

Concepts and Experiments in Computational Reflection

Pattie Maes

Computational Reflection:

Activity performed by a computational system when doing computations about its own computation.

A reflective system is a system which incorporates structures representing (aspects of) itself.

The self representation is causally connected to the aspects of the system it represents:

The system has always an accurate representation of itself

The status and computation of the system are always in compliance with this representation. This means that a reflective system can actually bring modifications to itself by virtue of its own computation.

Use of reflection:

Keep performance statistics

Debugging purposes

Stepping and tracing facilities

Interfacing

Reasoning about control

Self-optimisation

Self-modification

Self-activation

Reflective architecture

Provides tools for handling reflective computation explicitly

The interpreter of such a language has to give any system that is running access to data representing the system itself.

The interpreter also has to guarantee that the causal connection between these data and the aspects of the system they represent is fulfilled.

Existing reflective architectures

3-LISP and BROWN: Introduces the concept of a reflective function that specifies computation about the currently ongoing computation.

FOL and META-PROLOG: Uses Meta-theories that are about the deduction of other theories.

TEIRESIAS and SOAR: Incorporates Meta-rules that specify computation about ongoing computation.

These languages operates by the means of a meta-circular interpreter: a representation of the interpretation of the language which is also used to run the language.

This is a procedural reflection: a representation of the system in terms of the programs that implements the system.

Problem with procedural reflection: It has to be designed so it provides a good basis to reason about the system **and** it has to be effective and efficient, which are often contradicting requirements

Another type of architecture: Declarative reflection.

Self-representation merely consisting of statements about the system. The self-representation does not have to be a complete procedural representation.

The causal connection requirement is more difficult to realise here. The interpreter has to find ways to translate declarative representations about the system into the interpretation-process that is implementing the system.

The concepts of reflection fits most naturally in the spirit of object orientation. An important issue is abstraction, an object is free to realise its role in the overall system in whatever way it wants to. Thus it is natural to think than an object not only performs computation about its domain, but also about how it can realize this computation.

Motivations to provide reflective facilities in OOL:

An reflective architecture can be used to create local specialized interpreters.

Reflective information can be used to provide documentation, history and explanation facilities. It can keep track of relations among representations. It encapsulates value of the data-item, and guards its status and behavior.

OOLs have responded in ad hoc ways (SMALLTALK-72 and FLAVORS).

Two problems:

Only a fixed set of reflective facilities are supported. Mixed object-level and reflection-level, which may lead to obscurity.

SMALLTALK-80 introduces meta-classes, but a class still mixes information about domain and implementation.

Another step: Languages that represent everything as objects (class, instance, meta-class, method etc)

A reflective architecture in an OOL

Architecture for procedural reflection introduced in KRS. Result: 3-KRS

Every object is given a meta-object. The structures in an object exclusively represents information about the domain entity that is represented by the object. The structures in the meta-object contains all the reflective information that is available about the object.

Selfrepresentation of an OO system is uniform. Every entity is an object, so every aspect of the system can be reflected upon.

3-KRS provides a complete selfrepresentation.

Self-representation in a 3-KRS system is consistent.

Self-representation can be modified at run-time, and this has an impact on the run-time computation

Examples of use of 3-KRS

1: Tracing

One can temporarily associate a meta-object with a program such that during its evaluation various tracing or stepping utilities are performed.

2: A local deviating interpreter

Multiple-inheritance (not supported by 3-KRS) can be realized by a specialized meta-object.