Vorlesung Aspektorientierte Softwareentwicklung (AOSD)
– Sommersemester 2008 –

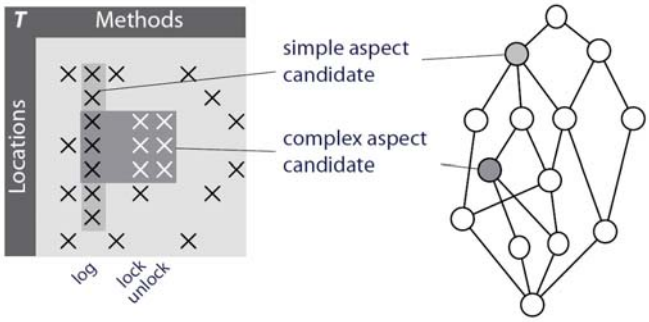
Dr. Günter Kniesel, Daniel Speicher

Übungsblatt 4

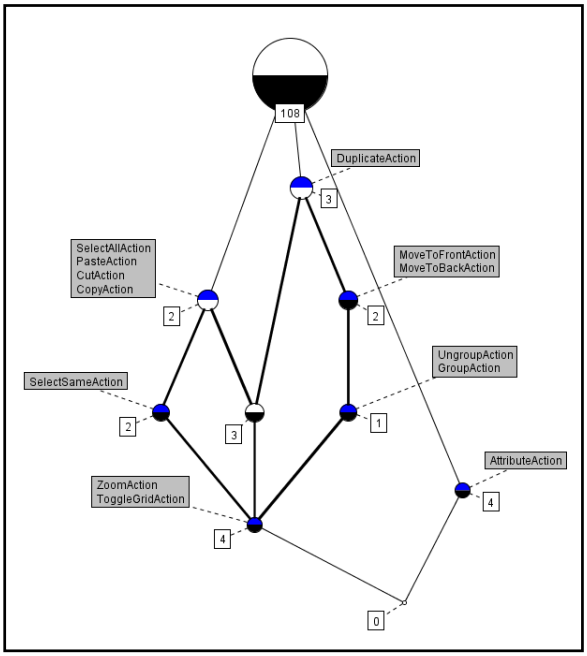
Abgabe: Dienstag 01.07.2008, 23:59:59 per SVN
(Tutorien: Freitag 04.07. + Montag 06.07.)

Aufgabe 1 (*FCA Leseübung*) (6 Punkte)

Formale Begriffsanalyse wird vielfältig zum Aspect Mining eingesetzt. Dabei sticht die *Grouped calls analysis*[1] hervor. Wie im nebenstehenden Bild[2] angedeutet bildet man aus der binären Relation „Methode ruft Methode auf" einen Begriffsverband. Die Methoden werden dabei als Aufrufende als Gegenstände und als Aufgerufene als Merkmale aufgefasst.



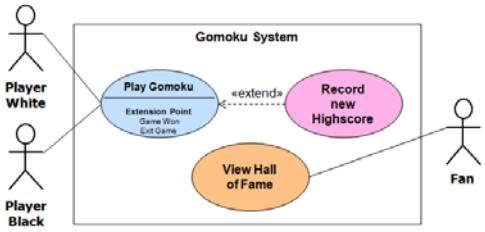
Als Beispiel sehen Sie hier einen Verband[3], beim dem als Merkmale Aufrufe der Konstruktoren von `*Action` Klassen gewählt sind. Folgende Aussagen sind wahr. Finden Sie heraus, wie man sie aus dem Diagramm ablesen kann:



(1) Es gibt genau 13 Methoden die `DuplicateAction` aufrufen.
(2) Die Aussage (1) ist repräsentiert durch einen Begriff. Dieser Begriff hat zwei direkte Unterbegriffe.
(3) Es gibt genau zwei Begriffe, die genau zwei nichttriviale Oberbegriffe haben.
(4) Der Begriff in der Mitte, bei dem keine Merkmale notiert sind, hat die Merkmale `{SelectAllAction, PasteAction, CutAction, CopyAction, DuplicateAction}`.
(5) Wenn ein Gegenstand das Merkmal `ZoomAction` hat, hat es auch das Merkmal `ToogleGridAction`.
(6) Wenn ein Gegenstand das Merkmal `SelectSameAction` und `GroupAction` hat, hat es auch die Merkmale `PasteAction` und `DuplicateAction`.

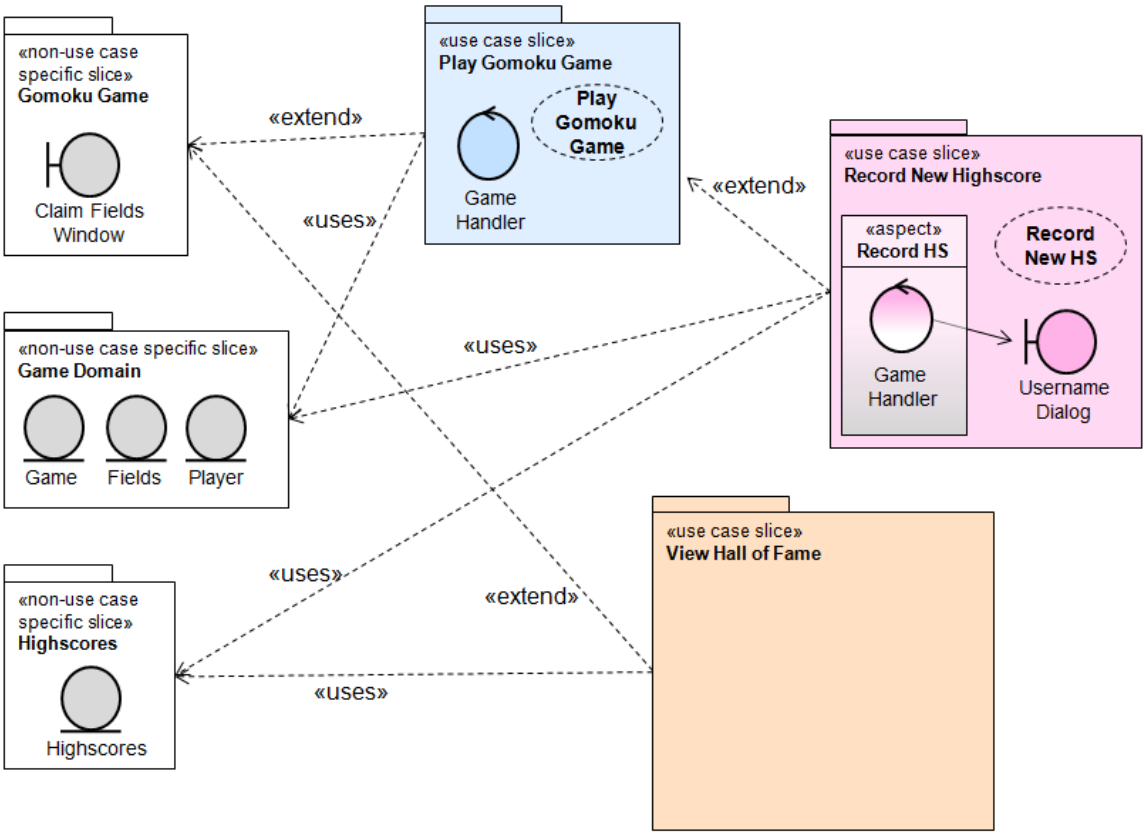
[1] M. Marin, L. Moonen, and A. van Deursen. A common framework for aspect mining based on crosscutting concern sorts. In Proceedings of the 13th Working Conference on Reverse Engineering (WCRE), 2006
[2] Breu, S., Zimmermann, T., and Lindig, C. 2006. HAM: cross-cutting concerns in Eclipse. In *Proceedings of the 2006 OOPSLA Workshop on Eclipse Technology Exchange*. eclipse '06. ACM, New York, NY, 21-24.
[3] Generiert mit FINT. Visualisiert mit ConceptExplorer. Codebasis: JHotDraw. Andere Merkmale gefiltert.

Aufgabe 2 (*Use Case Slices*, **AspectJ**) (8 Punkte)

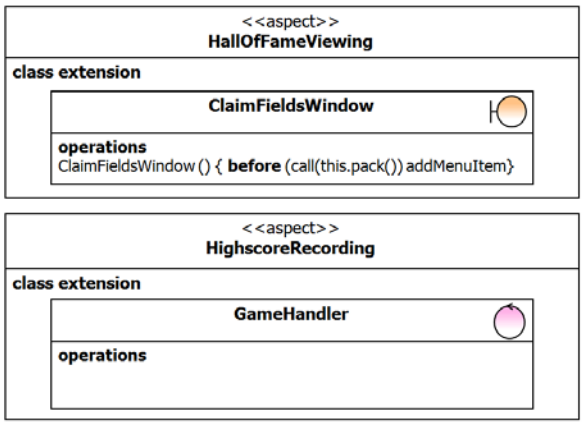


Wir haben eine Use Case-getrieben Analyse und Implementierung einer Gomoku Anwendung („Fünf in einer Reihe“) vorgenommen. Vervollständigen Sie die Lücken:

i.) Welche Analyse Objekte gehören in das „View Hall of Fame“-Use Case Slice? Ihre Lösung soll nicht detailierter aber auch nicht weniger detailiert als die Darstellung des „Record New Highscore“-Use Case Slices sein.



ii.) Geben Sie die class extensions für `HighScoreRecording` an, so wie wir sie für `HallOfFameViewing` angeben haben.



iii) Vervollständigen Sie die Implementierung von `HighScoreRecording` und `HallOfFameViewing` in `B04A2_GomokuApp`. Wir haben viel vorbereitet. Ihnen bleibt der Kern: Lediglich der eigentliche Aspektcode fehlt noch.

(Bei Interesse:) Die vorliegende Implementierung ist schon sehr weitgehend nach Use Case Slices strukturiert. Sogar die technischen Hürden einer Distribution mittels RMI sind genommen, wie Sie `B04A2_GomokuAppClient` und -`Server` entnehmen können. Allerdings lässt die Anwendungslogik in diesem Fall noch zu wünschen übrig. Mitwirkung bei der Perfektionierung dieser komplexen Anwendung sind willkommen!

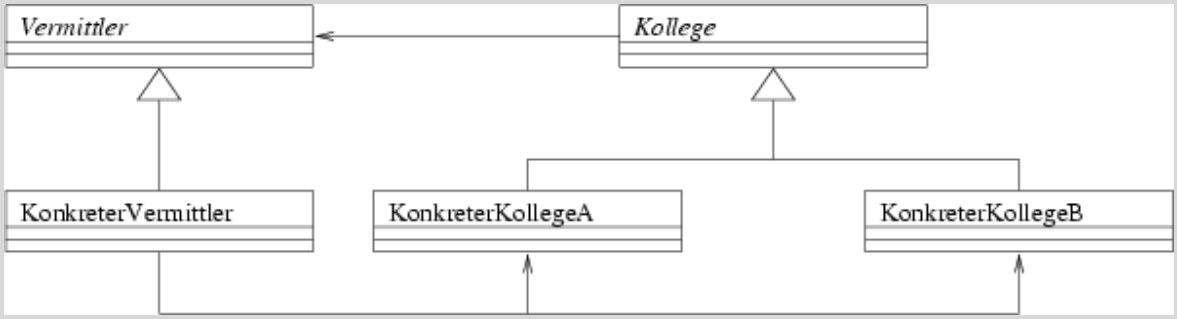
Aufgabe 3 (*Visitor Pattern*, **AspectJ**) (6 Punkte)

i.) In der Vorlesung wurde behauptet, dass mit dem OO-Visitor:
"Das Hinzufügen neuer Operationen einfach wird, das Hinzufügen neuer Objektklassen aber sehr aufwändig." Inwiefern ist das der Fall? Kann man das im Abstractness-Stability-Diagramm erkennen?

ii.) Implementieren sie eine aspektorientierte Version eines Visitors. Dieser soll die Baumstruktur in Package visitor „besuchen" und eine HTML Ausgabe erzeugen. Die Zahlen sollen dabei **fett** dargestellt werden (123) und die Operatoren *kursiv* (*).
Sourcecode: B04A3_SimpleVisitor

Aufgabe 4 (*Mediator Patter - Abstractness-Stability-Diagramm*) (8 Punkte)

Das Mediator (Vermittler-) Pattern



Das Mediatorpattern ist ein sogenanntes Behavioral pattern bei dem die Interaktion zwischen mehreren Kollegenobjekten in einem Vermittlerobjekt kapselt werden. Dadurch vermeidet man, dass die einzelnen Kollegen die Interaktion untereinender definieren müssen. In einem Vermittlerinterface werden die zur Kommunikation zwischen den Kollegen notwendigen Methoden definiert. In dem konkreten Vermittler werden dann die Beziehungen zwischen den einzelnen Kollegen definiert. Die Kollegen müssen nicht mehr alle Kollegen kennen sondern kommunizieren nur noch mit dem Vermittler. Der Vermittler erhält und vermittelt also die Anfragen der einzelnen Objekte.

i.) Stellen Sie das Mediator Pattern in einem Abstractness-Stability-Diagramm dar. Ist es möglich es so einzuzeichnen, dass alle Abhängigkeiten „gut" sind? Wenn ja, unter welchen Annahmen an Abstraktheit und Stabilität der einzelnen Klassen?

ii.) Wie sieht eine entsprechende aspektorientierte Variante aus? Zeichnen sie das Diagramm.

iii.) Nennen sie mindestens zwei Beispiele bei denen das OO-Mediatorpattern geeignet oder ungeeignet ist. Warum ist es in den von Ihnen genannten Fällen geeignet oder ungeeignet? Wäre eine aspektorientierte Variante in Ihren Beispielen besser?

Aufgabe 5 (*Annotationen*) (7 Punkte)

Durch Java 5 Annotationen kann man Deklarationen mit zusätzlichen semantischen Meta-Informationen anreichern. Diese implizieren oft Verträge. Mit Aspekten ist es nun möglich, einige dieser Verträge statisch oder dynamisch zu überprüfen. Bei der Nicht-Einhaltung der Verträge wäre eine Compiler-Warnung oder sogar ein Compiler-Fehler wünschenswert.

Wir verwenden noch einmal unsere Gomoku-Anwendung. Das Programm soll die folgenden Verträge erfüllen:

1) Methoden des Controllers, die zum Spielfluss gehören, dürfen nur von Methoden außerhalb des Controllers oder von anderen Spielfluss-Methoden aufgerufen werden.

2) Bestimmte Methoden einiger Klassen sollen ausdrücklich nur von Methoden anderer Klassen und nicht durch Methoden derselben Klasse aufgerufen werden.

Identifizieren Sie die jeweiligen Methoden (der zweite Fall ist nicht sehr eindeutig, aber im ersten sollten sich die Methoden klar identifizieren lassen) und annotieren Sie sie mit der aus Ihrer Sicht sinnvollsten Annotation aus der folgenden Liste (die beiden Annotationen müssen implementiert werden):

```
@Boundary, @Update, @NoInternalCallsAllowed, @GameFlow,
@TakeField, @ChangePlayer, @Controller, @ModelClass,
@ClassBoundary, @BoundaryClass
```

Schreiben Sie Pointcuts und darauf aufbauende Warnungsdeklarationen, die die oben genannten Verträge sichern. Wenn sich der Vertrag nicht statisch prüfen lässt, schreiben Sie Laufzeitchecks.