An Efficient Implementation Technique for a Sequential Reflective Language

Yutaka Saeki

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

February 14, 1997

Keywords: reflection, scheme, reuse, modularization, compiler, meta-level architecture.

1 Introduction

The goal of this work is to establish an efficient implementation technique for a sequential reflective language Flect — a language that is capable of extending its semantics in modular way.

In this work, we try to satisfy the following requirements.

1. The user should be able to program/understand reflective features easily.
   In the 'traditional' methodology of defining reflective languages (*i.e.*, construction using meta-circular interpreters), it is generally difficult for users to understand all the mechanism of language construct. We need a good abstraction for reflective programming.

2. Meta-level modules should be composable.
   Practically, the programmer might want to apply several extensions to the language at the same time. To help this, there must be a facility to compose metalevel modules.

3. The implementation of the language should be efficient.
   Implementing the language by constructing a tower of meta-circular interpreters is not satisfactory.

The key idea of our work is to provide a semantics of the language using two kinds of objects. We represent each object as a set of modules that realize the states of the language, so behavior of each object (*i.e.*, semantics of the language) is given by its modules.

2 Our Approach

As an approach to satisfy the requirements we described above, we will give four facilities to Flect.

- We provide a way for user to describe a semantics of the language as meta-level modules, and user can use them to extend the language.
- We classified meta-level modules to two categories, so that we can compose them safety.
- In implementation, we could represent the construct of a semantics of Flect as first-class object, because we provide it as set of modules, so it is possible to pass the structure during an execution of Flect program, and could provide a facility for user to extend the language in a dynamic extent.
- We divide the structure that represents a semantics of the language from a compiled-code and made it possible to extend them by compiled meta-level modules, so we can apply reflective facilities to the compiled code.

2.1 Semantic Modules

In Flect, we expressed an extension of the language as the inclusion of new states of computation to the language. For example, we can regard an environment of interpreter as a state of system that is running on the interpreter.

Conceptually, we can say that each module is the evaluator of interpreter that is divided with some states into the independent and composable representation, namely meta-level module. In Flect, we can provide a flexible structure of language, because Flect can treat some parts of interpreter that specialized to some state, and user can compose them at run time to extend the language.

Users give a characteristic of each modules by a structure that constructed with states of computation and operations for the states. These operations are executed at atomic computation and compound computation.

- Atomic computation is that can complete its work in single step.
- Compound computation is that needs partial computation for its entirely computation.

Set of modules belongs to each object actualize the action of each object, so we can explain each object as interpreters. One is that specialized to states of controls (control-object) and the other is that specialized to rest states (reflective-object).

2.2 Reflection

Flect is a kind of reflective language. A reflective system can understand/compute its structure and behavior at run time. Giving the facility of reflection to OS or programming languages, it is possible to construct the advanced and complex systems, like mobile computing systems or distributed systems, with more systematic manner. Especially, much kind of reflective languages is implemented and the effectiveness of giving the reflective facility for programming languages is proved. Reflective languages allow the user to write a program that can change the language's semantics against the various run-time contexts.

2.3 Reflection in Flect

A system is said to be causually connected to its domain if the internal structures and the domain they represent are linked in such a way that if one of them changes, this leads to a corresponding effect upon the other.

To provide the reflective facility, there must be the causually-connection between a model of system and the system itself.

We prepare some interface for user to access each object with following operations.

- Put some modules that have already declared into an object
- Define modules and include it in the context of program
- Access to the states of objects

With these operations, we can say that there is a causually-connection. Because, each object represents a model of Flect, and set of modules that belongs to one of objects is the representation of the object, so we can change a behavior of objects by these operations, so user becomes to be able to change the language's semantics indirectly by the operations enumerated above.

2.4 Module Composition

We have provided a mechanism to compose each module safety, because we separate a semantic structure that represents control flow from the other structure, and provide a facility to extend Flect during a user-specialized dynamic extent. We can use some extension as we need, and can eliminate some of them if it seems to cause some conflict.

2.5 Structure of Flect System

We constructed the system of Flect by Flect-to-Scheme translator, run-time system and compiler of Scheme. We used MacGambit2.2 system as a compiler for the translated code and as a run-time environment.

It is difficult to implement a reflective system efficiently, because the semantic model of reflective language is usually given with a tower of meta-circular interpreter, so the

overhead of interpretation is caused. Therefore, we implemented Flect system as compiler. We arranged system to treat a semantics of Scheme immediately, (treatment of variables, dispatching with a type of expression, etc.) and separate a structure that represents the semantics of Flect into modules that can be compiled independent, so we can compile a part of source code by a compiler of Scheme, and improved its performance.

3 Conclusion

The characteristics of Flect system are that each module could be reused. For this, we provide following facilities in Flect and succeed to actualize the reusability.

- We have presented the model of reflective language, as the system that has modifiable semantics represented by meta-level modules.
- We gave a design of a language of the style and implemented this as a compiler.
- We could give user a tidy way to describe the modification, because we provide a unit of extension as a module.
- We have provided a framework for safety composition of meta-level modules.
- We have provided a facility to give an extension for a dynamic extent.