SWE 212 Analysis of Programming Languages

Week 5 Summary of 'Concepts and Experiments in Computational Reflection' by Pattie Maes

Samyak Jhaveri

This paper takes the reader on a deep dive into the concept of reflection in programming languages - starting from explaining what it is to proposing an experiment to implement in the new object oriented programming language. Taken for granted in modern day programming languages like Python, Java, C# etc., the concept of reflections in object oriented languages was a radical concept for its time, born out of necessity as later elaborated in the paper.

Computational reflection is a behavior of a reflective system that gives it the ability to perform computation about its own computation and modify itself during runtime. The source code is treated as mutable data to which modifications can be made as per the programmer's implementation of reflections in the code. Such programs are also called metaprograms. This saves the programmer a lot of time.

The author's opinion is that reflection has much value in real-world systems where most systems have some form of self-information and self-reflection in them about things like performance statistics, interfacing, self-optimization just to name a few. Although reflective computation won't solve problems by itself, it may be tremendously helpful in managing the internals of the system and its components, during runtime. Might I dare to draw an analogy, computational reflection may be thought to be analogous to introspective meditation - it does not solve our problems in the outside world for us, but may help organize our thoughts and control over our mind and body.

A reflective system / architecture is one which incorporates structures that enable reflections as part of its fundamental set of tools. This means, that such a system, implemented into the interpreter of a programming languages, must :

I. Be able to access data representing the system itself, and
II. Guarantee that the causal connection between the data and the aspects of the system

A reflective architecture helps make the reflective component of a programming language become more modular, which in turn makes these systems better manageable and more readable.

By the time this paper was written, procedure-based, logic-based and rule-based reflection architecture already existed. 3-LISP and BROWN were procedural languages that introduced the concept of a reflective function that specified computation about the currently ongoing computation. FOL and META-PROLOG were logic-based languages that used meta-theory to deduct other theories. TEIRESIAS and SOAR were rule-based languages that implemented meta-rules to specify computation about ongoing computation. all these languages operate on the concept of meta-circular interpreters which is essentially an infinite tower of circular interpreters with one circular interpreter interpreting the one below it. This makes it convenient to establish a causal connection. A system's self-representation is the meta-circula interpretation process itself and that this is a procedural representation, it is said that the architectures support procedural reflection.

The problem with such a system architecture is the duality of the self- representation in its responsibility to be a reliable basis of reasoning about the system while at the same time being an efficient implementation of the system. This is the motivation behind declarative reflection - a concept in which the self-representation of a system is not implementation of the system - effectively, enabling the architecture to provide reflection functionality with a handful of statements which are a collection of constraints that the status of the system has to fulfill.

Further, the author emphasizes the motivation behind implementing reflection in object oriented languages. Firstly, as the programming language community was still debating about what would be the most essential aspects of object oriented languages, it was unclear what purposes object oriented languages would serve. Reflective features allowed the object oriented languages to be open-ended - i.e. it was possible to create a local version of the language's interpreter for the task(s) at hand. Secondly, an object would be responsible for representing it's thing's domain information as well as meta information about itself. This would enable users of OOLs to provide documentation, history, explanations. Also, it would allow them to keep track of relations among the representations, encapsulate the value of the data-item and guard its status and behavior.

Finally, the authors discuss a new architecture for an OOL KRS, called 3-KRS, which has procedural reflection features baked into it. The languages has the following properties:

1) Every object in the languages is given a met-object. The structures inside an object exclusively represent information about the domain entity represented by the object. These structures hold all the reflective information about the object while the meta-object has implementation and interpretation information about the object.
2) The self-representation of the OO system in uniform i.e. every entity in the languages is an object.
3) The languages 3-KRS provides complete self-representation
4) The self-representation can be modified at runtime such that these modifications can affect the runtime computation.