Hyper/J

Übersicht

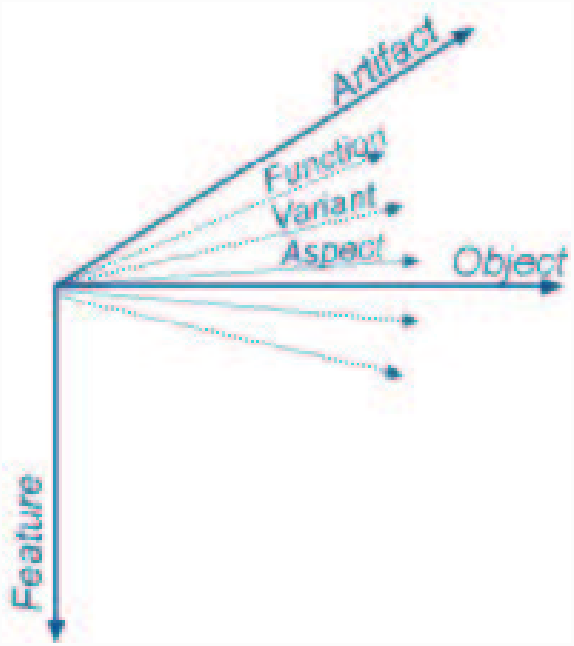
1. Multi-Dimensional Separation of Concerns
   1. Multi-Dimensional
   2. „Tyranny of the Dominant Decomposition“
   3. Lösung MDSOC
2. Hyper/J
   1. Grundlegende Begriffe
   2. Anwendungsbeispiel
3. Literatur

Multi-Dimensional - Motivation

- Grundlegendes Konzept: Separation of Concerns
- Verschiedene Arten von concerns:
  - Z. B. Konzept der Programmiersprache: Klassen, Funktionen, Regeln, ...
  - Z. B. Funktionalität: Anzeigen, Ausführen, ...
  - Z. B. Aspekte: Synchronisation, Tracing, ...
  - ...
- Es gibt also viele „Dimensionen“

Multi-Dimensional – Motivation

- Bild von der Webseite [2]:



Multi-Dimensional – Praxis

- Übliche Methoden erlauben *Separation of concerns* in nur einer Dimension
  - Concerns ändern sich im Laufe des Software-Lifecycles -> Design, Implementierung, Produktion, ...
  - Concerns ändern sich durch Bedürfnisse der Anwender
- Probleme v. a. dann, wenn Änderungen in einer anderen Dimension vorgenommen werden

„The Tyranny of the dominant Decomposition“

- Festlegung auf eine Dimension führt zu „dominanter Dekomposition“ (Bsp: Klasse in Objektorientierung). Folge:
  - Andere Dimensionen müssen diese Dekomposition verwenden (Funktionalität in eine Klasse?, Objekte in regelbasierten Sprachen ?, ...)
  - Änderungen (v. a. in anderen Dimensionen) können erschwert werden, im Extremfall Restrukturierung erforderlich (Hinzufügen neuer Funktionalitäten, ...)

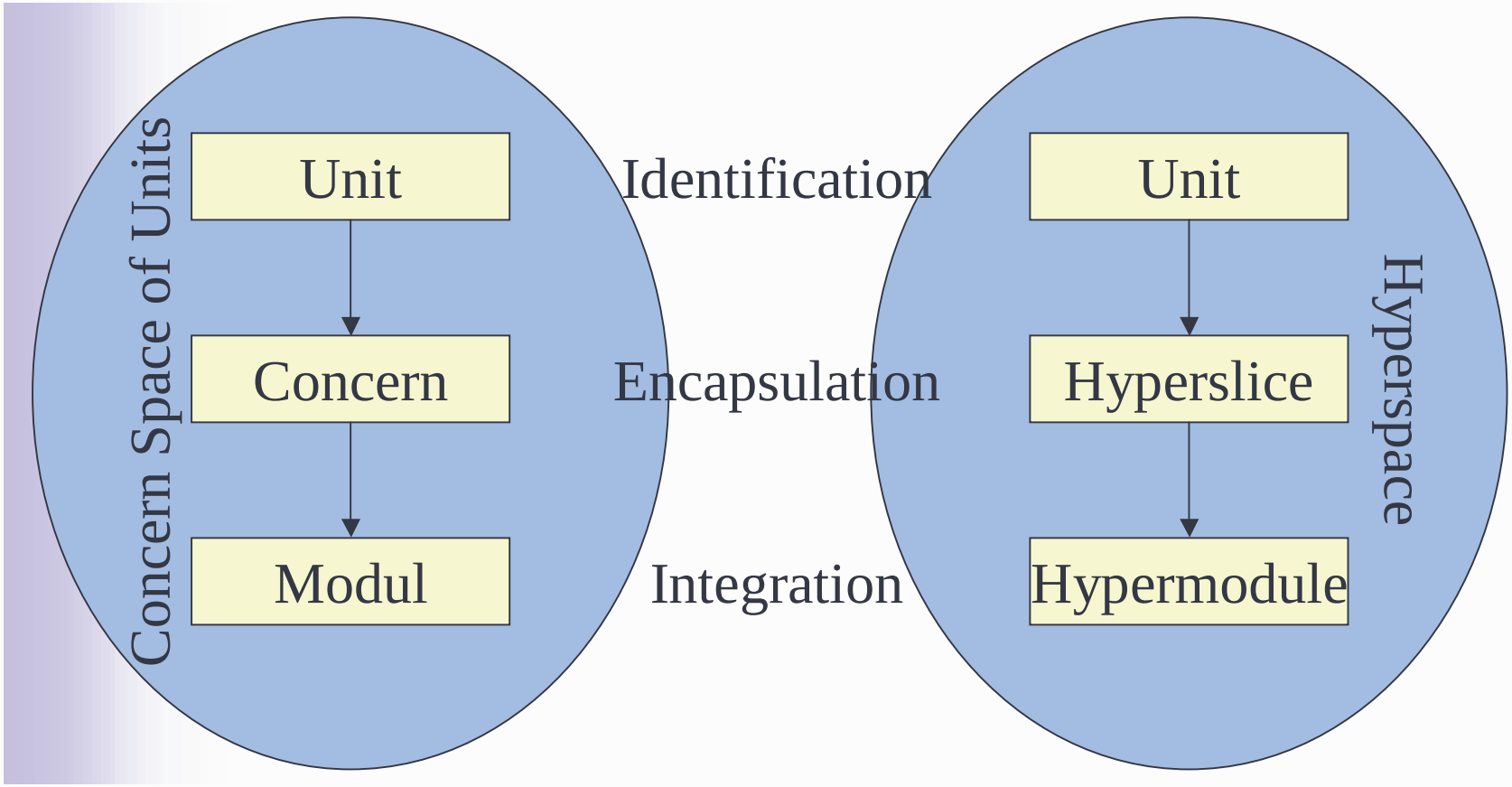
Lösungsansatz

- Multi-Dimensional Separation Of Concerns:
  - Dekomposition in mehrere Dimensionen gleichzeitig
  - Überlappungen und Interaktion zwischen diesen Dimensionen
  - *Dynamisches* Hinzufügen neuer Concerns innerhalb einer Dimension oder sogar neuer Dimensionen

Hyper/J

- IBM-Ansatz, um MDSOC zu erreichen
- Werkzeug für Java, also keine Spracherweiterung
- Erzeugt Java-Bytecode aus kompilierten Java-Klassen und Integrationsvorschriften aus Eingabedateien (Beschreibung später)

Hyper/J - Grundlagen



Hyper/J – Units und Concerns

- Primitive Einheit: Member-Variablen und –Methoden
- Zusammengesetzte Einheit: Interfaces, Klassen, Packages
- Concern Mapping: Zuordnung von Einheit(en) :

  X: dimension.concern
- Hyper/J baut daraus eine mehrdimensionale „Concern Matrix“ auf

Hyper/J – Concern Matrix

- Die „Achsen“ der Concern Matrix entsprechen der Dimension
- „Punkte“ auf den Achsen entsprechen Concerns
- Units werden den Concerns zugeordnet

Hyper/J – Concern Mapping File

- Concern Mapping (*.cm) beispielhaft:

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

Hyper/J - Hyperslices

- Ein Hyperslice kapselt einen Concern ein und ist nach außen abgeschlossen
- Hyperslices sind „deklarativ vollständig“, d. h. sie enthalten alles, was sie benötigen:
  - Alle Methoden, die aufgerufen werden
    - Evtl. nur abstrakt deklariert, ohne Implementierung
  - Alle Variablen, die benutzt werden
- Hyper/J vervollständigt Hyperslices: „declaration completion“

Hyper/J – Hyperslices (1)

- Hyperslices sind zwar unabhängige Einheiten, Beziehungen können durchaus bestehen
  - Z. B. müssen abstrakte Deklarationen mit Implementierung verknüpft werden
- Hyperslices können in Hypermodules integriert werden

Hyper/J - Hypermodules

- Zwei Funktionen:
  - Integration von Hyperslices in größere Blöcke
  - Bekanntgabe, wie die Hyperslices integriert werden sollen (z. B. überschreibt eine Methode eine andere, sollen beide ausgeführt werden, ...)
- Hypermodules sind auch wieder „deklarativ vollständig“ -> Hyperslice
- Weitere Integration möglich

Hyper/J – Hypermodule Spec.

- Hypermodule Specification File (*.hm):

```
hypermodule hypermoduleName
    hyperslices:
        dimensionName1.concernName1,
        dimensionName2.concernName2,
        ...
    relationships:
        mergeByName | nonCorrespondingMerge |
        overrideByName;
        other relationships
end hypermodule;
```

Hyper/J – Hyperspace Spec.

- Hyperspace Specification File (*.hs):

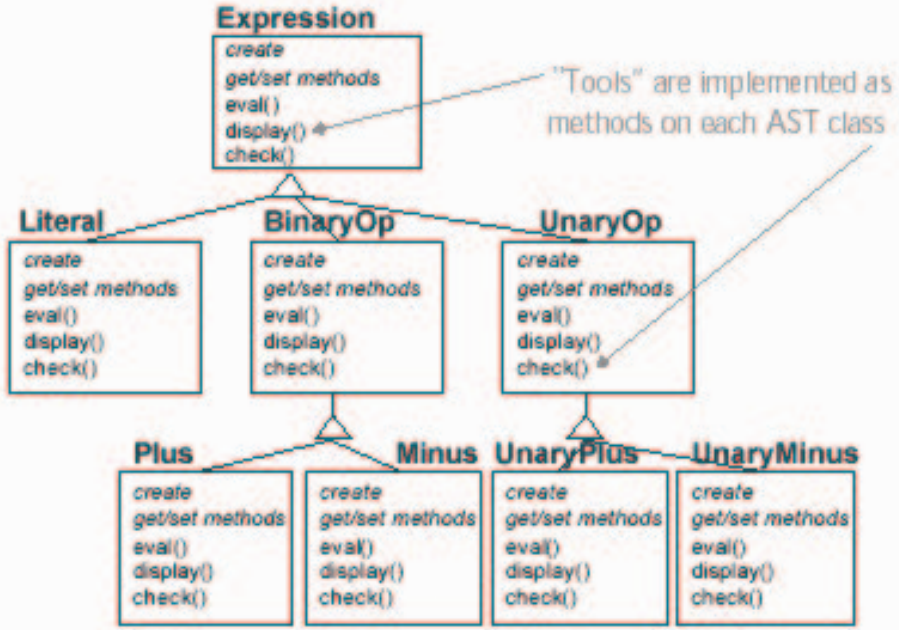
```
hyperspace hyperspaceName
classFileSpecification;
classFileSpecification;
...
```

Mit classFileSpecification (Beispiel):

```
class package1.className1,
  package1.className2;
composable class package2.* except
  package2.someClass;
uncomposable class java.lang.*, java.io.*;
```

Hyper/J - Anwendungsbeispiel

- Anwendungsszenario (aus Hyper/J Doku): Implementierung eines abstrakten Syntax-Baums



Hyper/J – Anwendungsbeispiel

- Anfängliche Funktionen:
  - Syntax prüfen
  - Ausgabe des Baums
  - Auswerten des Baums
- Demonstration:
  - Standardimplementierung ohne Hyper/J
  - Remodularisierung mit Hyper/J

Hyper/J – Anwendungsbeispiel

- Erweiterung der Funktionalität um „Style Checking“:

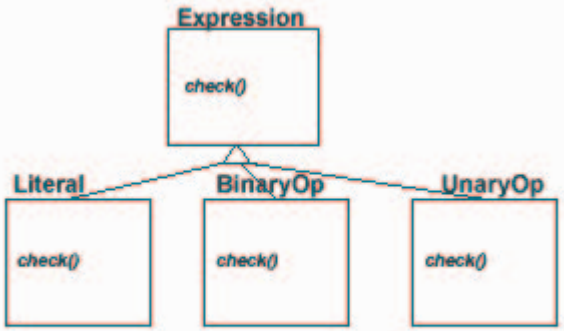


Figure 3. The Style Checking Hyperslice Package

- Neue Funktionalität wird in „Hyperslice Package“ implementiert

Hyper/J - Anwendungsbeispiel

- Weitere Erweiterung: Logging der Methodenaufrufe; Implementierung ebenfalls in Hyperslice Package

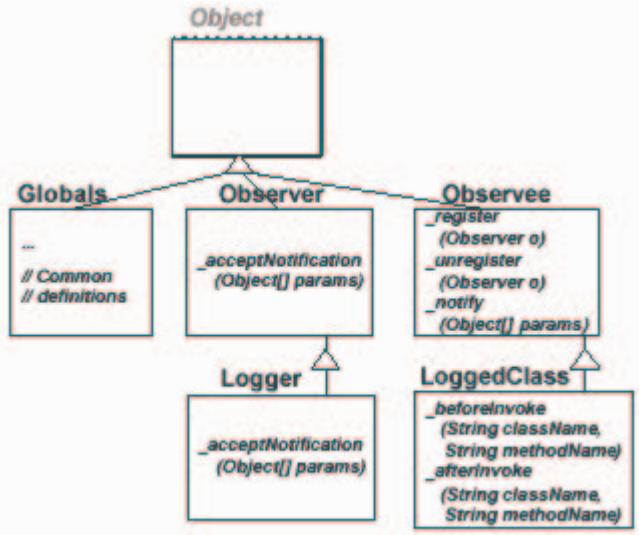


Figure 4. The Logging Hyperslice Package

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