# Vergleich von Hyper/J und AspectJ

Steffen Harbich
Fakultät für Informatik
Universität Magdeburg, Deutschland
E-Mail: Steffen.Harbich@st.ovgu.de

***Zusammenfassung* — Ausgehend von dem in der Vorlesung behandelten populären Vertreter der aspektorientierten Programmierung AspectJ untersuchen wir das von IBM Research entwickelte Tool Hyper/J bezüglich der Unterstützung der Entwicklung von Produktlinien und vergleichen beide Ansätze.**

## I. EINFÜHRUNG

Bei der Entwicklung von Softwareproduktlinien ist die Aufteilung der Software in Module gemäß wichtiger Belange (Features) wesentlich. Diese Modularisierung bringt einige Vorteile mit sich [8]:

- Die Software ist insgesamt leichter verständlich, da einzelne Teile separat verstanden werden können.
- Die Komplexität der Implementierung wird hinter Schnittstellen versteckt.
- Auch sind Änderungen, die aufgrund der Trennung meist lokal erfolgen, leichter wartbar.
- Die Wiederverwendbarkeit des Codes ist erhöht.

Man stößt allerdings auch auf Probleme bei der Umsetzung mittels objektorientierter Programmierung. So gibt es etwa querschneidende Belange, die verhindern, dass alle Belange gleichzeitig im Programm modularisiert werden können. Dies führt zum Verstreuen von Implementierungsteilen eines Belangs über verschiedene Implementierungseinheiten und zur Vermischung von Implementierungsteilen mehrerer Belange innerhalb einer Implementierungseinheit. Dadurch sind Features unter Umständen nicht mehr leicht im Quelltext auffindbar und dies wiederum erschwert z.B. die Suche von Fehlern im Programm. Außerdem müssen zukünftige Erweiterungen der Software so gut wie möglich vorausgeplant werden, da sonst der Basiscode nachträglich angepasst werden muss. In bestimmten Fällen ist auch die Vererbung zu unflexibel, um Erweiterungen von Klassen vorzunehmen.

Wir haben uns in der Vorlesung verschiedene Ansätze angeschaut, wie man diese Probleme überwinden kann. Die aspektorientierte Programmierung ist einer dieser Ansätze, dem ich mit dieser Ausarbeitung, dem Vergleich von AspectJ und Hyper/J, detaillierter betrachten möchte. Zur Illustration der beiden Tools verwende ich das aus der Übung bekannte „Chat"-Programm als Implementierungsgrundlage für ein praktisches Beispiel. Der Quelltext befindet sich im mitgesendeten Zip-Archiv.

## II. ASPECTJ

AspectJ ist eine aspektorientierte Erweiterung der objektorientierten Sprache Java. Ein Aspekt beschreibt Änderungen am Basiscode, die zur Implementierung eines Belangs benötigt werden. Pro Aspekt wird eine Datei angelegt, die neben Standard-Java verschiedene AspectJ-spezifische Konstrukte beinhaltet, mit denen definiert wird, welcher Quelltext an welcher Stelle eingefügt werden soll. Nehmen wir ein Beispiel zu Hilfe: In der Chat-Anwendung aus der Übung soll es Varianten geben können, bei denen die Kommunikation von Server und Clients verschlüsselt vonstatten geht. Zur Vereinfachung wird nur ein einfaches Verschlüsselungsverfahren, nämlich Rot13, verwendet. Betrachten wir folgenden Auszug aus der Klasse `TextMessageImpl`, die eine Textnachricht darstellt, die zwischen Server und Client verschickt werden:

```
private String content;

public String getContent() {
        return content;
}

public void setContent(String content) {
        this.content = content;
}
```

Um den Inhalt der Textnachricht zu verschlüsseln, könnte man nun beim Getter von `content` eine Methode zum Entschlüsseln aufrufen und beim Setter eine Methode, die den Parameter verschlüsselt, bevor er in die Membervariable `content` gesetzt wird. Wir wollen allerdings die Änderung nicht im Basiscode vornehmen, denn dann würde der Belang (das Feature „Verschlüsselung") nicht sauber vom Rest getrennt sein, was aber ein erklärtes Ziel war. Mittels eines Aspekts „Verschlüsselung" wird dies erreicht:

```
public aspect Encryption {

private EncryptionAlgorithm encryptionAlgorithm =
        new Rot13();

String around() : execution(String
        TextMessageImpl.getContent()) {

        String content = proceed();
        return
        encryptionAlgorithm.decrypt(content);
}

void around(String content) : execution(void
```

```
        TextMessageImpl.setContent(String)) &&
        args(content) {

        proceed(
        encryptionAlgorithm.encrypt(content));
}
}
```

Mit dem **around**- und **execution**-Schlüsselwort wird signalisiert, das der entsprechende Codeblock *anstelle* der Methode ausgeführt wird, die angegeben wurde (hier: `getContent` und `setContent` der Klasse `TextMessageImpl`). Per **proceed**-Schlüsselwort kann man den originalen Methodenaufruf durchführen und somit die Ver- und Entschlüsselung einfügen. Der AspecJ-Compiler webt schließlich die Aspekte in den Basiscode, sodass das Programm wie üblich mit der Java Virtual Machine ausgeführt werden kann.

Mit dem AJDT–Plugin für Eclipse beinhaltet AspectJ ein ausgereiftes Tool für die Entwicklung von Aspekten in Java. Für einfache und komfortable Produktlinienentwicklung ist allerdings eine Kombination mit einem Tool wie Feature IDE sinnvoll, um die Modellierung der Features und die Variantengenerierung zu unterstützen.

## III. HYPER/J

Hyper/J ist ein von IBM Research entwickeltes Tool zur Unterstützung von „Multi-dimensional Separation of Concerns“ in Java [1][2]. Als Dimension wird hier die Art eines Belangs verstanden, beispielsweise Klassen, Aspekte und Features. In jeder Dimension gibt es eine Menge von Belangen, nach denen die Aufteilung erfolgt. Die ausführlichen theoretischen Grundlagen findet man in der 60-seitigen Dokumentation zu Hyper/J [3]. Wir beschränken uns im Folgenden auf die praktische Anwendung bezüglich der Produktlinienentwicklung. In der Literatur findet man übrigens vorrangig die Zuordnungen von Hyper/J zum themenorientierten Programmierparadigma [3][4]. In [5] wird aber auch die Verwendung im Sinne der Aspektorientierung beschrieben. Hyper/J besitzt auf jeden Fall einige Umsetzungsmöglichkeiten, die aus der aspektorientierten Programmierung bekannt sind, wie wir noch am Beispiel sehen werden.

Beginnen wir zunächst mit dem Beispiel der Implementierung des Verschlüsselungs-Features im Chat-Programm. Auch mit Hyper/J wurde zuerst der Basiscode geschrieben (bzw. übernommen, denn es ist der gleiche wie im AspectJ-Beispiel). Anschließend wurde ein Java-Package `feature.encryption` angelegt, in dem neben den eigentlichen Klassen für die Verschlüsselungsalgorithmen noch eine weitere mit dem Namen `TextMessageImpl` zu finden ist. Diese enthält die Methoden `getContent` und `setContent`, die beim Kompilieren die entsprechenden Methoden im Basiscode ersetzen:

```
private String content;
private static EncryptionAlgorithm
        encryptionAlgorithm = new Rot13();

public String getContent() {
        return
encryptionAlgorithm.decrypt(content);
}

public void setContent(String content) {
        this.content =
        encryptionAlgorithm.encrypt(content);
}
```

Wie Hyper/J den Code letztlich zusammenstellt, wird über mehrere Konfigurationsdateien angegeben. Im Beispiel ist etwa angegeben, dass die beiden obigen Methoden die Methoden aus dem Basiscode ersetzen. Neben der Ersetzung ist auch eine Zusammensetzung der Methoden möglich sowie die Angabe, welche Reihenfolge dabei verwendet wird. Bei Methoden mit Rückgabewert kann zusätzlich bestimmt werden, wie die Ergebnisse der einzelnen Methoden zusammengerechnet werden. Leider besteht keine Möglichkeit, wie beim **around** von AspectJ die originale Methode aus der Implementierung im Feature heraus aufzurufen. Das macht es schwierig noch ein weiteres Feature hinzuzufügen, das den Parameter `content` modifizieren und dann die eigentlich Methode aufrufen soll.

Zu den Konfigurationsdateien gehören eine Datei zur Spezifikation des Hyperspace (Variant1.hs), eine weitere für die Angabe des Hypermodules (Variant1.hm) und mehrere Dateien für die Abbildung von Quelltexteinheiten auf die Belange (*.cm). In der Hyperspace-Datei wird festgelegt, welche Klassen Hyper/J bei der Komposition der Features berücksichtigen soll. Im Hypermodule stehen die Auswahl der Features sowie eine Reihe von Beziehungen zwischen den Belangen, mit denen die Komposition beeinflusst werden kann. Ein Beispiel für solch eine Beziehung ist *override*, mit der etwa das Überschreiben von Methoden realisiert werden kann. Ein weiteres Beispiel ist *bracket*. Mit Hilfe von *bracket* können Methodenaufrufe in dem Sinne „geklammert“ werden, dass davor und danach festgelegte Methoden aufgerufen werden. Die Auswahl der zu klammernden Methode geschieht per Name, wobei bestimmte Platzhalter und dergleichen als Muster auftauchen dürfen. Auch ist die Angabe erlaubt, woher der Methodenaufruf der zu klammernden Methode kommt. Dies erinnert stark an die Möglichkeiten von AspectJ, nur dass, wie oben bereits erwähnt, leider kein **around**-ähnliches Konstrukt zur Verfügung steht. Die Dokumentation verweist dazu auf einen zukünftigen Release von Hyper/J. Die Entwicklung scheint allerdings eingestellt worden zu sein, da seit dem Jahr 2000 keine neuen Veröffentlichungen getätigt wurden.

Wollte man die Gespräche in unserem Chat-Beispielprogramm aufzeichnen lassen, so müsste man folgende Zeile als Beziehung in das Hypermodule mit aufnehmen:

```
bracket "*"."{~_,~<}*broadcast" with
 after
Feature.History.HistoryAspect.serverBroadcast;
```

Gesucht werden dann Methoden mit dem Namen `broadcast` und so angepasst, dass danach immer `serverBroadcast` der Klasse `HistoryAspect` ausgeführt wird. Dort kann dann das eigentliche Aufzeichnen der Chat-Textnachricht beim Server erledigt werden. Bedauerlicherweise wollte es auch nach langwieriger Fehlersuche nicht gelingen, dies umzusetzen. Laut Dokumentation und De-

monstrationsprogramm soll dies aber prinzipiell machbar sein.

Die Installation und die anfängliche Einrichtung von Hyper/J sind nicht trivial, da es keine Integration in eine Entwicklungsumgebung wie Eclipse gibt. Die Ausführung von Hyper/J ist deshalb im Beispielprogramm per Windows-Batch-Datei *RunHyperJ.bat* umgesetzt. Die Pfade müssen vor dem Ausprobieren angepasst werden.

## IV. Vergleich beider Ansätze

Wir wollen nun beide Ansätze anhand ausgewählter Kriterien vergleichen. Der Vergleich basiert auf [7].

In AspectJ gibt es einen klassischen Compiler, der die Java-Quelltexte benötigt und mit den aspektorientierungsspezifischen Konstrukten verwebt. Die Ausgabe ist der übliche Java-Byte-Code und kann ganz normal von der Java Virtual Machine interpretiert werden [6]. Hyper/J unterscheidet sich insofern, dass auf den vom Java-Compiler generierten *class*-Dateien gearbeitet wird. So kann es auch dann noch verwendet werden, wenn der ursprüngliche Quelltext nicht vorhanden ist.

Introduktionen – Einfügungen von Methoden, Feldern usw. in den Basiscode – werden von beiden Ansätzen im selben Umfang unterstützt. In AspectJ wird dies explizit über Sprachmittel realisiert, in Hyper/J geschieht dies implizit über die Komposition von Klassen. Zudem ist es in AspectJ möglich per Aspekt einer Klasse neue Interfaces hinzuzufügen.

Bereits angesprochen wurde der Unterschied, dass Hyper/J ein Verhalten, welches in AspectJ per `around` erzielt werden kann, nicht unterstützt, wodurch die Ausdrucksmöglichkeiten nicht unerheblich eingeschränkt sind.

Pointcuts, die sehr umfangreich und feingranular festgelegt werden können in AspectJ, kennt das Tool von IBM Research nicht. Entsprechende Basisfunktionalität ist aber über diverse Kompositionsregeln wie der *bracket*-Beziehung verfügbar. Während Pointcuts und Advices direkt im Aspekt geschrieben werden, verwendet man bei Hyper/J zusätzliche Konfigurationsdateien.

## V. Fazit

Wir haben am Chat-Beispiel gesehen, wie mit den zwei unterschiedlichen Ansätzen Produktlinien entwickelt werden können. Nach meiner persönlichen Einschätzung ist AspectJ dabei eindeutig zu bevorzugen. Dies begründet sich zum einen durch die größere Ausdruckskraft der Spracherweiterung und zum anderen durch die sehr gute Tool-Unterstützung, die sich problemlos in die Entwicklungsumgebung Eclipse integriert.

## Referenzen


[1] IBM Research, *Hyper/J™: Multi-Dimensional Separation of Concerns for Java™*, www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm, Zugriff: 03.01.2011

[2] IBM alphaWorks, *HyperJ*, www.alphaworks.ibm.com/tech/hyperj, Zugriff: 03.01.2011

[3] P. Tarr, H. Ossher, *Hyper/J™ User and Installation Manual*, IBM Research, 2000

[4] Wikipedia, *Subjectoriented Programming - Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/w/index.php?title=Subject-oriented_programming&oldid=395380548, Version: 2011, Zugriff: 04.01.2011

[5] C. Chavez, A. Garcia, C. Lucena, *Some Insights on the Use of AspectJ and Hyper/J*, http://reference.kfupm.edu.sa/content/s/o/some_insights_on_the_use_of_aspectj_and__765991.pdf, Zugriff: 05.01.2011

[6] Wikipedia, *AspectJ – Wikipedia, The Free Encyclopedia*, http://de.wikipedia.org/w/index.php?title=AspectJ&oldid=71034382, Version: 2011, Zugriff: 05.01.2011

[7] N. Heller, *HyperJ*, Seminararbeit des Seminars *Konzepte und Werkzeuge für die Softwareentwicklung mit Java, SS 2005*, http://ebus.informatik.uni-leipzig.de/www/media/lehre/seminar-javatools05/semtools05-heller-text.pdf, Zugriff: 04.01.2011

[8] C. Kästner, S. Apel, G. Saake, Folien zur Vorlesung *Erweiterte Programmierkonzepte für maßgeschneiderte Datenhaltung*, Wintersemester 2010/11