Description of an inventory management problem in CafeOBJ and assessment

Atsushi Sakamoto (s0810027)

School of Information Science,
Japan Advanced Institute of Science and Technology

February 8, 2010

Keywords: Formal Method, CafeOBJ, Rapid prototyping, C, inventory management problem.

In the field of software development, many problems have occurred on the specification. As their measures, developers are interested in formal methods. The methods are a mathematical and logical technique to write rigorous specifications of systems and formally analyze such specifications, leading to high quality systems. As a case study of formal methods, FeliCa IC chip firmware can be mentioned. FeliCa is used in a very large area. For example, it is used as electronic cash, tickets and passes for public transportation, credit card, door key, and the identification card. This is beacese these areas cannot tolerate any small faults. To embed FeliCa in mobile phone without trouble, the developer wrote and tested the mobile FeliCa specification using a formal methods language, VDM++. Then, they created a FeliCa IC chip firmware based on the specificatin. In the investigation after creating this firmware, they reported that there were few troubles related to the specification. CafeOBJ is a formal methods language as VDM++, and it is an executable language based on algebra. But so far any serious programs have not been written based on CafeOBJ specifications. In this paper, we do rapid prototyping of a system using CafeOBJ. Then, we create a program of the system using the rapid prototyping as a specification, and we review and consider the obtained effect and impact. Rapid prototyping is a technique to solve problems in programming languages. Rapid prototyping makes it possible to clarify the request, the detailed, the design, and more in development. Therefore, it allows to discuss specifications of systems at earlier states of the developments and write programs that conforms to the clients' requirements.

This paper uses the inventory management problem as an example. The inventory management problem was raised in Information Processing Society, and is a problem with the program design techniques. The purpose of this problem is to clarify the differences and effects of new programming techniques by solving a common programming problem with such techniques, so this problem has been studied a lot as a program design techniques. But most of these studies are either to write a specification of problem or to do rapid prototyping of the problem in a programming language. We should not only design a system solving this problem and describe the design in a specification language, but also write a program based on the design (or the specification) in a real programming language. By

describing a specification of a system to solve the problem based on the requirements in CafeOBJ and writing a program based on the specification in a programming language in this study, CafeOBJ is assessed in terms of its effects to system development. We have selected the C programming language as our implementation language because the language is used in various fields.

Before we write the specification in CafeOBJ, we must have a deep understanding of the requirements of the inventory management problem. In our study, we used UML class diagrams and activity diagrams to clarify what data structures and procedures should be used for the inventory management problem.

we wrote a specification of a system to solve the problem in CafeOBJ based on the UML diagrams. As a result, when we were writing specifications in CafeOBJ, we noticed that the activity diagrams should have been changed. This change is reasonable for writing a programs of the system but we did notice it when we wrote the activity diagrams. This experiment demonstrates that writing specifications in CafeOBJ lets us make better understanding of the problem.

Then, we wrote a program to manage the inventory problem in C based on the CafeOBJ specification. We have come up with a set of transformation rules to systematically transform CafeOBJ specifications to C programs. We have proposed three transformation rules: the use of linear lists, handling function and modules, and data syntax. We used the transformation rules to write programs in C.

We noticed six effects and problems on how CafeOBJ affects program developments from the experiment. First, writing a specification of a system in CafeOBJ lets us understand the system better. When we made activity diagrams before writing specifications in CafeOBJ, we use two different functions to treat the situations when a receptionist receives a request sheet and a cargo sheet, respectively. However, when we were writing the CafeOBJ specification, we noticed that we can combine the two functions as one function. In this case, we could not notice it before writing specifications. Second, parameterized modules can be used in CafeOBJ and can be instantiated with other modules that correspond to more concrete data types. For example, generic lists can be described as a parameterized module and can be instantiated with module Nat, producing lists of natural numbers. On the other hand, C is not equipped with this functionality. Hence, each instantiated module should be written in C. Third, the module should not be fragmented. C++, Java and other programming languages are equipped with modules and classes. Thus, we do not fragment modules and should write some functions together. Fourth, the variable name should be made to the name meaning. Names given to variables used in the CafeOBJ specification written in this experiment do not have meaning. However, variables are given meaningful names, we can constrain the variable and argument names used in function and it also can be more precisely specified. Fifth, it is worth following some conventions to write programs. In the field of development, function and variable names often have to decide on the usage of capitalization rules. Most programming languages distinguish capital letters from small letters and then, for example, a variable whose name is "foo" is different from a variable whose name is "Foo". So we should write specifications in CafeOBJ by following some naming rules. Finally, when a program is written based on a specification, some subsidiary functions may have to be made. CafeOBJ specifications do not need to have such subsidiary functions, but C may need. For example, CafeOBJ

automatically shows results on the display. On the other hand C only calculates, but does not show the results on the console. We also need to consider other things like file operations. In this case, before we write the CafeOBJ specifications, we should consider those things during the problem definition.

In conclusion, it is worth writing a specification of a system to be developed in CafeOBJ, leading better understanding of the system, but it is necessary to come up with some rules to write specifications in CafeOBJ to write better specifications. One piece of our future work is to conduct more case studies to confirm our results obtained from the experiment and to obtain more skills and knowledge on specification in CafeOBJ.