{what} Support for domain-specific aspect languages

XAspects is the xajc command together with a library of plug-ins

```
> xajc *.java
```

The xajc command is a wrapper around the AspectJ compiler that introduces macro-like features to allow domain-specific aspect languages to be implemented as if they were an extension to the AspectJ language itself.

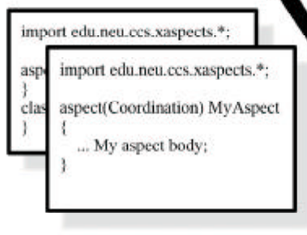
A plug-in implements this interface in order to be used by the xajc command

```
abstract class AspectPlugin {
  public AspectPlugin(
            CompilationEnvironment ce);
  abstract void recieveBody(AspectInfo ai,
              String aspectID, String body);
  abstract File[] generateExternalInterfaces();
  abstract File[] generateCode(File[] bytecodes);
  abstract void cleanUp();
}
```

The phases of compilation



(1) Programmer uses domain-specific aspects embedded in their regular AspectJ/Java code



(2) The xajc command separates the domain-specific aspects from the rest of the code and passes the aspects to their defining plug-ins

(3) Each plug-in exports the new interfaces (fields, methods, classes) it will define.

(4) The program is compiled with these stub interfaces.

(5)The entire program in byte-code form is passed back to each plug-in so they can do whole-program analysis on the byte-codes.

(6) Each plug-in provides implementations for the new interfaces it declared in step 3.

Finally, the program is compiled again.

{how} Use macros that generate AspectJ source code

```
xaspects.jar
import edu.neu.ccs.xaspects.*;
```

The XAspects library currently has plug-ins to support the domains of adaptive programming (as provided by DemeterJ), thread coordination (as provided by COOL), and RMI (as provided by RIDL).

Plug-ins are also being written to support " Personalities" (a dynamic alternative to Multiple Inheritance) and the Gang of Four Design Patterns.

```
import edu.neu.ccs.xaspects.*;

aspect(Coordination) MyAspect
{
   ... My aspect body;
}

aspect MyAspectJAspect {
}

class MyJavaClass {
}
```

The domain-specific aspect is used by " passing" a type-name to the aspect keyword. Here, the Coordination plug-in is being used.

The body defined in the source is passed to the Coordination plug-in at compile time and the plug-in then generates AspectJ code. The generated AspectJ code can introduce new classes, new fields, and new methods or change the semantics of existing code.

The Coordination plug-in is based off of the COOL language.

{why}

Achieve separation of concerns for several domains simultaneously

Direct support for programming techniques in the language eases the development effort. For example, DemeterJ shows how "new language constructs can make design patterns disappear" [Hannermann, Kiczales].

Many of the problems that can be solved in AspectJ can only be done so by an expert in the language. The plug-in mechanism allows for aspect programming patterns to be encapuslated as custom languages that can check that the pattern is being used properly.

In the past, several domain-specific languages were created to solve problems. While these domain-specific languages were too narrow to be included in the AspectJ programming language they were useful enough to warrant custom tools to support them. Now with XAspects, these languages can exist in a framework where they can be composed with other languages.

XAspects

Ankit Shah, Macneil Shonle
{ankit,mshonle}@ccs.neu.edu

Prof. Karl Lieberherr
Northeastern University, Boston MA

http://www.ccs.neu.edu/home/mshonle/xaspects