The Design and Implementation of an Extensible Preprocessor for CafeOBJ

Yoshiyuki Asaba (110002)

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

February 14, 2003

Keywords: CafeOBJ, preprocessor, module, extension, policy.

1 Background and Purpose

The purpose of this work is to support the description of specifications in CafeOBJ by extending the CafeOBJ syntax. CafeOBJ is an algebraic specification language which has the powerful module system and the type system. Although CafeOBJ has high expressive power, the description of specifications in CafeOBJ may become complex. Such complexity may have a bad influence on readability and maintainability of specifications, which we cannot ignore. One of the solutions is to extend the CafeOBJ syntax itself. By extending it, we can expect to reduce the complexity. Unfortunately, the current CafeOBJ system does not have an extension mechanism of the CafeOBJ syntax. From the reason, CafeOBJ users cannot extend the CafeOBJ syntax easily. However, CafeOBJ users want such an extension mechanism. Moreover, in the description of extensions of the CafeOBJ syntax, it should be realized as independent modules. This modularization achieves the extension of the CafeOBJ syntax in incremental way. Here, we have to solve conflicts between extension syntaxes. In this work, we aim to realize an extension mechanism of the CafeOBJ syntax in easy and incremental way without conflicting between the extension syntaxes.

2 Our Approaches

In this work, the extension mechanism of the CafeOBJ syntax is realized as follows:

- the pseudo-extension of the CafeOBJ syntax by a preprocessor.

It is difficult to extend the current CafeOBJ system itself. Thus, we try to realize pseudo-extensions of the CafeOBJ syntax by a preprocessor. Concretely, CafeOBJ users define the extension of the CafeOBJ syntax by using the preprocessor. The preprocessor translates from the extension syntaxes into CafeOBJ code. Namely, the extension of the CafeOBJ syntax is equal to the extension of the preprocessor. Therefore, we need to make the architecture of the preprocessor extensible. To realize such a preprocessor, we adopt the following approaches:

- modularization of an extension syntax and its translation rule into CafeOBJ code (called *extension module*),
- the description of extension modules in CafeOBJ,
- usage policies of extension modules.

Firstly, an extension syntax and its translation rule are encapsulated as an independent module. However, because of the convenience of the description, extension modules can include one or more extension syntaxes and their translation rules. An advantage of this modularization is to improve reusability and maintainability of extension syntaxes. Secondly, by using CafeOBJ as the description language of extension modules, CafeOBJ users can realize the pseudo-extension of the CafeOBJ syntax in CafeOBJ. They can also use the CafeOBJ's powerful module system. In this work, the preprocessor consists of many extension modules, and we can add extension modules to the preprocessor incrementally. However, we need to solve above problem that means conflicts between extension syntaxes. It is conflicts of names of extension syntaxes in extension modules. In this work, the preprocessor solve conflicts by specifying translation rules as usage policies of extension modules. Then, we introduce the concept of a

context such as a block in a program. The preprocessor avoids conflicts between extension modules through the usage policies. Preprocessor's users describe a policy that is separated from extension modules and specifications including extension syntaxes. Consequently, by changing a policy, the preprocessor can change flexibly a meaning (translation rule) of extension syntaxes without changing extension modules and specifications including them.

In this work, we realize the preprocessor as follows. It has a basic module that defines the information of the CafeOBJ syntax. Preprocessor's users describe extension modules and extend the preprocessor by using them. It has also a system that controls extension modules. We call it *module control system*. This system solves conflicts between extension syntaxes according to the usage policies.

3 Espresso : An Implementation of the Preprocessor

We have designed and implemented the preprocessor based on our approaches. We call it *Espresso*. Firstly, we have implemented the basic module as the core system of Espresso in CafeOBJ. It has a CafeOBJ parser and library functions to support the development of extension modules. Then we have implemented the module control system of Espresso in Ruby that is a script language. The module control system needs to interpret usage policies of extension modules. Thus, we have designed and implemented a simple description language for the policies.

Secondly, we have described some examples by using Espresso and experimented the pseudo-extension of the CafeOBJ syntax actually. One example is based on a specification of a bank account system. We have written it in pure CafeOBJ at first. Then we have extracted complex parts of the description from it, and we have extended the CafeOBJ syntax to make the description simple. In this example, we have defined two extension modules. One is an extension module including syntaxes for declaring multiple predicates at a time; the other is an extension module including ones for declaring equations written state transition of CafeOBJ projection operations only. These modules have been added to Espresso, then we have revised the specification of the bank account system by using these syn-

taxes. As the result, we have confirmed that the code size of revised version of the specification have become about a half size of the code of previous version (written in pure CafeOBJ). Moreover, we have caused a conflict of extension syntaxes intentionally in this example. We have confirmed that Espresso have avoided the conflict appropriately.

4 Conclusion

In this work, we proposed a mechanism to realize a pseudo-extension of the CafeOBJ syntax easily & incrementally and developed an extensible preprocessor named Espresso. Espresso avoided conflicts between extension syntaxes according to usage policies of extension modules. Then, by introducing the concept of a context into Espresso, Espresso users can change flexibly translation rules of extension syntaxes. Moreover, we realized pseudo-extensions of the CafeOBJ syntax in CafeOBJ. We have also experimented extensions of the CafeOBJ syntax by using Espresso. These syntax extensions were realized by using the specification of a bank account system in pure CafeOBJ only at first. Then we have revised the specification of the bank account system by using these syntaxes. We showed the effectiveness of syntax extensions by using Espresso.

The following are our future work. We describe many examples by using Espresso. We experiment to incorporate a lot of extension modules into Espresso, and we consider limitations of our approaches. Moreover, we plan to improve the design and implementation of the current Espresso according to the results of the experiment.