Zweite Hausübung Softwaretechnik III

Kontext	Hausübung 2 SWT III WS03/04
Teilprojekt	Erfahrungsbericht
Autor	Ansorg Matthias
	Dibaj Babak
Team	**Ansorg Matthias**
	Dibaj Babak
Erstelldatum	12.12.03-21.12.03
Version	0.1
Ablage-Name	Diskussion-0.1.sxw

1.Einführung

Im Abschnitt »Anmerkungen zum Verständnis« stehen einige Dinge, die wir zum eigenen Verständnis zusammengetragen, zusätzlich zu dem was Fowler in seinem Artikel schreibt. Im Abschnitt »Diskussion« geht es dann um Erfahrungen bei eigenen Erweiterungen der kleinen Anwendung aus Fowlers Artikel, natürlich hauptsächlich bezogen auf den Wert von Mustern und den Umgang damit.

2.Anmerkungen zum Verständnis

Fowler verwendet seine Schlüsselabstraktionen »Phenomenon« und »PhenomenonType« für ganz unterschiedliche Zwecke, ohne die Gemeinsamkeit, die das rechtfertigt, klar darzustellen. Etwa so:

- PhenomenonType ist eine Größe:
  - In der Assoziation mit Measurement die Größe eines quantitativen Merkmals. Beispiele: heart rate, breathing rate.
  - In der Assoziation mit CategoryObservation über Phenomenon die Größe eines qualitativen Merkmals. Beispiele: blood group, level of consciousness.
- Phenomenon ist eine kategorisierende Eigenschaft von PhenomenonType:
  - In der Assoziation mit Measurement über PhenomenonType die kategorisierende Eigenschaft einer Messung. Beispiele: slow, nomal, fast.
  - In der Assoziation mit CategoryObservation die kategorisierende Eigenschaft, die durch die Beobachtung festgestellt wurde. Beispiele: A, B, AB oder 0 bei PhenomenonType blood group.

3.Diskussion

Flexibilität der Trennung in Präsentation und Applikationsfassade

Das Muster »application facade« erwies sich als wirklich hilfreich. Unsere erste lauffähige Version hatte nur die von Fowler beschriebene Funktionalität, stellte also auch nur den kleinen Dialog aus Fowlers Artikel bereit.
Diesen Dialog durch einen umfangreicheren zu ersetzen und nebenbei von AWT zu Swing als GUI framework überzugehen, erwies sich dank »application facade« als sehr einfach. Denn die Fassade hängt nicht vom GUI framework ab, muss also nicht angepasst werden. Die Dialogklasse `VitalsWindow` enthält keine weiter benötigte Funktionalität, sondern nur die Oberfläche, und kann darum einfach gelöscht werden. Der vom anderen Dialog wiederverwendbare Teil ist ja in der Fassade enthalten, die einfach vom neuen Dialog `RecordPatientWindow` benutzt wird.

Bindung der Fassade an einen Dialog

Fowler implementiert eine Fassade und einen Dialog. Ob aber eine Fassade die Schnittstelle zum Domänenmodell für genau einen Dialog bereitzustellen hat, bleibt dabei natürlich offen. Fowler äußert sich in seinem Artikel auch nicht klar darüber:

> »We will do this by creating an application facade that converts from Figure 2 to a form ready for Figure 3 and a presentation object that gives the display in Figure 3.« [AppFacades], S.4.

> »The sample window in Figure 3 is based on your vital signs, so I will call it a VitalsFacade.« [AppFacades], S.7.

Wir hatten zwei verschiedene Usecases entworfen: »Patientendaten erfassen« und »Patientendaten anzeigen«. Sie sollten durch zwei verschiedene Dialoge (oder zwei Tabs eines Dialogs) bedient werden. Es scheint wenig sinnvoll, hier zwei Fassaden zu schreiben; Patientendaten zu erfassen verlangt ja dieselbe Funktionalität von der Fassade wie sie anzuzeigen, zusätzlich nur Schreibzugriff. Also haben wir nur eine Fassade `appfacade.PatientDataFacade` geschrieben. Fassaden sollten überhaupt nicht an einzelne Dialoge gebunden werden, sondern thematisch kohärente Funktionen als Schnittstelle zum Domänenmodell bereitstellen. Damit kann es nötig sein, dass ein Dialog mehrere Fassaden verwendet. Trotzdem ist diese thematische statt strukturelle Gliederung natürlich vorteilhaft, denn eine bloße Umstrukturierung der GUI hat keine Auswirkungen auf die Fassaden.

Bindung der Fassade an ein Subject

Fowlers Ansatz ist, dass jede Fassade auf einem Subject operiert:

> »When I construct a facade I give it a subject: a reference into the domain model. In this case the subject is the patient.« [AppFacades], S.7.

Mit diesem Ansatz ist es unmöglich, eine Fassadenobjekt ohne Subject zu erzeugen: der einzige Konstruktor von `VitalsFacade` verlangt ein Patient-Objekt als Subject. Das Subject einer Fassade wurde mit der Angabe im Konstruktor auch ein für allemal festgelegt.
Diese Konstruktion der Fassade, nicht das Konzept der Fassade an sich, erwies sich als zu unflexibel als wir versuchten, einen Dialog zu implementieren, in dem der Benutzer einen Patienten auswählen und dann seine Daten ansehen oder bearbeiten kann. Auch ist das Konzept eines Subjects unverträglich mit der Aufgabe einer Fassade: eine Fassade als Schnittstelle zum Domänenmodell ermöglicht es, Domänenobjekte aus entsprechenden Daten anzulegen. Nun verlangt die Fassade aber bei Instanziierung ein Domänenobjekt als Subject, das zu seiner Erzeugung die Instanz einer Fassade benötigt! Natürlich kann man dies mit mehreren Fassaden und unterschiedlichen Typen von Subjects lösen, etwas eleganter ist es aber, wenn eine Fassade kein Subject braucht: es kann fehlen, von ihr aus entsprechenden Benutzerdaten erzeugt werden, und es kann dynamisch geändert werden. Diesen Weg, bei dem Subject unter den Attributen der Fassade keine ausgezeichnete Stellung mehr hat, haben wir verfolgt.

Refactoring der Fassadenattribute

In »Refactoring the Level of Consciousness« und »Refactoring the Heart Rate« [AppFacades], S.18-22, hat Fowler schon einige Refactorings an der Klasse `VitalsFacade` vorgenommen, es entstanden einige Satelliten. Wir erweiterten `VitalsFacade` zu unserer Fassade `PatientDataFacade` und führten weitere Refactorings durch. Fowler hatte schließlich alle `Measurements` wie Puls- und Atemfrequenz in der Fassade durch eine einheitliche Klasse `MeasurementAttribute` dargestellt.
Auch bei `Category Observations` ist eine solche Vereinheitlichung möglich. Fowler hatte bisher die `PhenomenonType`-spezifischen Attribute `LocAttribute` und `ShockAttribute` verwendet. Als wir für die erweiterte GUI in der Fassade etwas wie `BloodGroupAttribute` benötigten, wurde die Gemeinsamkeit offensichtlich. Es entstand die Klasse `CategoryAttribute`, und nun können der »level of consciousness«, der Schockzustand und die Blutgruppe einfach durch Instanzen dieser Klasse dargestellt werden. Sie enthält dazu einfach den zugehörigen `PhenomenonType` und das zuletzt beobachtete `Phenomenon` als Attribute.

Flexibilität bei Aufnahme neuer Phenomena

Hier zeigt sich, wie leistungsfähig die im Domänenmodell verwandten Muster *Quantity*, *Measurement*, *Observation*, *Range* und *Phenomenon with Range* (vgl. [AppFacades], S.3) sind. Die für jeden Patienten abgespeicherten Daten sollten nun auch die Blutgruppe nach dem AB0-System und die Körpertemperatur enthalten. Aufgrund der eben erwähnten Muster musste für diese

Erweiterung keine einzige Klasse definiert oder geändert werden. Es reicht, Objekte von existierenden Klassen zu erzeugen:

```
Unit degree = new Unit("degree Celsius").persist();

PhenomenonType temperature =
  new PhenomenonType("temperature").persist();

new Phenomenon("low", temperature).range(
  new QuantityRange(
    null,
    false,
    new Quantity(36.5, degree),
    false
  )
);
new Phenomenon("normal", temperature).range(
  new QuantityRange(
    new Quantity(36.5, degree),
    true,
    new Quantity(37.5, degree),
    true
  )
);
new Phenomenon("high", temperature).range(
  new QuantityRange(new Quantity(37.5, degree), false, null,
false)
);

PhenomenonType bloodGroup = new PhenomenonType("blood group").per-
sist();
new Phenomenon("0", bloodGroup);
new Phenomenon("A", bloodGroup);
new Phenomenon("B", bloodGroup);
new Phenomenon("AB", bloodGroup);
```

Es war etwas gewöhnungsbedürftig, mit den Mustern *Phenomenon with Range* und *Measurement* zu arbeiten. Denn was umgangssprachlich als Typ identifiziert wird (»der Typ der Messung ist eine Blutgruppenbestimmung«), ist hier ein Objekt. Natürlich ist das nicht anders machbar, wenn »Typen von Messungen« zur Laufzeit erzeugt werden sollen, und die so gewonnene Flexibilität ermöglichte ja tatsächlich die o.a. sehr einfache Implementierung von Blutgruppe und Körpertemperaturwert.

Fowlers Testansatz und JUnit

Die Fassade `PatientDataFacade` und `PhenomenonType` als zentrale Klasse des Domänenmodells wurden zusammen mit allen assoziierten Klassen mit JUnit-Testfällen versehen.
Interessant dabei war, wie einfach das Test-Framework aus Fowlers Artikel per Refactoring in den JUnit-Testfall `PatientDataFacadeTest` überführt werden konnte. Denn die Klasse `Tester` aus Fowlers Artikel verwendet dieselbe Gliederung eines Testfalls wie JUnit: zuerst die zu testenden Objekte erzeugen und mit geeigneten Werten versehen (sog. »fixture«), dann eine zusicherungszentrierte Testmethode.

Verwendung der Facade für die GUI

Für die Erstellung der GUI wurde Swing benutzt. Eine der Vorteile des Patterns Application Facade sind, dass der Implementierer der GUI keinerlei Kenntnisse über die Domäne benötigt. Diesen Vorteil konnten wir in unserer Gruppenarbeit ausnutzen, da wir unabhängig voneinander GUI und Domäne implementieret haben. Die Facade wurde dann schließlich für die Ereignisbehandlung der GUI eingesetzt und vereinfachte die Arbeit enorm.

Literaturverzeichnis

AppFacades: Martin Fowler, Application Facades, 1998