5th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence and Informatics
January 25-26, 2007 ░ Poprad, Slovakia

Static and Dynamic Approaches to Weaving

Michal Forgáč, Ján Kollár

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia
Michal.Forgac@tuke.sk, Jan.Kollar@tuke.sk

Abstract: In this paper we present current state of principles and applications in static and dynamic weaving. We are concentrating on static weaving in AspectJ and dynamic weaving in PROSE - PROgrammable extenSions of sErvice. The contribution of this paper is in analyses of both approaches to weaving which we aim to apply as essential mechanisms when constructing software systems by automatic evolution.

Keywords: Aspect oriented programming, systems engineering, weaving mechanisms, AspectJ, PROSE

1 Introduction

As a complexity of software systems grows, the need for advanced methods and tools of their specification and the development is the task of the high importance. Object oriented programming (OOP) has increased the productivity of systems rapidly. Benefits of object orientation include support for modular design, code sharing, and extensibility [15]. Object orientation has enabled using inheritance and polymorphism in sence of Strachey's original definition of polymorphism [16]. However, the key issue in application of object approach is that it is not sufficient for crosscutting concerns.

One of the fundamental principles of the software engineering is a principle of separation of concerns.

A concern is a particular goal, concept, or area of interest, it means that it is in substance semantical concern. From the structural point of view a concern may appear in source code as [4]:

This work has been supported by the Scientific grant agency project (VEGA) No. 1/1065/04 ‘Specification and Implementation of Aspects in Programming’

- A component, if it is cleanly encapsulated in building block of the programming language.
- An aspect, if it cannot be cleanly encapsulated in any construct of the programming language.

Components are structuraly compact (core) concerns, aspects are properties that crosscut components and tend to affect component performance and semantics.

A general software system in OOP is composed by multiple core and crosscutting concerns. Core concerns are crosscutted by secondary (crosscutting) concerns and thus source codes become less clear. This problem has been solved by a new aspect oriented paradigm which overcame the consideration of a concern as inevitably beeing described in component module.

Aspect oriented programming (AOP) [1, 2, 3, 5, 6, 13, 14] enables to represent a concern by an aspect which is semantically tangled or scattered. In this sence AOP paradigm extends OOP paradigm and AspectJ extends Java.

Since the design of programs created by AOP should be easier, source codes should be more transparent and so maintaining of programs should be without radical problems.

According to [5], AOP support can be added to languages that are not only object-oriented, so if AOP is added to a procedural language, the constructs that crosscut block structure of procedural programs must be added. AOP support can be added also to functional languages [12].

This paper is structured as follows: In section 2 AOP terminology is presented. In section 3 general information about static and dynamic weaving is presented. Section 4 deals with AspectJ framework and PROSE platform.

2 Aspect-oriented Principles

In aspect oriented programming an object oriented language is exploited, as a basis which is extended by aspect features, forming an aspect language. These features consist essentially of: pointcut designators and advices.

A pointcut is a term given to the points of execution in the application at which a cross-cutting concern needs to be applied. Base on this selection term a group of join points is selected which is related to the concern.

A join point is a well-defined point in the base program (component language) that can be modified by an aspect. Join points may include calls to a method, a conditional check, a loop's beginning, or an assignment. They have a context associated with them.

An advice is an additional code which is executed at join points that are selected by corresponding pointcut. Advices in general can be executed before, after, or around the join point.

An aspect is a combination of the pointcut and the advice. Hence, an aspect module encapsulates one or more crosscutting concerns. Aspect oriented language, such as AspectJ is a base unit of modularization as classes in Java and enables static or dynamic crosscutting.

Static crosscutting enables inserting declarations of additional attributes, additional methods and additional constructors into existing classes. It is not important where an insertion is performed, because this additional properties can be inserted somewhere into the class declaration.

In dynamic crosscutting it is very important, where additional behavior is added. New behavior is inserted in well-defined points - join points, which are defined by pointcuts as mentioned before.

Separated crosscutting concerns are mutually woven by a specialized compiler extended by a weaver. The weaving can be static or dynamic which depends on when the weaver weaves. If weaving is performed before compiling, it is static weaving, if it is performed after compiling, it is dynamic weaving.

Of course dynamic weaving can be performed just after compiling since it is performed in run-time.

Masive expansion of the aspect oriented software development can bring new possibilities to systems engineering.

3 Approaches to Weaving

Some approaches to weaving can be found in [1, 2, 3, 6]. Essentially weaving acts as a composition mechanism for merging of a new aspect with an original system to produce a system with a new semantics. Hence, weaving is a process of composition performed practically by an aspect weaver. An original system (its code) is affected by the aspect weaver automatically.

3.1 Static Weaving

According to [6], static weaving means the modification of the source code of a class by inserting advice - aspect specific statements at selected join points. Advice code is inserted into classes. The result is a highly optimized woven code whose execution speed is comparable to that of code written by traditional methodologies (without AOP or related techniques). There is one disadvantage in

static weaving – it makes difficult to later identify aspect-specific statements in a woven code. As a consequence, adapting or replacing aspects dynamically can be time-consuming or not possible at all.

In static weaving, compile-time approaches are used. The code of the original application can be composed with the aspect code and a result is a new code that includes additional functionality. This approach uses pre-processing or integrates the aspect weaver and the source compiler. Compile-time AOP produces well-formed programs, since the resulting code must pass a compiler. Therefore only few additional checks must be performed at run-time. The compiler may even optimize the result code, leading to improved performance [2].

This type of weaving is represented for example by AspectJ, AspectC, AspectC++, HyperJ. The application of static weaving in AspectJ language will be presented in more details in section 4.

3.2 Dynamic Weaving

Dynamic weaving is performed in load-time or run-time [1]. These approaches allow modification of a program while it is running.

Load-time approaches as Binary Component Adaptation and Java Object Instrumentation Environment allow transforming an application at load-time. They replace the class loader of a Java Virtual Machine and change the classes at load-time.

Java defines its semantics in terms of bytecodes that can be interpreted by an appropriate bytecode interpreter – the Java Virtual Machine (JVM). A straightforward run-time approach to AOP for Java is to locate the support for weaving and unweaving aspects directly in the JVM. JVM must provide an interface for weaving aspects at run-time which name is Java Virtual Machine Aspect Interface (JVMAI). But another approaches are possible too [1, 2, 3].

Four requirements are presented for efficient dynamic weaving for Java in [2].

These requirements are as follows:

- Efficiency under normal operations (no woven aspects)
- Secure and atomic (in the same time) weaving
- Efficient advice execution
- Flexibility

A potential drawback of dynamic AOP is the performance overhead. In addition, dynamic AOP may be insecure since it can allow weaving of malicious advice [2].

A complex system created by dynamic weaving could behave as a live organism which could change its properties during its life. This issue relates to Genetic algorithm [7,11].

Dynamic weaving is applied for example by TinyC, Java Aspect Component, PROgrammable extenSions of sErvice (PROSE). This type of weaving can be used for example in Real-Time systems [9,10].

4 Application of Weaving in Aspect Oriented Programming

In this section we will present essential representatives of static weaving – AspectJ and dynamic weaving - Programmable extensions of service (PROSE). AspecJ extends a component language with new constructs, but in PROSE aspects are expressed with a component language. The component language in both cases is Java.

4.1 AspectJ

AspectJ integrates specification and implementation AO framework. According to [5], AspectJ is designed as a compatible extension to Java. The following properties are meant to be compatible:

- **Upward compatibility**: All legal Java programs are legal AspecJ programs.
- **Platform compatibility:** All legal AspectJ programs run on standard Java virtual machines.
- **Tool compatibility**: It is possible to extend existing tools to support AspectJ in a natural way (IDEs, documentation tools, design tools).
- **Programmer compatibility**: Programming with AspectJ feels like a natural extension of programming with Java.

AspectJ extends Java with support for two kinds of crosscutting implementations [5]. It makes possible to define an additional implementation to run at certain well-defined points in the execution of a program (dynamic crosscutting) and it enables to define new operations on existing types (static crosscutting).

Core constructs of AspectJ are join points, pointcuts and advices.

Join points are well-defined points in the program's execution. The dynamic join points of AspectJ, which are used for dynamic crosscutting, are:

- method and constructor call
- method and constructor call reception
- method and contructor execution
- field get and set
- exception handler execution
- class and object initialization

Pointcuts select sets of join points and optionally some of the values in the execution context of those join points.

Join points are not selected separately but by pointcuts designators (anonymous or named user-defined).

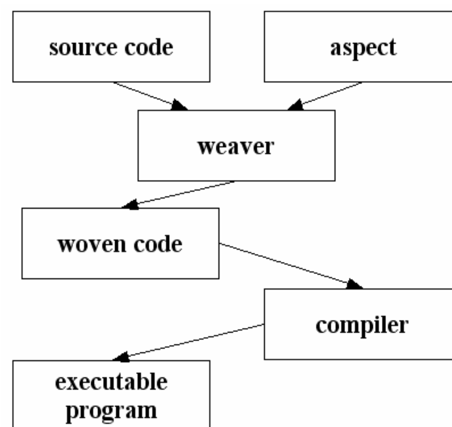

Figure 1
Generalized scheme of static weaving

An advice is a language construct which declares that certain code should be executed at each of the join points in a pointcut. AspectJ supports before, after, and around advices. It means that pointcut and advice determine weaving rules.

Aspect is a modular unit of crosscutting implementation. Aspect declarations may include pointcut declarations, advice declarations and other kinds of declarations permitted in class declarations.

Opposite to dynamic crosscutting which requires exact location of advice code, static crosscutting covers member introduction and class hierarchy modification for which it is not important where it will be located in class exactly [5, 8].

In Fig. 1 a generalized scheme of static weaving is exposed. In the current version of AspectJ (in compile-time weaving), weaver and compiler are integrated together into ajc compiler/weaver. Thus the weaver, the woven code, and the

compiler in Fig. 1 could be figured as one modul. Ajc compiler/weaver compiles source codes (classes and aspects) and produces woven class files. The invocation of the weaver is integrated into the ajc compilation process.

But another approaches as post-compile weaving (also called binary weaving), and even load-time weaving are in AspectJ possible too.

4.2 PROSE

PROSE is a generic platform for software adaptation by aspect oriented paradigm [3]. Dynamic AOP extends the original notion of AOP by allowing weaving at load or run time. This platform has been developed for several years, so previous versions of PROSE explored the issue of dynamic AOP (interception through the Java Virtual Machine Debugger Interface (JVMDI) and Just-in-time (JIT) based weaving), current version represents complete and flexible adaptation platform.

The first version of PROSE used the Java Virtual Machine Debugger Interface (JVMDI) to convert join points into stop points. Once the application had been stopped, the advice was executed externally to the application although the advice had access to the context where it was being executed (e.g., stack frames, calling parameters for methods, etc.) [1,3].

The second version of PROSE extended this model by giving the option of, instead of using the debugger, using the baseline JIT compiler. The idea was to weave hooks into the application at native code locations that correspond to all potential join points. When executed, the hooks determine whether an advice needed to be invoked for that particular join point and called the advice [2,3].

Current version supports different forms of weaving: stub weaving and advice weaving. These forms are combined to give the ability to fine tune trade-off between performance and flexibility in the adaptation.

PROSE aspect language is very similar its component language – Java. All aspects extend relevant base classes. An aspect may contain one or more crosscuting objects, which correspond to pointcuts and an associated advice in AspectJ. A crosscut object defines an advice method and a pointcut method which defines a set of join points where the advice should be executed.

PROSE currently supports the following set of join points [3]:

- **Method boundaries:** Encompassing method entry and exit (extending application by adding new functionality).
- **Method redefine:** Replacing method bodies (changing to the current behavior).
- **Field access and modification:** Tracking acess to variables.

- **Exception join points:** Including exception catch and throw.

The architecture of PROSE is divided into AOP engine layer and execution monitor layer, as is shown in Fig. 2.

The AOP engine accepts aspects and transforms them into join point requests. It activates the join point requests by invoking methods for the execution monitor. The bytecode advice weaver module is responsible for the bytecode manipulations and the methods instrumentation. When the program execution reaches one of the activated join points, the execution monitor notifies the AOP engine which then executes the advice.

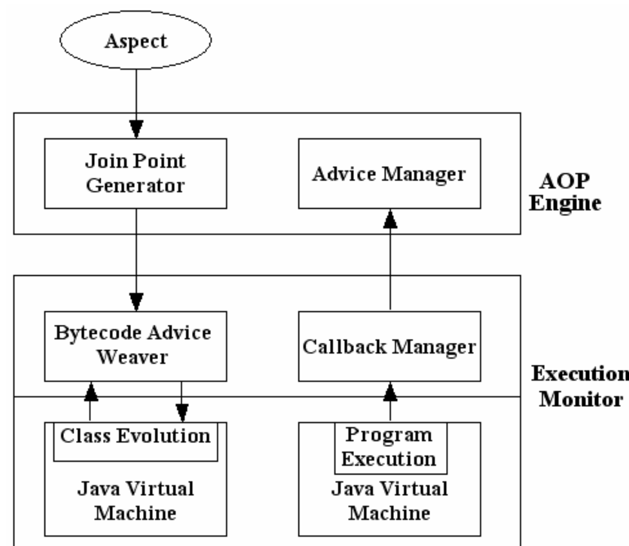

Figure 2
PROSE stub and advice weaving architecture

Conclusions

At this point of our research we are starting with a new method for automatic construction of software evolution driven by rules. The idea comes not just from the specification methods exploiting predicate and temporal logic, but also from the idea of AOP. In this paper we have been concentrated on weaving because as we feel it is not just the mechanism for composition of crosscutting concerns – aspects but it represents also a human activity while programming software systems. To be able to perform this composition driven by rules automatically, the detailed analysis of weaving mechanism is needed. This paper is a particular contribution and a step to further research in this area.

Acknowledgement

This work has been supported by the Scientific grant agency project (VEGA) No. 1/1065/04 'Specification and Implementation of Aspects in Programming'.

References

[1] Andrei Popovici, Thomas Gross, Gustavo Alonso: Dynamic weaving for Aspect-Oriented Programming, In: 1st International Conference on Aspect-Oriented Software Development, Enschede, The Netherlands, 2002, pp. 141-147

[2] Andrej Popovici, Gustavo Alonso, Thomas Gross: Just in Time Aspects: Efficient Dynamic Weaving for Java, In: 2nd International Conference on Aspect-Oriented Software Development, Boston, USA, 2003, pp. 100-109

[3] Angela Nicoara, Gustavo Alonso: Dynamic AOP with PROSE. In: Proceedings of International Workshop on Adaptive and Self-Managing Enterprise Applications (ASMEA 2005) in conjunction with the 17th Conference on Advanced Information Systems Engineering (CAISE 2005), Porto, Portugal, June 2005

[4] G. Kiczales, J. Lamping, A. Mendekar, C. Maeda, C. Videira Lopes, J.-M. Loingtier, J. Irwin: Aspect-Oriented Programming. In Mehmet Aksit and Satoshi Matsuoka, editors, 11th European Conf. Object-Oriented Programming, volume 1241 of LNCS, Springer Verlag, 1997, pp. 220-242

[5] G. Kicszales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. Griswold.: An overview of AspectJ, In Proceedings of ECOOP'01, European Conference on Object/Oriented Programming, Springer-Verlag (LNCS 2072), 2001, pp. 327-355

[6] Kai Böllert: On weaving aspects, In Proceedings of Aspect-Oriented Programming Workshop at ECOOP'99, Lisbon, Portugal, June 1999

[7] V. Fabera, V. Jáneš: Automata Construct with Genetic Algorithm, Proc. of the 4-th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence, Herľany, Slovakia, Jan. 20-21, 2006, pp. 164-176

[8] Hui Wu, J. G. Gray, S. Roychoundhury, M. Mernik: Weaving a debugging aspect into domain-specific language grammars, Proc. of the 2005 ACM symposium on applied computing, 2005, pp. 1370-1374

[9] D. Zmaranda, G. Gabor, C. Rusu: Evaluation method algorithm used to improve real-time control systems stability. Analele Universitatii din Oradea, Proc. 8-th International Conference on Engineering of Modern Electric Systems, Felix Spa-Oradea, May 26-28, University of Oradea, Romania, 2005, pp. 170-175

[10] D. Zmaranda, G. Gabor, M. Gligor: A Framework for Modeling and Evaluating Timing Behavior for Real-Time Systems, Proc. SINTES 12 –

Int. Symposium on Systems Theory, October 20-22, University of Craiova, Romania, 2005, pp. 514-520

[11] Vladimír Siládi, Michal Vagač: A Parallell Genetic Algorithm for Optimization an Irregular Connecting Network Topology, Proc. of ECI'2006, 7-th International Scientific Conf. on Electronic Computers and Informatics, Košice-Herľany, September 20-22, FEII TU Košice, Slovakia, 2006, pp. 298-302

[12] Marek Běhálek, Petr Šaloun: Paralelization of Process Functional Language, Proc. of ECI'2006, 7-th International Scientific Conf. on Electronic Computers and Informatics, Košice-Herľany, September 20-22, FEII TU Košice, Slovakia, 2006, pp. 168-173

[13 Ján Kollár: Structural Propositon for Aspect Oriented Software Evolution, Proc. of ECI'2006, 7-th International Scientific Conf. on Electronic Computers and Informatics, Košice-Herľany, September 20-22, FEII TU Košice, Slovakia, 2006, pp. 180-185

[14] Damijan Rebernak, Marjan Mernik, Pedro Rangel Henriques, Maria Joao Varanda Pereira: AspectLISA: An Aspect-oriented Compiler Construction System Based on Attribute Grammars, Electronic Notes in Theoretical Computer Science, Volume 164, Issue 2, 24 October 2006, pp. 37-53

[15] Scott Danforth, Chris Tomlinson: Type theories and object-oriented programmimg, ACM Computing Surveys (CSUR), v.20 n.1, March 1988, pp. 29-72

[16] C. Strachey: Fundamental concepts in programming languages. Lecture Notes, International Summer School in Computer Programming, Copenhagen, Denmark, August 1967