An Algebraic Approach to Validation of Class Diagram with Constraints

Kazuki Munakata and Kokichi Futatsugi
Graduate School of Information Science,
Japan Advanced Institute of Science and Technology,
1-1 Asahidai, Tastunokuchi, Nomi, Ishikawa, 923-1292 Japan
E-mail: {kmunaka, kokichi}@jaist.ac.jp

Abstract: In this paper, we propose Class Diagram With Constraints (CDWC) as an object oriented modeling technique which makes validation possible in software development. CDWC is a simple and basic model for the object oriented analysis, and has a reasonable strictness for software developers. CDWC consists of class diagrams and constraints (invariant and pre/post conditions), using UML and a subset of OCL. We introduce a method of validation of CDWC using the verification technique of algebraic formal specification language CafeOBJ.

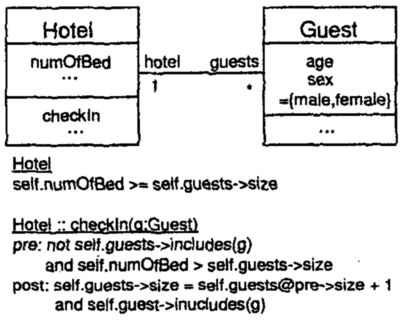
1. Introduction

In object oriented analysis, the target system is specified in the view point of *objects*, traditionally with graphical notation. Therefore, it is easy for a practical developer to specify with the notation. As object oriented modeling language, Unified Modeling Language (UML)[5] is standard one. UML is based on some diagrams for graphical modeling. Such an object oriented modeling becomes popular in developers, and contributes to the productivity of software development. However, validation of the specifications (e.g. specifications using UML diagrams) is not available because of the lack of strictness. Therefore, object oriented modeling method with reasonable strictness which makes validation possible, is desired.

One of the approaches for that is the textual description of *constraints* (i.e. invariant, pre/post conditions for attributes or operations of objects) as the annotation for graphical diagrams. The advantage of describing constraints to object oriented models in the analysis phase is as follows: Firstly the ambiguity of specifications is removed and the improvement of the consistency is expected, keeping object oriented concepts. Secondly the property satisfied in the specification can be clarified by the invariant constraints. This point is important as requirement analysis. Finally, the description of constraints can be used as the information for validation if constraints is described formally.

UML contains an annotation language for UML diagrams, Object Constraint Language(OCL)[4]. OCL is a textual language to write constraints on diagrams, declaratively. The language specification of OCL may become big and complex one, and its formal semantics is not defined sufficiently. Therefore it is very difficult to support the full set of OCL for validation. On the other hand, there are some methodologies which attaches importance to the description of constraints (e.g. Catalysis, Syntropy and so on). In these methodologies, the effective usage of constraints in object oriented modeling is discussed. In Catalysis, the use of UML and OCL is assumed. However, these methodologies are not concerned with how to validate whether constraints are satisfied in the specification.

In this paper, we propose class diagram with constraints (CDWC) as an object oriented model which makes validation possible. CDWC is a simple and basic model for the object oriented analysis, and has a reasonable strictness for practical developers. CDWC consists of class diagrams and constraints (invariant and pre/post conditions) using UML and a subset of OCL. We restrict OCL so as to verify, keeping the concepts and benefits of original OCL , and we call this *Simple OCL*.



```
Hotel
self.numOfBed >= self.guests->size

Hotel :: checkIn(g:Guest)
pre: not self.guests->includes(g)
     and self.numOfBed > self.guests->size
post: self.guests->size = self.guests@pre->size + 1
      and self.guest->inuclude(g)
```

Figure 1. the Class Diagram with Constraints

Our basic approach for validation of this model is to translate it into a formal specification language. CafeOBJ[1] is an algebraic specification language. It has a rigorous logical semantics, and can treat object oriented concepts. Moreover it can execute specifications as an environment for supports of its verification. Therefore, we use CafeOBJ for validation of CDWC.

The remainder of this paper is organized as follows: Section 2 introduce an overview of CafeOBJ as preliminaries. Section 3 gives the basic concept and the notation of CDWC (including simple OCL). Section 4 presents the translation rule from CDWC into CafeOBJ specification (called behavioural specification), and by this we give semantics to CDWC. Section 5 shows a method of validation of CDWC with CafeOBJ environment. Section 6 mentions related works. Finally we concludes this paper in Section 7.

2. Preliminaries

In this section, we are going to explain briefly the basic concepts of algebraic specification language CafeOBJ, and especially the idea of behavioural specification.

2.1 Algebraic Specification Language CafeOBJ

CafeOBJ[1] is multi-paradigm algebraic specification language which successor of OBJ. CafeOBJ is executable which means that it can be used for rapid prototyping and theorem proving. CafeOBJ is based on the combination of several logic consisting of order sorted algebra, hidden algebra and rewriting logic.

2.2 Behavioural Specification

Generally, behavioural specifications are specifications that abstract their structure and focus on their behaviour by using observations of the systems. In CafeOBJ, we call a specification *behavioural specification* if it is based on hidden algebra[2]. One of the advantage of this is we can abstractly specify systems and focus on their important properties. In hidden algebra, a system is regarded as a kind of black box in the sense that we can observe its state only by using special operators called *observation.* The state of a system can be changed only by using special operators called *action.*

3. Class Diagrams with Constraints (CDWC)

In this section, we propose Class Diagram With Constraints (CDWC), as an object oriented model which makes the semantic analysis possible.

3.1 Basic Concept

In general, a class diagram characterizes only the structural aspect of a system. However, CDWC supports not only the description of static structure but also a dynamical aspect, that is the description of how the state of an object changes by the apply of the method. We regard a object as an abstract state machine, and we think that the basic concept of object oriented modeling is the description of the whole effect which each objects have during changing the state of the other object. Therefore, it is reasonable to suppose that CDWC is basic model for object oriented analysis. In CDWC, a semantic analysis is available, that is CDWC provides validation of invariant conditions and simulated execution.

3.2 Notation

A CDWC specification consists of the graphical part (written with UML class diagram) and the textual part (written with Simple OCL).

Class Diagram:

A Class diagram consists of the class boxes and the associations. A class box has class the name field, the attribute field and the operation field, basically according to UML class notation. In CDWC, there are some restrictions. The visibilities of all attributes and operations are public. Each attribute has an initial value. Moreover, each operation is a setting method, i.e. the state of the class must be changed when the operation is applied. In Simple OCL, it is possible to refer the value of attributes without getting methods.

There are three kinds of associations, a relation, an aggregation and an inheritance. An association consists of the role names and the multiplicities. A relation has bi-directional navigability. The notation and semantics of association accord with UML's association basically.

Simple OCL:

We can declare invariant conditions to attributes and pre/post conditions to operations with OCL. Simple OCL is a subset of OCL, whose expressions are specialized for validations. Simple OCL supports original OCL important concepts (such as a model type, a collection type, a navigation through the associations, and a multiplicity constraint). Although Simple OCL is restricted, we can describe a constraint for attribute values and effects of operations. The effects of a operation is described by changes between the values of attributes immediately before and after the operation has happened. The most typical assertion for describing the effects is a Simple OCL expression (s_ocl exp) whose form is

```
context Class :: operation(...)
[object].[attribute] =
                    F([object].[attribute]@pre)
```

where `[object].[attribute]` in the left-hand side and `[object].[attribute]@pre` in the right-hand side are the attribute values immediately after and before the operation has happened, respectively, and `F` is a function. A s_ocl exp has the context (`Class :: operation(...)`) which constraints belong to.

Simple OCL has types as well as original OCL, basic types (Boolean, Integer, String and Enum), Object type and Collection type. `[object]` has Object type. We can refer an attribute of the object by the dot notation like `[object].[attribute]`. An element having Object type is *self*, *self@pre*, or *self.role.* By using *self.role* we can describe an associated class indicated by a role name on its association. By this notation, it is possible to refer an attribute value of an associated class with *dot navigation*, like `[object].[role]` ··· `.[role].[attribute]`. This is a key concept in describing the collaboration of objects. Moreover the type of *self.role* is dependent on the multiplicity of *role*'s association: Object type for `1` or Collection type for `*` (zero or more). When *self.role* has Collection type, it represents the set of associated objects. When *self.role* has Collection type, it represents the set of associated objects. For Collection type, we can use some useful collection operators: *size,include(o:object)*, and *isEmpty.* For example `self.role->size` represents the number of elements in the set *self.role.*

The following is an example of Simple OCL in the Hotel specification.

```
context Hotel::checkIn(g:Guest)
post: self.guests->size
                  = self.guests@pre->size + 1
```

This s_ocl exp shows that after checking in a hotel, the total of guests increases by one. We show another example of Simple OCL, which represents an invariant condition which attributes should always satisfy in all states.

```
context Hotel
self.numOfBed >= self.guests->size
```

This s_ocl exp shows that the number of the guests does not exceed the number of the Hotel's beds.

Since these constraints are specified declaratively (i.e. side effect freely), CDWC is suitable for high abstract specifications (e.g. requirement specifications).

4. Translation from CDWC into CafeOBJ

In this section, we introduce the guiding principle of the translation from CDWC into behavioural specification. By the definition of the translation rule into behavioural specification based on hidden algebra, we attempt to give the precise semantics to CDWC (i.e. UML class diagram and Simple OCL). The basic principle of the translation is that a class diagram corresponds to a signature, a pre/post condition corresponds to an equation in behavioural specification. An invariant condition corresponds to the predicate which should be validated. A summary of the guiding principle is as follows.

Translation 1: Class Diagrams

A class corresponds to a module based on *hidden algebra*, where state space is treated as *hidden sort*. A class's name corresponds to the name of the module and the principal sort. An attribute and an operation correspond to an *observational operator* and an *action operator* respectively. However class diagram basically corresponds to *signature*, some *equations* are derived from attributes if they have the initial values. For basic types used by the definition of attributes or operations, built-in modules in CafeOBJ environment are imported.

An association between two classes has navigability to each other. In behavioural specification, we use a *projection operator*[3] as a special operator to refer the other class. We prepare a meta-module (SYSTEM) for realizing mutual references of associated classes. SYSTEM module can refer each class module by the projection operator. The multiplicity about an association is realized by the parameterized projection operator with object identifier (OID).

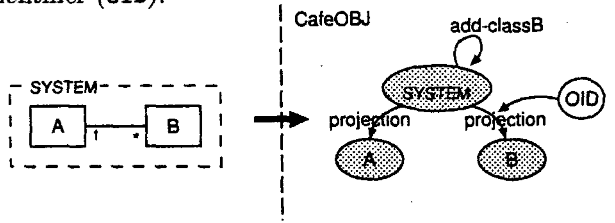


Figure 2. the treatment of multiple association

Translation 2: Simple OCL descriptions:

A s_ocl exp is translated into *equations* in behavioural specification. Simple OCL has basic type, Object type, and Collection type). A simple type (such as Boolean, Integer, and so on) corresponds to *abstract data type* (*visible sort* in behavioural specification), and an object type corresponds to a *observational* or an *projection* operator. For collection type we prepare COLLECTION module (which is imported to SYSTEM module). Some examples of translation are as follows:

Example 1 of translation:

```
<Simple OCL assertion>              <in Class module in CafeOBJ>
context class::op(p1,...,pn)   ➡   eq atr(op(p1,...,pn,S))
post self.atr                                    = F(atr(S),...,pn)
     = F(sefl@pre,p1,...pn)
```

A post condition is translated into an *equation*, naturally. Here, F is a function such as basic type's functions, and S is a variable of the hidden sort, which corresponds to self.

Example 2 of translation:

```
<Simple OCL assertion>         <in SYSTEM module in CafeOBJ>
context class::op(p)       ➡  ceq atr(op(P,S)) = P
pre: self.role.atr' = true                 if atr'(class(S)) == true
post: self.atr = p             ceq atr(op(P,S)) = S
                                           if atr'(class(S)) == false
```

A pre condition is translated into a conditional part of *conditional equations*. Navigations with association correspond to *projection operators* from SYSTEM module to each Class module.

The result of the translation from Hotel's specification into behavioural specification is as follows:

```
mod* SYSTEM{
  pr(GUEST + HOTEL + COLLECTION(OBJECTID))
  *[ System ]*
  -- projection
  op hotel : System -> Hotel
  op guest : Oid System -> Guest
  op collection : System -> Collection
  -- observation
  bop numOfBed : System -> Nat
  bop age : Oid System -> Nat
  bop sex : Oid System -> Sex
  -- action
  bop checkIn : Oid Nat Sex System -> System
   ....
-- equations for OCL constraint
  ceq collection(checkIn(O, N, S, C))
    = add(O, collection(C))
      if not(includes(O,C))
 and (numOfBed(C) > size(C)) .
  ceq collection(checkIn(O, N, S, C))
    = collection(C)
      if not(includes(O,C))
 and (numOfBed(C) == size(C)) .
  ceq collection(checkIn(O, N, S, C))
    = collection(C)
      if not(includes(O,C))
 and (numOfBed(C) < size(C)) .
  ceq collection(checkIn(O, N, S, C))
    = collection(C)
      if includes(O,C) .
    ...
}
```

5. Validation

In this section, we show validation of an invariant condition, as an example of semantic analysis of CDWC with CafeOBJ environment. An invariant condition is the declaration of a property to be satisfied through all reachable states (all *snap shots*). In our framework, a snap shot including current states of all objects is defined as the term of `System` sort in `SYSTEM` module.

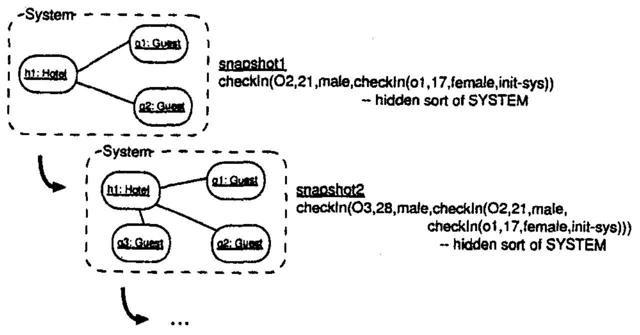


Figure 3. snap shots

Consider the following invariant condition.

`self.numOfBed >= self.guests->size`

This condition is described in behavioural specification as follows:

`numOfBed(s) >= size(s)`

for all s : `system`. We prove the invariant condition by the induction on `System`. That is, it suffices to show the following two cases:

- Base Case: It holds for the initial state (`P(init-sys)`).
- Inductive Case: Each action operator preserve it (`P(S) => P(operation(s)) for all operations`).

The following is a proof score of CafeOBJ according to this principle.

```
-- opening module SYSTEM
open SYSTEM
**> prove numOfBed(s) >= size(s)
**> by induction on s

**> base case ( s=init-system )
    : numOfBed(init-system) >= size(init-system)
red numOfBed(init-system) >= size(init-system) .
-- should be true

**> induction hypothesis ( s = sys )
    : numOfBed(sys) >= size(sys)
eq numOfBed(hotel(sys)) >= size(collection(sys))
    = true .

**> induction step ( s = checkIn(o, n, s, sys) )
**> case analysis for checkIn operation
**> case : include(o,sys) = true
eq includes(o, sys) = true .
red numOfBed(checkIn(o, n, s, sys))
          >= size(checkIn(o, n, s, sys)) .
-- should be true

**> case : include(o,sys) = false
eq includes(o, sys) = false .
red numOfBed(checkIn(o, n, s, sys))
          >= size(checkIn(o, n, s, sys)) .
-- should be true

**> QED for OCL invariant
close
```

By this validation of invariant condition, it is shown that the property is always satisfied under the pre/post conditions

6. Related Works

There are some studies about the semantical analysis of an object model which consist of a class structure and some constraints. The concepts of Alloy[6] are micromodels (light weight), analyzable, declarative and structural. The concepts of our study are similar to Alloy's ones. Alloy treats only static data structures (not objects or classes). CDWC is suited to popular object oriented modeling. Because a class is modeled as an abstract state machine. In [7], the semantics of ICL (OCL specialized in the description of an interface) is defined with an algebraic specification language. However, the validation of invariant constraints is not discussed.

7. Conclusion

In this paper, we have proposed CDWC as object oriented model which make validation possible, the principle of the translation to CafeOBJ specification, and the method to validate the model with CafeOBJ environment. Several studies had been made on validation of object oriented modeling (such as UML) with a formal approach. The key points of these studies are whether the formalism treats naturally object oriented models, and what is able to be validated on the formalism. In OCL constraints are basically described by the values of attributes. In behavioural specification the effect of an action is specified by the result of observation. Therefore CDWC is naturally treated in this formalism. Moreover we have confirmed that the proof of the invariant condition by structural induction is useful for validation of an important property on object oriented analysis.

References

[1] Răzvan Diaconescu and Kokichi Futatsugi, CafeOBJ Report: The Language, Proof Techniques, and Methodologies, World Scientific, 1998

[2] Joseph Goguen and Grant Malcolm. A hidden agenda: Technical Report CS97-538, UCSD Technical Report, 1997

[3] Shusaku Iida, Michihiro Matsumoto, Răzvan Diaconescu and Kokichi Futatsugi. Concurrent Object Composition in CafeOBJ: JAIST Research Report, IS-RR-98-0009S,1998

[4] J.Warmer and A.Kleppe. The Object Constraint Language Precise Modeling with UML: Addison-Wesley, 1999

[5] G.Booch et al. The Unified Modeling Language User Guide: Addison-Wesley, 1998

[6] Daniel Jackson. Alloy: A lightweight object modelling notation: Technical Report 797, MIT Laboratory for Computer Science, 2000

[7] Michel Bidoit et al. Correct Realizations of Interface Constraints with OCL: UML'99, number 1723 in LNCS, pages 399-415, Springer Verlag, 1999