Fragen, Antworten und Kommentare zur aktuellen Vorlesung

Zum Blatt 6 befinden sich Lösungsskizzen in ILIAS (Objektorientierte Analyse und Design (Vorlesung) _MI im SoSe 2026), https://lms.hs-osnabrueck.de/ilias.php?baseClass=ilrepositorygui&cmdNode=zf:o6&cmdClass=ilObjFileGUI&cmd=sendfile&ref_id=779666. Beachten Sie, dass es sich nur um sinnvolle Lösungsvorschläge handelt. Wir sind längst in dem Bereich, in dem es oft nicht die eine optimale Lösung gibt. Die Fähigkeit Vor- und Nachteile von Lösungen diskutieren zu können, ist ein zentrales Ziel dieser Veranstaltung

Frage: Soll ich bei Assoziationen immer beide Multiplizitäten angeben, auch wenn durch die Pfeilspitze klar ist, dass die Objekte sich in der anderen Klasse nicht kennen?

Antwort: Die Antwort ist etwas komplizierter. Wird konsequent die Analyse Schritt für Schritt durchgeführt, erstellen Sie am Anfang ein Analyseklassenmodell, bei dem es wichtig ist, dass alle Anforderungen in das Modell aufgenommen werden. Bei diesem Analysemodell werden beide Multiplizitäten angegeben und es wird meist auf die Angabe von Pfeilspitzen verzichtet, da es sich hierbei um eine Implementierungsentscheidung handelt. Wir zeichnen meist direkt das nachfolgende Designmodell, was zur Implementierung genutzt wird. Hier muss für jede Assoziation entschieden werden, in welche Richtung sie verläuft oder ob sie bidirektional ist/bleibt. Hier reicht es aus, die Multiplizität nur bei der Pfeilspitze anzugeben, da dies für die Implementierung benötigt wird. Die andere Multiplizität kann aber stehen bleiben, sie ist dann ein Constraint (Randbedingung), die z. B. später durch Tests überprüft werden kann.

Frage: Wie baue ich die Validierung ein?

Antwort: Dies kann von verwendeten Frameworks abhängen, die teilweise ein Validierungskonzept mitbringen. Die zentrale Regel lautet ungültige Objekte immer soweit wie möglich weg von der Persistierung zu halten. Schreiben Sie einen WebService wird die annehmende Methode die Parameter, soweit ohne die Nutzung weiterer Objekte möglich, auf Plausibilität prüfen und erst dann ein Controller-Objekt aufrufen. Dieses sollte dann, eventuell in Zusammenarbeit mit anderen Controllern, dann die Validierung vollständig durchführen. So ist meist eine konkrete Validierung in einer Entitätsklasse, z. B. in set-Methoden, vermeidbar. Oft werden Validation-Frameworks eingesetzt, dabei werden Validierungsregeln dann an Objektvariablen annotiert, die dann von zentral nutzbaren Validierungsklassen des Frameworks überprüft werden. Ein Beispiel ist das Jakarta Bean Validation-Framework (https://beanvalidation.org/). Generell ist der Validierungsansatz einheitlich in den Designregeln eines Projekts vor Start der Implementierung festzulegen.

Frage: Mir ist der View-Begriff beim MVC im Zusammenhang mit dem Beispiel mit den Plus- und Minus-Tasten unklar.

Zunächst ist wichtig, dass es sich bei MVC um ein Konzept handelt, eine oder mehrere Klassen übernehmen die Aufgaben des Views, des Controllers und des Models. Bei dem graphischen Beispiel ist es irritierend, dass es eigentlich zwei Modelle gibt. Einmal der verwaltete Wert im XModel-Objekt um das es eigentlich geht und ein Modell für die graphische Oberfläche (genauer die Knöpfe, z. B. Buttonfarbe, Buttonposition, Buttonzustand, Buttongroesse, …). Der Button hat damit seine eigene View, die diesen Button auf den Bildschirm malt. Das ist aber bei der Aufgabe weniger vom Interesse. Hier ist das GUI etwas vereinfacht die View zum XModel. Views stellen Modelle dar, nehmen Befehle an, können eventuell eine (Teil-)validierungen vornehmen und den Controller benachrichtigen. Eventuell wird die Vorstellung einfacher, wenn das GUI durch eine Konsolen-Ein-und-Ausgabe ersetzt wird. Dann stellt der View das Model dar und ermöglicht den Zugang zu Controller-Aktionen. Wenn Sie eine enge Verwandtschaft zur Boundary bei Boundary-Control-Entity spüren, passt das, der View zeigt zusätzlich etwas in irgendeiner Form an und wird meist bei GUIs genutzt. Als Hintergrund vielleicht interessant: https://stackoverflow.com/questions/32912341/entity-control-boundary-ecb-vs-model-view-controller-mvc

Frage: Welche Informationen stelle ich im Sequenzdiagramm dar?

Antwort: Die für die Beschreibung relevanten. Genauer, stehen in der Kopfzeile des Diagramms alle wichtigen Objekte, die nicht im Laufe des Diagramms erzeugt werden. Woher die Objekte kommen ist damit für dieses Diagramm unwichtig. Dann zeichnen Sie alle Methoden detailliert ein, die zum Verständnis der dokumentierten Funktionalität wichtig sind. Gleiches gilt für Rückgabepfeile, auf die verzichtet werden kann, wenn es void-Methoden sind oder die Information irrelevant sind. Es muss aber nicht darauf verzichtet werden, wenn der bekannte rote Faden durch das Diagramm erkennbar sein soll. Generell darf nichts dazu erfunden werden, wenn ein detaillierter Ablauf m1;m2; m_n gegeben ist, sind alle Folgen wie m2;m4;m5;m8 generell denkbar, solange die Reihenfolge eingehalten wird. Generell ist es natürlich auch möglich UML-Kommentarkästen in das Diagramm zu setzen.

Frage: Können wir zur Erzeugung von Klassendiagrammen nicht einfach die Generierungsmöglichkeit von UMLet oder einem anderen UML-Tool nutzen?

Antwort: Da Sie eigentlich erst das Klassendiagramm entwickeln und dann den Code sollte das für die OOAD-Hausarbeit kaum möglich sein. Generell können natürlich Generatoren genutzt werden, dann sind sie aber verantwortlich dafür, dass die korrekte UML-Syntax eingehalten wird. UMLet versagt da leider in vielen Punkten, ähnliches gilt auch für die mir bekannten kommerziellen Werkzeuge.

Nachfrage: Ok, aber das sind ja nur Details, auf die bei Zeitmangel verzichtet werden kann?

Antwort: Nein. Der Sinn der UML ist, dass jede Person die Diagramme exakt gleich interpretiert bezüglich der dargestellten Informationen. Da abhängig vom Detaillierungsgrad Informationen weggelassen werden können, kann es allerdings immer noch Interpretationsmöglichkeiten geben. Da sie generell lernen sollen die Wünsche auftraggebender Personen zu respektieren und Ihre Freiheitsgrade in den Umsetzungsmöglichkeiten zu sehen, könnten nur solche Personen Anforderungen als unwichtige Details klassifizieren.