Albedo
A meta-object infrastructure for Smalltalk

Carel Bekker* Roelf van den Heever
Department of Computer Science, University of Pretoria,
0083, Hillcrest, Republic of South Africa
Voice: (+27) (12) 420-2361
Fax: (+27) (12) 43-6454
Internet: cbekker, rvandenh@cs.up.ac.za

ABSTRACT

Reflection in the object-oriented paradigm became prominent after the publication of Pattie Maes' thesis on Computational Reflection. In her thesis Maes provides a *framework* for introducing computational reflection into the object-oriented paradigm. Maes observed that *"it must be said that the concept of reflection* ***fits most naturally in the spirit of object-oriented programming****"*. In an attempt to experiment with reflection, a meta-object infrastructure was implemented to support reflective computation. **ALBEDO**, a meta-object infrastructure based on Smalltalk is described in this paper. Smalltalk supports various ad hoc reflective facilities, but does not support uniform reflective facilities. **ALBEDO** was implemented without changing the Smalltalk virtual machine. Meta-objects may be explicitly associated with any existing object in the Smalltalk system. **ALBEDO** is measured against the *properties* proposed by Maes describing a full reflective architecture. A class library of meta-objects describing various reflective behaviours, including: debugging, logging, triggering of messages and atomic accessing meta-object classes was implemented. Finally issues regarding the **ALBEDO** implementation are discussed, conclusions are drawn and future research directions are noted.

Keywords: *object-oriented reflection, meta-objects, meta-object infrastructure*

1. Overview

ALBEDO, an infrastructure for meta-objects is implemented in Smalltalk and was used to perform various experiments regarding reflection in the object-oriented paradigm. Smalltalk's architecture consists of a virtual machine and a virtual image. The *virtual machine* implements accessing of hardware devices and other low level routines that must be written in machine language. The *virtual image* consists of all the objects in the Smalltalk system [Gold 89]. These include: the Smalltalk kernel objects, compiler objects and the users' objects. A prerequisite for implementing reflective facilities in an OOPL, is run-time access to the *message handling primitive* of the OOPL [Ferb 89]. In contrast to CLOS, Smalltalk's message handling primitive is part of the virtual machine and therefore inaccessible to users.

Various authors have proposed the use of the **doesNotUnderstand:** method to side-step the normal *message handling* of Smalltalk [McCu 87], [Brio 89]. Normally the **doesNotUnderstand:** message is sent to the receiver of an invalid message, i.e., if the receiver cannot handle a message. The **doesNotUnderstand:** method is implemented in Smalltalk's **Object** root class and takes one argument, **aMessage**. **aMessage** consists of the selector and the arguments of the invalid message. This message may be re-sent by performing

---

* Unit for Software Engineering associated with the Department of Computer Science

explicit message passing using variations of the message, **perform: with:**. This method and its variations, take as arguments a selector and its arguments, and is implemented in the **Object** class.

In order to gain run-time access to Smalltalk's message passing, the class **MinimalReferent** is used to bypass the primitive message passing. Instances of the **Referent**[1] class are used as a shell to encapsulate object computation and reflective computation. **MinimalReferent**, the superclass of class **Referent**, is used as a root class, instead of class **Object**. This is needed to override the normal **doesNotUnderstand:** method. Class **MinimalReferent** is created with the least number of methods recompiled from class **Object** and then **MinimalReferent**'s superclass chain is terminated to form a new hierarchy of classes. Class **MinimalReferent** does not have a superclass, i.e., its superclass is **nil**. The **doesNotUnderstand:** method, implemented in class **Referent**, will forward its argument, **aMessage,** to the object's reflective computation, i.e., its meta-object. Figure 2 shows a model of the meta-object infrastructure using OMT notation [Rumb 91a]. The classes to the right of the dotted line show the meta-object infrastructure and the classes to the left of the line show a part of the meta-object class library.

The objectives of **ALBEDO** used as basis for implementing meta-objects are discussed in the next section.

2. Objectives

ALBEDO was designed and implemented with the following *objectives*:

(i) No changes should be made to the Smalltalk virtual machine, because the *virtual machine* is inaccessible.
(ii) *Seamless integration* with current Smalltalk objects; and meta-objects may be attached to any object.
(iii) *Current objects* need no or minor changes to have a reflective behaviour.
(iv) It should be possible to create *libraries* of meta-objects.
(v) A user should be able to create new meta-object classes.
(vi) The *same notation* (language syntax) should be used for both object and reflective computation.
(vii) The reflective computation implemented via meta-objects must be implemented *independently* from the object computation.
(viii) An object should not have any knowledge of its associated meta-object. *Reflective behaviour* is activated when a meta-object is associated with an object.
(ix) It should be possible to associate a meta-object with *any object*, including meta-objects.

All the above *objectives are achieved* with **ALBEDO**, however, some issues need to be addressed. No changes were made (or could be made) to Smalltalk's virtual machine. Meta-objects may be associated with any current Smalltalk object. The only changes to current objects, are the addition of two methods, **reflect** and **reflect:** to class **Object.** Method **reflect** associates the default meta-object with an object, while method **reflect:** takes a specific meta-object as argument. The association of a meta-object with an object activates reflective behaviour for an object. The method **components** is added to class **Object** and is used by the **TriggeringMetaObject** class.

A *library of meta-objects* were implemented. The current library contains the following meta-object classes: DebuggingMetaObject, LoggingMetaObject, AccessingMetaObject, TypedOrderedCollection, LIFOOrdered-Collection, FIFOOrderedCollection and AsynchronousMetaObject. This library may be *enhanced and extended by a user*. Meta-objects are implemented with the *same notation* used to implement objects. Any of the *Smalltalk foundation classes* may be used by meta-object classes. A user may use the meta-object classes

[1] Alternative Names for these classes are **MinimalEncapsulator** and **Encapsulator**.

of the library as examples to implement new meta-object classes. The following are *steps to follow when implementing a new meta-object class*:

(i) The new meta-object class should be a subclass of class **MetaObject**.
(ii) The **handleMsg:** is typically overridden to implement the control flow of the new reflective behaviour.
(iii) Methods and variables may be added to the new meta-object class.
(iv) The new meta-object subclass may use the instance variable, referent to access the instance methods of the referent.

Meta-objects are implemented *independently* from objects and objects need not have any knowledge of their associated meta-objects. The *meta-object relationship* established by a meta-object with its referent, may be coupled by using interface coupling or protocol coupling. *Protocol coupling* is preferable (and enhances reusability) and is illustrated by meta-object classes: **DebuggingMetaObject**, **LoggingMetaObject** and **AccessingMetaObject**. These meta-objects use the message **performMessage:** ***aMessage***, to perform message passing to their referents explicitly. *Interface coupling* exists between the meta-object instances of **TypedOrderedCollection** and **LIFOOrderedCollection**; and their referents. These meta-objects need to use the interface of a specific class, in this case the class **OrderedCollection**. Examples are given to illustrate the association of meta-objects with meta-objects. A simple example of associating a meta-object with an object is described in the next section.

3. Using a meta-object

A simple example of associating a meta-object with an object is given in figure 1. In the example given in figure 1, a debugging meta-object instance of class **DebuggingMetaObject**, is associated with an object **p**, an instance of class **Person**. In statement (1) a new **DebuggingMetaObject** is created and in (3) it is associated with object **p**. In (3), the sending of the message **reflect:** to **p** returns an instance of class **Referent**, called **p'**, which consists of a meta-object **pMeta** and **p** (the object **p** of (2)), the object being reflected upon. Messages now sent to object **p'** will, as in (4), activate the meta-object **pMeta.** After **pMeta** is finished with the message (tracing the message sent on the **Transcript**), it will send it using explicit message passing to its referent, **p**. The *message* ***handleMsg:*** is used to perform *reflective message passing*, while the *message* ***performMessage:*** is used to perform *explicit message passing* to referents. In (5) the object **p'**, is destroyed and **p** becomes the object of statement (2) with no associated meta-object.

```
"Using Albedo, the meta-object infrastructure to associate a debugging meta-object with an object."

| p pMeta |

pMeta := DebuggingMetaObject new.          "(1)"
p := Person new.                           "(2)"
p reflect: pMeta.                          "(3)"

p name: 'John'.                            "(4)"

p noMeta.                                  "(5)"
```

Figure 1: Associating a meta-object with an object.

The classes used to implement the **ALBEDO** meta-object infrastructure are described in the next section.

4. Classes MinimalReferent, Referent and MetaObject

The three most important classes used to realize reflective computation in Smalltalk using meta-objects, are now described. These are classes **MinimalReferent**, **Referent** and **MetaObject**. The minor additions that are needed to class **Object** are also described. Figure 2 shows an OMT model of the meta-object infrastructure and other associated classes.

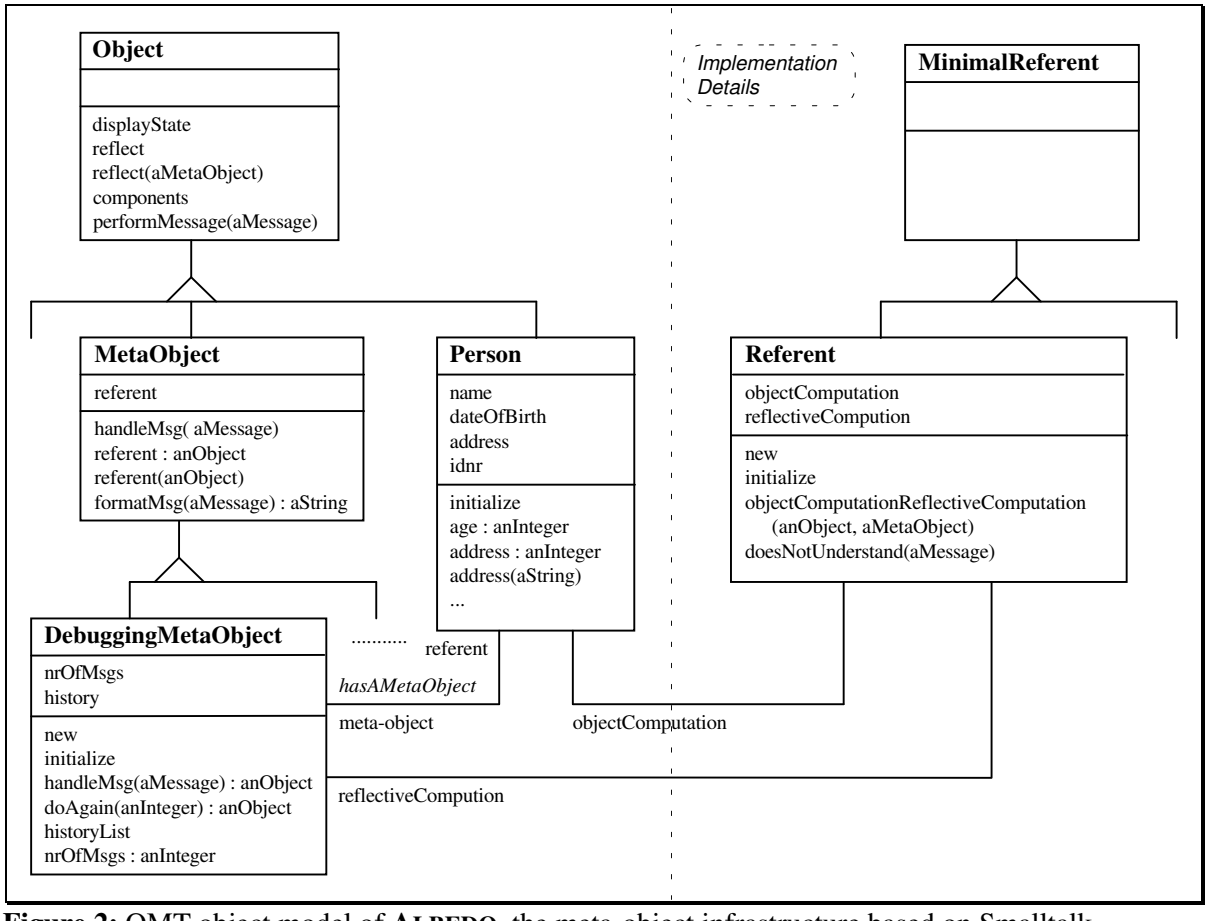

Figure 2: OMT object model of **ALBEDO**, the meta-object infrastructure based on Smalltalk.

4.1. MinimalReferent

This class is the minimum needed to create a new root class, used for message forwarding. Message forwarding is achieved, because messages sent to this class's instances (excluding the minimum methods recompiled from class **Object**), will not be understood and will trigger the **doesNotUnderstand:** message. This method is re-implemented in this class's subclasses and messages are forwarded to associated meta-objects. Subclasses of this class are used to encapsulate the object and reflective computation of a referent object. Messages sent to it will be forwarded to an object's meta-object. The class is initially created with the following definition, before the superclass chain is broken by sending message **initialize** to the class.

4.2. Referent

Referent is a subclass of **MinimalReferent**. Instances of this class consist of an object computation and a reflective computation. An object computation may be any Smalltalk object and defines the structure, i.e., methods and instance variables of a referent. Reflective computation may be instances of class **MetaObject** or any subclasses thereof. A meta-object defines computation about the computation of an object. The meta-objects of a referent may be changed dynamically. Objects may also dynamically terminate the use of reflective computation.

The methods **reflect** and **reflect:** added to class **Object** are used to encapsulate any object with a shell referent object and associate a meta-object with it. Subsequent messages are sent to the encapsulated object and thus forwarded to the referent's meta-object.

4.3. MetaObject

This class describes the default reflective behaviour for an object. It forms the root of a library of reflective behaviours. An instance of this class is used when no meta-object is specified when a meta-object is associated with an object.

4.4. Additions to class Object

The following methods are added to the class **Object**. The only methods necessary for meta-object realization, are **reflect:** and **components**. The other methods are used for convenience.

5. Comparison with Maes and Ferber

In this section the *meta-object infrastructure,* **ALBEDO**, *is measured* against the *properties* defined by Maes [Maes 87b] for a full reflective architecture. The *issues* that Ferber [Ferb 89] states should be addressed when defining a reflective architecture are also discussed.

In the following discussion we ascertain to what extent **ALBEDO** measures up to the *five properties* forwarded by Maes for a full reflective architecture:

(i) **Disciplined split between object-level and reflective level:** **ALBEDO** may be used to *associate a meta-object with any object* in the Smalltalk system. Every object in the system does not, however, have a meta-object, implying that an object may *exist without a meta-object*. Meta-objects are *physically separated* from their associated objects.

(ii) **The uniform self-representation of the object-oriented system:** This property is partially complied with, because in Smalltalk *every entity is an object*. The self-representation is, however, *not always consistent*, e.g., the use of metaclasses. Every aspect of Smalltalk may be reflected upon, using **ALBEDO** and Smalltalk's reflective facilities. This property also states that each object needs a meta-object to exist. This is not so when **ALBEDO** is used in Smalltalk. Meta-objects are only *created when needed*, thus lazy creation of meta-objects is not needed because meta-objects are created *explicitly*.

(iii) **Complete self-representation:** The meta-objects created using **ALBEDO** do not contain all the information needed by its referent. The meta-object does, however, have access to the self-representation of the object and the reflective facilities of Smalltalk.

(iv) **Consistent self-representation:** Most of Smalltalk is implemented by using its self-representation. There are, however, parts of Smalltalk implemented in its *virtual machine*. Thus Smalltalk's self-representation is not always consistent. Each object also does not have an associated meta-object. Therefore an *infinite tower of reflection* avoided with **ALBEDO**, because meta-objects are created *explicitly*.

(v) **Modified self-representation impacts run-time computation:** Smalltalk's self-representation is *modifiable at run-time*. Using meta-objects to modify the self-representation will impact the run-time computation of a system.

Ferber states that *three issues* should be addressed when implementing a reflective architecture. They are now discussed in regard to **ALBEDO**:

(i) **Which entities of the system should be reified?** - Every entity in Smalltalk is an object, i.e., is reified, and by using **ALBEDO**, *any object* in the system may have an associated meta-object. The meta-object classes are subclasses of class **MetaObject** and may be implemented by using the standard Smalltalk classes and constructs.

(ii) **How is the causal connection realized?** - If a meta-object is associated with an object, all the messages sent to the object will be *handled by the meta-level* (reflective level). *Metalevel message handling* is performed by using the message **handleMsg: aMessage**, while *basic message handling* is performed, by using the message **performMessage: aMessage**.

(iii) **What activates a system to use its metalevel?** Any object may be reflected upon by sending it either the message **reflect**, for default reflective behaviour, or **reflect: aMetaObject** for specific reflective behaviour. Thus, sending an object the messages **reflect**, or **reflect:** will cause *subsequent messages sent to the object to activate the metalevel.*

It may be said that **ALBEDO** supports a *fine grained or explicit* form of reflective programming. This means that reflection is only used in a system when it is needed and that objects may exist without associated meta-objects. A full reflective architecture, as described by Maes, results in the uniform and consistent reflection of a system. Thus meta-objects are created implicitly and objects cannot exist without a meta-object. Both forms of reflection are useful, and may be used to solve different problems. A full reflective architecture may be used in a dynamic environment where objects are changed and updated at run-time, e.g., *explanatory programming*. The explicit creation of meta-objects may be used in *production environments* where the metalevel is static and where it may be configured before a system is executed.

6. Issues

Some of the more *prominent issues* regarding **ALBEDO**, are:

(i) Messages sent to ***self*** and ***super*** are not intercepted by the meta-object infrastructure and do not activate reflective computation. This is already noted as a problem [Brio 89, Foot 89]. One solution proposed [Brio 89] for the Actalk system is that messages to **self** be replaced with messages to **aself**.

In **ALBEDO**, **aself** may be called, **rself**. **rself** may be a handle to an instance of class **Referent** that encapsulates the object and reflective computation. **rself** may be added as an instance variable of class **Object** and used instead of **self**. The drawback of this mechanism is that all references to **self** in Smalltalk's foundation classes must be changed to **rself**. An elegant solution is still lacking.

(ii) Instances of the class **MinimalReferent** and its subclasses need a *minimal number of methods* to exist in the Smalltalk system. These methods are: **doesNotUnderstand:**, **error:**, **~~**, **isNil**, **=**, **==**, **printString**, **printOn:**, **class**, **inspect**, **basicInspect**, **basicAt:**, **basicSize**, **instVarAt:**, **instVarAt:put:** and **primBecome:**. If these messages are sent to a referent with an associated meta-object, the reflective computation will not be activated, because these messages are understood by the encapsulating instance of class **Referent**. It is, however, essential that the message **doesNotUnderstand:** is understood by the instance of class **Referent**. This is not a major problem, but it should be kept in mind when using **ALBEDO**.

(iii) When Smalltalk's *debugger* is used to debug an object with an associated meta-object, the debugger will debug the encapsulating object. This object, in return, contains the object computation (the actual object to be debugged) and reflective computation of the object. This is also true for Smalltalk's *inspector*. This does not render these tools obsolete when using meta-objects, it only adds a level of indirection. Specialized debuggers and inspectors may be implemented for use with meta-objects.

7. Conclusions and Future Research

The following conclusions may be draw from experiments with *reflection using **ALBEDO.*** The main objective of these experiments were to ascertain to what extent reflection in object-oriented programming may enhance *reusability, extensibility* and *understandability*.

(i) **Reusability:** Reflection in object-oriented programming, as it is illustrated with the meta-object infrastructure, *enhances reusability*. Meta-object classes may be *implemented separately* from associated objects and may usually be associated with any object in the system. *Class libraries* of meta-objects describing various reflective behaviours may be implemented. The small class library implemented, using **ALBEDO**, illustrates this.

(ii) **Extensibility:** The default behaviour of an object and a complete object-oriented system may be *extended by using reflection*. This is illustrated using **ALBEDO**, with the **TypedOrderedCollection**, **FIFOOrderedCollection**, **AccessingMetaObject** classes. The standard mechanisms of an object-oriented system may also be extended. For example method lookup for inheritance may be changed, or *multiple inheritance* could be implemented in Smalltalk. Object-oriented systems may also be extended to support c*oncurrency* by using reflection, as it is illustrated with the **AccessingMetaObject** and **AsynchronousMetaObject** classes.

(iii) **Understandability:** The use of meta-objects to implement reflective computation instead of mixing reflective computation with object computation, *enhances the understandability* of a system. Meta-objects are implemented separately. Another factor that enhances understandability is the *explicit implementation and use* of meta-objects. Meta-objects are implemented by using the same Smalltalk syntax as normal objects and therefore providing *orthogonality of notation*, and enhancing the understandability of a system. The *declarative notations* that may be used for specifying the triggering of messages and the accessing of shared resources, assist in the understanding of a system.

The experiments performed, using meta-objects to implement reflection in Smalltalk, illustrates that reflection is a *powerful programming tool* and that it enhances various quality factors.

Future research includes:

(i) The enhancement and refinement of the **ALBEDO** to address the issues mentioned.

(ii) The association of multiple meta-objects to one object.

(ii) The interaction of objects and meta-objects using **ALBEDO** in a larger system.

Acknowledgements

We are indebted to the Unit for Software Engineering for supporting this research. All glory be to Jesus Christ without whom this work would not have been possible.

8. References

[Bekk 93] **Bekker, Carel:** *Relationships and Reflection in the Object-Oriented Paradigm, M.Sc. dissertation*, Department of Computer Science, University of Pretoria, May 1993.

[Brio 89] **Briot, Jean-Pierre:** "Actalk: a Testbed for Classifying and Designing Actor Languages in the Smalltalk-80 Environment" in *Selection of Publications on the Actalk Project*, Research Report LITP 92.68, Institut Blaise Pascal, Université Paris IV, September 1992.

[Gold 89] **Goldberg, Adele & Robson, David:** *Smalltalk-80. The Language*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[Foot 89] **Foote, Brian & Johnson, Ralph E.:** *"Reflective Facilities in Smalltalk-80"* in [Oops 89].

[Maes 87a] **Maes, Pattie:** *Computational Reflection*, Technical report 87.2, Artificial Intelligence Laboratory, Vrije Universiteit Brussel, Belgium, 1987.

[Maes 87b] **Maes, Pattie:** "Concepts and Experiments in Computational Reflection", in [Oops 87].

[McCu 87] **McCullough, Paul L.:** *"Transparent Forwarding: First Steps"*, in [Oops 87].

[Oops 87] **OOPSLA '87:** *Conference Proceedings, Object-Oriented Programming: Systems, Languages, and Applications*, ACM SIGPLAN Notices, Vol. 22, Nr 12, December 1987.

[Oops 89] **OOPSLA '89:** *Conference Proceedings, Object-Oriented Programming: Systems, Languages, and Applications*, ACM SIGPLAN Notices, Vol. 24, Nr 10, October 1989.