Multi-Dimensional Separation Of Concerns mit Hyper/J

Martin Schauer (martin.schauer@stud.informatik.uni-erlangen.de)

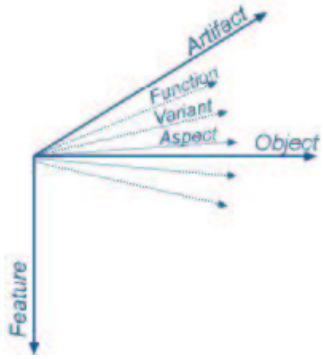
Friedrich-Alexander-Universität Erlangen-Nürnberg
Hauptseminar Aspektorientierte Systemprogrammierung

05. Mai 2003

In großen Softwareprojekten ist eine der größten Herausforderungen, durch konsequente Dekomposition und Modularisierung die Komplexität des Projekts in Schranken zu halten. Dabei gelingt es im allgemeinen nicht, die Wartbarkeit, die Erweiterbarkeit und das Anpassungsvermögen auf individuelle Bedürfnisse - z. B. des Anwenders, des Entwicklers, ... - aufrechtzuerhalten. Hyper/J ist ein von IBM entwickeltes Werkzeug für Java, um die Entwickler dabei zu unterstützen, die größtmögliche Flexibilität in allen Phasen des Software-Lifecycles zu gewährleisten. Das Schlagwort, das die Idee hinter Hyper/J illustrieren soll, ist Multi-Dimensional Separation Of Concerns. Dieses Dokument beschäftigt sich deshalb, vor einer kleinen Einführung in das Werkzeug selbst, mit dieser Art und Weise der Dekomposition.

Im Software-Engineering gilt Separation Of Concerns als das Mittel der Wahl, um die Komplexität eines Software-Projekts so gering wie möglich zu halten und um die Last des Projekts auf mehrere Schultern zu verteilen. Die Idee dahinter ist, zusammengehörige Teile zu identifizieren und einzukapseln. Änderungen müssten dann nur in einem klar abgegrenzten Bereich vorgenommen werden. Die klassische Art und Weise zur Kapselung ist das zentrale Ausdrucksmittel, das die Implementierungssprache zur Verfügung stellt, also z. B. die Klasse in objektorientierten Sprachen, die Funktion in funktionsorientierten Sprachen, ... Weniger beachtet, da es kaum geeignete Darstellungsmöglichkeiten dafür gibt, sind andere Arten der Modularisierung. Man könnte eine Trennung nach Funktionalität vornehmen, z. B. Anzeige-, Überprüfungs-, Auswertungsfunktionen, ... Oder auch Trennung nach Aspekten: z. B. Synchronisation, Tracing, ... Es gibt also verschiedene „Dimensionen“ nach denen man Separation Of Concerns vornehmen könnte. So hätte z. B. die Dimension „Funktionalität“

also die Concerns „Anzeigen“, „Überprüfen“, „Auswerten“, ... Die Autoren der Dokumentation zu Hyper/J nennen die gleichzeitige Aufteilung entlang mehrerer Dimension „Multi-Dimensional Separation Of Concerns“. (Abbildung von Website [2])



Der Effekt, der dabei gewünscht wird, ist der, dass Änderungen in irgendeinem Concern in irgendeiner Dimension automatisch auch entlang der anderen Dimensionen übernommen werden, ohne zusätzliche Arbeit leisten zu müssen. Die Menge der Concerns, die zur Bearbeitung anliegt, ändert sich im Software-Lifecycle ständig, z. B. sind während der Entwicklung andere Belange von Interesse, als in der Produktivversion. Nicht zuletzt ändern sich Concerns durch die Vorstellungen des Anwenders, so dass manche Funktionalität verändert werden muss, manch neue hinzukommt. Wenn Separation Of Concerns nur entlang einer Dimension vorgenommen wird – wie es in der Praxis derzeit üblich ist – führt das früher oder später zu Problemen. Wenn z. B. die Funktionalität in einem objektorientierten Softwareentwurf erweitert werden soll, ist es sehr wahrscheinlich, dass Erweiterungen in Klassen kreuz und quer über die Vererbungshierarchie vorgenommen werden müssen. Um wiederum die Autoren der Hyper/J-Dokumentation zu zitieren: das Problem ist die „Tyranny of the Dominant Decomposition“.

Hat man sich also einmal für eine bestimmte Dekomposition entschieden, muss man immer Rücksicht darauf nehmen. Es wird also schwierig werden, Funktionalität in eine einzige Klasse einer objektorientierten Sprache abzubilden, oder ein Objekt in einer regelbasierten Sprache zu verwenden, ... Ebenso schwerwiegend können Änderungen sein, z. B. beim Hinzufügen neuer Funktionalität. Im Extremfall kann sogar eine komplette Restrukturierung nötig werden. In diesem Fall wäre es einfacher, man hat die Dimension „Funktionalität“ bereits berücksichtigt. Dann wäre Erweiterung nur an einer Stelle nötig.

Multi-Dimensional Separation of Concerns (MDSOC) wäre also die Lösung, um die „Tyranny of the Dominant Decomposition“ zu brechen. MDSOC erlaubt Dekomposition entlang mehrerer Dimensionen gleichzeitig, wobei die Dimensionen gegenseitig überlappen

oder interagieren dürfen (neue Funktionalität wird natürlich mit Auswirkungen auf Klassen in OO-Sprachen einhergehen -> die Dimensionen „Funktionalität“ und „Klassen“ überlappen sich). Außerdem ist es mit MDSOC möglich, dynamisch neue Concerns innerhalb einer Dimension, oder sogar neue Dimensionen hinzuzufügen oder zu entfernen. Der IBM-Ansatz für MDSOC ist Hyper/J.

Der gesamte Raum aller Concerns bildet den sog. Hyperspace. Der Raum der Concerns ist durch die Anwendung bestimmt, zu ihm gehören alle Dinge, die für die Anwendung von Interesse sind. Die Concerns selbst sind wiederum aus Einheiten („Units“) aufgebaut. Bei Hyper/J gibt es zwei Arten von Einheiten. Zum einen die primitiven Einheiten, das sind Member-Variablen und –Methoden einer Java-Klasse, zum anderen zusammengesetzte Einheiten, also Interfaces, Klassen und Packages. Um nun aus allen Einheiten den Raum der Concerns aufzubauen gehören drei Schritte:

- Identifizieren (Identification): Auswählen der benötigten Concerns und Verbinden dieser mit den zugehörigen Einheiten, in Hyper/J das sog. „Concern Mapping“

- Kapseln (Encapsulation): Zusammenfassen mehrerer Concerns. Könnte in Java z. B. eine Klasse sein. In Hyper/J ist das ein sog. Hyperslice.

- Integrieren (Integration): Integration eines Concerns in ein aktuelles Projekt. In Java können nur ganze Klassen oder Interfaces integriert werden. In Hyper/J heißen integrierbare Einheiten Hypermodules.

Beim Identifizieren wird Hyper/J bekannt gemacht, welche Dimensionen mit welchen Concerns von Interesse sind, und welche Units zu welchen Concerns gehören. Hyper/J verwaltet aus diesen Informationen eine mehrdimensionale „Concern Matrix“. Für jede Dimension gibt es eine „Achse“ in der Matrix, jeder Punkt auf so einer Achse ist ein Concern innerhalb dieser Dimension. Jeder Einheit (Unit) werden Adressen innerhalb der Concern Matrix zugeordnet, um auszudrücken, dass diese Einheit zu diesen Concerns gehört. Um Hyper/J diese Zuordnung bekannt zu machen, gibt es die „Concern Mappings“. In einer Eingabedatei (*.cm) liegen Einträge der folgenden Form vor:

X: dimension.concern

Wie intuitiv zu vermuten, ordnet diese Anweisung die Einheit X dem Concern „concern“ in der Dimension „dimension“ zu. Dabei kann X sowohl eine primitive, als auch eine zusammengesetzte Einheit sein:

```
package someProject.util : Feature.Utilities;
class FooClass : Feature.FooClassConcern;
operation SomePackage.SomeClass.someMethod : Feature.SomeConcern;
field fooVar : Feature.Foo;
```

Um die so definierten Concerns einkapseln zu können, fehlt ihnen noch eine wichtige Eigenschaft, die Unabhängigkeit von anderen Komponenten. Erfüllen Concerns diese Eigenschaft, heißen sie in der Welt von Hyper/J Hyperslices. Ein Hyperslice ist „deklarativ vollständig“ („declaratively complete“), d. h. ein Hyperslice enthält alles, was er benötigt. Ruft also eine Methode innerhalb des Hyperslice eine Methode auf, die außerhalb des Hyperslice definiert ist, muss diese ebenfalls zum Hyperslice hinzugenommen werden. Dabei spielt es keine Rolle, ob die Methode im Hyperslice selbst implementiert wird oder nicht, eine abstrakte Definition genügt. Hyper/J kann alle Concerns zu gültigen Hyperslices durch „declaration completion“ automatisch ergänzen.

Hyperslices sind also von vornherein nicht miteinander gekoppelt, es können aber durchaus Beziehungen zwischen Hyperslices bestehen. Diese müssen bei der Integration mehrerer Hyperslices in ein sog. Hypermodule explizit angegeben werden. Hypermodules können prinzipiell nach dem „mix-and-match“ Prinzip beliebig aus Hyperslices zusammengesetzt werden, im Normalfall gibt es dabei schon Beschränkungen, die beachtet werden müssen: Wenn also, wie eben beschrieben, ein Hyperslice nur eine abstrakte Deklaration einer Methode enthält, muss für diese Methode eine Implementierung angegeben werden, um ein lauffähiges Hypermodule zu erhalten.

Aus der Eigenschaft der deklarativen Vollständigkeit eines Hyperslice folgt induktiv auch die deklarative Vollständigkeit eines Hypermodules. Ein Hypermodule kann also auch als Hyperslice angesehen werden und in andere Hypermodules integriert werden, so dass sukzessive größere Blöcke aufgebaut werden können.

Hypermodules werden in Hyper/J in Form einer Eingabedatei (*.hm) dem Werkzeug bekannt gemacht. Neben einer Menge an Hyperslices müssen auch die Beziehungen der Hyperslices untereinander definiert werden. Da es eine Vielzahl solcher Beziehungen gibt, sei der Leser

auf die Dokumentation von Hyper/J verwiesen. Hier nur eine Beispieldatei aus der Demo der Dokumentation:

```
hypermodule DemoSEE
   hyperslices:
      Feature.Kernel,
      Feature.Check,
      Feature.Display,
      Feature.StyleChecker;
   relationships:
      mergeByName;
      equate operation Feature.Kernel.process,
                       Feature.Check.check_process,
                       Feature.Display.display_process;
      set summary function for operation DemoSEE.check
           to external com.ibm.hyperj.SummaryFunctions.booleanAnd;
end hypermodule;
```

Dieses Hypermodule fasst durch die „mergeByName“-relationship die vier Hyperslices über gleichartige Namen zusammen, in diesem Fall also alle Methoden, die zur gleichen Klasse gehören. Die „equate“-relationship besagt, dass die Operationen Feature.Kernel.process (Feature ist der Name der Dimension, Kernel der Name des Concerns, process der Name der (hier primitiven) Einheit) Feature.Check.check_process und Feature.Display.display_process gleichgesetzt, durch das „mergeByName“ also zusammengefasst werden. Die letzte relationship zeigt, wie man zwei (verschiedene) Operationen ausführt und deren Rückgabewerte zu einem zusammenfasst (im Beispiel gibt es eine Methode, die die Syntax einer Eingabe überprüft, eine andere Methode, die den „Stil“ der Eingabe überprüft). Da beide Methoden eine boolean zurückgeben, ist hier ein einfaches logisches UND ausreichend.

Um nun noch den Hyperspace zu vervollständigen fehlt noch eine weitere Datei, die den eigentlichen Hyperspace definiert. Auch dieser wird über eine Eingabedatei (*.hs) definiert, die folgendes Format hat:

```
hyperspace hyperspaceName
classFileSpecification;
classFileSpecification;
...
```

Dabei kann *classFileSpecification* folgendermaßen aussehen:

```
class package1.className1, package1.className2;
composable class package2.* except package2.someClass;
composable class package3.* including subclasses;
composable file c:\users\smith\someProject\*;
uncomposable class java.lang.*, java.io.*;
```

Das Schlüsselwort **composable** (Standardeinstellung, falls nicht anders spezifiziert) bedeutet dabei, dass diese Klasse an relationships wie bei den Hypermodules vorgestellt teilnehmen dürfen. Das Schlüsselwort **uncomposable** verhindert dies und sollte auf jeden Fall bei Klassen der Java-Bibliothek verwendet werden.
In manchen Fällen kann die Definition des Hyperspace weggelassen werden. Hyper/J erzeugt dann automatisch eine Hyperspace Definition aus den Concern Mappings. Um zu sehen, in welchen Fällen das möglich ist, sei der Leser abermals auf die Hyper/J-Dokumentation verwiesen.

Hyper/J ist ein Werkzeug für Java und keine Erweiterung der Sprache. Hyper/J liest die Eingabedateien und erzeugt aus den getrennt übersetzten Klassen den Java-Bytecode passend zu den Integrationsvorschriften. Außerdem wird Pseudo-Java-Quelltext für das zusammengesetzte Projekt erzeugt, der aber nur für Debugging-Zwecke geeignet ist und sich nicht vom „normalen“ Java-Compiler übersetzen lässt. Hyper/J selbst ist ebenfalls in Java geschrieben, kann also von der Java-VM ausgeführt werden.

Hyper/J kann also dazu genutzt werden, große Projekte zum einen besser zu strukturieren, zum anderen die Wartbarkeit, Skalierbarkeit und Erweiterbarkeit zu verbessern. Natürlich ist es auch kein Allheilmittel und kann einen ordentlichen Entwurf nicht ersetzen.

Literatur:

[1]: P. Tarr, H. Ossher: Hyper/J Dokumentation

[2]: http://www.research.ibm.com/hyperspace

[3]: P. Tarr, H. Ossher: Multi-Dimensional Separation of Concerns in Hyperspace, April 99

[4]: W. Harrison, H. Ossher: Subject-Oriented Programming (A Critique on Pure Objects), ACM 93

[5]: http://www.alphaworks.ibm.com/tech/hyperj