Meta-Object Protocols for C++: The Iguana Approach

Brendan Gowing & Vinny Cahill

ECE 750 Topic 8, MPLSA

June 2, 2004

Claus W. Spitzer

Contents Overview

- 4 major sections:
  - Reasons for introducing fine-grained MOPs and explicit reification into Iguana.
  - Iguana syntax (Largest section).
  - Examples.
  - Performance analysis.
- Sort of related to OpenC++, but not quite.

Contents Detail

- Introduction.
- Previous implementations (MPC, OpenC++ versions I and II, MC++, CLOS).
- Reasons for doing it the way the did it.
  - Wanted adaptable operating system software.
  - Flexibility while maintaining performance.

Content Detail (cont'd)

- Iguana Syntax.
  - Meta-Level Classes and Objects.
  - Reification Categories.
  - MOP Declarations.
  - Protocol Selection.
  - Meta-Level Invocations.
  - Meta-Level Class Library.
- Examples.

Contents Detail (cont'd)

Related Work

- Implements ideas from previous systems
  - CLOS: Dynamic reflection.
  - OpenC++ et al: Compile-time reflection methods.

Contributions and Novelties

- Is a tool for defining MOPs, doesn't limit us to one MOP.
- Dynamic reflection while maintaining performance.
  - Selective Reification.
  - Multiple MOPs.

The Good

- Interesting concept.
- Good organization.
- Decent explanations.
- Feels like a programming manual.
- Syntax is very similar to C++, feels familiar.

The Bad?

- The paper doesn't really go into much detail about Iguana's internals. **I** don't mind, but I guess that someone wanting to learn more about *how* it works would be disappointed.

Q1

- What are the main distinguishing features of Iguana?

A1

- The ability to support multiple MOPs and MOP instances
- Selective reification.

Q2

- What mechanisms are used to maintain performance while providing dynamicity?
  - (Hint – I just said it)

A2

- Selective Reification
- Fine-grained MOPs.

Q3

- What is "meta-level locality of change"?

A3

- The ability of objects to alter their meta-level implementation without affecting other objects.

Q4

- What is the syntax for choosing a reification category?

A4

- The keyword "reify" followed by the single name of the category that must be reified, followed by an optional alternative class name, followed by an optional alternative name for the instance of the meta-class.

```
reify <Category> [: <ClassName>] [instanceName];
```

Q5

- Which member components of a MOP definition are required?

A5

- If you answered anything but “none”, then you need to go over the text again.

Q6

- How does Iguana implement instance protocol selection?

A6

- By replacing the declared class of the object with a subclass containing the necessary meta-level adjustments. For example

```
protocol Distributed;
Integer i ==> Distributed;
```

becomes

```
Integer__Metai i;
```

Q7

- What was the problem with reifying invocation using class or instance protocols for context switching?

A7

- All the invocations in an object would also trigger a context switch.

Q8

- What construct is used to gain access to meta-level objects?

A8

- The `meta` class. Example:

  To access the `bar` method of `foo` (which is an instance of the `Mfoo` class) one would use

```
 meta->foo->bar(...)
```

Q9

- What are the dangers of replacing one object's meta-level with the one of another object?

A9

- There is no checking for compatibility, so the reification categories of the new meta-level may be different.

Q11

- What happened to Q10?

Q12

- Why did the authors decide to use dynamic bindings for the meta-level (which use expensive register-memory moves) instead of a flat non-reference member object?

A12

- The performance gain was minimal.
- Advantages of dynamic binding are lost.
  - It is adaptable.
  - It is encapsulated and can easily be replaced as a whole.