Metalevel Tool Support in AspectS

Robert Hirschfeld, Matthias Wagner

DoCoMo Euro-Labs
Landsberger Strasse 308-312
80687 Munich, Germany

(hirschfeld|wagner)@docomolab-euro.com

Abstract. AspectS is an approach to general-purpose aspect-oriented programming in the Squeak environment. It extends the Squeak metaobject protocol to accommodate the aspect modularity mechanism. Weaving and unweaving in AspectS happens dynamically at runtime, employing metaobject composition. This paper outlines how this metaobject protocol extension was utilized to enhance Squeak's programming environment allowing developers to become aware of system parts affected by aspects and to traverse structural relationships between aspects and affected system parts.

1 Introduction

Aspect-oriented programming (AOP) is a new software technology [1]. Based on the assumption that crosscutting is inherent to complex systems, it introduces new units of modularity to support separation of concerns more adequately. These units, called aspects, capture crosscutting structures explicitly and with that make it possible to organize systems in a more modular way.

So far, various approaches supporting AOP concepts have focused mainly on the creation of programming languages and aspect composition support, ranging from general-purpose to domain-specific. However, only a few have complemented theses efforts with sufficient tool support.

This paper outlines how the Squeak metaobject protocol (MOP) was extended to assist in the augmentation of Squeak's code browsers to become aware of aspects introduced by AspectS and their effects in the system.

2 AspectS

AspectS is an approach to general-purpose AOP in the Squeak/Smalltalk environment [2]. It extends the Squeak metaobject protocol to accommodate the aspect modularity mechanism. Weaving and unweaving in AspectS happens dynamically at runtime, employing metaobject composition [5].

AspectS is based on Method Wrappers by John Brant, a mechanism to add behavior to compiled methods in Squeak [3]. With Method Wrappers, one can introduce code that is executed before, after, or instead of an existing method. Instead of modifying Squeak's standard method lookup process, Method Wrappers change the objects the lookup mechanism returns. A method wrapper replaces an entry in the method dictionary of a class (that is, a compiled method or another method wrapper), adds behavior to the method invocation, and may eventually invoke the wrapped method itself.

3 Metalevel Extensions

In AspectS an aspect is responsible for the configuration and distribution of a set of method wrappers into the Squeak image. Each aspect references its method wrappers, and each method wrapper refers back to the aspect it originated from (Figure 1) [4].

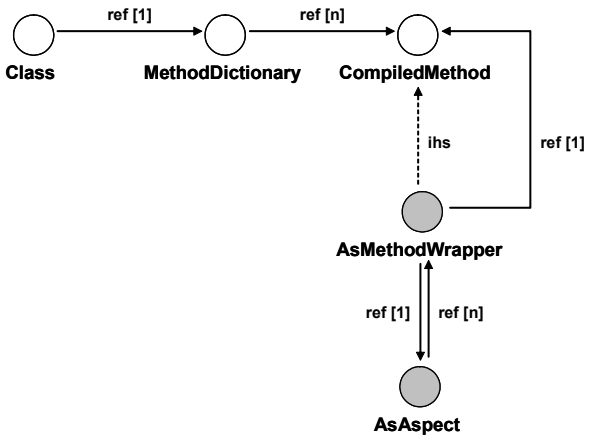

Figure 1: Squeak and AspectS Metaobjects

The relationships between classes, their method dictionaries and compiled methods, and in addition to that the relationships between method wrappers (which are specialized compiled methods) and their aspects make such metalevel structures straightforward to traverse.

AspectS' extensions to the regular Squeak MOP benefit from the uncomplicated metalevel traversal possibilities to detect classes and methods affected by aspects, to access individual aspect and wrapper instances as well as their classes.

In the following, MOP extensions essential to the analysis of classes and methods affected by aspects are presented.

If it is only of interest if a particular class is affected by one or more aspects, but not by which, Behavior>>isAffectedByAspects can be applied. isAffectedByAspects iterates over the class' method dictionary to see if at least one method there is a method wrapper. If so it concludes that there is an aspect installed in the system affecting this class, and with that some of its instances:

```
Behavior>>isAffectedByAspects
  ^ self methodDictionary anySatisfy: [:each | each isAsMethodWrapper]
```

If one is interested in all wrapper and aspect instances or their classes, Behavior>>wrappers, Behavior>>aspects, and Behavior>>aspectClasses can be used.

```
Behavior>>wrappers
  ^ self methodDictionary inject: Set new into: [:wrappers :each |
      wrappers addAll: each wrappersX; yourself]

Behavior>>aspects
  ^ (self wrappers collect: [:each | each aspect]) copyWithout: nil

Behavior>>aspectClasses
  ^ self aspects collect: [:each | each class]
```

CompiledMethod, AsMethodWrapper, and AsIntroductionWrapper all implement wrappersX which returns a set of all wrappers affecting one particular method:

```
CompiledMethod>>wrappersX
  ^ Set new

AsMethodWrapper>>wrappersX
  ^ self clientMethod wrappersX add: self; yourself

AsIntroductionWrapper>>wrappersX
  ^ Set with: self
```

The implementation of aspectsX to access all aspects affecting one particular method is similar:

```
CompiledMethod>>aspectsX
  ^ Set new

AsMethodWrapper>>aspectsX
  ^ self clientMethod aspectsX add: self aspect; copyWithout: nil

AsIntroductionWrapper>>aspectsX
  ^ (Set with: self aspect) copyWithout: nil
```

No object recursion is needed to get all aspect classes a particular method. CompiledMethod>>aspectClassesX makes use of the previously introduced aspectsX method:

```
CompiledMethod>>aspectClassesX
  ^ self aspectsX collect: [:each | each class]
```

The next section will show how these MOP extensions were used to adjust the Squeak development environment.

4 Tools With Metalevel Support

Most tools in software development environments rely on metalevel support. And so does AspectS by modifying Squeak's code browsers, the system and the hierarchy browsers in particular, to assist in aspect oriented programming and making the new structures introduced by the aspect modularity construct navigable.

In Squeak, code browsers essentially work over the metaobject structure involving the system organization, classes, method dictionaries and methods. If one selects a class in a browser, the browser will then reference the corresponding class object. From here, the enhanced browsers make use of the MOP extensions introduced in the previous section (Figure 2).

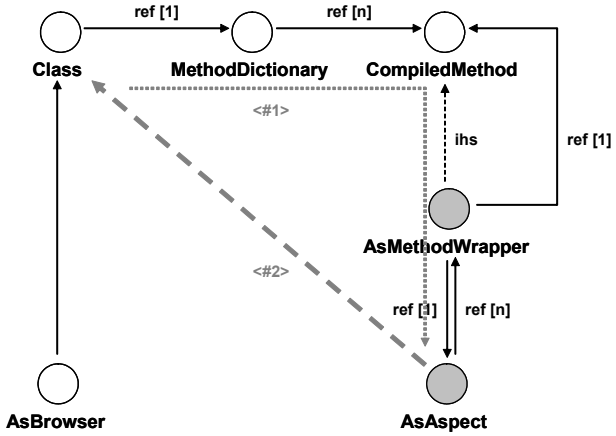

Figure 2: Aspect and Wrapper Metaobject Traversal

If a class is affected by one or more aspects (Behavior>>isAffectedByAspects), the class, its class category as well as its affected methods and their method categories are emphasized by applying a bold font setting. Menus of the class and method list offer to inspect effective aspect instances (Behavior>>aspects and CompiledMethod|AsMethodWrapper|AsIntroductionWrapper>>aspectsX) as well as to browse their classes (Behavior>>aspectClasses and CompiledMethod>>aspectClassesX) in addition to their usual entries (Figure 3).

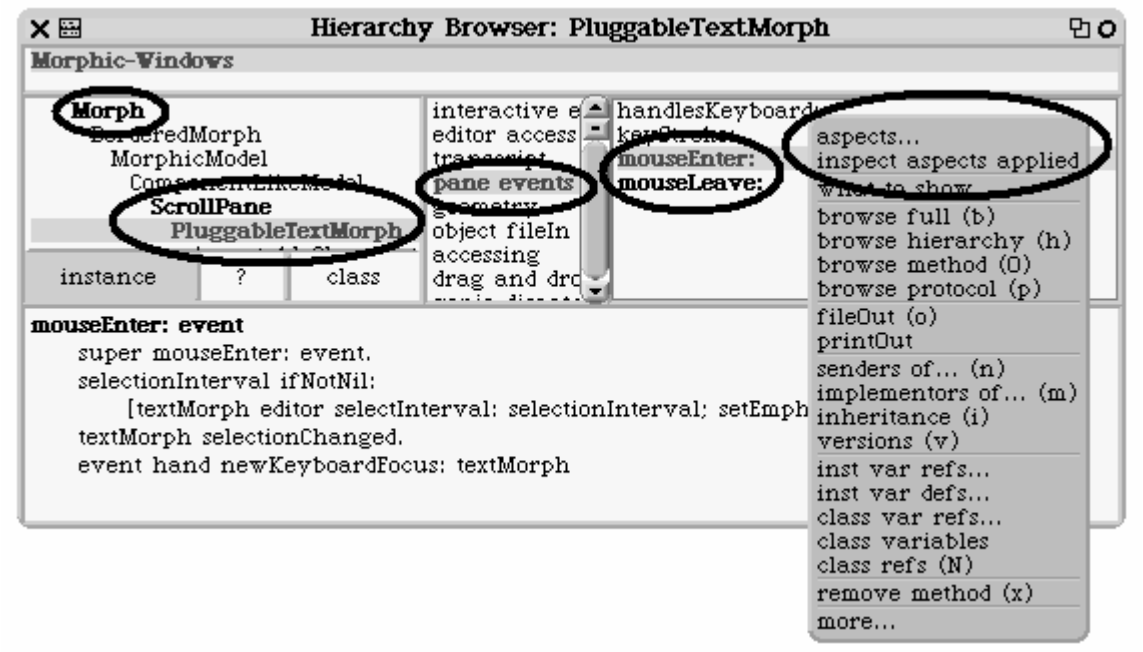

Figure 3: Browser Extensions Acting on System Parts Affected by Aspects

The extension explained above start their navigation from system parts already affected by aspects and allow to trace them back to the aspects and aspect classes they originate from.

It is also possible to start from an aspect and its advice code to examine its actual (if already installed) or potential (in not installed at the moment) impact on the system. For that, the pointcut of an aspect or an advice of an aspect is computed on-demand. The result is then displayed using message lists which allow for further navigation and traversal if possible.

5 Conclusion

The paper illustrated how AspectS extended Squeak's metaobject protocol to allow Squeak's code browsers to become aware of parts of the system affected by aspects and to traverse potential or actual relationships between aspects and affected classes and methods in the system. Changes done at the metalevel were small and concise which leads to the conclusion that the more powerful the metaobject protocol of the base system the easier it is to build tool support for newly introduced language constructs. Freundschaft!

References

1 *Aspect-Oriented Software Development* homepage (http://www.aosd.net/)
2 *AspectS* homepage (http://www.prakinf.tu-ilmenau.de/~hirsch/Projects/Squeak/AspectS/)
3 *Method Wrappers* homepage (http://st-www.cs.uiuc.edu/users/brant/Applications/MethodWrappers.html)
4 Hirschfeld, R.: "*Aspect-Oriented Programming with AspectS*." In: Proceedings of Net.ObjectDays (NODe), Erfurt, October 2002
5 *Squeak* homepage (http://www.squeak.org/)