,L_X)), P_{(G_{Y,A_{Y}})}((O_Y,L_Y)), \]
\[ (\forall e \in C_X \cap C_Y, \forall o \in O, P_e(o) \Rightarrow (o \in O_X \cup O_Y)), \]
\[ (\forall a \in A_X \cap A_Y, \forall l \in L, P_a(l) \Rightarrow (l \in L_X \cup L_Y)) \]

The semantical merger operation \((\hat{\vee})\) is closed on a structure \(\mathcal{I}\) mapped from class diagrams which satisfy the consolidation condition. This semantical operation is simple enough for merging. Although we omit the proof because of space limitations, the following relation is formed.

**Proposition 2**

Let \((C_X,A_X)\) and \((C_Y,A_Y)\) be class diagrams and \(I\) be an interpretation mapping. There is a relation between the syntactical \((\vee)\) and semantical \((\hat{\vee})\) merger operations on class diagrams with the consolidation condition as followed:

\[ (I((C_X,A_X)) \vee I((C_Y,A_Y))((O,L)) \Leftrightarrow (P_{(G_{X,A_{X}})} \hat{\vee} P_{(G_{Y,A_{Y}})}((O,L)) \Leftrightarrow P_{(G_{X\cup Y,A_{X\cup Y}})}((O,L)) \Leftrightarrow \]
\[ I((C_X \cup C_Y,A_X \cup A_Y))((O,L)) \Leftrightarrow I((C_X,A_X) \hat{\vee} (C_Y,A_Y))((O,L)) \]

The semantical merger operation \((\hat{\vee})\) is associative and commutative as well as the syntactical merger operation. It is also a commutative semi-group that the algebraic structure of class diagrams which satisfy the consolidation condition and the semantical merger operation \((\hat{\vee})\). All results are same where any sequence of merging with the semantical merger operation \((\hat{\vee})\).

**CONCLUSION**

We present syntax and semantics of a class diagram describing a data model for foundation for class diagram algebra. Then we introduce algebraic structure for a merger operation on class diagrams and the consolidation condition. Satisfying the consolidation condition keeps consistency of class diagrams syntactically and semantically. We also show associative law and commutative law of the syntactical and semantical merger operation for class diagram algebra that keeps avoiding inconsistency depending on order of merging.

It means that the merger operation has good properties for avoiding inconsistency to handle class diagrams if the consolidation condition is satisfied. The consolidation condition and the merger operation are simple but very powerful because it is only necessary for any modelers to check the consolidation condition for keeping consistency during designing enterprise systems. Moreover, it is possible to implement functions of checking the consolidation condition and the merger operation into
UML modeling tools. However, another operation like difference is needed because the merger operation isn’t able to divide a big class diagram into partial class diagrams.

REFERENCES
UML 2.0 Superstructure Specification (formal/05-07-04), (2005), Object Management Group.