Seminarie 1

Adalet Adiljan

Objekt-Oriented Design, IV1350

KTH

Kista, Stockholm adalat@kth.se

Innehåll

1. Introduktion

Den här rapporten handlar om det första seminariet i en kurs, där vi lärde oss att tolka och förtydliga en given specifikation genom att skapa två olika typer av diagram: en domänmodell (DM) och ett systemsekvensdiagram (SSD). DM visar relationerna mellan olika enheter inom en organisations affärsdomän och används för att designa system som är anpassade till det. SSD:n visar samspelet mellan aktörer och systemkomponenter i form av händelser eller meddelanden och används för att tydliggöra krav och kommunikation mellan utvecklare och användare.

I deluppgift 1 skapade vi en DM för ett detaljhandelssystem baserat på beskrivningen "Process Sale" med hänsyn till huvudflödet och alternativt flöde i systemet. I deluppgift 2 skapade vi en SSD utifrån nämnda krav och specifikationer. Rapporten kommer att förklara hur SSD:n designades i metodavsnittet, visa resultaten i resultatavsnittet och utvärdera SSD:n med hjälp av utvärderingskriterier i diskussionsavsnittet.

2. *Metod*

Beskrivning av domänmodellen

I detta avsnitt av rapporten ska jag diskutera kring hur vi använde mig av två olika sätt och metoder för att identifiera samtliga klasser till domänmodellen, vilket var relevant för deluppgift 1 (task 1). Vid utformningen uppmärksammades vi även till att vara försiktiga att inte använda oss av programmeringsrelaterade begrepp då det finns en risk att göra domänen för programmatisk. De två metoderna så kallade substantivsidentifikation (*noun identification)* och *category list.* Substantivsidentifikationen går ut på att se om man kan sätta dit ett ”the” framför ett ord, i syfte för att se om den kan användas som ett klassnamn. På så sätt har man identifierat en potentiell klasskandidat, då man är beroende av att inneha en klass / klasser som kan ha tillstånd, beteenden och egenskaper. När det kommer till val av klassnamn har man även lagt betoning i att välja lämpliga namn utifrån de specifika krav som ställs för systemet. En ytterligare metod för att hitta klasskandidater kallas category list, vilket är en tabell som innehåller en lista över kategorier som varje potentiell klasskandidat kan tilldelas. Metoden hjälper oss att utvidga perspektivet för att hitta de icke-självklara klasser via noun identification genom att identifiera substantiv i beskrivningen av problemområdet.

När vi väl har hittat alla klasser som kunde tilläggas i vårt DM kunde vi sedan förenkla vårt DM ytterligare genom att bort klasser som lika gärna kunde vara ett attribut till en annan klass. Detta kunde göras genom att hitta särskilda attribut och associationer för klasser. För att kunna identifiera associationer mellan klasser använde vi oss av verb i problemområdet och se hur begreppen och entiteterna är relaterade till varandra. Associationerna är en fördel för att förtydliga DM ytterligare tills vi inte längre kunde hitta fler associationer tills det blev förvirrande. När det kom till attribut tittade vi på egenskaper som beskriver entiteten och vilka som är viktiga för systemet, där ett exempel på en klass med attribut (enligt vår domänmodell i resultatsektionen) kan vara ”item” klass med *quantity* och *price* som attribut.

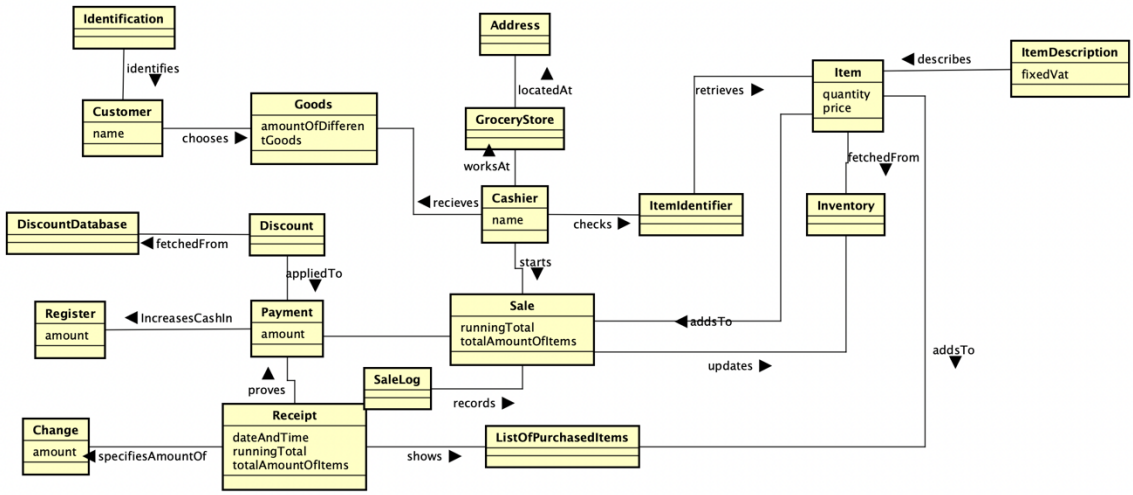
Beskrivning av systemsekvens diagrammet (Explain how you worked when developing your SSD)

Under vår utformning av vårt SSD tog vi hänsyn till vad SSD innebär och hur den ska tillämpas. Under tiden har vi även lärt oss av viktiga begrepp som kan vara bra att känna till. SSD i UML visar interaktionen mellan användaren och systemet, där användaren i vårt fall är kassörskan och programmet. Interaktionen kan bestå av ett meddelande som beskriver ett verb, vad som görs från användaren till systemet eller vice versa. När vi skulle påbörja det sekventiella händelseförloppet behövde vi även se till att följa interaktionen mellan aktören och systemen(a) som beskrevs i huvud- och alternativa flödet steg för steg tills vi uppnår det så kallade målet, vilket motsvarar syftet av interaktionen.
I vår basic- och alternative flow har vi lagt märke till att systemet har en del interna programmatiska interaktioner, vilka har lett oss till att skapa flera **externa** klasser i vårt SSD. De externa klasserna har identifierats i händelseförloppet med kännedomen att vi är endast intresserade av att veta hur externa aktörer (system, enheter eller användare) interagerar med system som designas. Externa klasser till systemet enligt bilaga 2 i resultat-avsnittet

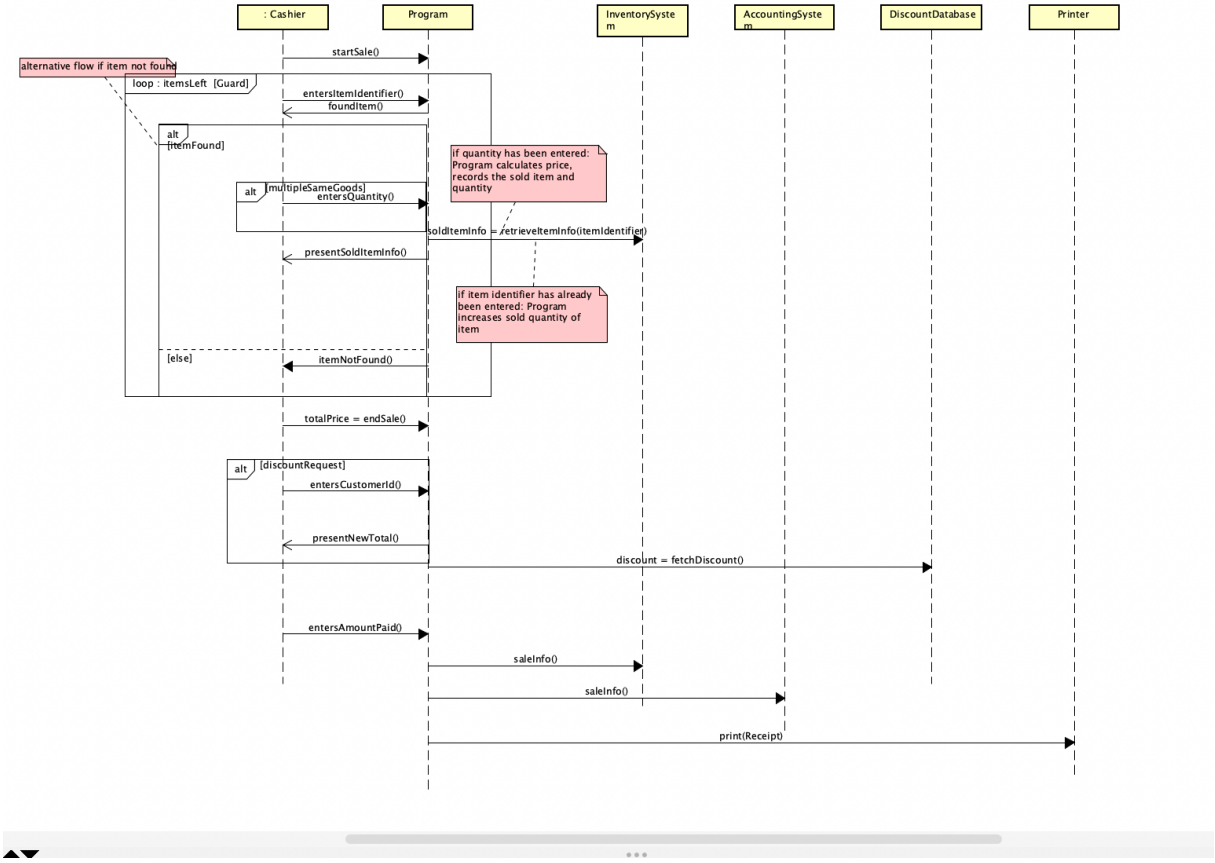
3. Resultat

I detta avsnitt av rapporten ser ni vår resultatet av vår modell för task 1 respektive task 2 för basic & alternative flow. Den första bilagan motsvarar 19 klasser och 9 klasser med attributer.

Bilaga 1: Resultatet av vår domänmodell för task 1 för basic & alternative flow.



Bilaga 2: Resultatet av vår systemsekvensdiagram enligt task 2. Vårt slutliga SSD användargränssnitt bestående av länkar mellan kassörskan och systemet, där våra externa system blev totalt life-lines.



4.Discussion

I detta avsnitt av rapporten använder vi oss av de tillämpade kriterierna från utvärderingsdokumentet som anvisats på uppgiftsinstruktionen. Den första utvärderingsfrågan handlar om vårt slutliga DM råkar vara mer programmatisk än vad den ska vara. Till denna fråga kan jag besvara utifrån ett perspektiv med utan någon tidigare programmeringserfarenhet. När man för första gången ska lära sig att konstruera ett DM i UML kan det vara vanligt förekommande att tänka utifrån ett programmeringstänkande sätt, speciellt när man har en redan tidigare erfarenhet inom programmering. Det blir som att lära sig att förbise sina erfarenheter i kodning i synnerhet man ska hitta tillämpa olika metoder för att finna attribut och associationer för klasser eller dra en linje mellan olika klasser för att visa att det finns en relation mellan dem. Då jag är relativt ny i programmering kan jag utifrån mitt perspektiv besvara på frågan att vårt DM inte är särskilt programmatisk utan kan tänkas begripas relativt väl av en person utan någon särskild programmeringskunskap. I allra första början av utformningen hade dock andra i min grupp svårt att inte få diagrammet att se mindre system-aktigt ut, vilket till slut löste sig när vi tog exempel på DM som inte är godkända från boken.

Den andra frågan lyder om vi kan förstå den DM, och om den är en korrekt representation för problemdomänen. Någonting som jag har lagt märke till vid utformningen var att verkligheten kan variera en del, och det finns ingen stor sannolikhet som vi kan säga att domän-modellen kommer att motsvara helt den ideala domänmodellen. Detta, då man fick veta att man vid alternative flow kunde inkludera discount-delen i vilket steg som helst utav 9a, 10a eller 11a. Problemet med detta är att det här kan uppstå spider-web i en klass som redan hanterar indata från andra klasser redan (trafikflöde av indata) vilket leder till att indata kan registreras in allt för sent för att kunna hinna gå till en annan klass. Detta kan påverka ordningen på de indata och utdata, vilket i sin tur inte kommer motsvara den exakta ordningen utav flödet som sker i verkligheten.

Enligt min uppfattning och upplevelser från vår gruppdiskussion tycker jag att vårt DM har tillräckligt många klasser som behövs för att vi ska kunna följa både basic och alternative flow. Dock kan det finnas en risk eller möjlighet att vårt program att hamna i *spider-in-the-web* sitiuation. Spinder in the web betyder att en klass innehåller betydligt mer länkar än de andra klasser. Vi kan försöka förbättra vårt program med hjälp av att separera stora klasser med mindre klasser, och på så sätt kan programmet bli jämnt fördelat. Således minskar risken att hamna i spider-in-web. Trots det tycker jag att vi efter diskussion som grupp har dragit slutsatsen att majoriteten av vår val av klasser och attribut har blivit förenklad tillräckligt. Vi har även oss av bokens metod som nämnde att vi bör stoppa när vi har hunnit förenkla tillräckligt för att bli förvirrade. Vi såg även i hänsyn till att attribut måste vara en string, boolean och number datatyp. Dock tror vi att det finns tillräckligt många associationer, där vi har lyckats förstå vad som sker i systemet i ett så förenklat sätt som möjligt. En sak som är nödvändig att veta är att vi försöker minska att använda samma association i flera olika ställen. Vi försöker minska att använda associationer som ”have” och ”do”. När det kommer till namnkonventionen tycker jag att vi har följt den korrekt, vilket innebär att vi ska ha en stor bokstav för själva namnet för klassen och ”life-lines” dvs. små bokstäver som då gäller för associationer och attribut. Något vi har haft problem med när det kommer till SSD är

reply-messages, dvs. returmeddelanden kunde inte jämföras med korrekt signal. Vi får heldragna linjer i stället dvs. messenger-linjer vilket inte motsvarar den exakta pilen vi behöver använda. Därmed kan detta påverka hur läsaren förstår diagrammet eller hur systemutvecklare vårt diagram. Frågan kring om vi har valt rätt objekt inkluderat i vår SSD är en klurig fråga eftersom den innehåller inget objekt i SSD men i klassen kan det innehållas objekt som t.ex. en kassör som då är en människa, vilket då indirekt kan motsvara ett objekt. Någonting som har påpekats tidigare, så har vi haft problem med retursignalerna, men vi tänkte på ett korrekt sätt att vi får ett returvärde när det behövs. Däremot har det funnit stunder som vi har haft svårt att förstå i vilka händelseförlopp som vi behövde använda oss utav retursignaler för varje loop-iteration eller efter att vi har loopat igenom en hel iteration.