OOPSLA 1993

Addendum to the Proceedings

Washington, DC—26 September–1 October, 1993

Workshop

Object-Oriented Reflection and Metalevel Architectures {Fourth Annual}

Report by:

Brian Foote
University of Illinois

Introduction

This year's workshop on object-oriented reflection and metalevel architectures generated an unprecedented level of interest, with thirty-one submissions, and in excess of fifty people in attendance. This summary presents but a flavor of the issues that were addressed at the workshop. Instructions for obtaining electronic copies of the attendance list, and the position papers themselves, appear at the end of this summary.

Session I-A: Theory

Moderator: Dan Friedman
Presenter: Anurag Mendhekar
Respondents: John Simmons, Constantin Laufer

Anurag Mendhekar spoke on representing reflection in the lambda calculus. His system implements an operator reifying the current continuation of an expression, represented by its lexical context. This operator corresponds to Scheme's *call/cc*. Another operator is provided to reinstall the abstracted context. Only the context is reified, and there is no explicit tower. A calculus for the system is developed and proved correct relative to its operational semantics.

John Simmons spoke on extending computational reflective systems by making the interpreter first-class and extensible. The interpreter is represented as two procedures, a *prelim* and a *dispatch*. The dispatch does the main work of interpretation by dispatching on the type of the expression and performing the proper computation for it. The prelim allows additional actions to be performed before or after the dispatch. This representation extends the power of traditional reflective systems. It permits the addition of new special forms, the modification (by shadowing) of basic interpreter actions on constants, variables, and applications, and the inclusion of actions preceding or following the main computation.

Constantin Laufer spoke on implementing reflection in a statically typed system. Using Haskell, he implements a system providing reflection on expressions. Expressions are represented as pairs, containing both uncompiled and (lazily) compiled versions of the expression. The compiled version can be used for computation, the uncompiled version for reification. The reify and reflect operators provide reification and reflection, with the reify operator recursively handling motion up the implicit tower.

Session I-B: Architecture

Moderator: Dan Friedman
Presenter: Jeff McAffer
Respondents: Carel Bekker, Patrick Steyaert

Jeff McAffer presented the *CodA MOP*, a framework for creating, controlling, and understanding concurrent computing environments. CodA is built atop Smalltalk-80. Its essential component is a simple, clear, and consistent interface protocol specification for basic system objects, such as schedulers, objects, and meta-objects. CodA uses the operational decomposition method of defining the metalevel. CodA embodies a complete system model which is independent of any implementation environment or language. Metadefinition, rather than meta-interpretation is emphasized.

Carel Bekker presented *ALBEDO*, a meta-object infrastructure for Smalltalk. ALBEDO adds support for a family of Maes-style meta-objects to Smalltalk-80. These meta-objects may be associated with any existing Smalltalk object. A library of meta-objects has been implemented using ALBEDO. Because meta-objects are distinct from the objects they are attached to, reusability, extensibility, and the understandability of the system are enhanced.

Patrick Steyeart took the view that rather than viewing extensibility as a consequence of a reflective architecture, we should instead consider extensibility as an a priori condition for the definition of reflection. He presented a two stage approach for introducing reflection into a hypothetical programming language, ASEL. The first stage is to define an open implementation for this language. The second stage is to make this open implementation explicit, or first-class.

Session II-A: Concurrency

Moderator: Pierre Cointe
Presenter: Hideaki Okamura
Respondent: Ken Wakita

Hideaki Okamura presented a paper discussing how shared resources can be modified in *AL-1/D*. The mechanism used employs the concept of grouping. Objects in a group share the resources that belong to the group. Examples of shared resources might be schedulers or garbage collectors. The paper also addressed how user-defined and default system resources cooperate.

Ken Wakita discussed how first-class messages, or *message continuations* provide object-oriented concurrent programming languages with extensibility in modeling and programming communication schemes such as asynchronous communication, multicasting, sophisticated synchronization constraints, inter-object synchronization, concurrency control, resource management, and the like. Despite this extensibility, the framework guarantees that no program can undermine the built-in communication primitives.

Session II-B: Massive Parallelism

Moderator: Pierre Cointe
Presenter: Hidehiko Masuhara
Respondent: Takashi Tomokiyo

Hidehiko Masuhara discussed the design of an object-oriented reflective language for massively parallel processors. One of the major problems in massively parallel programming is dynamic resource management, such as load balancing and scheduling. Simple management policies sometimes lead to lackluster performance under unexpected conditions, while complicated ones, which are effective under any situation, often impose considerable overhead. What is needed is an ability to provide abstractions for a variety of policies. Reflective languages provide this flexibility.

Masuhara et. al.'s language is based on ABCL/R2, which is extended so that the user can control the parallel programming primitives. In this architecture, the user can build his or her own resource management system—load-balancing, scheduling, etc.—at the metalevel, controlling the policies of primitives such as object creation, object migration, and scheduling.

Takashi Tomokiyo presented the metalevel architecture of a parallel object-based language called *OCore*. This language has simple, statically typed semantics to make compiled programs more safe and efficient. An object has a single thread, called the *normal thread,* for normal operations. Exceptions are handled by an *exception handling thread.* Communication between objects is based on asynchronous, one-way message passing.

Classes in OCore are merely templates, as in C++, and are not themselves first-class objects. The metalevel architecture of OCore handles three kinds of events: state transition events, user-defined events, and exceptions. Certain elements of an object's internal representation are visible at the metalevel via pseudo-variables.

Session III-A: C++

Moderator: Jacques Malenfant
Presenter: Shigeru Chiba
Respondents: Roger Burkhart, Roger Voss

There is growing evidence that users are chafing under the limitations that relatively closed languages such as C and C++ impose upon them. For instance, the current interest in runtime type information in the C++ community, and the widespread practice of building dynamic object systems *atop* C and C++, support the observation that real-world problems demand first-class, dynamic access to metalevel information and mechanisms.

Shigeru Chiba described the language *Open C++,* which extends C++ to allow programmers to define their own method call and variable access mechanisms. He then showed how they used these Open C++ mechanisms to support distributed programming.

The second paper, by Roger Voss, illustrates the dilemma that faces the frustrated user who would like to have a hook into some part of a language's implementation to be able to solve a particular problem. Mr. Voss is confronted with the problem of class evolution in C++ class libraries. His proposal for addressing this issue requires a modification to the usual C++ v-table-based dispatching scheme. In non-reflective languages, even a simple change can entail a protracted struggle with various standards committees.

Roger Burkhart discussed request functions for C. A request function is a new kind of C function that creates a context, which is a set of data concerning the function call, that is made accessible to the program. The primary point of this "conservative" extension is to have a means, readily implementable in C, for building more comprehensive reflective facilities.

Session III-B: Partial Evaluation

Moderator: Jacques Malenfant
Presenter: Erik Ruf
Respondent: Kenichi Asai

Erik Ruf feels that partial evaluation is far from the point where it could be used to efficiently implement reflective languages. In fact, existing partial evaluators are weak, bulky and inefficient. Moreover, there is a major flaw in the current work in partial evaluation for reflective languages. When reifying portions of the implementation information, decisions are made about the way this information will be represented. For instance, when reifying environments, it is standard practice to reify them as association lists. Currently, there is no way a partial evaluator will be able to get rid of this data structure and the resulting program (after partial evaluation) may get stuck with inefficient access to environments because the implementation decision survived the partial evaluation process.

Kenichi Asai reported on some experiments in partial evaluation using a reflective extension to Scheme that allowed reflective towers in which the interpreter at each level could be modified dynamically. Asai noted that they succeeded in applying partial evaluation, but in a static way (once and for all before the execution of the program).

During the discussion, Dan Friedman asked Mr. Ruf if he considers it impossible to build a partial evaluator that would be able to cope with the environment access problem discussed above. Friedman conjectured that with sufficient knowledge and complexity, an interpreter could deal explicitly with storage management and make it possible for the partial evaluator to generate code which would compare favorably with compiled code. Mr. Ruf conceded that he could imagine such an interpreter, but it would be quite a challenge to partial evaluate it using the current partial evaluation technology.

Session III-C: Garbage Collection

Moderator: Jacques Malenfant
Presenter: Barry Hayes

The final part of the session was devoted to a paper by Mr. Hayes on reflection and garbage collection. He claimed that the state of the art in garbage collectors is to provide "knobs" that allow users to feed in application level information that can be used profitably by the garbage collector. Unfortunately, these "knobs" are introduced in an ad hoc manner. Maybe reflection can help here, but it is also a challenge to open garbage collectors to users. While he noted that reflective garbage collectors could probably be implemented efficiently, the questions lie more on the side of the users. Will they be able to manipulate such a crucial part of the system profitably and correctly?

Session IV-A: Object Integration

Moderator: Yasuhiko Yokote
Presenter: Mireille Fornarino
Respondents: H. Justin Coven, Ming Peng

Existing object-oriented languages provide relatively weak facilities for representing dynamic relationships among objects. Users, however, may build their own using reflection.

Mireille Fornarino described a CLOS/MOP-based meta-architectural solution to the problem of keeping track of dependencies between objects. They introduce links to define dependencies, metalinks to define the structure of links, controllers to invoke control procedures and manage interactions between instance-of-links, and generic functions to invoke the current controller. Links are created incrementally. Thanks to metalinks, users can extend and customize the behavior of links.

H. Justin Coven discussed how interconnections, the links among pieces of information, and control issues might may be addressed using reflection.

Ming Peng described an object-oriented system that illustrated how mechanisms to dynamically represent interacting forces to address the "frame problem" might be constructed.

Session IV-B: Systems and Applications

Moderator: Yasuhiko Yokote
Presenter: Gary Lindstrom
Respondents: John Gilbert, Francois Rousseau

Ubiquitous object support in truly *open* systems will require that we refactor our computing systems so that support now provided on a per-process basis is elevated to the level of permanent system service. Gary Linstrom proposed that module management be elevated to this level, and that these objects be cast themselves as first-class objects. His factoring draws on insights from the programming language as well as system architecture communities. Although many systems are (assumed to be) implemented in the context of a single programming language, their work investigates how to support multiple programming languages.

John Gilbert discussed the importance of reflective facilities in open-ended scientific, realtime, and multimedia applications. Here, he claimed, reflection is essential to facilitate computational resource management, software configuration, fault tolerance, and reuse.

Francois Rousseau illustrated how meta-information is vital to the construction of modern tools for browsing programming languages such as ClassTalk.

Concluding Remarks: Dave Thomas

Reflection and meta-object programming provide an elegant and principled perspective on computation. The breadth of presentations at this workshop serve to illustrate the wide applicability to both applied and theoretical problems. These concepts, however, need to be clearly illustrated and explained to students, researchers and software engineers if this promising perspective of computation is going to have a major impact on mainstream computation. Everyone I know who has experienced computation through the exploration of even a simple metacircular interpreter has a different, deeper view of language semantics and computation.

We need to use common reference implementations such as TinyClos and Classtalk to illustrate applications and new ideas. Only if we communicate our ideas will others be able to understand the true benefits of Open Architectures. Failure to do so will leave reflection as a computational cult "going meta." One need only look at how few schools teach Scheme, ML or Smalltalk to realize the challenge which we have if meta-programming is to be part of every software engineer's toolkit/thinking process. The well-attended and highly successful MOP tutorial earlier this week is an excellent beginning.

Epilog: Brian Foote

Just as objects are good for building programs and for aiding in their evolution, so too are they good for building programming languages and computing systems. Unfortunately, the reflection community has developed a not entirely undeserved reputation for abstruseness, even as meta-architectural ideas are increasingly being incorporated into real object-oriented languages, databases, and systems, such as IBM's SOM.

One of the more interesting discussions of the day raised the question of whether terms such as "reflection" and "metalevel" have unnecessarily confused the broader perception of our open object-oriented architectural agenda. Gregor Kiczales remarked that these are good terms, but we shouldn't lead with them. In any case, as the computing industry prepares to build its next-generation systems out of objects, it is

imperative that we reach out and strive to more widely and clearly communicate our architectural vision.

Workshop Organizers:

Pierre Cointe	*E. des Mines de Nantes*
Brian Foote (chair)	*U. of Illinois at UC*
Dan Friedman	*Indiana University*
Jacques Malenfant	*U. of Montreal*
Dave Thomas	*OTI*
Yasuhiko Yokote	*Sony CSL*

Electronic Proceedings:

Anonymous ftp to:
p300.cpl.uiuc.edu
(128.174.72.1) pub/washington
camille.is.s.u-tokyo.ac.jp
(133.11.12.1) pub/oopsla93/reflection
or EMail to:
reflection-workshop@p300.cpl.uiuc.edu
foote@cs.uiuc.edu
Reflection Mailing List
Join: **reflection-request@p300.cpl.uiuc.edu**
Post: **reflection@p300.cpl.uiuc.edu**