Datavetenskap

Christoffer Karlsson

Jonas Östlund

Utveckling av ett grafiskt användargränssnitt för pekskärmar

Development of a graphical user interface for touch screens

Examensarbete 15 hp

IT-design: Programvarudesign/Dataingenjörsprogrammet

Datum/Termin: 10-06-09

Handledare: Katarina Asplund

Examinator: Martin Blom

Ev. löpnummer: C2010:07

Karlstads universitet 651 88 Karlstad

Tfn 054-700 10 00 Fax 054-700 14 60

Information@kau.se www.kau.se

Utveckling av ett grafiskt användargränssnitt för pekskärmar

Christoffer Karlsson

Jonas Östlund

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Christoffer Karlsson

Jonas Östlund

Godkänd, 10-06-09

Handledare: Katarina Asplund

Examinator: Martin Blom

Sammanfattning

ECS är ett detaljhandelssystem som används av olika typer av butiker runt om i Norden. ECS underlättar arbetet vid kassan samt vid lagerhantering. ECS togs fram i slutet 90-talet då pekskärmar inte var aktuellt att användas inom handeln. Idag finns dock en större efterfrågan av system som är anpassade för pekskärmar, vilket ECS med sitt nuvarande gränssnitt inte kan tillfredsställa. Det nuvarande gränssnittet förlitar sig på användandet av mus och tangentbord då knappar och menyer i gränssnittet är för små för fingertryckningar. Utöver knapparna och menyernas nuvarande utseende önskades en generell uppfräschning av gränssnittet, för att få till en modern design.

Under detta projekt har vi tagit fram ett nytt gränssnitt för ECS som är anpassat för pekskärmar. Med hjälp av våra tidigare kunskaper och informationsökning har vi tagit fram ett stilrent och användarvänligt gränssnitt.

Development of a graphical user interface for touch screens

Abstract

ECS is a system used by different kinds of retail stores to provide simplicity during sales and storage management. At the time when ECS first was released, the retail stores found it useful enough to satisfy their needs. But today the retail stores are more anxious to use touch screens in combination with their cashier, which ECS really cannot deliver. The interface of ECS is designed to be used with mouse and keyboard and it would be too difficult to use your fingers trying to perform a task in the system. The buttons are simply too small to handle and the interface overall looks out of date.

During this project we have designed a new graphical interface to ECS with a solution for touch pads in our minds. We have searched for information and used our own acknowledgment to provide a good looking interface, but still with options enough to be useful. The interface has just enough space between the buttons to make the interface user friendly, even when using it with fingers on a touch screen.

Innehållsförteckning

Figurförteckning

1 Inledning

Stamford är en koncern där det i dagläget finns tre stycken produktutvecklade och branschinriktade företag; Stamford Consult AB som har fokus på fastighetsförvaltning och Stamford Retail AB med inriktning på detaljhandeln. I det tredje företaget Stamford ECS Retail AB där tillhandahåller man butiksdatasystemet ECS som köptes från Benefit AB 2008.

Benefit började utveckla ECS i slutet av 90-talet och gränssnittet är idag omodern och dessutom ej anpassat för pekskärmar. Systemet används idag av bl.a. av Fagmöbler i Norge, Gallerix i både Sverige och Finland, Landstinget i Värmland och Verner & Verner i Sverige[1].

ECS är något mitt emellan ett affärssystem och ett butiksdatasystem. Detta innebär att det främst är anpassat för t ex sportaffärer och inredningsbutiker där det inte är ett lika stort informationsflöde som hos t ex mataffärer. ECS är en klient/server-lösning där man använder så kallade ”tunna klienter” som hämtar och lagrar information på en server. Systemet är helt enkelt inte snabbt nog för att användas vid en utkassa i en mataffär då det kan finnas många aktiva kassor samtidigt i butiken som alla kommunicerar med samma server. Det nuvarande gränssnittet är inte heller optimalt för att snabbt kunna betjäna många kunder efter varandra, utan passar bäst för detaljhandel med färre kunder. Man kan lägga in i princip hur mycket information som helst om exempelvis varor och priser och systemet passar därför utmärkt för lagerhantering. ECS kan styras från butik, centralt eller av Stamford själva. ECS är skapat i programspråket Java och kan köras på ett flertal olika operativsystem och är dessutom kompatibelt med flera olika databaser[2].

Även om systemet konstant utvecklas har Stamford ECS Retail AB diskuterat om att förnya gränssnittet i ECS. Det nuvarande gränssnittet togs fram för många år sedan och är inte anpassat för att användas tillsammans med pekskärmar, vilket är något som många kunder efterfrågar idag. Stamford har tidigare tagit hjälp av en designbyrå för att ta fram en prototyp för hur ett eventuellt nytt gränssnitt, anpassat för pekskärmar, skulle kunna se ut. Prototypen blev dock inte som de förväntat sig då gränssnittet bland annat var för dåligt utrustat med knappar.

1.1 Syfte

Syftet med detta examensarbete var att vi skulle ta fram en prototyp för ett användarvänligt gränssnitt för ECS som samtidigt är snyggt och anpassat för att kunna användas för pekskärmar. Användarvänligheten, det modernare utseendet och pekskärmsanpassningen var några av grundkraven från Stamford. Att utveckla ett gränssnitt för pekskärmar skiljer sig en del från att utveckla ett gränssnitt för endast musanvändande. Det kräver att man har knappar som är anpassade för att trycka på, dvs. tillräckligt stora och med lagom mycket mellanrum mellan varandra. Samtidigt vill man försöka få ut så mycket som möjligt av den yta man har att jobba på. Vad vi även ville försöka uppnå var att göra något unikt som skiljer sig från andra gränssnitt för den här typen av kassasystem. Vi var tidigt inne på att göra ett så användarvänligt gränssnitt som möjligt, men inte på bekostnad av funktionalitet eller ett tilltalande utseende. Att kunna anpassa gränssnittet för vänsterhänta var en funktionalitet som implementerades redan från start, tillsammans med möjligheten att låta användaren ändra på knappars utseende för att tillgodose just den användarens behov. Detta var även ett av kraven från Stamford.

Rent implementationsmässigt var syftet att det skulle vara enkelt att integrera vårt gränssnitt med ECS. Därför har vi valt att implementera gränssnittet med en öppen lösning, M-V-C (se avsnitt 4.3), för att i slutändan kunna integrera vårt gränssnitt på deras system med mål att inte behöva förändra koden allt för mycket.

1.2 Resultat

Vi har skapat ett grafiskt gränssnitt som vi tycker uppfyller kraven som ställdes ifrån Stamford. Vi delade upp vårt nya gränssnitt i olika delar och gjorde vissa uppgifter mer tillgängliga än andra, beroende på hur ofta de skulle kunna tänkas använda. Vi skapade två olika knappsatser; ”WallButtonLayout” och ”FloorButtonLayout”. ”WallButtonLayout” består av maximalt 35 knappar, fria att kopplas till vilken funktion som helst i systemet. ”FloorButtonLayout” består av 12 knappar som i sin tur kan byta ut uppsättningen av knappar i ”WallButtonLayout” vid en knapptryckning. Detta innebar att vi hade totalt sett 420 möjliga knappar att använda och koppla till olika funktioner i systemet. Vi implementerade också möjligheten att kunna inaktivera knappar som kunde ses som överflödiga, samt möjligheten att ändra storleken på knappar för att göra vissa mer tydliga och lättåtkomliga än andra. På så viss har vi skapat ett gränssnitt som har tillräckligt många möjligheter, samtidigt som det är skalbart för att göra gränssnittet tydligt och användarvänligt.

1.3 Disposition

Efter detta inledande kapitel går vi i kapitel 2 igenom vad man ska tänka på när man utvecklar användargränssnitt för att göra de så användarvänliga som möjligt. Det handlar om att t ex ta hänsyn till att alla människor inte använder ett gränssnitt på samma villkor då vissa kanske har sämre syn än andra, varav olika textinställningar bör kunna justeras. När man använder en pekskärm kan detta spela stor roll beroende på hur gränssnittet är uppbyggt.

I kapitel 3 beskriver vi de två prototyperna vi tog fram som förslag till nytt gränssnitt i ECS. Vi går igenom för- och nackdelar med de två lösningarna och slutligen förklarar vi varför vi valde den ena. Efter att den ena prototypen valts går vi i kapitel 4 in på detaljer om hur vi utvecklade denna prototyp till ett riktigt gränssnitt. Vi börjar beskriva översiktligt vad de olika delarna i gränssnittet fyller för funktion, och fortsätter sedan djupare ned i detalj att beskriva hur de är implementerade.

I kapitel 5 presenterar vi kortfattat det färdiga gränssnittet, vad vi lärt oss och vad vi stött på för problem. Slutligen i kapitel 6 summerar vi hela projektet och tar upp lite idéer kring eventuella framtida arbeten på gränssnittet.

2 Bakgrund

I avsnitt 2.1 kommer vi gå igenom olika principer att tänka på för att skapa ett användarvänligt gränssnitt. Dessa är bland annat att ta hänsyn till människors olikheter, säkerhet hos gränssnittet och effektivitet vid användandet av gränssnittet. Därefter i avsnitt 2.2 går vi igenom Stamfords nuvarande gränssnitt och även deras prototyp.

2.1 Användargränssnitt

”*What is design?...It's where you stand with a foot in two worlds – the world of technology and the world of people and human purposes – and you try to bring the two together”* [3]

Ett användargränssnitt ska underlätta interaktionen mellan användaren och system. Det krävs mer än bara ett snyggt utseende för att skapa ett gränssnitt.
När man ska utveckla ett användargränssnitt måste man dels ta hänsyn till de vetenskapliga principerna och den tekniska specifikationen. Dels har man den mer konstnärliga aspekten där innovation och fantasi spelar en stor roll. I vårt fall har det handlat om att ta fram en design anpassad till ett detaljhandelssystem där tanken var att det skulle gå lätt att använda det grafiska gränssnittet på pekskärmar. Nedan går vi igenom några olika aspekter på användarvänligt gränssnitt.

2.1.1 Människors olikhet

8 % av männen i västvärlden är färgblinda[4], de har alltså svårt att skilja på rött och grönt. Detta kan vara extra problematiskt då den röda och gröna färgen oftast har en ganska betydelsefull funktion då grönt oftast associeras till ”Godkänn” och rött till ”Avbryt”. Vi människor har sällan samma syn heller, då vissa är närsynta och andra långsynta. Alla är vi inte högerhänta och vi har inte heller samma storlek fingrarna. Att i slutändan kunna byta färger, typsnitt, textstorlek och storlek på knapparna är alltså inte bara en estetisk fråga utan fyller alltså en viktig funktion för att systemet ska vara användbart.

Vidare har vi de kulturella skillnader som finns runt om i världen. I kombination med den röda färgen är många människor vana att se ett kryss för att understryka att det betyder ”Stopp/Avbryt”, problemet är att i t.ex. England kan ett kryss betyda just ”Acceptera/Godkänn”[5]. I kombination med den gröna färgen för acceptans kan man dock

alltid använda en bock för att förtydliga att det handlar om ”Acceptera/Godkänn”. Vi har valt en design där man kan modifiera layouten efter sina egna behov.

2.1.2 Säkerhet

Säkerhetsaspekten kanske inte är den första man tänker på när man talar om gränssnitt anpassat för olika system och självklart berörs detta område olika mycket beroende på miljö. Det finns extrema exempel där både flygplans- och tågolyckor inträffat p.g.a. bristande design bland verktyg och indikationssystem som använts av föraren. För att inte tala om en olycka i USA i början på 80-talet där ett kärnkraftverk nästan orsakade härdsmälta p.g.a. att en kontrollpanel indikerade att valvet var stängt när det egentligen var öppet[6].

Det kan kanske tyckas överdrivet att försöka dra en parallell till situationer som våra kunder kan tänkas hamna i, men det vore en katastrof i deras ögon mätt om ett kundregister skulle råka försvinna eller att ett kassasystem skulle börja räkna fel vid betalning, rabatter etc. Varningsdialoger vid viktiga händelser och aktioner från användaren är minst lika viktigt som att gränssnittet skall vara vackert att titta på. Vi har, tack vare möjligheten att modifiera våra knappar, valt att särskilja kritiska knappar. I vår standard-layout som vi tagit fram har vi avskiljt knappar som ”Lås kassa” och ”Avbryt” från andra knappar, samt gjort dem röda. Detta för att minska risken att användaren trycker fel i hektiska situationer.

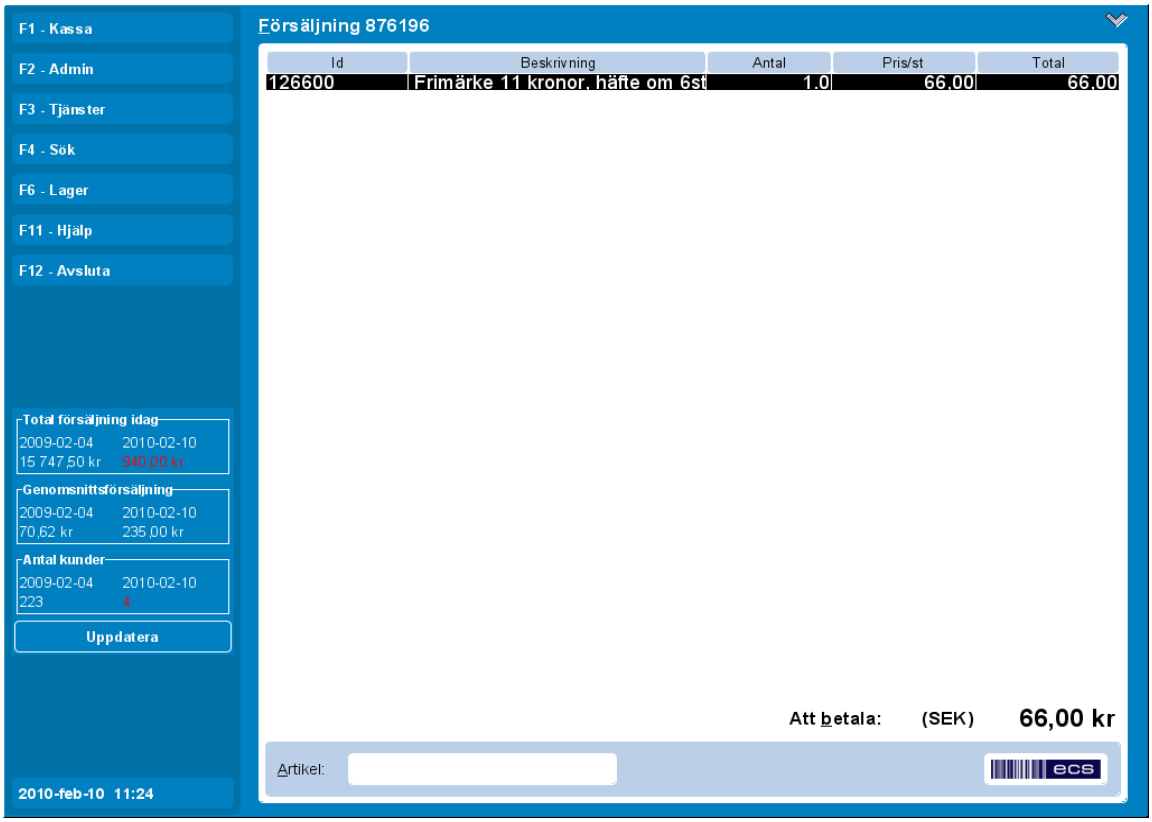
2.1.3 Effektivitet

Anledningen till att det ens finns användargränssnitt idag är just för att underlätta det dagliga arbetet som många användare utför. Ju lättare det är att förstå systemet och dess gränssnitt, desto fortare kan användaren börja bli produktiv. Vid framtagandet av våra prototyper (se kapitel 3) har vi bland annat hämtat inspiration från det nuvarande gränssnittet som används i ECS. Detta har vi gjort för att användaren lätt skulle känna igen sig. Man kan även hjälpa användaren att lära sig gränssnittet genom att vara konsekvent i sin design. Vi har t ex. knappen för att betala på samma ställe i vårt färdiga gränssnitt, oberoende av i vilket sammanhang man vill använda knappen. I vissa fall vi vill även fånga användarens uppmärksamhet. Detta gäller när t ex. det virtuella tangentbordet ska användas eller när en varningsruta visas. Då hamnar varningsrutan alltid i mitten av skärmen medan all bakomliggande grafik tonar bort, just för att användaren snabbt ska se vad som sker.

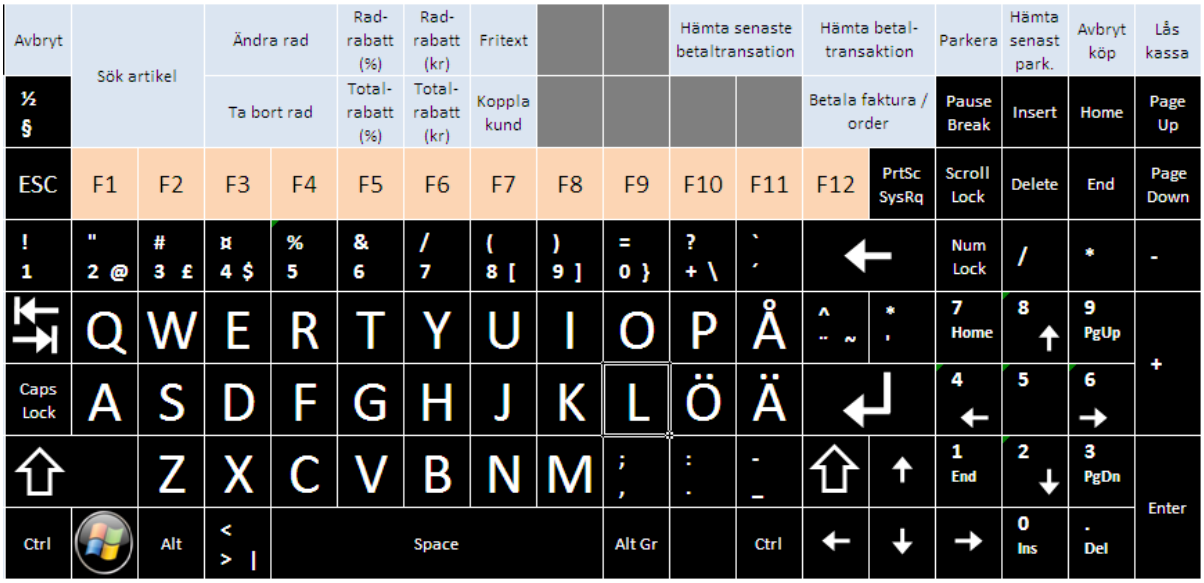
2.2 Beskrivning av det nuvarande gränssnittet

I Figur 2.1 ser vi det gränssnitt som används i ECS idag. Systemet tillsammans med gränssnittet började utvecklas för många år sedan och var då inte anpassat för att användas med pekskärmar hos butikerna. Att gränssnittet skulle kunna användas med en pekskärm var just ett av kraven från Stamfords sida. Gränssnittet (se Figur 2.1) förlitar sig delvis på användandet av mus och vanligt tangentbord, men Stamford har även specialdesignade tangentbord (Figur 2.2) som kan anpassas efter kundens behov. Dessa tangentbord har snabbknappar till vissa funktioner i systemet istället för att man måste klicka sig fram med musen. Några typiska funktioner som ECS erbjuder är att lista alla varor som man är på väg att köpa, att kunna söka efter artiklar på lagret, att visa statistik på försäljning och mycket annat.

Något som är värt att notera är att gränssnittet som finns nu inte har några möjligheter att förändras; det du ser är det du får. Eftersom ingen av oss tyckte att gränssnittet såg direkt modernt ut, fick vi ju redan här tanken att göra våra prototyper både snyggare och framför allt med möjlighet att kunna anpassas efter kundens behov.



Figur 2.1: Nuvarande gränssnittet för ECS



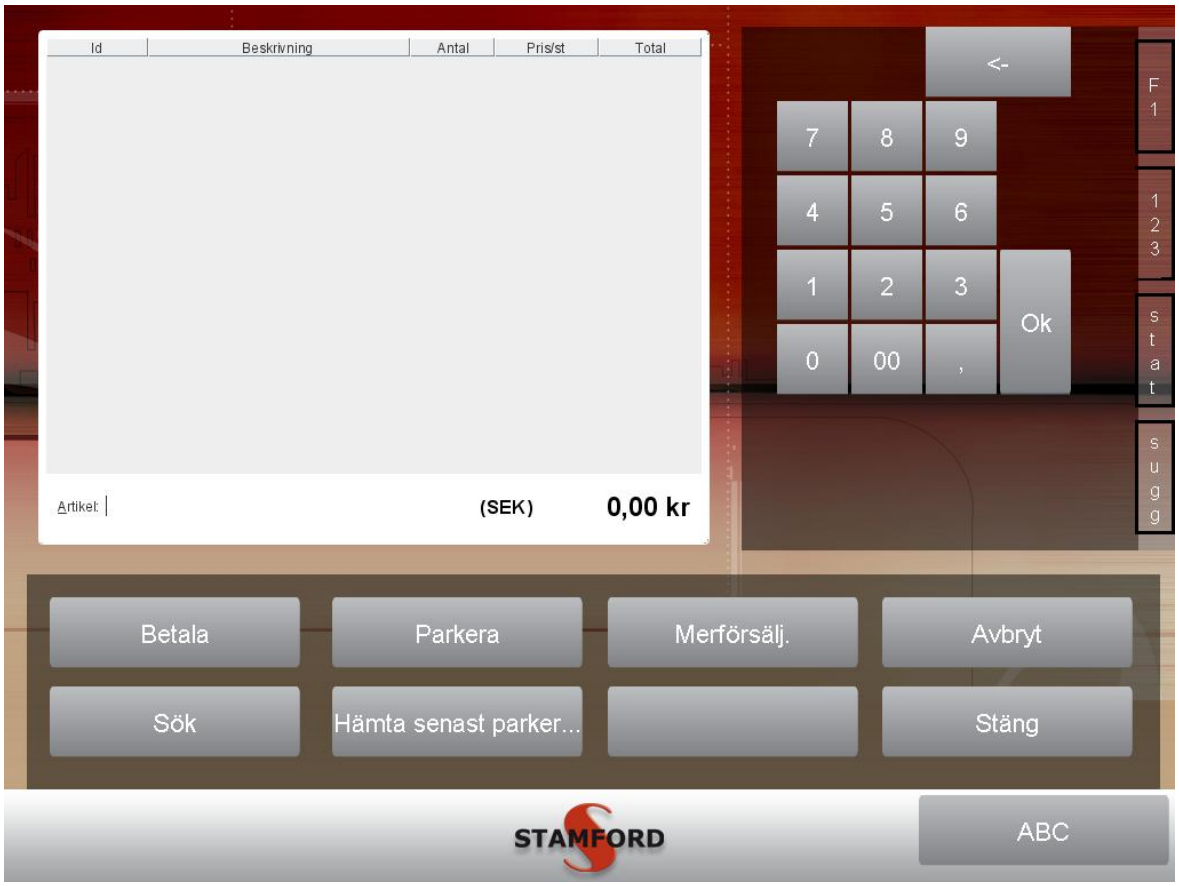
Figur 2.2: Layouten på Stamfords anpassade tangentbord

Tangentbordet ovan (se Figur 2.2) som Stamford kan specialanpassa efter kundens behov fyller samma slags funktion som vi erbjuder i vårt färdiga gränssnitt (se avsnitt 5.2). På samma sätt som kunden kan önska att t ex. ha en stort knapp för att söka efter artiklar, kan vi erbjuda detta i vårt pekskärmsgränssnitt.

2.3 Stamfords prototyp

Prototypen nedan (se figur 2.3) är framtagen av Stamford med hjälp av en designbyrå som bidrog med bilder till gränssnittet. Prototypen blev dock inte som Stamford önskat då t ex. bakgrundsbilden kändes överflödig då den inte syntes så bra när alla komponenter kom på plats. Vidare så tyckte de att det fanns för få alternativ till knappar och möjligheten att anpassa knapparna saknades.

Dessa var några av anledningarna till att Stamford ville att vi skulle ta fram en ny prototyp. På längre sikt har Stamford också planer på att erbjuda ett gränssnitt som även passar andra affärer än de som de är gjord för idag. Då är det viktigt att gränssnittet kan anpassas och passa så många olika butiker som möjligt, vilket inte denna prototyp skulle göra.



Figur 2.3: Stamfords prototyp

2.4 Summering

Vi beskrev översiktligt vad man bör tänka på vid framtagandet av ett grafiskt gränssnitt. Flera olika faktorer spelar in för att uppnå ett tillfredsställande resultat. Vid framtagandet av vårt gränssnitt har vi reflekterat över punkter som effektivitet, säkerhet och anpassningsmöjligheter för att underlätta användarens jobb. Vi beskrev därefter hur Stamfords nuvarande gränssnitt samt prototyp såg ut för att kunna visa på vad som behövdes förbättras (Stamfords krav) inför vår prototyp-utveckling.

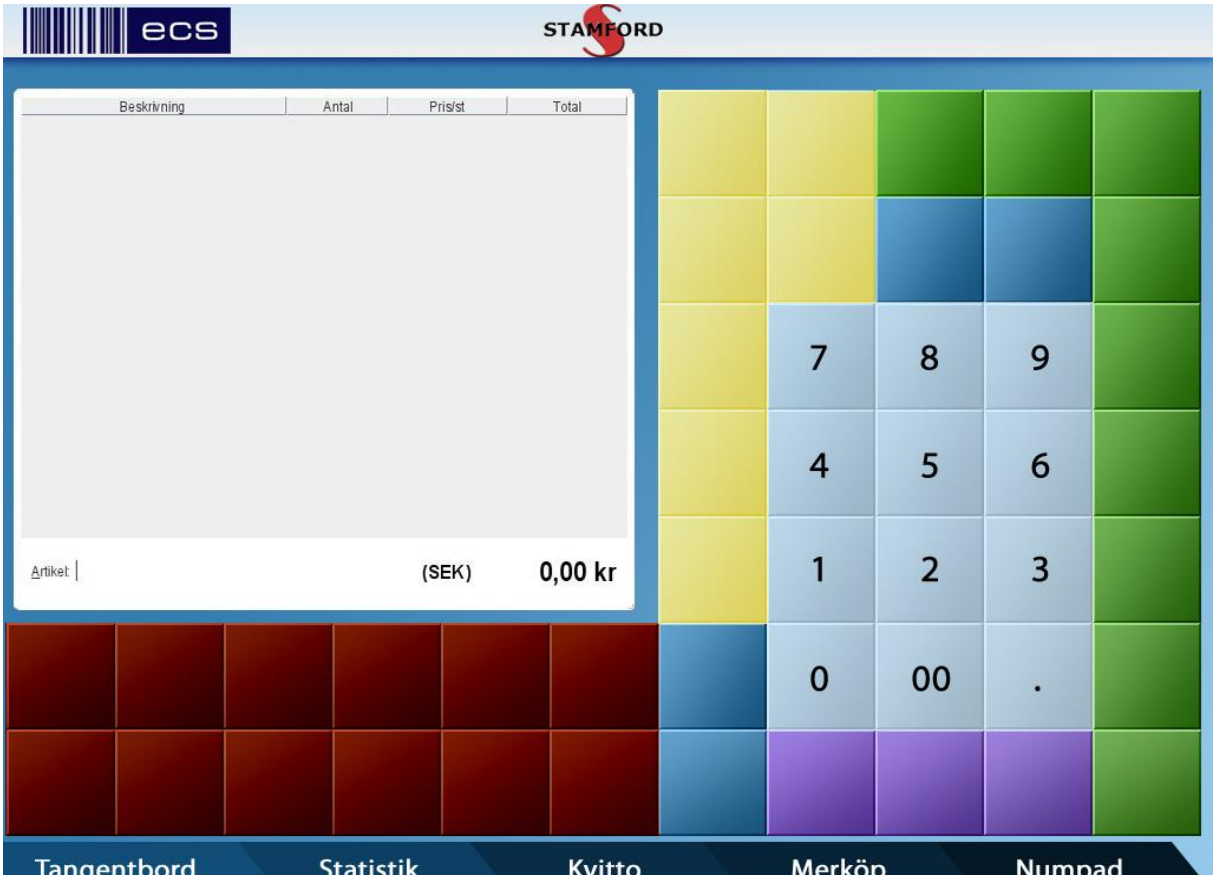
3 Prototypdesign

När ett gränssnitt ska skapas är det viktigt att ha något att utgå ifrån för att veta hur det grafiska ska lägga ut och designas på den ytan man har att tillgå. Vid introduktionen till projektet hade projektledarna kraven att det nya gränssnittet till kassasystemet skulle ha support för pekskärmar, vara anpassningsbart för kund, företag och eventuellt ha en mer tilltalande design. Vi utvecklade två olika prototyper för detta ändamål.
I avsnitt 3.1 beskriver hur vi kom fram till den första prototypen och även hur den förhåller sig till de krav som ställdes. Och i avsnitt 3.2 beskriver vi den andra prototypen och hur den passar in i kraven. Avsnitt 3.3 jämförs därefter dessa två prototyper och vi motiverar varför den prototypen var passande bäst för ändamålet. Dessutom tar vi upp några av de förbättringar som behövdes göras.

3.1 Första prototypen

Den första tanken som dök upp för oss vid visningen av Stamfords nuvarande prototyp (se avsnitt 2.6) var att den hade för få knappar. Detta gjorde att de fick kategorisera alla funktioner, dvs. göra så att en knapp också förändrade de andra knapparna. T ex att knappen ”Order” ändrade andra knappar till ”Rabatt” och ”Betala”. Detta kunde vara lite förvirrande då man, som användare, lätt kunde tappa bort sig.

Det vi valde att göra i den första prototypen (se Figur 3.1) var att skapa så många knappar som möjligt, men där varje knapp har en tillräckligt bra träffyta för att ingen ska kunna trycka fel då en pekskärm används. En fråga som kom upp när prototypen var klar var om att det möjligen fanns för många knappar, dvs. om många av dessa knappar inte skulle användas. Detta löstes dock med att flera knappar skulle kunna sammanfogas till en enda knapp vilket både hjälper träffsäkerheten för rätt knapptryckning och anpassningsbarheten. Nedan beskrivs hur prototypen uppfyller de krav som ställdes och vilka fördelar respektive nackdelar designen har.



Figur 3.1: Prototyp 1

3.1.1 Support för pekskärmar

Den grundläggande storleken för en pekskärmsknapp är ca 1,9 cm och den storleken är också fastslagen som en standard av ISO och ANSI [7]. Dock är standarden inte är anpassad till personer med svårigheter gällande syn och motorik. Därför förstorade vi varje knapp till 3,14 cm, vilket gav en mycket god träffyta samtidigt som man tydligt ser var knapparna är.

3.1.2 Anpassningsbarhet

Prototypen har goda anpassningsmöjligheter då det först och främst är möjligt att ändra storleken på varje knapp, ändra färg på knappar och bakgrund, inaktivera knappsatser och omplacera komponenter (vilket främst riktar sig till vänsterhänta). Med dessa möjligheter kommer varje kund att kunna anpassa sitt kassasystem utifrån vad som passar bäst för butiken.

3.1.3 Tilltalande design

Att man har möjlighet att ändra både färgtema och knappfärger gör att varje företag själv kan designa sitt kassasystem utifrån tycke och smak. Dessutom är prototypen ren i designen med vilket menas att den inte innehåller några avancerade grafiska detaljer. Detta gör att prototypen är lugn och välkomnande där man enkelt ser vad varje sak har för funktion utan att ha en större kompetens inom grafik eller utveckling. Prototypen har även ett modernt utseende, vilket är viktigt för att hålla samma standard som andra gränssnitt inom liknande områden. Detta kan ge ett försprång gentemot övriga företag med äldre gränssnitt. Det moderna utseendet uppstod genom att skapa flera olika färgövergångar vilket gav en 3D-känsla på alla objekt.

3.1.4 Fördelar

Tanken bakom denna prototyp var att främst att skapa en design som skulle kunna anpassas för varje företag på så sätt att alla skulle kunna känna sig bekväma med det visuella och med orienteringen i programmet.

Den största fördelen är att nästintill allt man ser går att ändra på något sätt, antingen genom omplacering eller genom storleksändring. Genom detta blir programmet dynamiskt och mycket mer användarvänligt. Dessutom så ska varje knapp ha en funktion knuten till den. Denna funktion kan bytas till det som passar knappen, vilket i korthet betyder att alla knappar kan bindas till vilken funktion som helst. Flikarna har även en bra placering då dessa liknar ett aktivitetsfält som de flesta operativsystem idag har. Genom att de är placerade där de är så ger de användaren ett igenkännande intryck och på så sätt kan dessa flikar kännas familjära.

3.1.5 Nackdelar

Den största nackdelen med den här prototypen är att det inte finns något mellanrum mellan knapparna, vilket kan göra det lättare att trycka på fler knappar än vad man vill. Det kan även vara en nackdel att det finnas får många knappar vilket kan skapa förvirring.

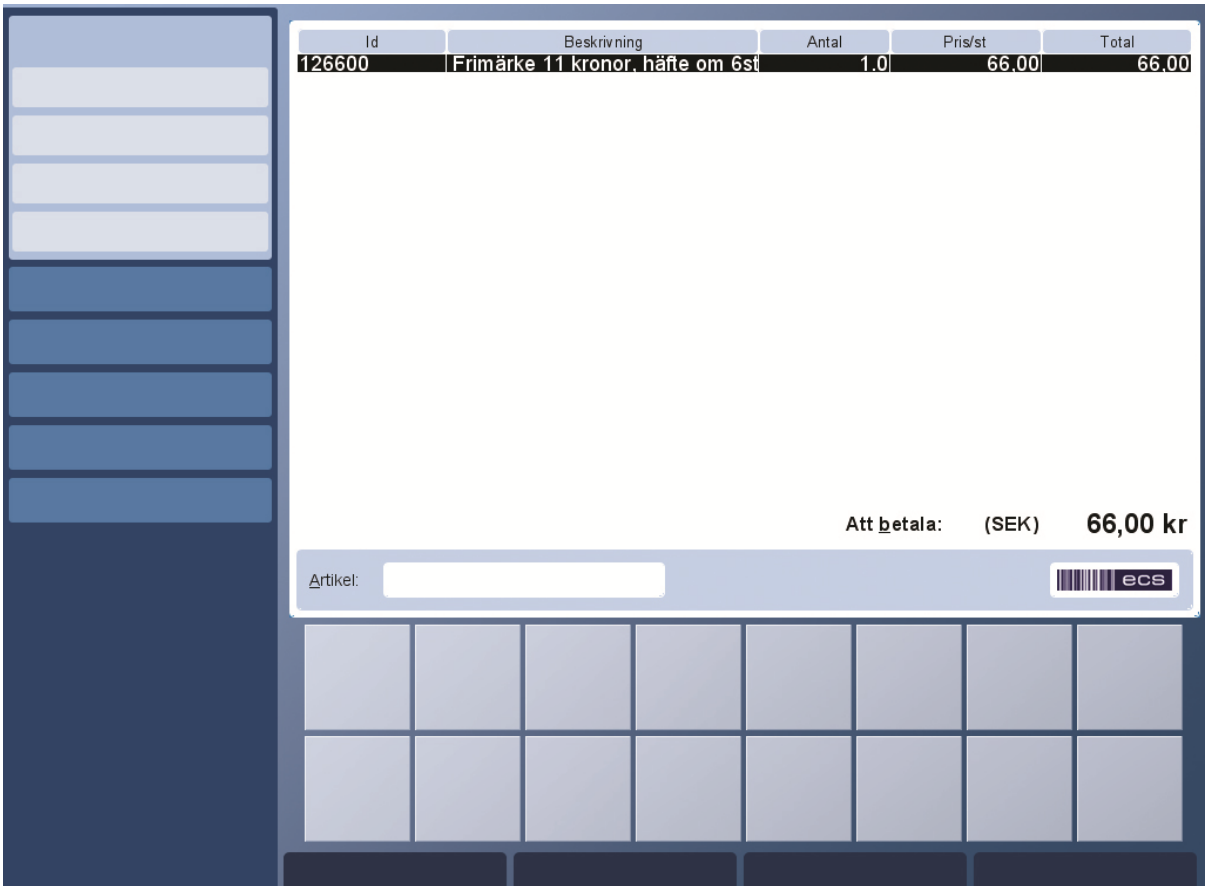
Produktfönstret (den vita rutan på Figur 3.1) har ett par mindre nackdelar. Dels så passar kolumnerna i fönstret inte in om man ser till resterande utseende på prototypen, vilket ger ett intryck att de inte hör hemma i det resterande utseendet. En annan nackdel med

produktfönstret är att fältet där man söker artiklar inte syns tydligt nog. Bakgrunden på fältet är vitt och väldigt svårt att lokalisera om man är ny användare.

3.2 Andra prototypen

Den andra prototypen (se Figur 3.2) bygger till största delen på Stamfords nuvarande gränssnitt. Tanken här var att alla användare skulle få ett så välbekant gränssnitt som möjligt vid en eventuell uppgradering.

När denna prototyp skapades så försökte vi sammanfoga den föregående prototypens idéer tillsammans med Stamfords nuvarande gränssnitt för att både få funktionaliteten som vår föregående prototyp hade, men även behålla det ursprungliga utseendet. Menyn (rutan till vänster på Figur 3.2) ändrades till kategoritabbar där varje tabb expanderas vid en knapptryckning som ses i den ljusa rutan i Figur 3.2. Utifrån denna funktion kan man enkelt se vilka undertabbar som finns. Dessutom lades idén om flikar och snabbknappar till även i den här prototypen för att ha vissa funktioner mer lättillgängliga. Sist ändrades bakgrunden till en färgövergång för att skapa ett trevligare utseende. Nedan beskrivs hur prototypen uppfyller de krav som ställdes och vilka fördelar respektive nackdelar designen har.



Figur 3.2: Prototyp 2

3.2.1 Support för pekskärmar

Prototypen har tilldelats större menytabbar (menyn till vänster på bilden) än i Stamfords nuvarande gränssnitt. Dessa har en höjd på 1,9 cm tillsammans med en separation på 0,4 cm, vilket gör att varje tabb är enkel att trycka på och har en relativt stor träffyta. Dessutom har vi inkluderat en panel med samma knappstorlek som på föregående prototyp (3,14 cm). Dessa knappar används som snabbknappar dvs. knappar som behövs för snabb åtkomst t ex när det gäller att lägga till en speciell vara fort.

3.2.2 Anpassningsbarhet

När det gäller anpassning var tanken att det skulle gå att ändra färgtema, ändra knappars färg och form, ändra på vilka tabbar som ska synas och sist att det även skulle kunna gå att ändra på placeringen av objekt. Eftersom denna prototyp har hämtat många idéer ifrån den första prototypen så är anpassningen densamma i båda fallen.

3.2.3 Tilltalande design

Denna prototyp använder sig av designteman, vilket betyder att om man ändrar färg så ändras också alla komponenter, bl.a. flikarna och bakgrunden, till just den färgen. Dock så får komponenterna olika nyanser för att kunna skilja dem åt. Designtemat är dock inte tillämpat på knapp-panelen där varje knapp istället har en egen färg. Då den här prototypens design påminner mycket om Stamfords nuvarande gränssnitt så är inte mycket av utseendet på prototypen något nytt. Dock så finns det vissa delar som skiljer sig, t ex bakgrunden som har en färgövergång. Dessutom så gör snabbknapparna och flikarna att programmet är mer funktionellt.

3.2.4 Fördelar

Prototypen designades utifrån det nuvarande gränssnittet på kassasystemet, som är välbekant för användarna sedan tidigare. Detta kan göra det lättare att förstå sig på det nya gränssnittet.
I den här prototypen finns det även möjlighet att dela in knapparna i kategorier (menyn till vänster), vilket kan skapa en enklare överblick av var en viss funktion finns.
Den här prototypen har en fördel av att den har ett stort produktfönster (den vita rutan till vänster på Figur 3.2), vilket kan ge en större överblick över artiklarna och även ha möjligheten att visa mer information.

3.2.5 Nackdelar

Trots att menytabbarna är på 1.9 cm med ett mellanrum på 0,4 cm så är träffytan liten vilket gör det enklare med feltryckningar. Dessutom finns problemet att menyn har en begränsad yta, vilket gör att alla tabbar har ett begränsat antal undermenyer. Det finns heller inget stöd för att undermenyer ska kunna ha egna undermenyer. Prototypen har knappt om utrymme om många funktioner ska få plats i samma gränssnitt vilket i vissa fall kan ställa till problem om den här prototypen utvecklades för en livsmedelsbutik. Menytabbarna kan även göra det svårt att hitta rätt funktion, vilket även ökar antalet knapptryckningar för att komma dit man ska.

3.3 Val av prototyp

Vi skapade två olika prototyper för att ge Stamford en möjlighet att välja vilken prototyp som de tyckte passade bäst in i ECS. Båda prototyperna hade var sin unik design och båda hade vissa element ifrån både Stamfords nuvarande gränssnitt och från Stamfords prototyp.

Varje prototyp utgick från de fyra krav som ställdes av Stamford; prototypen skulle ha stöd för pekskärmar, vara anpassningsbart för kund och företag, eventuellt ha en mer tilltalande design samt vara användarvänligt.

De två prototyperna jämfördes med varandra utifrån kriteriet om vilken som bäst uppfyllde de fyra kraven. Vi jämförde även fördelar och nackdelar med varje prototyp. Utifrån jämförelsen och även utifrån diskussioner med Stamford så utsågs den första prototypen till den som skulle implementeras till ett gränssnitt.

Prototyp ett är bättre anpassad för pekskärmar då knapparna är större, vilket skapar en mindre risk att trycka fel. Då det dessutom finns möjligheter att skapa knappar som ändrar de övriga knapparnas funktion, har vi även större möjlighet att kategorisera funktioner än för den andra prototypen. Dessutom finns det möjligheten att kunna integrera ett tangentbord för siffror direkt i gränssnittet. Dock så behövde vissa förbättringar göras. Knapparna behövde mellanrum, främst för att kunna minska feltryckningsrisken ännu mer, men även för att fördela ut alla komponenter över ytan för att få en mer ”luftig” känsla i designen. Produktvyn behöver även den ändras då huvudcellerna inte gick i samma färgnyans som resterande celler i gränssnittet. Dessutom så var det väldigt svårt att se var artikelfältet låg då det hade samma färg som bakgrunden.

3.4 Summering

Vi skapade två prototyper för att vi tillsammans med Stamford skulle ha möjligheter att välja vilken prototyp som passade bäst till ändamålet. Den prototypen som därefter valdes var den första prototypen.

4 Implementation

Vårt gränssnitt är uppdelat i fyra centrala delar (se Figur 4.1) som separerar de olika komponenterna. I avsnitt 4.1 beskriver vi hur vi gjort uppdelningen och på vilket sätt den speglar tanken bakom vår design av gränssnittet. Implementationen är gjord i Java och består av många klasser med tydligt fördelade uppgifter. Anledningen till detta är att det ska vara enkelt att vid ett senare tillfälle modifiera funktionalitet utan att behöva störas av beroenden från andra klasser och funktioner. Detta diskuterar vi i avsnitt 4.2 för att sedan i avsnitt 4.3 förklara hur man kan uppnå oberoenden av klasser genom designmönstret M-V-C. För att specificera hur knapparna ser ut och vilken funktionalitet de har använder vi en XML-fil som vi går in i detalj på i avsnitt 4.4. I avsnittet 4.5 berättar vi om en egenskaps-fil som kan användas för att anpassa gränssnittet ytterligare. Flikarna är modifierade knappar som används för att kunna skapa menyer. Dessa erbjuder ytterligare funktioner i ECS som dock inte behöver vara lika snabbt tillgängliga. Detta diskuterar vi om i avsnitt 4.6 Produktfönstret tillsammans med dess inre komponenter listar alla varor vid ett köp. Det presenterar id, namn, pris och antal av varan som säljs. Vi har anpassat dessa delar för att passa in rent designmässigt med resten av gränssnittet. Produktfönstret beskriv i avsnitt 4.7.

4.1 Översikt av systemet

Då vårt gränssnitt ska vara kompatibelt med det befintliga systemet framtaget av Stamford, är gränssnittet programmerat i Java, JDK 5.0.

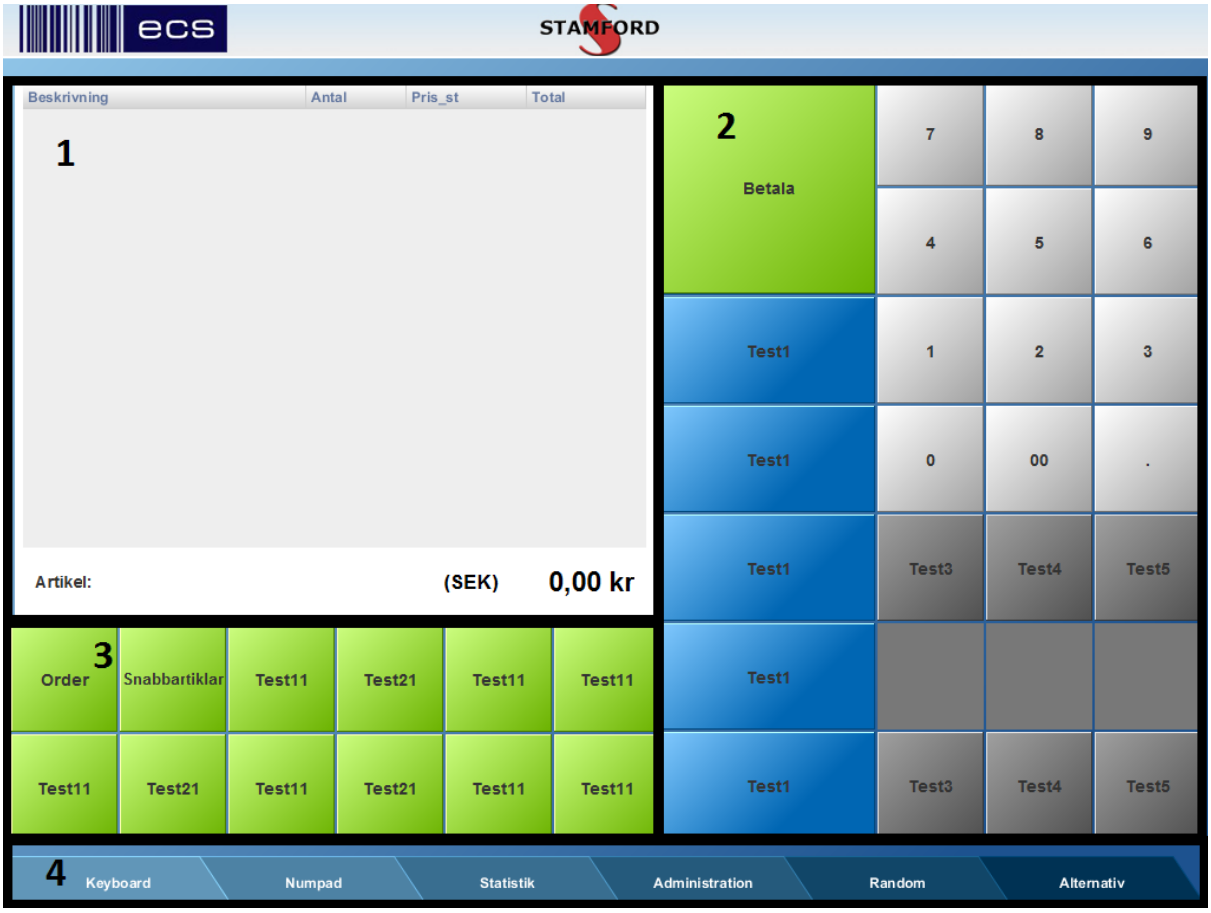
Vår huvudklass som håller ihop hela gränssnittet heter ”ECS.java” och är implementerad med hjälp av en Frame[8]. En ”Frame” i Java kan ses som själva ramen i ett grafiskt program, där man i princip bara får ett tomt fönster med en tillhörande knapp i högra hörnet för att kunna stänga ned programmet. Eftersom programmet är menat att köras i helskärmsläge har vi valt att ta bort den synliga ramen runt programmet och endast lagt till Stamfords logotyp högst upp i gränssnittet. Detta gör att man får en bättre helhetskänsla när man använder systemet. Inuti denna osynliga ram har vi sedan lagt till grafiska komponenter där vi bygger upp grunden för var alla knappar, menyer och fönster skall finnas. Man kan se det som att gränssnittet är uppdelat i 4 separerade områden (se figur 5.1):

1. **Produktvisningsfönstret** är rutan där all information presenteras för användaren, till exempel alla varor som registrerats när en kund ska göra ett köp. Där ser man även kostnaden för köpet.

2. **WallButtonLayout** kallar vi området där vi har alla knapparna som syns vertikalt till höger i gränssnittet. Dessa knappar är modifierbara på så sätt att man till exempel kan ändra färg, storlek och text på dem. Varje knapp som skapas och läggs till denna högra panel fyller dock en bunden funktion, till exempel att betala, vad man än väljer att det synliga namnet på knappen skall vara. Knapparna beskrivs mer detaljerat i avsnitt 4.6.

3. **FloorButtonLayout** är knappsatsen som ligger nedanför produktvisningsfönstret. Dessa knappar används för att bestämma vilken uppsättning av knappar som skall synas under ”WallButtonLayout”. Om ingen av dessa knappar är nedtryckta visas en standarduppsättning av knappar som användaren antagligen kommer att använda flitigast under en vanlig session. I standarduppsättningen finner vi bland annat en knapp för att betala och en uppsättning av knappar som fungerar som ett numeriskt tangentbord. Om användaren väljer att trycka på t ex. ”Snabbartiklar” i WallButtonLayout (se figur 5.1: ruta 3) kommer en ny uppsättning av knappar synas som alltså är anpassade för att sälja vissa artiklar. Det kan handla om rea-varor som för tillfället säljs väldigt flitigt och som man då vill ha nära till hands. Rent tekniskt fungerar kopplingen mellan ”FloorButtonLayout” och ”WallButtonLayout” på det sättet att en specifik XML-fil läses in vid interaktion med knapparna i ”FloorButtonLayout” och på så sätt ritas rätt knappar ut i ”WallButtonLayouten”. Detta förklaras mer i detalj i avsnitt 5.2.

4. Längst ned i gränssnittet finner vi ett antal flikar. De fungerar som menyer som expanderas när man trycker på en specifik flik. Under fliken ”Alternativ” får man till exempel upp menyval för att byta kassa eller för att stänga ned kassan.



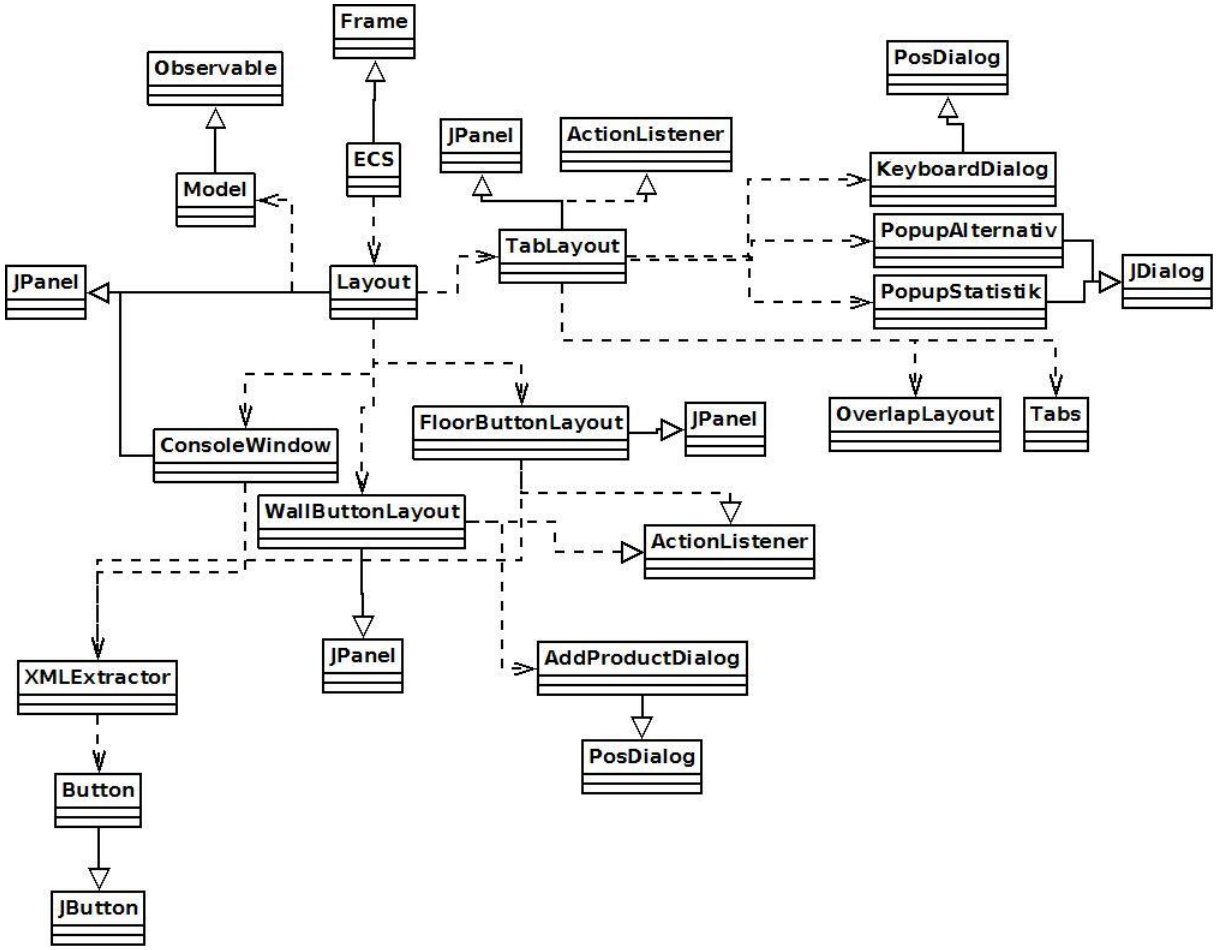
Figur 4.1: Beskrivning av layout

4.2 Generellt om implementationen

En förutsättning för att kunna gör en så generell lösning som möjligt var att undvika bilder i gränssnittet och att istället använda sig av Javas egna färger och funktionalitet för att modifiera utseendet. Knapparna, som är helt vanliga ”JButtons” [9], innehåller alltså inga bilder utan endast färger i olika nyanser och hela bakgrundsbilden är också en vald färg som går från en ljus till mörk toning. Dessa färger går sedan att ändra på efter eget tycke och smak och vi har på så sätt skapat färgteman som går att ändra på bara några klick, just för att användaren skall kunna trivas med systemet. Detta val gör dessutom gränssnittet mer lättviktigt och tar mindre plats då inga bilder behövdes laddas in. Den enda bilden som används är Stamfords egna logotyp längst upp i bild i gränssnittet. Detta gjordes efter önskemål från kundens sida.

Vi har försökt att hålla lösningen så flexibel som möjligt genom att låta varje klass fylla en specifik funktion (se Figur 4.2). Till exempel finns det separata klasser för knappar, flikar och

layout. Dessa klasser bygger i sin tur till viss del på klasser som redan tagits fram av Stamford, t ex klasser för att skapa paneler och knappar med rundade hörn. Ser vi till ett längre perspektiv ville vi också försäkra oss om att det skulle vara enkelt att lägga till eller ta bort funktionalitet i designen. Meningen är att man i slutändan helt enkelt ska kunna lyfta ut ett befintligt gränssnitt från ECS och ersätta det med vår lösning i framtiden. Det krävs då att vår lösning är flexibel och delarna löst kopplade för att skapa så lite problem som möjligt vid den slutgiltiga integrationen. Ett sätt att underlätta denna integration var att använda oss av designmönstret M-V-C som vi beskriver i detalj i följande avsnitt.



Figur 4.2: Klassdiagram

4.3 M-V-C

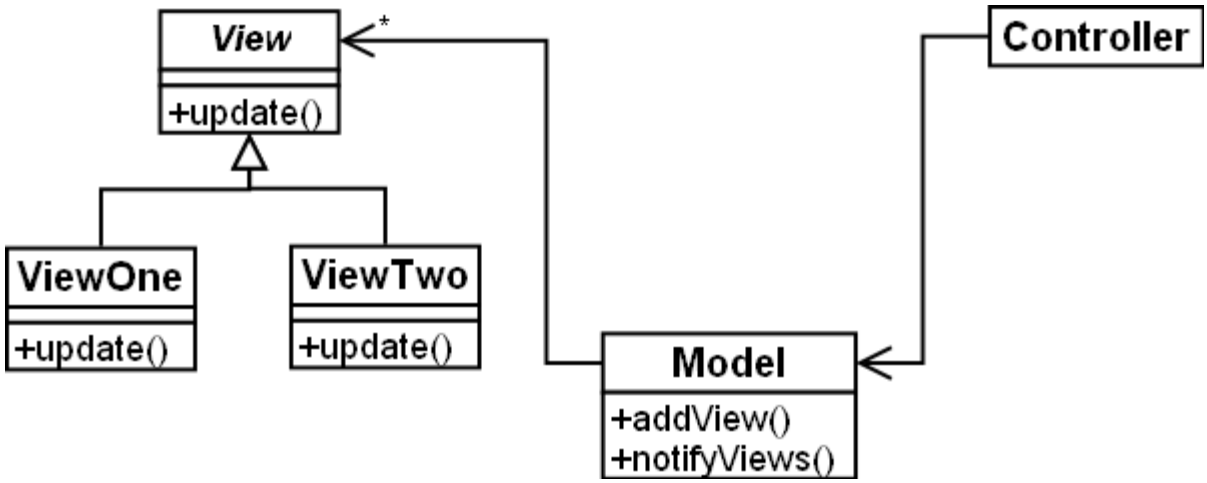
MVC är ett vanligt designmönster som används för att separera de grafiska komponenterna i ett system från själva logiken som systemet erhåller. Nedan beskrivs de olika delarna i designmönstret:

- **M – Model:** Denna del av designmönstret ska motsvara logiken i programmet, dvs. där all information sparas och där alla eventuella uträkningar sker. Det skulle t ex kunna vara att modellen håller reda på totalsumman av alla varorna vid ett köp.
- **V – View:** I denna del samlas alla vyer som kan tänkas vara intresserade av att veta ändringar som sker i logiken. Det skulle t ex kunna vara att produktfönstret i vårt gränssnitt presenterar totalsumman av det köp som nämns ovan. Produktfönstret har heller ingen kontroll på vilka varor som egentligen ska visas i fönstret, utan dessa hämtas från logiken/modellen. Allt detta sker automatiskt när användaren valt att lägga till en vara.
- **C – Controller:** Denna del har som uppgift att agera som en mellanhand mellan vyerna och modellen. Den tar emot händelser som sker i vyn, t ex knapptryckningar, skickar vidare händelsen till modellen som uppdaterar sig och sedan informerar vyn om att de ska uppdatera sig för att visa om den nya informationen för användaren.

Detta designmönster, eller arkitektur, är bara en princip över hur alla delarna ska hänga ihop och det finns egentligen inget officiellt sätt att implementera designmönstret på. En anledning att implementationen till viss del beror på vilket programmeringsspråk man använder och vilka tekniker det språket erbjuder.

Som vi ser i figur 4.3 innehåller Model funktioner för att addera ”addView” lyssnare till sig, samt ”NotifyViews” för att beordra sina lyssnare att uppdatera sig. Vyerna har då funktionen ”update”, där de uppdaterar sig beroende på vad som hänt. Figur 4.3 är ett exempel och en förenklad version av hur det går till i denna koppling.

Anledningen till att vi har använt detta designmönster under implementationen är naturligt då vi bara skapat ett nytt gränssnitt till ECS, utan att påverka dess egentliga funktionalitet. Vi ansåg att ECS var ett såpass invecklat och stort system att vi inte hade den tid vi önskade att sätta oss in i koden. Istället för att använda logiken som finns i ECS kunde vi alltså tack vare M-V-C skapa en fiktiv modell där vi bland annat lagrade påhittade varor som vi kunde använda för att visa hur vårt gränssnitt fungerade.



Figur 4.3: M-V-C[10]

Vi har valt att implementera M-V-C med hjälp av Javas ”Observer”[11] och ”Observable”[12] som kan ses som två olika ramverk eller tekniker. Med ”Observer” låter man en eller flera vyer ”lyssna” på förändringar i datan som tillhandahålls av logiken i systemet. Det är logiken som är ”Observable” där systemets tillstånd förändras och tolkas av vyerna Vårt produktfönster (se avsnitt 4.2) har bland annat till uppgift att visa varor som ska säljas vid ett tillfälle. Vi har därför implementerat denna komponent som en ”Observer”, som observerar datan som i detta fall tillhandahålls av klassen. Logiken behöver inte bry sig om vem det är som vill ha informationen den har, utan den skickar ut den till alla registrerade vyer, oavsett om de är grafiska eller terminalbaserade vyer. Produktfönstret behöver i sin tur inte veta hur eller var informationen kommer ifrån, dvs. var den är lagrad. T ex skulle varorna som visas upp i produktfönstret kunna ligga lagrade i en databas, på en fil, eller som i vårt fall, i en provisorisk ArrayList[13]. Varje gång någonting händer i systemet som produktfönstret skulle kunna vara intresserad av, skickas ett anrop ut till alla registrerade vyer att de ska uppdatera sig . I och med denna lösning hamnar ”Controller” inbakat i vyerna där alla händelser (events) skapas vid knapptryckningar.

4.4 Layouten på knapparna

Som vi redan nämnt tidigare (se avsnitt 4.1) spelar XML(e**X**tensible **M**arkup **L**anguage)[14] en stor roll för gränssnittet. Med hjälp av XML-filer specificerar vi hur olika knappsatser ska se ut och även hur varje enskild knapp ska se ut och bete sig. Man specificerar vilka olika element som filen ska hantera och använder sedan dem som attribut för hur knappen ska se ut och fungera. XML-filen lagrar information om hur knapparna ska se ut innan knapparna har

blivit skapade. Vid uppstart av gränssnittet har vi en standarduppsättning av knappar som läses in till ”WallButtonLayout”. Vi ser då bl.a. knappar för att betala, söka, lägga till vara och låsa kassan. Om användaren trycker på en av kategoriknapparna i ”FloorButtonLayout”, t.ex. ”Snabbartiklar”, läses motsvarande del in av XML-filen med hjälp av klassen ”XMLExtractor”. Nya knappar, anpassade för snabbartiklar, skapas nu och visas upp i ”WallButtonLayout”. Här nedan visas exempelkod på hur en knapp i vårt gränssnitt skulle kunna se ut:

- **<FloorButtonPanel> -** Namnet anger vilken del av XML-filen som ska läses av. Man skulle kunna ha många olika uppbyggnader i en och samma XML-fil.
- **<Row id="0"> -** Anger på vilken rad knappen ska ligga på
- **<Button> -** Attributet som representerar alla egenskaper från den specifika knappen.
- **<Title color="white">Order</Title>** - Anger färg och namnet på knappen
- **<ForeGround>*Hexadecimalt*</ForeGround>** - Sätter förgrundsfärgen
- **<BackGround>*Hexadecimalt*</BackGround>** - Sätter bakgrundsfärgen
  Anledningen till att det finns två olika typer av färgsättning är att skapa en färgövergång, där förgrunden står för den ljusa tonen och bakgrunden för den mörka.
- **<Width>1</Width>** - Anger hur bred knappen skall vara, dvs. hur många platser utifrån en vanlig knapps storlek den ska ta upp på bredden.
- **<Height>1</Height>** - Anger hur hög knappen skall vara, dvs. hur många platser utifrån en vanlig knapps storlek den ska ta upp på höjden.
- **<Position>0</Position> -** Anger vilken plats på den angivna raden knappen ska ligga på.
- **<TextSize>14</TextSize>** - Anger textstorleken på knappen. Oftast räcker det att ha ett kort förklarande ord på knappen som får plats, men om man behöver mer utrymme kan man alltså minska textstorleken.
- **<Enabled>true</Enabled>** - Anger om knappen ska vara aktiv eller inte. Då vår designlösning innebär att det finns väldigt många knappar på skärmen samtidigt, varav många kanske inte ens används, låter vi här användaren avaktivera en knapp. Den kommer fortfarande synas på skärmen, men den går inte att klicka på så att inga missförstånd kan inträffa.

- **<isToggle>true</isToggle>** - Denna funktion används för knappar som är skapta för ”FloorButtonLayout”. Toggle i detta fallet syftar på en av/på-mekanism och innebär då att man kan aktivera en ny vy för ”WallButtonLayout” genom att trycka på någon av toggle-knapparna. I exempeldatan som är ifylld för just denna knapp kommer layouten anpassad för ”order” att hamna i fokus. Trycker man en gång till på knappen kommer man tillbaks till standard-layouten och trycker man på någon annan toggle-knapp aktiverar man istället den motsvarande layouten.
- **<Function>Order</Function>** - Istället för att knyta knappens namn till en funktion, alltså vad som ska triggas i systemet, använder vi ett funktionsattribut för det.

Eftersom det är ganska tydliga element/attribut som kan redigeras i XML-filen känns det väldigt logiskt att med tiden bygga en användarvänlig redigerare som användaren kan använda för att modifiera informationen i. Klassen ”XMLExtractor” tar emot information om vilken XML-fil det är som ska läsas in när användaren gjort ett val. En funktion löser igenom filen och skapar alla knapparna med rätt värden.

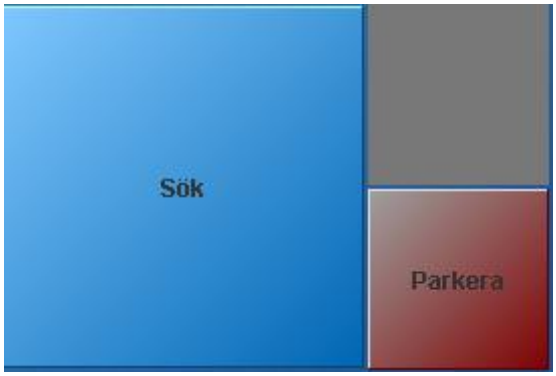
4.4.1 Utseendet på knapparna

I den valda prototypen var ett av målen att gränssnittet skulle vara så flexibelt som möjligt. För att detta även skulle gälla knapparna så utformade vi dessa så att det fanns möjlighet att både kunna ändra form, position och färg.

Knapparna (se Figur 4.3) skapades av den inbyggda ”JButton”-klassen i Java. För att sedan ändra utseendet på knapparna till det som var angivet på prototypen krävdes det att vi skapade en egen knapp-klass som ärvde av klassen ”JButton”. Utifrån den nya klassen kunde vi därefter utforma en egen design på knappen genom att skapa en ny komponentritare[1]. Det viktigaste som skulle göras designmässigt var att skapa en färgövergång mellan två färger i en diagonal position, vilket gjordes genom att rita upp en tom yta för att sedan lägga på rätt färgövergång och integrera det med knappens ritkomponent. För att ändra på en knapps egenskaper krävdes det att programmet skulle kunna minnas tidigare inställningar. När programmet startas så läses XML-filen in via objektet ”XMLExtractor” som vi skapat. Varje knapps egenskaper extraheras och läses in till ett nytt JButton-objekt som sedan ritas upp på rätt plats i dess respektive JPanel[15]. Varje knapp har också möjligheten att skapas som inaktiv och fungerar då som ett mellanrum mellan två knappar för att kunna minska

[1] En funktion i varje grafisk komponent som har till uppgift att bygga upp och visa komponentens grafiska representation.

eventuella feltryckningar.
När en knapp har en inaktiv status så ritas knappen om för att ge den ett mer inaktivt intryck. I vårt fall betydde det att göra knappen så platt som möjligt, dvs. vi tog bort alla ramar runt knappen så att endast en rektangel med rätt angiven färg ritades upp. Den inaktiva knappen placeras ut på angiven position för att fungera som en avgränsare mot andra knappar.
Knapparna har blivit utformade så att gränssnittet i sig ej kan ändra på en knapps egenskaper under tiden kassan exekveras, då dessa funktioner är tänkta att hanteras av en gränssnitts-redigerare i framtiden.



Figur 4.4:Knappar

I Figur 4.4 ser vi ett exempel på hur olika egenskaper kan påverka knapputseendet. Knappen längst upp till höger är inaktiv.

4.5 Egenskaps-filen

Klassen ”Layout” används för att läsa in en egenskaps-fil[2] som innehåller inställningar för hur layouten för själva huvudsidan ska se ut. Det vill säga, om användaren är vänster- eller högerhänt ska denne kunna flytta layouten så det passar dennes behov bäst. Det går även att ändra grundfärgen i programmet via denna fil. Denna egenskaps-fil är även den enkel att modifiera om mer funktionalitet skulle behövas läggas till. Denna fil kommer kunna modifieras med hjälp av en gränssnittsredigerare som vi pratar mer om i avsnitt 5.3.1.

4.6 Flikarna

I prototypen (se Figur 4.1) har flikarna ett utseende som består av att översta högra hörnet är diagonalt klippt, vilket ger dem ett utseende liknande en ”tabb” inom diverse olika program.

[2] En fil som tillhandahåller värden för objekt eller variabler.

Förutom detta skulle även varje flik ha en färg och vara helt platt, dvs. helt utan kanter. Till sist så skulle också varje knapp överlappa den andra.

Flikarna implementerades som vanliga JButtons då dessa har egenskaper för nedtryckning respektive uppsläpp, vilket behövdes för att kunna visa grafiskt om man har klickat på en flik eller inte. Därefter lades en komponentritare till för att kunna skapa det speciella utseende på knappen. I komponentritaren så skapade vi en polygonyta[3], vilken formades till det rätta utseendet och därefter lade vi till färg. Dock så blev det väldigt svårt att se hur flikarna såg ut, vilket dock rättades till genom att rita till en tunn kantlinje som gick runt hela polygonytan.
Vi fick även lägga till en kantutjämning på varje knapp för att få polygonytan så mjuk som möjlig.

För att kunna få flikarna att överlappa varandra på rätt sätt krävdes det att varje komponent ritades från höger till vänster med en inskjutning på ca 30 % av flikens storlek (för att det skulle se ut precis som på prototypen). Detta löstes med att lägga till en ny layout-klass vid namn ”OverlapLayout”. Layouten skapar den sista komponenten först och lägger den längst ut till höger och lägger därefter på komponenter som överlappar den föregående komponenten med hjälp av en komponentritare.
Till sist så krävdes det att varje flik skulle ha en egen nyans av förgrunden där den första fliken hade den ljusaste nyansen och den sista fliken hade den mörkaste nyansen.
För att kunna skapa denna färgövergång mellan knapparna så hämtades föregående fliks röd, grön och blå. Därefter adderades varje värde med 20 och sedan sattes de nya värdena till den nya fliken. Detta gav en tillräckligt jämn färgövergång mellan knapparna för att de skulle synas, men inte för mycket så att kontrasten mellan knapparna blev för stor.

4.6.1 Flikarnas menyval

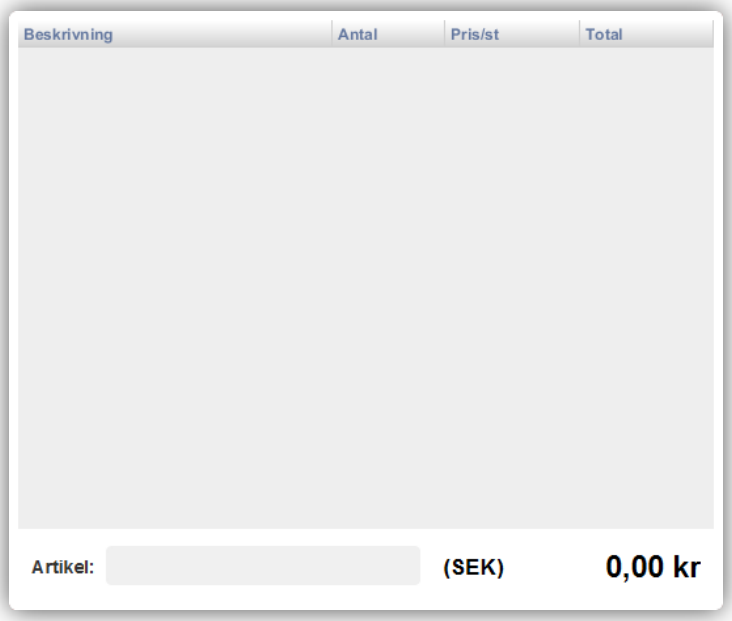
Vid klick på flikarna visas en tillhörande meny för användaren där ett visst antal menyval kan göras. Antalet är beroende av vilken flik det gäller. Tanken var från början att använda Javas egna klass för att skapa menyer som kan visas, men vi ansåg att dessa skulle bli svåra att modifiera rent utseendemässigt, så vi skapade helt egna istället. Detta gjorde vi genom att använda JDialog[16] som fungerar lite som ett pop up-fönster. Ramen runt fönstret skalades bort och sedan lades paneler till för att motsvara varje menyval. Varje panel använder s.k.

[3] Polygoner är ett samlingsnamn för geometriska figurer i form av slutna kurvor bestående av ett antal sträckor, detta skapar då en s.k polygonyta.

lyssnare för att upptäcka om användaren klickar på just den panelen i menyn. Tack vare att flikarna redan fanns där kunde man hämta både positionen och bredden på fliken och med lite modifikation beräkna var detta fönster sedan skulle visas. I och med att fönstret visas precis ovanför rätt flik och med rätt storlek, får man känslan av att det är en meny som expanderas.

4.7 Produktfönstret

Produktfönstret (se Figur 4.5) är den enda grafiska komponent som togs från Stamfords nuvarande gränssnitt, av den anledningen att den matchar resterande design väldigt bra och är dessutom väldigt enkelt att förstå sig på som användare. Bakgrunden på produktfönstret är vit och får sin avrundade form utav klassen ”SmoothJPanel.java”, vilken även fanns i Stamfords nuvarande gränssnitt. Klassens enda funktion är att skapa rundade hörn och därefter applicera en kantutjämning för att göra hela panelen mjuk i formerna. Därutöver finns det en tabell (en s.k. ”JTable”[17]) som visar upp hela listan med tillagda produkter. Nedanför tabellen finns en textruta för att snabbt söka fram en produkt med hjälp av namn eller produkt-id. Till höger om sökrutan visas det vilken valuta kassan är inställd på och även totalbeloppet på produkterna i tabellen.



Figur 4.5: Produktfönstret

4.7.1 Tabellen

För att tabellen ska kunna visa några produkter krävs en lista där varje produkt kan hanteras. Den lista som används i detta fall heter ”DefaultTableModelNotEditable” och är en expandering av listan ”DefaultTableModel”[18] som finns med i Javas API. Skillnaden mellan de två är att den översta är inställd på att ej kunna ändra på cellernas värden när de väl ligger inuti listan.

Genom att man skapar en lista och kopplar den till tabellen så kan man nu både skapa celler och kolumner. I det här gränssnittet har vi skapat nya visuella utseenden på dessa objekt för att tabellen lättare ska kunna smälta in i det övriga utseendet på gränssnittet.

4.7.2 Kolumnerna

Kolumnernas utseende (se Figur 4.6) byggs upp av en inbyggd klass som heter ”DefaultTableCellRenderer”[19]. Genom att man skapar en expandering av denna klass kan man ändra utseendet på kolumnerna i en tabell. Den klassen kallar vi för ”HeaderRenderer.java”.

Klassen hämtar först varje kolumn ur tabellen och läser av dess storlek. Därefter skapas en ny rektangel med exakt samma storlek där man skapar den färgövergång som är angiven, vilket i vårt fall är ljust grått till vitt. När rektangeln är skapad så ritas den upp i tabellen och nästa kolumn skickas in. Detta händelseförlopp pågår ända tills alla kolumner har ändrats.

Texten i varje kolumn fick även en blåaktig färg genom klassen för att kunna stå ut från resterande text i produktlistan.

4.7.3 Cellerna

Cellerna har även ett egenkomponerat utseende som skapades genom en expandering av klassen ”DefaultTableCellRenderer”. I detta gränssnitt är endast textens placering ändrad i varje cell för att få ett mer lättöverskådligt resultat när dessa skapas. Det finns dock stöd för att kunna ändra utseendet på alla celler, men i detta fall behövs det inte då resterande gränssnitt har tillräckligt med effekter. Varje cell har helt enkelt det standardutseende som finns i tabellen från början.

Figur 4.6: Cellerna i produktfönstret

4.8 Summering

Vi har utgått ifrån den prototyp som valdes i kapitel 3 och därefter implementerat alla grafiska komponenter för att få ett likadant utseende på implementationen som på prototypen. De komponenter som krävde mest tid var produktfönstret, knapparna och flikarna, vilka alla behövde förändra utseendet helt utifrån vad som var standardformen för varje komponent.
Knapparnas olika egenskaper styrs av en XML-fil som lagrar all den datan. När filen därefter läses in extraheras datan och skapar varje knapp med sina speciella egenskaper.
Det är inte endast knapparna som har egenskaper lagrade, utan även layouten på programmet är lagrad i en egenskapsfil där varje panel kan flyttas och även inaktiveras utifrån vad som ställs in i egenskapsfilen. Sist så implementerades även designmönstret MVC på gränssnittet för att dels skapa en testmiljö och dels göra det lättare att integrera gränssnittet med ECS.

5 Resultat och utvärdering

Att gå från prototyp till färdigt gränssnitt har varit en intensiv resa där vi stötte på en hel del problem men tillslut blev det färdiga gränssnittet väldigt likt prototypen som valdes. Vi utgick ständigt från hur komponenterna i prototypen såg ut och försökte återskapa utseendet hos dessa i det färdiga gränssnittet. Detta gjorde att vi fick utforska den visuella delen av Java mer ingående.
Vi diskuterar ingående i avsnitt 5.1 om hur utseendet på vårt gränssnitt blev där vi ser till vilka krav och önskemål som ställdes från Stamford. I samma avsnitt jämför vi även vårt färdiga gränssnitt med både Stamfords prototyp och deras nuvarande gränssnitt. Vidare i avsnitt 5.2 diskuteras implementationen och funktionaliteten i gränssnittet och även där jämförs vårt gränssnitt med Stamfords prototyp och nuvarande gränssnitt. Därefter beskriver vi i avsnitt 5.3 vilka problem vi stött på och hur vi löste dessa. Därefter gå vi in på framtida arbete i avsnitt 5.4 där vi ser på hur utvecklingsmöjligheterna är för gränssnittet och alternativa tillägg.

5.1 Gränssnittsdesign

Gränssnittet hade flera krav på sig då det skulle vara mer användarvänligt, ha bättre utseende, vara anpassat för pekskärmar och vara enkelt att förändra.
Användarvänligheten förbättrades i vårt gränssnitt genom att det finns fler knappar, vilket gör att knapptryckningarna för en viss händelse är färre än tidigare. Dessutom förbättrades användarvänligheten genom att utseendet är enkelt och konsekvent.
Vi skapade också färgövergångar vilket gjorde utseendet modernt. Anpassningen till pekskärmar skapades genom att knapparna fick storlekar som passade in på den standard som ISO och ANSI fastställt. Anpassningen för förändringar i gränssnittet löstes med hjälp av egenskapsfilen och XML-filen, vilka gjorde det möjligt att ändra färg och utseende på gränssnittet.

Den prototyp vi skapade har några skillnader mot det färdiga gränssnittet. En sådan skillnad är att alla knappar har ett mellanrum i färdiga gränssnittet, vilket kom som ett önskemål då vi visade upp prototypen. Detta gjorde dock att alla knappar fick bli mindre för att alla skulle få plats. Knapparna visade sig dock vara i helt rätt storlek.
En annan ändring som gjordes var att utseendet på kolumnerna i produktfönstret då

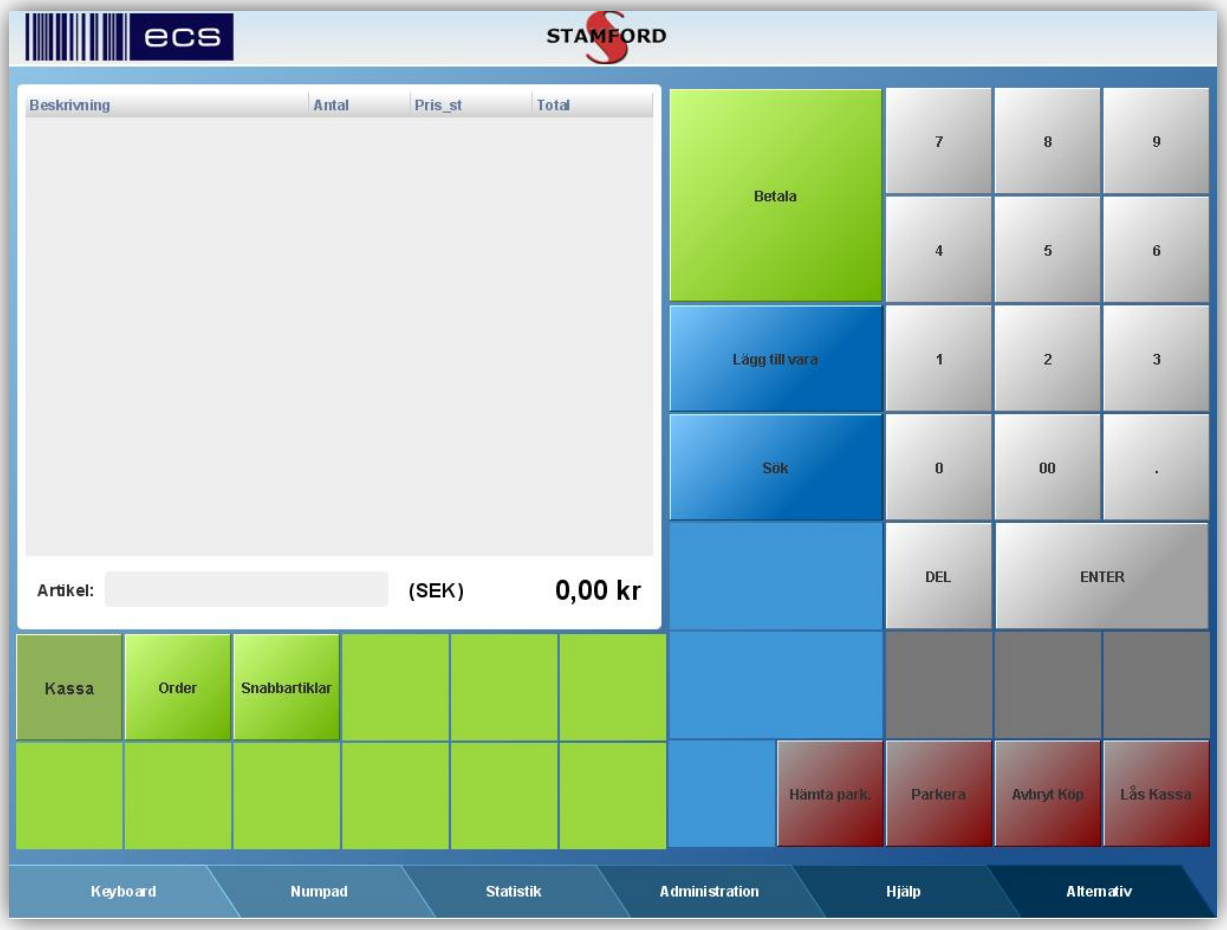
prototypens utseende på dessa inte passade med resterande komponenter om man ser det ur en grafisk synvinkel. Dessa ändrades till en gråtonad färg vilket smälte in utmärkt med de övriga komponenterna i produktfönstret.

Om vi ser tillbaka på Stamfords prototyp (se avsnitt 2.7) och jämför med det färdiga gränssnittet, så har flera förbättringar gjorts rent grafiskt. Den första stora förändringen är hur vi, i vårt färdiga gränssnitt, har lyckats utnyttja hela programytan och just av den anledningen ser vår prototyp större ut om man ser till längd och bredd.

En annan stor skillnad vid en jämförelse av Stamfords prototyp och vårt färdiga gränssnitt är att vårt gränssnitt har en större flexibilitet, vilket gör att samma gränssnitt kan utformas olika för olika behov och önskemål. Denna flexibilitet fanns icke i Stamfords prototyp då den var till mestadels uppbyggd på bilder.

Även om vi har försökt att förbättra gränssnittet utifrån de tidigare versionerna så har vi även använt av oss vissa delar av tidigare gränssnitt. En av dessa komponenter är produktfönstret som redan fanns i Stamfords nuvarande gränssnitt av kassan. Då denna komponent både fanns med i det nuvarande gränssnittet och deras egen prototyp så var det ett självklart val att vi även skulle använda den. Dessutom har vi även tagit med idén om att kunna kategorisera funktioner på samma sätt som i det nuvarande gränssnittet. Ett exempel är när man klickar på ”Order”-knappen ändras ”WallButtonLayout” till orderrelaterade knappar. Just av den anledningen skapades kategoriknapparna (se de gröna knapparna i figur 5.1). Den sista liknelsen med det nuvarande gränssnittet är att båda går i blåa nyanser som standard, vilket är till för att ge nuvarande användare ett välbekant gränssnitt.

Det färdiga gränssnittet var över förväntan och vi är väldigt nöjda över resultatet just för att vi har skapat ett gränssnitt som känns bekvämt och välkomnande.



Figur 5.1 Slutresultat av gränssnittet

5.2 Implementationsresultatet

Förutom det visuella så utvecklade vi även nya funktioner för att få gränssnittet så pass flexibelt som vi ville ha det från början. Det färdiga gränssnittet var väldigt bra då gränssnittet har många fler möjligheter jämfört med föregående gränssnitt. Dessa är främst att vi utvecklat egenskapsfilen och XML-filen som sköter om positioner och egenskaper för varje objekt, vilket gjorde det mycket enkelt att förändra utseende på komponenterna utan att i själva verket förstå sig på den underliggande koden.

En annan fördel med vårt gränssnitt är att endast en bild används i gränssnittet. För det första skapade vi möjligheten att kunna förändra varje komponent och för det andra så sparas

utrymme i både programmet och på det lagringsutrymme som används. Detta gör att vårt program är snabbare i längden då dessa bilder aldrig behöver läsas in och ritas om.

Genom att en M-V-C-modell har använts har vi gjort det enkelt att integrera detta gränssnitt med resterande system.

Det var en del arbete med att genomföra de idéer vi hade för gränssnittet då det fanns vissa områden i Java där vi inte hade någon erfarenhet. Vi fick läsa oss till hur det skulle lösas, vilket både löste våra problem och även lärde oss en hel del om hur Javas grafiska komponenter fungerar i grunden.

Något som vi ej hann med under utvecklingstiden var att skapa den gränssnittsredigerare som vi berättar om i nästa avsnitt. Denna redigerare skulle ha kopplas ihop med gränssnittet som ett externt tillägg och gjort det ännu enklare att redigera utseendet på gränssnittet.

5.3 Problem

Under implementationen av gränssnittet har vi stött på flera både större och mindre problem. De flesta problemen har oftast löst sig med hjälp av information som funnits på Internet eller genom att vi har rådfrågat någon. Dock fanns det några få problem som var svårare att lösa och krävde sin tid för att lösas. Följande var de problem som tog mest tid:

5.3.1 Byte av ”WallButtonLayout”

Ett av de mest tidsödande problemen var att implementera funktionen att kunna ändra ”WallButtonLayout”. Problemet var att ”WallButtonLayout” inte ändrades när vi tryckte på en kategoriknapp, trots att XML-filen lästes in korrekt. Med hjälp av utskrifter vid felsökandet kunde vi konstatera att rätt knappar skapades då vi tryckt på en kategoriknapp. Det visade sig att funktionerna ”removeAll” och ”repaint” som finns i ”JPanel”-objektet inte var tillräckliga för att rensa de tidigare komponenterna (knapparna) som fanns lagrade i ”WallButtonLayout” och rita ut de nya. Vi fann till slut funktionen ”doLayout” som tvingar en JPanel att lägga ut sina nya komponenter efter att de har rensats med ”removeAll”. Därefter anropas ”repaint” för att uppdatera grafiken i ”WallButtonLayout”. Detta löste problemet att kunna använda kategoriknapparna för att byta ”WallButtonLayout”.

5.3.2 Flikarnas utseende

Prototypen som vi valde hade flikar som hade det översta högra hörnet avkapat och dessutom överlappade flikarna varandra. Det första problemet vi hade var att skapa den rätta formen på flikarna om man utgick ifrån en ”JButton”. Avkapningen av knapparna var svårt att skapa då standardformen för en ”JButton” är rektangulär. Problemet löstes genom att ta bort den ursprungliga formen och skapa en ny utifrån en polygonyta. Det andra problemet som inträffade var när flikarna skulle bli överlappade. I Java finns ingen layout som stöder att komponenter överlappar varandra. Vi testade då att se varje flik en fast position men detta skapade ingen överlappande effekt. Lösningen var då att vi fick skapa en ny layout,”OverlapLayout”, som ritade ut varje komponent med en överlappning från höger till vänster. Med hjälp av dessa lösningar kunde vi skapa flikarna med samma utseende och placering som på prototypbilden.

5.3.3 MVC

Vid implementationen av MVC upptäckte vi att det fanns flera olika tillvägagångssätt för att implementera detta designmönster. Det generella sättet är att ha en eller flera vyer och en modell. Vårt fokus låg hela tiden i gränssnittet dvs. vyn. Vi hade därför problem med att se en framtida koppling till en modell eftersom vi inte hade någon. Då vår tid under projektet var ganska begränsad valde vi att skapa en fiktiv modell att knyta till MVC, istället för att bearbeta hela ECS-koden. Den befintliga koden för ECS var väldigt svår att förstå då vi aldrig tidigare varit insatta i ECS, och vi kände att detta gick utanför ramarna för vad vi skulle göra under detta projekt. Som vi beskrev i avsnitt 4.3 föll valet på Observer/Observable för att uppnå MVC.

5.4 Framtida arbete

Genom att de krav som ställdes på gränssnittet är uppfyllda så är även gränssnittet komplett. Däremot så finns de mindre ändringar som skulle ha gjorts om tiden räckte till. En av dessa andringar är att rutan längst upp i gränssnittet (se Figur 5.1) just nu är en bild. Eftersom vi har använt bilder i den resterande delen av gränssnittet skulle det även vara bra om den enda bilden även skapades när gränssnittet exekverades. Något som även skulle behöva utvecklas är att skapa enklare kopplingar till ett system för att göra en installation av gränssnittet mycket enklare. Dessa två ändringar var dock varken krav eller önskemål ifrån Stamford utan är egna synpunkter från oss som utvecklade detta gränssnitt.

Utifrån dessa förbättringar så diskuterades det tidigt om att skapa ett tilläggsprogram där man kunde skapa och förändra utseendet och placeringarna på komponenter. Detta tilläggsprogram som vi kallar för en gränssnittsredigerare förklaras nedan.

5.4.1 Gränssnittsredigerare

En viktig aspekt vid framtagandet av vårt nya gränssnitt var att erbjuda ett externt tilläggsprogram, en gränssnittsredigare, för att kunden enkelt skulle kunna ändra utseendet på gränssnittet. Denna var möjlig att utveckla tack vare att grunden till gränssnittet ligger i XML-filen som talar om var knapparna skall placeras och hur de ska se ut, och i egenskaps-filen där grundfärgen för gränssnittet sätts och där man även kan ställa in om man är höger- eller vänsterhänt. Dessa filer är väldigt enkla att redigera och vi har undvikit att lägga in för mycket inställningar i koden. Att erbjuda möjligheten att ändra färgtema på sitt gränssnitt ansåg vi vara både väldigt enkelt och kul för kunden. Många företag kan ha vissa färger starkt associerade till sina namn och då kan det kanske vara intressant för dem att få välja en viss nyans efter eget tycke. Vi ser inga som helst hinder kring detta, förutom den aspekten att vissa färgskalor inte är lika användarvänliga och tydliga som andra.

Tanken var att vi bara skulle utveckla ett enkelt program med ett tydligt gränssnitt som presenterar alla inställningarna som kunden kan göra. T ex kunde alternativet att välja på höger- och vänsterjusterat gränssnitt ha gjorts med en enkel ”bock” i en kryssruta. Programmet skulle då ha ändrat koden som finns i egenskaps-filen automatiskt och vi hade på så sätt möjliggjort inställningsmöjligheter även för den som inte är inbiten datorfantast. På samma sätt har vi idéer om knapparna som skapas i XML-filen. Det handlar till stor del bara om att ändra vanlig text och siffror för att påverka knapparnas utseende. Vi hade tänkt oss att man skulle kunna få fram ramen till hela gränssnittet, fast mer eller mindre tomt på knappar. Därefter skulle man kunna lägga till knappar en efter en och fylla i information som namn, färg och storlek. Med knappen utlagd inuti ramen skulle man sedan kunnat ”dra-och-släppa” knappen till den plats i gränssnittet där man vill ha den. Ångrar man någonting kring t ex färgval eller storlek är det bara att klicka på knappen och få fram den redigerbara informationen igen.

Självklart är det inte meningen att man ska behöva skapa alla knapparna själv från starten, utan det handlar om att kunden bara ska kunna ändra några småsaker som de inte är nöjda

med i standard-uppsättningen av knappar. Det skulle kunna fungera på samma sätt som när Stamford ECS Retail AB erbjuder specialanpassade tangentbord efter kundens behov. D.v.s. att Stamford själva skulle kunna använda denna redigerare för att anpassa gränssnittet till kunden efter deras önskemål, eller att överlåta det till kunden.

5.5 Summering

Vi har i det här kapitlet jämfört vårt gränssnitt med Stamfords prototyp och nuvarande gränssnitt där vi främst ser till vilka förbättringar vi har gjort. Dessutom har vårt gränssnitt jämförts med kraven och önskemålen som tidigare ställdes av Stamford, vilka har varit grunden för utseendet och funktionaliteten hos gränssnittet.
Vi har även diskuterat hur implementationen och hur de nya funktionerna kan hjälpa både företag och utvecklare genom gränssnittets flexibla sätt att kunna ändras. Vi tog även upp de problem vi hade under vägen och även vad vi gjorde för att lösa dessa problem. Gränssnittsredigeraren var något som vi gärna ville utveckla om vi hade haft mer tid. Denna redigerare skulle då ha möjligheten att enkelt kunna ändra på både layouten av gränssnittet och även varje knapps egenskaper.

6 Slutsats

Gränssnittet som vi hade i uppgift att ta fram åt Stamford har i slutändan blivit fullt implementerat att fungera som en testprototyp till ECS. Den uppstod genom att vi lyckades skapa ett färdigt gränssnitt utifrån den valda prototypen som vi vid ett tidigt skede tog fram och presenterade. Under projektets gång har vi lyckats uppfylla alla kraven från Stamford genom att ha skapat ett anpassningsbart och funktionellt gränssnitt, som dessutom både vi och Stamford tycker är modernt till utseendet. Det nya gränssnittet är testat och bekräftat användbart på en pekskärm, vilket var en av de största förändringarna gentemot Stamfords nuvarande gränssnitt.

7 Referenser

[1] Stamford. http://www.stamford.se/Mftg/referenser.asp?cat=referenser 2010-05-10

[2]Stamford.http://www.stamford.se/Mftg/system.asp?cat=system&pageid=ecsretail&prod=ECS 2010-05-10

[3] David Benyon, Phil Turner och Susan Turner. *Designing Interactive Systems*, 1:10, 2005

[4] David Benyon, Phil Turner och Susan Turner. *Designing Interactive Systems*, 2:31, 2005

[5] David Benyon, Phil Turner och Susan Turner. *Designing Interactive Systems*, 1:32, 2005

[6] David Benyon, Phil Turner och Susan Turner. *Designing Interactive Systems*, 1:24, 2005

[7] GUIFX http://blog.guifx.com/2009/08/12/touchscreen-button-dimensions-and-spacing-hell-if-phi-know/.

[8] Oracle/Sun http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Frame.html 2010-04-27

[9] Oracle/Sun http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JButton.html, 2010-04-26

[10] Linköping universitet. http://www.ida.liu.se/~TDDB58/timetable/DiagramMVC.png 2010-05-10

[11] Oracle/Sun http://java.sun.com/j2se/1.4.2/docs/api/java/util/Observer.html 2010-05-10

[12] Oracle/Sun http://java.sun.com/j2se/1.4.2/docs/api/java/util/Observable.html 2010-05-10

[13] Oracle/Sun http://java.sun.com/j2se/1.4.2/docs/api/java/util/ArrayList.html 2010-05-10

[14] W3Schools. http://www.w3schools.com/xml/default.asp, 2010-04-27

[15] Oracle/Sun http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JPanel.html 2010-05-10

[16] Oracle/Sun http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JDialog.html 2010-04-27

[17] Oracle/Sun http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JTable.html 2010-05-10

[18] Oracle/Sun http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/table/DefaultTableModel.html 2010-05-10

[19] Oracle/Sun http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/table/DefaultTableCellRenderer.html 2010-05-10