A Formal Software Development Approach Based on COOZ and Refinement Calculus

Wang Yunfeng Li Bixin Pang Jun Zha Ming Zheng Guoliang *
National Key Laboratory for Novel Software Technology at Nanjing University
Nanjing, 210093, PR China
E-mail: zhenggl@nju.edu.cn

Abstract

Including Refinement Calculus into COOZ complements its disadvantage during design and implementation. The apartment between design and implement for construct and notation is removed as well. Then the software can be developed smoothly in the same frame. There is not corespondent object-oriented construct in existing Refinement Calculus. The combination of COOZ and Refinement Calculus can build an object-oriented frame, in which the specification in COOZ is refined stepwise to code by calculus. In the paper ,two development models are argued ,which are based mainly on COOZ and Refinement Calculus respectively. The first model is debated primarily. The data refinement and operation refinement is analyzed by example; the two method of operation refinement for OO formal specification is discussed simply; the frame transition rule from COOZ to C++ is argued.

1 Introduction

In early 90s, to extend specification Z with object-oriented is a hot point in the field and several extensions to Z is argued[1]. *COOZ* [2]is a object-oriented extension to Z that is based on the former with more advantage and desirable technology. Such specification can then be used as a basis for a provably corrected development of the program.

The development can be conducted in small step, thus allowing the unavoidable complexity of the final program to be introduced in manageable pieces. The process, called refinement, by which specifications are transformed into program has been extensively studied.

The extension from Dijkstra's language to the *refinement calculus* was made by R.J.R Back (1978)[3],then redeveloped independently by J.M.Morris [4], C.C.Morgan[5] .

Previous attempts to address in formal development fall into two categories. In the first, a specialized calculus is developed within Z or COOZ to allow algorithm refinement. The second approach involves a translation stage in which the Z specification is transformed into another notation , which is more amenable to refinement. Both of the approaches seem somewhat wasteful, on one hand, because the translation stage needs much effort but not contribute directly to development, and on the other because so much work has gone into the development of various flavors of refinement calculus which already exist.

So, for refinement of objected-oriented specification, we hope to develop a approach which integrate the Morgan's refinement calculus seamlessly. Since Z can not support implement of the system directly, that how to develop executable program from Z specification has been a valuable research field.

The advantage of COOZ is to specify a large scale software, but not support refinement by calculating and need to proof in refinement. But the proof is very hard for OO specification,

*This work supported by National Natural Science Foundation of China.

especially for the large an complex one. Thus its application is confined and it can not be taken as the whole method for software development. Including Refinement Calculus into COOZ complements its disadvantage during design and implement. The apartment between design and implement for construct and notation is removed as well. Then the software can be developed smoothly in the same frame. There is not corespondent object-oriented construct in existing Refinement Calculus. The combine of COOZ and Refinement Calculus can built a object-oriented frame, in which the specification in COOZ is refined stepwise to code by calculus.

In the paper ,two development model is argued ,which are based mainly on COOZ and Refinement Calculus respectively. The first model is debated primarily. The data refinement and operation refinement is analyzed by example; the two method of operation refinement for OO formal specification is discussed simply; the frame transition rule from COOZ to C++ is argued. For second model ,a class model and the class data refinement calculus is argued, which can be as a base of OO extension for Refinement Calculus.

2 COOZ and Refinement Calculus Notation

- COOZ (Complete Object-Oriented Z)

COOZ[2], an object-oriented extension to Z, which stands for "Complete Object-Oriented Z", as a way of combining formal methods with object- oriented techniques. A class schema ($Figure 1$)includes class name, class parameter, list of super class, object interface, local definition, state schema, initialization schema, method schema and class invariant.An example is given($Figure 2$), which models the students in a exercise class. Where just the registering method $Enroll - OK$ is given ,the others are omitted.

$Class Classname TypeParameters$
$SuperclassList$
$ObjectInterfaces$
$LocalDefinitions$
$AnonymousStateSchema$
$InitializeSchema$
$MethodSchemas$

$Real - TimeandHistoryConstraints$

Figure 1. Class Schema in COOZ

[Student]

$max : \mathbb{N}$

$max \geq 0$

$ClassStudent$
...
$y, n : \mathbb{P}\ Student$

$y \cap n = \{\}$
$\# y \cup n \leq max$

...

$Enroll - OK$
$\Delta ClassStudent$
$s? : Student$

$s? y \cup n$
$y' = y$
$n' = n \cup \{s?\}$

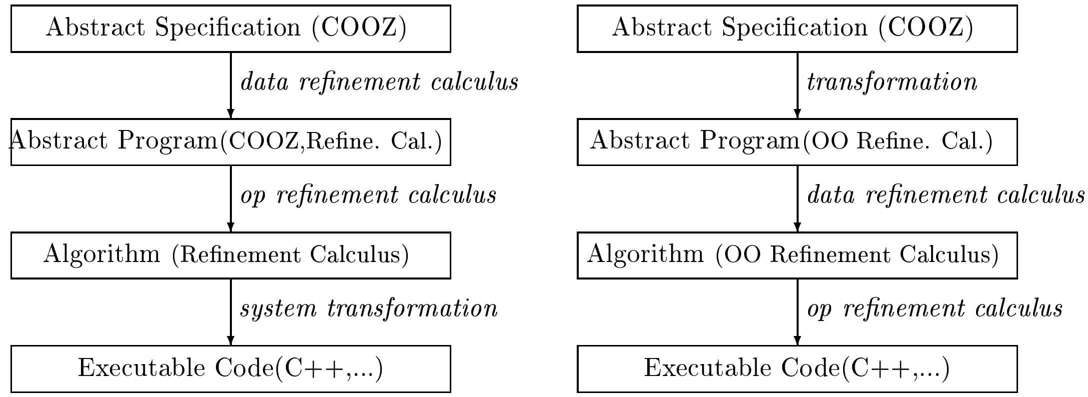
Figure 2. class schema ClassStudent

The calculus includes a specification statement in addition to the usual executable constructs. This integration of specification and exection in one languange is the key to a smooth development process.

- Specification statement

One program statement that is particularly important in the refinement calculus is the specification statement, which provides a convenient way of embedding abstract specification into programs. The frame of the statement (x) is a variables list that can be updated, all other variables must remain unchanged. The specification statement is defined as:

$wp[x : [pre, post], P] \equiv (pre \cap \forall x \cdot post \Rightarrow P)$



a. Develpment Mode 1 *b. Develpment Mode 2*

Figure 3. development model

- Data refinement

Data refinement can be viewed as a special case of program refinement, in which the abstract local variables of a program are replaced by the concrete set of variables, while the structure of the program remains largely unchanged. It is also useful to have a direct data refinement relation between components of the abstract and concrete program. More recently still, the relational approach has been generalized to allow the abstract and concrete state spaces to be related an arbitrary program [6]This program is called a simulation if it converts from the abstract state space the concrete, or a co-simulation if it converts from the concrete state space the abstract. In the following discussion we assume that the state space of program S is variable to be refined (a) with some globe variables (g) and the state space of program S' is the concrete variables (c) with g, where a, c and g are all disjoint.We choose any predicate transformer sim that takes predicate on the variables a, g to predicates on the variables c, g. For program P and P',P is data refined by P', written as $P \preceq P'$.

Data refinement definition 1: $P \preceq P'$ iff $sim;\ P \sqsubseteq P';\ sim$
Where the operator ';' is functional compositional (of predicate transformers). In fact the above sim is a co-simulation. If the sim is a simulation, then the data refinement id defined as:
Data refinement definition 2: $P \preceq P'$iff $P;\ sim* \sqsubseteq sim*;\ P'$

3 Software Development Model 1

The development starts from the software specification in COOZ. (see Figure 3 (a)) By refinement calculus of state schema and operation schema in the class, the abstract level is reduced stepwise until the enough concrete specification is archived. The model is based on COOZ frame. Firstly, to specify the system in COOZ, transforming the operation schema into specification statement, then after the state schema refined , the operation refined is calculated by data refinement calculus. According to the refinement law, the data refined specification statement is operation refined and the abstract program in COOZ frame is archived. At last , the abstract program is transformed into a programming language code, such as C++.

- Data Refinement

By data refinement ,abstract class A is refined by class C, noted as $A \sqsubseteq C$. But in Z and COOZ, the process need to be proofed[10]. As the refinement calculus included, the operation schema in A can be noted as specification statement, after state schema in a refined , the correct operation in C can be calculated and no more proof. Morgan [7] extended the program definition by taking both specification and code as program. In the program, unexecutable code which will be refined is noted by specification statement. We extend that by allowing include mathematic data type into the program, which is called abstract program. There is a directed corespondent relation between specification statement and operation schema in COOZ. In operation schema, the changeable variable is noted as $\triangle Id\_list$, for the predicates, the deference precondition is apart by a line and the precondition and postcondition is apart by a key word if. So the operation schema can be transformed into specification statement automatically. The specification statement is modified: for deferent preconditions, given the postcondition respectively. Then in a specification statement, there are several pairs [pre, post]. For example , The operation schema in *Figure 2* Enroll-OK can be transformed into a specification statement:

$$n : [s? \notin y \cup n, n = n0 \cup \{s?\}] \qquad (3.1)$$

where n0 notes the reference to pre-state variable n. The concrete representation for the sets given in the specification of *Figure 2* will consist of two arrays, one for students, and one for boolean values, and a counter to say how much of the arrays is in use. It is intended that the values in the second array will be true for those who have done the exercises ,and false for those who have not. The arrays is modeled by total functions whose domain is the index set (1..max). Then the class schema ClassStuden is refined as ClassStudent-1, the state schema in which is given as Figure 4:

$$\begin{array}{|l}
cl : 1 \mathinner{..} max \rightarrow Student \\
ex : 1 \mathinner{..} max \rightarrow Bool \\
num : 0 \mathinner{..} max \\
\hline
((1 \mathinner{..} num) \lhd cl) \in (N \rightarrowtail\!\!\!\!\!\rightarrow Student)
\end{array}$$

Figure 4.state schems of ClassStudent

class $classname : ancestor\_names$
{public: $constants$
$class\_attributes$
$methods\_signature$ }
$classname :: classname()\{ini\_val\}$
$methods\_implementation$
$classname ::\sim classname()\{\ \}$

Figure 5. class frame of COOZ in form of C++

The concrete state invariant says that there will be no duplicates in the first num elements of the array of students. The retrieve relation ,relating the concret and abstract states, is as follow:

$$R \equiv (y = \{i : 1..num \mid (exi) = true \cdot (cli)\}) \wedge \{i : 1..num \mid (exi) = false \cdot (cli)\}$$

Following the data refinement ,and with R and formula (3.1),the operation on state of Fig.5 can be calculated.

$$n : [s? \notin y \cup n, n = n0 \cup \{s?\}]$$
$$\preceq cl, ex, num : [\exists\, n \cdot R \wedge s? \notin y \cup n, \exists\, n \cdot R \wedge n = n0 \cup \{s?\}]$$
$$= cl, ex, num : [(num < max) \wedge (s? \notin \{1..num(cli\}), (num = num0 + 1) \wedge$$
$$(cl = cl0((num(s?))((ex = ex0((num(false))] \qquad (3.2)$$

¿From formula (3.2), we can get the refined operation schema in ClassStudent-1 easily.

- Operation Refinement

For OO formal specification, the operation has two contents: The first is based on Refinement Calculus, by which following refinement law, the specification is refined stepwise into executable code. The second is based on OO technology: following subtyping definition, refining the class by subtype inheritance. Both of above can refine the class, but the first is about refinement of the implement, and the second is usually confined in specification. Including refinement calculus into COOZ will combining the above methods, and will not distinguish the specification and the code in refining. For example, the data refined operation schema, i.e. the corespondent specification statement (3.2) is refined into executable code easily by refinement law of Morgan:

$$(3.2) \sqsubseteq num, cl[num + 1], ex[num + 1] = num + 1, s, false$$

- Program Frame Translation

The refined specification in COOZ is composed by numbers of Class schema. It has a well corespondent relation with the class structure of an OO programming language such as C++. Taking

C++ as a target language, following shows how to transform the refined abstract program into the programing language code. The following explains how to transform Class schema into the class in C++. The general form of Class schema, it can be refined as *Figure*4, the class in C++, where *ancestor_names*, *constants*, *class_attributes*, *methods_signature*, *initial_values*, *methods_implementation* in COOZ are transformed by Superclass List ,Local Definitions,Anonymous State Schema ,interface,Initialize Schema,Method Schemas respectively, the transform way is as :

ancestor-names are archived by putting the key word public ahead of every class name in Superclass List

constants are archived by putting the key word const and constant type name ahead of every constant name in the class.

class-attributes are archived by putting the state variable type ahead of every state variable in Anonymous State Schema

methods-signature is archived directly since massage name, massage parameter name, parameter type ,and result type given in Object Interface.

initial-values is archived by rewriting Initialize Schema as specification statement.

methods-implementation is archived by rewriting every operations schema as specification statement and putting message parameter in message interface ahead of them.

destruct function is left for extension .

Following the above, the Class schema can be translated into the class frame of C++ automatically by the tools.

4 Software Development Model 2

The developing model 2 is based on refinement calculus mainly. (see Figure 3(b)) The advantage of COOZ is to specify a large scale software, but not support refinement by calculating and need to proof in refinement. But the proof is very hard for OO specification, especially for the large a complex one. So firstly ,the specification in COOZ is transformed to abstract program in refinement calculus, then in the refinement calculus frame ,it is refined stepwise into executable code by date refinement and operation refinement. We extend the refinement calculus by objects, classes [8]. Following gives a class model in refinement calculus, defines the data refinement calculus ,which can be a basis for OO extension of refinement calculus in development model 2.

- class model

Class A is refined by class C if any property of objects of class A is also a property of objects of class C. This means that class C has all methods of class A with corresponding parameters. Firstly ,for list of variables a ,the initialization formula Ia , and list of method Ma we model a class A as $[Sa \mid Ia \cdot Ma]$, where Sa represents states of class A. For class invariant Ivn, Sa is defined as $[Vara \mid \forall\, a \cdot Inv]$ For predicate Q not containing free variable a ,it is defined as:

Definition 4.1:$[Sa \mid Ia \cdot Ma]Q == (((Sa(Ia(< Ma > Q)$

Where $< Ma > Q$ likes $wp[Ma, Q]$, which means west pre-condition for Q, but former is confined in the class, it must holds the class invariant. Given $Ma = a : [pre, post]$,class invariant Inv,

$< Ma > Q = (pre \wedge Inv) \wedge (\forall\, a \cdot post \Rightarrow Inv \wedge Q)$

Definition 4.2: Class refinement Definition Given A and C are classes of the same type, Class A is refined by class C, i.e. $A \sqsubseteq C$, we mean

$[Sa \mid Ia \cdot Ma] \sqsubseteq [Sc \mid Ic \cdot Mc]$

- Data refinement between class

As explaining above, the data refinement transforms an abstract block to a concrete one, here it means transformation from a abstract class to the concrete one. We assume that the concrete variable c do not appear in the abstract class A, and vice versa .The class refinement has following feature: The abstract variable declaration var a are replaced by concrete variable declaration var c. The abstract initialization Ia is replaced by a concrete initialization Ic . The abstract method Ma

,referring to variable a not c, is replaced by the concrete method Mc referring variable c, more over ,the algorithmic structure of Ma is reproduced in Mc.

Definition 4.3 : Data refinement Definition a class A is data refined by another class C ,abstract variable a and concrete variable c,

$sim;\ Ma \sqsubseteq Mc;\ sim$

We write this relation $Ma \preceq Mc$, The data refinement definition can be proofed that it guarantees the refinement between classes, i.e. it is sound.

Theorem 4.1 soundness of class data refinement: for suitable sim ,and $Ma \preceq Mc$ then

$[Sa \mid Ia \cdot Ma] \sqsubseteq [Sc \mid Ic \cdot Mc]$

The recent trend in formal program development is to calculus program from their specifications; that contrasts with proving that a given program satisfies some specification By Definition 3,we can calculate data refinement rather than prove them. The following is a theorem for data refinement by calculus in the class:

Theorem 4.2 $a : [pre, post] \preceq c : [sim(pre), sim(post)]$

The proof of above theorems is in [8].

5 Conclusion and Future Work

For formal methods, the main concern is to get a precise, concise and unambiguous formal specification. At present, the abstract description language is the key part of formal methods.In the formal specification of each class, the data is described with abstract data structure of COOZ, and the operations are defined with pre and post conditions.

In fact, just the frame and the theory base of the two development models are researched. Many concrete technologies need to be studied. software reuse is one main characteristic of object-oriented method. In the refinement of class, we should use the implemented class in the class library as far as possible.

The model presented here requires the supports of tools, which include prototyping tools, management tools for class library, and refinement tools for formal specification. In fact, it is difficult for any software methods to be of practical use without the support of integrated CASE. The further work is to provide necessary tools and appropriate environment. The work is being done now.

References

[1] S. Stepney, R. Barden and D. Cooper, editors. Object Orientation in Z. *Workshops in Computing* , London: Springer-Verlag. 1992.

[2] Yuan Xiaodong, Hu Deqiang, Xu Hao, Li Yong and Zheng Guoliang. COOZ: an complete object-oriented extension to Z. *ACM Software Engineering Notes*, 1998, Vol.23(4):78 81

[3] R.J.R.Back. On the Correctness of Refinement in Program Development. *PhD thesis*, Department of Computer Science, University of Helsinki, 1978. Report A-1978-4.

[4] J.M.Morris. A theoretical basis of stepwise refinement and programming calculus. *Science of Computer Programming.* 9(2)287-306,1987.

[5] C.C.Morgan. The specification statement. *ACM Transaction on Programming Language and systems* ,10(3):403-419,July 1988.

[6] P.H.B.Gardiner and C. Morgan. Data refinement of predicate transformers. Theoretical Computer Science,87:143-162,1991.

[7] C. C. Morgan. Programming from Specifications, second edn. Prentice Hall, 1994.

[8] Wang Yunfeng et.al. On refinement of formal object-oriented specification. In Xu Baowen et.al. editors,proceedings of The Fifth International Conference for Young Computer Scientists. Nanjing ,China, International Academic Publisher,1999 .