6V−1

A Mapping System from Object-Z to C++ *

M. Fukagawa, T. Hikita and H. Yamazaki †
Dept. of Computer Science, Meiji University ‡
Higashimita, Tama-ku, Kawasaki 214, Japan

1 Introduction

The formal specification language Z is now gaining popularity [3]. Object-Z is based on Z, augmenting the class concept as a structuring facility [2].

Here we discuss on a structural mapping system from Object-Z to C++. The idea of structural mapping was initially proposed by Bagherzadeh Rafsanjani and Colwill [1]. The structural mapping translates classes of an Object-Z specification into class interfaces of C++ such as data members and (headers of) member functions. Thus it is not intended as a code generation system, but rather as a tool for analyzing specification (including syntax and type checking) and for aiding to obtain code from specification.

In [1] the primary framework and basic rules of the mapping was described, which were obtained through case studies. We follow up the idea, and we describe the rules of mapping more clearly and to the details. We have implemented the major part of the mapping system, on which we report here. The major facets of the mapping are the treatment of classes and their inheritance, types, and predefined operator symbols.

2 An Example of Structural Mapping

The following is an specification of a birthday book, which is originally written in Z (chap. 1 of [3]) and slightly modified in order to conform to Object-Z syntax.

This is mapped to the C++ code segment, consisting of declarations of a class and its members.

$[NAME, DATE]$

*Object-Z から C++ へのマッピングシステム
†深川雅和 正田輝雄 山崎浩
‡明治大学理工学部情報科学科

BirthdayBook

$known : \mathbb{P}\, NAME$
$birthday : NAME \pfun DATE$

$known = \mathrm{dom}\, birthday$

INIT

$known \neq \emptyset$

AddBirthday

$\Delta(known, birthday)$
$name? : NAME$
$date? : DATE$

$name? \notin known$
$birthday' = birthday \cup \{name? \mapsto date?\}$

FindBirthday

$name? : NAME$
$date! : DATE$

$name? \in known$
$date! = birthday(name?)$

Remind

$today? : DATE$
$cards! : \mathbb{P}\, NAME$

$cards! = \{n : known \mid birthday(n) = today?\}$

```
#include "GlobalDefs.h"
class BirthdayBook{
protected:
//DeclPart
  Power< NAME > known;
  PFun< NAME, DATE > birthday;
public:
  BirthdayBook();
  BirthdayBook(BirthdayBook& the_BirthdayBook);
  virtual ~BirthdayBook();
  BirthdayBook&
    operator=(BirthdayBook& the_BirthdayBook);
  virtual void
    AddBirthday(NAME& name_q, DATE& date_q);
  virtual void
    FindBirthday(NAME& name_q, DATE& date_x);
  virtual void
    Remind(DATE& today_q, Power< NAME >& cards_x);
};
```

Classes Power and PFun (actually, templates) are prepared in C++ as a class library for realizing the

Object-Z operators related to power set and partial functions, respectively. In these classes related Z symbols and operators like *dom* and # are prepared.

All member functions (corresponding to Object-Z operations) are declared as virtual functions in C++. Also note the suffixes "_q" and "_x" of the parameters of the operations `AddBirthday` and the others. They correspond to the decorations "?" and "!" of variables in operation schemas of Object-Z.

3 Structural Mapping

3.1 Basic rules

We basically follow the rules in [1] of the mapping from Object-Z to C++, which are as follows.

1. Constants and state variables in a class are mapped into the protected part of a C++ class.
2. All inheritances in Object-Z are mapped to public inheritances of C++.
3. In the case of multiple inheritance, a base class is mapped to a virtual base class of C++.
4. Operations in Object-Z classes are mapped to virtual functions in C++. The return values of the functions are of type `void`, and their parameters are passed by reference.
5. For each class of C++, null constructors, copy constructors, assignment operators, destructors, and invariants for constants are always supplied.
6. Constructors for types of constants are always supplied.

3.2 Multiple inheritance

Object-Z allows multiple inheritance, so it should be realized in the mapping. In the mapping of multiple inheritance in Object-Z specification, one has to determine a common ancestor class as a base class, starting from the far ends of derived classes. This procedure also applies in the case of multiple inheritance among generic classes.

3.3 Types

Power set and function space do not have direct coounterparts in C++. Moreover, all of these are generic. But genericity itself can be realized in C++ by the template construct. The generic symbol of power set is realized as a predefined class in C++, as

```
template <class T> class Power.
```

Thus, our solution for realizing these type construction methods in C++ is simple; we map each of these type constructions directly to a (template) class of C++. Of course there are many ramifications of actually realizing these classes in C++, which differ to each other in simplicity and efficiency.

4 Implementation of a Mapping System

4.1 Format of Object-Z specification

As a source text of Object-Z specification we use Latex source. Object-Z uses Tex style files `fuzz.sty` (originally for Z) and `oz.sty`. Generally, these two use the same symbol names of Z.

We found that an environment for generic classes of Object-Z is not included in `oz.sty`, so that we have prepared another new style file `oz2.sty`, which the specification writer should add. That is, when preparing an Object-Z specification one has to write in Tex:

```
\documentstyle[fuzz,oz,oz2]
.....
```

4.2 Imlementation

We use for preprocessing and scanning the automatic lexical analyzer generator *lex*. And also, we use for syntax analysis and transformation the automatic syntax analyzer *yacc*. (We have actually used *flex* and *bison*, instead of lex and yacc.)

In the current implementation of the mapping system the sizes of source code are approximately: 300 lines of lex text, 1,100 lines of yacc text, and 2,100 lines of semantic functions in C++.

References

[1] G.-H. Bagherzadeh Rafsanjani and S. J. Colwill : From Object-Z to C++: A structural mapping, in "Z User Workshop, London 1992", Springer-Verlag, 1993, pp. 166–179.

[2] R. Duke, P. King, G. Rose and G. Smith : The Object-Z Specification Language: Version 1, TR 91-1, Dept. of Computing Science, Univ. of Queensland, 1991.

[3] J. M. Spivey : *The Z Notation: A Reference Manual*, Prentice Hall, 1989; 2nd ed., 1992.