UniAspect: A Language-Independent Aspect-Oriented Programming Framework

Akira Ohashi Kazunori Sakamoto Tomoyuki Kamiya Reisha Humaira Satoshi Arai
Hironori Washizaki Yoshiaki Fukazawa

Waseda University

{akira-radiant@akane, kazuu@ruri, kamiya7140@akane, reisha@fuji, www.31o4-xy@fuji}.waseda.jp,
{washizaki, fukazawa}@waseda.jp

Abstract

Existing AOP tools, typified by AspectJ, are proposed as extensions of a single language. Therefore, most existing AOP tools cannot deal with cross-cutting concerns, which are scattered on many modules implemented in two or more languages. We propose a novel language-independent AOP framework named UniAspect. UniAspect translates programs written in various languages into a Unified Code Object, which is our common representation of source code. And it achieves the modularization of scattered cross-cutting concerns in multiple languages by weaving aspects through the Unified Code Object. In this paper, we introduce a case study of the implementation of logs in a web application that is implemented in Java and JavaScript. Its result shows that UniAspect achieves the modularization of these concerns by a single aspect.

Categories and Subject Descriptors D.3.2 [*Programming Languages*]: Language Classifications

General Terms Languages, Design

Keywords Aspect-oriented programming, Language independent, UniAspect, UNICOEN

1. Introduction

Programs run on various machines can be written in various programming languages according to the hardware characteristic or the purpose of the software, such as system software, middleware, mobile and web platform. Moreover large scale systems providing a large amount of functions could be realized by composing many program modules deployed on different machines.

In such a background, various aspect-oriented programming (AOP) tools corresponding to various different programming languages have been proposed. For example, AspectJ [4] is an extension of Java language, and AOJS[5] supports AOP for JavaScript. Developers can benefit from AOP in most major languages. However, since most existing AOP tools are implemented for a specific language, these tools cannot deal with cross-cutting concerns scattered on many modules implemented in multiple languages. Therefore, there is a possibility that a single concern is not always modularized to a single aspect. Moreover, the weaving mechanism and the description of the aspect are not unified among existing AOP tools. It leads to cost in terms of the time required for learning.

In this paper, we propose a language-independent AOP framework named UniAspect. UniAspect achieves language independence by translating programs written in various languages into a Unified Code Object (UCO), which is our common representation of source code, and weaving aspects through the UCO.

The Contributions of the paper are as follows:

- We show cross-cutting concerns that are written in multiple languages and scattered on many modules can be modularized into a single module using UniAspect.
- We explain the learning cost due to the introduction of AOP for a new language can be reduced because UniAspect supports AOP in multiple languages.

UniAspect is ongoing project as open-source software, and it can be downloaded from the UniAspect website [7].

In this paper we introduce UniAspect as follows. Section 2 illustrates the problems in existing AOP tools. Sections 3 and 4 give an overview of UniAspect and the UCO, respectively. We describe the join point model and the weaving process of UniAspect in Sections 5 and 6, respectively. Section 7 reports a case study. Section 8 refers to related work and Section 9 concludes the paper.

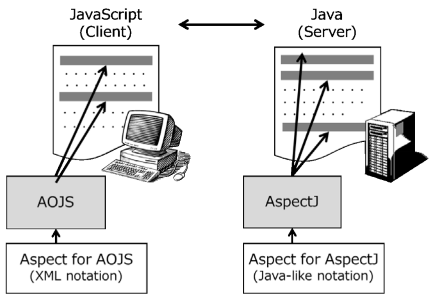

JavaScript (Client)
Java (Server)
AOJS
AspectJ
Aspect for AOJS (XML notation)
Aspect for AspectJ (Java-like notation)

Figure 1. Aspects for Web application

2. Background

In this section, we illustrate the problems in existing AOP tools. These are as follows.

- **P1: Most existing AOP tools cannot deal with cross-cutting concerns, which are scattered on many modules implemented in two or more languages.**

  For example, web applications are usually implemented by using multiple languages because the client side program and the server side program run on the different platform. To obtain logs of such a web application, it is necessary to use two AOP tools to support the implementation languages on both sides (Figure 1). In this example, AOJS is used for the client side and AspectJ is used for the server side. Therefore, the log code must be written as two aspects, and it is difficult to deal with these aspects as a single module. As a result, if the module has been modified about a concern, it is necessary to confirm its extent; how many languages are affected. Incorrect confirmation leads to missing some modifications.

- **P2: There is no consistency in the weaving mechanism and the description of the aspect among AOP tools.**

  Each existing AOP tool has its own mechanism for weaving. Therefore, developers need to pay a lot of attention to the consistency of weaving among multiple tools. In addition, the description of the aspect varies depending on the tool: an extended grammar of the specific language[4], a XML notation[5] and a function provided by AOP library within specific language's grammar[6]. Thus, the introduction of a new AOP tool results in a cost in terms of the time required for learning.

3. Overview of UniAspect

In this section, we give an overview of UniAspect. UniAspect translates programs written in various languages into a UCO and weaves aspects through the UCO. The details of the UCO will be described, in section 4.

Figure 2 shows an overview of UniAspect. The entire process of the system is as follows, where numbers correspond to those in the figure.

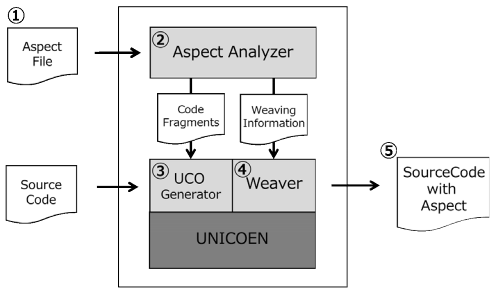


Figure 2. Overview of UniAspect

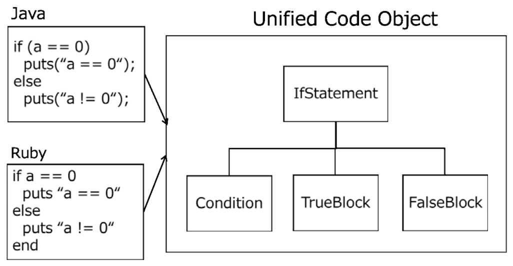


Figure 3. Example of unified code object

1. The developer inputs source code written in supported language and aspect.
2. Weaving information and code fragments from the aspect are extracted in the aspect analyzer.
3. The UCOs of the input source code and code fragments are generated in the UCO generator.
4. Code fragments are woven into the input source code in the UCO in the weaver.
5. The source code with aspect is regenerated from the UCO and outputted.

UniAspect performs weaving by transformation of the source code on the UCO. Therefore, developers can compile or execute output source code using any system.

4. Unified Code Object

UniAspect is based on UNICOEN [8], a source code processing framework for multiple languages. UNICOEN is a framework for developing source code analysis or transformation tools, and our research team is developing it as open-source software. UNICOEN supports the development of language-independent source code processing tools by supplying a common representation of source code for different programming languages. The common representation that UNICOEN supplies is called the unified code model (UCM), and objects generated from the source code according to the UCM are called UCOs.

For example, an If statement is composed of a conditional expression, a true block and a false block in all programming languages. Therefore, UNICOEN translates an If statement

```
1  aspect Sample {
2    Foo : @Java{
3      public void __debug() {
4        System.out.println(...);
5      }
6    }end
7
8    pointcut move() :
9      execution(double Foo.hoge());
10   pointcut init() :
11     execution(* *.init*());
12
13   before : init() {
14     @Java { __debug(); }end
15     @JavaScript { console.log(...); }end
16   }
17 }
```

Figure 4. Example of aspect in UniAspect

as shown in Figure 3. UNICOEN also translates elements that appear in particular languages such as Type and Class into a common object. As a result, a UCM consists of the union of a set of elements in programming languages. The latest version of UNICOEN deals with seven programming languages, C, Java, C#, JavaScript, Ruby, Python and Visual Basic, and a UCM consists from these seven languages. Developers are also able to implement new languages into UNICOEN by writing mapping rules for the UCO in UCM.

5. Join point Model

In this section, we present an example of an aspect in UniAspect and explain its features.

5.1 Sample Program

Figure 4 shows an example of an aspect in UniAspect. A grammar of UniAspect is designed to imitate that of AspectJ so that a developer used to an existing system can understand it easily. There is a problem that code fragments written in advice and interType declarations depend on the programming language of the weaving target. Since a UCO may hold language specific features such as foreach statement, this may become a source of error, for example, weaving advice written in C# into a source code written in Java. Therefore, advice and InterType declarations need to be written depending on the particular language. As a result, an aspect in UniAspect has both a language-independent portion and a language-dependent portion. A process that depends on a particular language is called a language-dependent block and is described as shown in lines 14 and 15 of Figure 4.

5.2 Pointcut

To weave an aspect into programs written in supported language, UniAspect is designed to deal with only common elements as join point in a UCO among multiple languages. Table 1 shows the relationship between programming languages and the main elements in a UCO. "Yes" means that the element appears in the corresponding programming language, and "No" means that the element does not appear.

Table 2. List of specifiable join points

Join point Type	Location in Source Code
call	When a function is called
execution	When a function is executed
get	When a variable is referenced
set	When a variable is assigned

Table 1 shows that function declarations, function calls and variables appear in all programming languages adopted by UNICOEN. Moreover, variable uses are categorized into two types, assignment and reference variables. Table 2 shows specifiable join points in UniAspect.

A pointcut is declared using the "pointcut" identifier shown in lines 8 and 10 of Figure 4, and a developer writes the pointcut name and the conditions for selecting join points. In the conditions, developers can specify the type of join point, the return type, the class name and the function/-variable name. Because UniAspect weaves aspects through a UCO, a pointcut specifies the join point from the UCO. Therefore, the description of a pointcut is independent of the programming language. For example, lines 10-11 of Figure 4 select function declarations whose name starts with "init" as a weaving point regardless of the language.

5.3 Advice

Developers can specify "before" and "after" as advice. Advice is declared to use a "before" or "after" identifier, as shown in line 13 of Figure 4, and the developer writes the declared pointcut and code fragments. Developers need to write code fragments in language-dependent blocks in the appropriate programming language.

For example, lines 14 and 15 of Figure 4 define language-dependent blocks for Java and JavaScript. If a join point selected from the corresponding pointcut is based on a Java program, the code shown in line 14 of Figure 4 will be woven, and if it is based on a JavaScript program, the code shown in line 15 of Figure 4 will be woven.

5.4 InterType Declaration

Lines 2-6 of Figure 4 define an interType declaration by describing a language-dependent block. The developer specifies the class name by describing an identifier before the language-dependent block in the case of a language with classes such as Java, or specifies the file name in the case of a language that does not have declarative classes such as JavaScript. Then, the corresponding function or variable will be woven into the specified class or file.

6. Weaving Process

The weaver identifies join points in a UCO according to aspect description. Since a UCO uses a tree structure to

Table 1. Relationship between languages and main elements in the unified code object

	C	Java	C#	JavaScript	Ruby	Python	VisualBasic
Function Declaration / Call	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Variable	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Exception	No	Yes	Yes	Yes	Yes	Yes	Yes
Class	No	Yes	Yes	No	Yes	Yes	Yes
Typed Variable Declaration	Yes	Yes	Yes	No	No	No	Yes

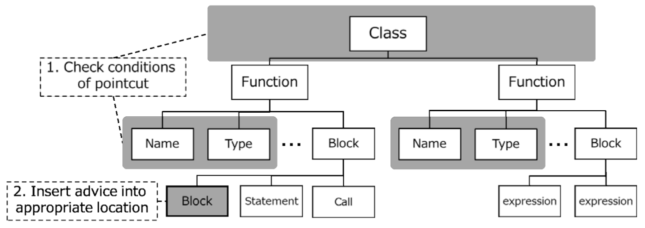


Figure 5. Process of execution weaving

represent the source code, as shown in Figure 5, it is easy to identify specific elements in the input source code by checking the parent and child object (Figure 5-1). The object which satisfies all given conditions will proceed to the next step. Then, the weaver inserts the advice into the join point specified in the previous step. Insertion of the advice is also performed on the UCO. The advice is translated into a UCO as a block, and the basic operation is to insert the advice into a block so that it becomes a previous sibling in the case of execution before (Figure 5-2).

On the other hand, there is a possibility that a conflict arises between the names of the variable/function of an aspect and those of target program. This conflict arises because the advice and interType declarations are woven directory into specified join points. As a result, the scope of an aspect depends on the join point used, and developers need to avoid these conflicts between aspects and source code.

7. Case Study

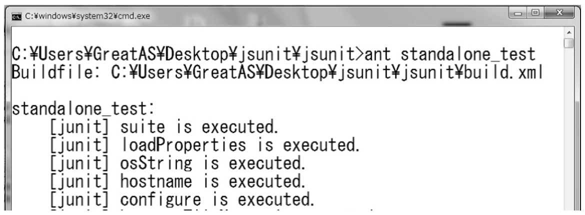
In this section, we give an example of implementing logs using UniAspect for JSUnit, which is a client-server-type test framework. The client side of JSUnit is implemented in JavaScript and the server side is implemented in Java. JSUnit is a system that performs tests on web browsers and displays the results on the server side. In this case study, we implement log code for all functions of JSUnit. We obtain the source code of JSUnit from its Github project page. Then, we weave the aspect shown in Figure 6 into the obtained source code. Finally, we compile the woven source code using the supplied Ant build file and execute it.

Figures 7 and 8 show the result of execution. It shows logs of the functions executed on both the server side and client side. We can confirm that the log code has been woven using UniAspect.

```
aspect Logger {
  pointcut allMethods() :
    execution(* *.*());

  before : allMethods() {
    @Java { System.out.println(
      JOINPOINT_NAME+" is executed."); }end
    @JavaScript { console.log(
      JOINPOINT_NAME+" is executed."); }end
  }
}
```

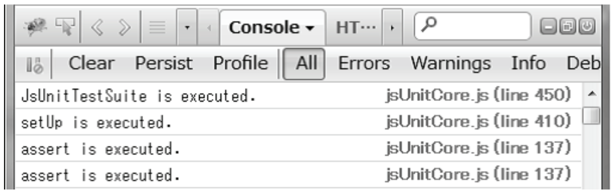
Figure 6. Aspect using in case study



```
C:¥Users¥GreatAS¥Desktop¥jsunit¥jsunit>ant standalone_test
Buildfile: C:¥Users¥GreatAS¥Desktop¥jsunit¥jsunit¥build.xml

standalone_test:
    [junit] suite is executed.
    [junit] loadProperties is executed.
    [junit] osString is executed.
    [junit] hostname is executed.
    [junit] configure is executed.
```

Figure 7. Result of server side logs



```
JsUnitTestSuite is executed.          jsUnitCore.js (line 450)
setUp is executed.                    jsUnitCore.js (line 410)
assert is executed.                   jsUnitCore.js (line 137)
assert is executed.                   jsUnitCore.js (line 137)
```

Figure 8. Result of client side logs

Table 3 shows the number of methods, files and lines in JSUnit. Using UniAspect, 786 log codes have been woven into server-side programs and 43 log codes have been woven into client-side programs. In addition, functions are scattered in 137 files written in Java and JavaScript. Table 4 shows the number of chunks of code for logs, and the number of modified files. In the case of using UniAspect, developers can summarize cross-cutting concerns scattered on multiple files written in multiple languages as a single aspect.

The considerations obtained from the results are as follows. Cross-cutting concerns scattered on many modules implemented in two or more languages can be summarized as a single aspect, and this shows that UniAspect solve the problem mentioned as P1. On the other hand, in the case of using

Table 3. Numbers of woven aspect

	Server side	Client side
Language	Java	JavaScript
Lines of Code	7106	2856
Number of Methods	786	43
Number of Files	128	9

Table 4. Comparison of scattered log codes

	Using UniAspect	Using Existing Tools	Without Using Aspect
Number of chunks of code	1	2	829
Number of modified files	1	2	137

existing AOP tools, developers must implement two aspects, for AspectJ and AOJS for example. Therefore, the amount of scattered log code is two as shown in Table 4, and developers have to manage these aspects separately.

The description of the aspect in UniAspect is unified in the form shown in Figure 4 or 6. Therefore, there is no cost of introducing an AOP tool to a new language when using UniAspect. On the other hand, for example AspectJ has a similar syntax to Java, but AOJS has a syntax based on XML. Therefore, the use of multiple AOP tools has a large cost including a learning cost. This shows that UniAspect solve the problem mentioned as P2.

8. Related Work

Several AOP tools for .NET Framework have been proposed to support multiple languages. SourceWeave.NET[2] is a weaver based on source code transformation through the CodeDOM; .NET standard for representing source code as abstract syntax tree. Weave.NET[3] weaves the aspect into the CIL of the .NET Framework. Although these tools are similar to UniAspect in terms of using a common representation for multiple languages, the languages that they support depend on the .NET Framework. On the other hand, UniAspect does not depend on a particular platform, and it can support more languages for weaving aspects.

Compose*[1] is a compilation and execution framework for the Composition Filters model, which supports multiple languages and platform: .NET platform, Java language and platform and C language. Although Compose* is similar to UniAspect in terms of creating a common structural language model to specify where aspect behavior should be applied, a weaver needs to be implemented for every specific target language. On the other hand, the weaver of UniAspect is also based on unified code object, which leads unified implementation.

9. Conclusion And Future Work

In this paper, we proposed a language-independent AOP framework, UniAspect. UniAspect achieves language independence by translating programs written in various languages into a UCO then weaving aspects through the UCO. As a case study, we gave an example of implementing logs for a system written in Java and JavaScript. It shows developers can summarize cross-cutting concerns scattered on modules in multiple languages as a single aspect.

Finally, we describe some future works.

Advice in UniAspect is dependent on the language of the join point. To achieve a language-independent description of advice, we should provide common specification for describing advice. Moreover, UniAspect does not support several pointcut adopted in existing tools; control flow pointcut, composition of pointcuts and so forth. We plan to support these pointcuts to make an aspect more expressive.

Acknowledgments

This research has been originally conducted as ”MITOH youth 2010” project in Information-technology Promotion Agency, Japan. After the end of the project, the research is partially supported by the Japan Prize Foundation.

References

[1] A. de Roo, et al. Compose*: a Language- and Platform-Independent Aspect Compiler for Composition Filters. In *First International Workshop on Advanced Software Development Tools and Techniques (WASDeTT '08)*, 2008.

[2] Andrew Jackson and Siobhan Clarke. SourceWeave.NET: Cross-Language Aspect-Oriented Programming. In *Proceedings of the Generative Programming and Component Engineering(GPCE '04)*, 2004, pages 369–393.

[3] Donal Lafferty and Vinny Cahill. Language-Independent Aspect-Oriented Proggraming. In *Proceedings of the 18th annual ACM SIGPLAN conference on Object-Oriented Programing, Systems, Languages, and Applications (OOPSLA'03)*, 2003, pages 1–12.

[4] Gregor Kiczales, et al. An overview of AspectJ. In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP'01)*, 2001, pages 327–353.

[5] Hironori Washizaki, et al. AOJS: aspect-oriented javascript programming framework for web development. In *Proceedings of the 8th workshop on Aspects, components, and patterns for infrastructure software (ACP4IS'09)*, 2009, pages 31–36.

[6] Rodolfo Toledo, Paul Leger, and Éric Tanter. AspectScript: expressive aspect for the web, In *Proceedings of the 9th ACM International Conference on Aspect-Oriented Software Development (AOSD'10)*, 2010, pages 13–24

[7] UniAspect website in UnicoenProject. http://www.unicoen.net/application/uniaspect.html

[8] UNICOEN. A framework for developing code processing tools. http://www.unicoen.net/