TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

Produktkonfigurator

Lucas Björkman

Jimmy Erixon

EXAMENSARBETE 2010

Datateknik

TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

PRODUKTKONFIGURATOR

PRODUCT CONFIGURATOR

Lucas Björkman

Jimmy Erixon

Detta examensarbete är utfört vid Tekniska Högskolan i Jönköping inom ämnesområdet Datateknik. Arbetet är ett led i den treåriga högskoleingenjörsutbildningen.
Författarna svarar själva för framförda åsikter, slutsatser och resultat.

Examinator: Inger Palmgren

Handledare: Karl Hammar

Omfattning: 15 hp (grundnivå)

Datum: 2010-06-24

Arkiveringsnummer:

Abstract

The report features the creation of a product configurator and several suggestions on how it can be improved. The assignment was given by System Andersson to two students at Jönköpings Tekniska Högskola who implemented this as their examination paper.

The report brings up possible solutions for a product configurator and demonstrates a result that origins from these solutions. The product configurator in the report consists of two parts.

First part is the configurator where the user creates new products from a set of articles. The configurator is designed to sort the articles in a way that only compatible articles can be connected to the product. This is achieved by a ”drag and drop” user interface.

Another essential part of the application is the compatibility part, which helps the user to define compatibility between articles. This can be done simultaneously while configurating a product. The user can choose to make two or more articles either compatible with each other, or entire categories.

In the final chapter of the report there is a discussion about the choices that has been made during the project and how things could have been done better.

Sammanfattning

Rapporten presenterar skapandet av en produktkonfigurator och flera förslag på hur den kan förbättras. Uppdraget gavs av System Andersson till två studenter på Jönköpings Tekniska Högskola som genomförde detta som sitt examensarbete.

Rapporten diskuterar möjliga lösningar på en produktkonfigurator och visar ett resultat som utgår från dessa lösningar. Produktkonfiguratorn i rapporten består av två delar.

Dels konfigurationsdelen som hjälper användaren att skapa nya produkter utifrån valda artiklar. Artiklarna sorteras automatiskt av produktkonfiguratorn så att endast kompatibla artiklar kan sättas ihop. Detta sker med ett ”drag and drop” gränssnitt.

Dessutom innehåller produktkonfiguratorn en kompatibilitetsdel, vars uppgift är att underlätta vid definierandet av kompatibilitet mellan artiklar. Detta kan göras löpande under tiden en produkt konfigureras om så önskas. Användaren kan välja på två eller flera artiklar som ska vara kompatibla med varandra, eller hela kategorier.

Sist i rapporten diskuteras de val som gjorts under arbetets gång och möjliga förbättringar som kan göras på produktkonfiguratorn.

Nyckelord

Produktkonfigurator

Konfiguration

Kompatibilitet

Windows Presentation Foundation

C#

Microsoft SQL-Server 2008 Express

Innehållsförteckning

Figurförteckning

1 Inledning

Antag att du skall konstruera en helt ny produkt. Till din hjälp har du ett bestämt antal artiklar. Vissa känns igen medans andra är nya och okända. Du kan nu antingen sätta ihop de artiklar du vet passar med varandra eller börja experimentera och pröva de okända artiklarna för att se vilka de passar ihop med. Antag att det skulle finnas ett verktyg tillgängligt som visste vilka artiklar som passade ihop innan du började bygga. Om du tog upp en artikel så blev alla artiklar som inte passar med den osynliga. Du kan helt enkelt inte göra fel. Det är vad en produktkonfigurator gör. Den sorterar artiklarna åt dig så att du inte kan göra fel.

Den här rapporten redogör för det examensarbete som gjordes på Jönköpings Tekniska Högskola åt System Andersson. Rapporten kommer att visa på resonemangen bakom en produktkonfigurator, dess skapande och utvärdering av den samma.

1.1 Bakgrund

System Andersson, ett företag i Jönköping med inriktning på att skapa och erbjuda administrationslösningar och rapportunderlag i form av IT-system åt små och medelstora företag i Sverige, har under 3-4 år arbetat med en ny produktserie. Som en del av den önskar System Andersson kunna erbjuda en produktkonfigurator.

System Andersson kom med en förfrågan till den datatekniska utbildningen på Jönköpings Tekniska Högskola om det fanns intresse för att skapa en produktkonfiguratorprototyp i Windows Presentation Foundation – WPF.

En produktkonfigurator kan hjälpa företag att designa nya produkter och även till att se vilken åtgång av material en viss produkt kräver. För att klara detta måste artiklars inbördes kompatibilitet definieras och användas i konfiguratorn för att enkelt kunna sortera fram möjliga förslag på artiklar som passar ihop.

1.2 Syfte och frågeställningar

Relationerna mellan artiklar måste hanteras på något sätt så att kompatibilitet kan finnas och skapas. Användargränssnittet måste vara enkelt och användarvänligt utan att tumma på produktkonfiguratorns potential.

Vår målsättning är först och främst att skapa en fungerade konfigurator. Vi har nästan inte alls programmerat för en grafisk miljö under utbildningen, utan vi har i princip utan undantag skrivit program för Kommandotolken i Windows. Vår egna målsättning innebär därför också att lära oss arbeta med den grafiska miljön WPF.

Till sist innebär det en utmaning att utan tidigare erfarenhet utröna logiska villkorsförhållande mellan artiklar för att få önskad kompatibilitet mellan dessa.

System Andersson hade följande tre krav på produktkonfiguratorn:

- Endast kompatibla artiklar ska kunna kopplas till varandra.
- Användargränssnittet ska vara av typen ”drag and drop” och artikellistan skall ligga till höger och komponeringsytan skall ligga till vänster.
- Vissa artiklars attributvärden är inte definierade, till exempel attribut som längd, bredd och vikt. Produktkonfiguratorn ska i de fall dessa artiklar är inblandade själv kunna räkna ut antalet kompatibla artiklar som behövs för att produkten ska bli komplett, beroende på vilket värde attributen får i just den konfigurationen. Om en garageport ska byggas kan antalet gångjärn som krävs bero på längden av garageporten. Längden definieras då vid konfigurationen av produkten.

Produktkonfiguratorn ska kodas i Visual Studio 2010 i C# som en WPF applikation och artiklarna lagras i en SQL-server 2008 Express databas.

1.3 Avgränsningar

Rapporten kommer inte gå in på produktkonfiguratorer i största allmänhet utan bara redogöra för skapandet av den produktkonfigurator som presenteras i rapporten och förslag på förbättringar för den samma.

1.4 Disposition

Teoretisk bakgrund tar upp de tekniska verktyg och metoder som användes under arbetet. Här beskrivs användningsområden för de olika verktygen och vissa kodexempel ges.

Under genomförande visas först designen eller resonemangen bakom arbetet, för att sedan visa hur detta användes under implementeringen det vill säga omvandlandet av strukturerna till kod.

I Resultatet presenteras sedan en produktkonfigurator och dess funktioner.

Till sist presenteras de slutsatser och förslag på förbättringar i kapitlet Slutsats och Diskussion.

2 Teoretisk bakgrund

2.1 Windows Presentation Foundation

Windows Presentation Foundation (även kallat WPF) är Microsofts senaste presentationssystem som är till för att ge Windows-applikationer ett rejält visuellt lyft. Tanken är att man ska utnyttja den moderna hårdvaran som klarar av kraftfullare effekter. Grafiken är vektorbaserad och det medför att applikationerna som är byggda med den här tekniken ser bra ut oberoende av bildskärmens upplösning [1].

I WPF skiljer man på utseende och beteende. För att märka upp objekt och bestämma applikationens utseende använder man sig av XAML, som kommer tas upp i ett senare avsnitt. Bakom fasaden som är uppbyggd av XAML styr man objektens beteende med hjälp av kod som skrivs i C#. Den här delningen gör att man kan spara både tid och pengar vid utvecklingsarbetet. Till exempel kan designers arbeta med applikations utseende samtidigt som programmerarna kodar [1].

Programmerare använder helst Microsoft Visual Studio som utvecklingsverktyg. Visual Studio 2010 släpptes nyligen och nu kan man bland annat arbeta över flera bildskärmar [8]. Designers föredrar oftast att arbeta i Microsoft Expression Blend eftersom man som icke-programmerare känner sig hemma i det verktyget. Det fokuserar en hel del på det grafiska och man behöver inte kunna så mycket kod [9].

I WPF använder man sig utav eventhandlers för att ta hand om de händelser som sker under en applikations körning. Ta till exempel eventhandlern MouseButtonDown som detekterar när man klickar med musen. Beroende på var man klickar i applikation så kommer olika saker att hända [1].

2.2 XAML

XAML står för Extensible Application Markup Language och bygger på språket XML. Tanken är att göra det enklare för utvecklaren att skapa ett användargränssnitt. Man deklarerar objekt genom att ge dem ett unikt namn samt eventuella värden. För att komma åt objekt från koden använder man det namn som man tilldelat i XAML-delen. Det går även att deklarera objekt direkt i koden [1].

För att vidare förstå sambandet mellan XAML och koden bakom se Figur 1.

```
<Window
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Fönster med en knapp"
       Height="100" Width="250">
    <Button Name="knapp" Click="knapp_Click">Klicka här!</Button>
</Window>
```

Figur 1. I kodexemplet skapas en enkel applikation med ett fönster som innehåller en knapp.

Nu har ett fönster skapats som innehåller en knapp. För att komma åt den här knappen används namnet vi döpt den till. I det här fallet döptes den till ’knapp’. När man klickar på knappen kommer ’knapp_Click’ att anropas (se Figur 2).

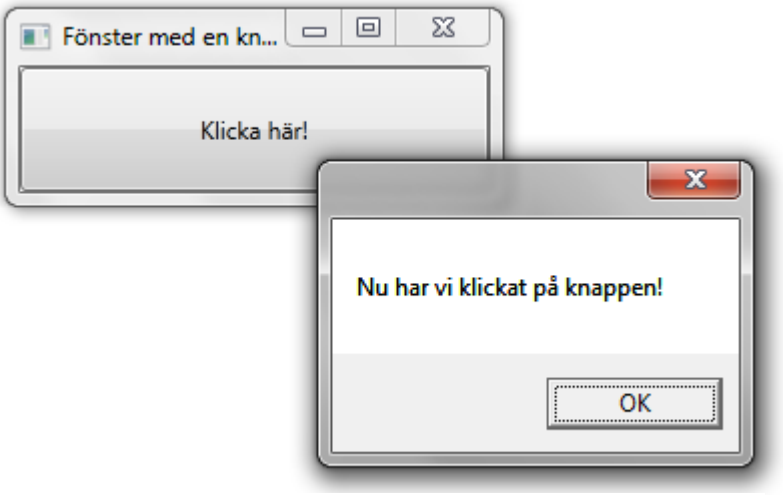
```
using System.Windows;

namespace Exempel
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            //Det här kommandot krävs för att förena
            //layouten med koden.
            InitializeComponent();
        }

        void knapp_Click(object sender, RoutedEventArgs e)
        {
            // Visa meddelande när man klickar på knappen
            MessageBox.Show("Nu har vi klickat på knappen!");
        }
    }
}
```

Figur 2. Applikationen initieras och knappen tilldelas en händelse som sker vid musklick.

Under knapp_Click är det nu definierat vad som ska hända när man trycker på knappen. I det här fallet ska det dyka upp en meddelanderuta som innehåller frasen ’Nu har vi klickat på knappen!’. I Figur 3 kan man klart och tydligt se vad som händer om man förverkligar koden.



Figur 3. Uppe till vänster syns knappen som skapades och nere till höger syns meddelanderutan som kommer upp när man klickar på knappen.

2.3 .NET-Framework

.NET-Framework är Microsofts plattform som är till för att hjälpa utvecklare att skapa applikationer som både är grafiskt tilltalande och kommunikationsmässigt säkra. Utvecklingen av ramverket började strax innan millenieskiftet och hette då New Generation Windows Service. Första betan släpptes år 2000 men det dröjde 2 år innan de släppte den första officiella versionen som kallades .NET-Framework 1.0. Ramverket består av följande vitala delar[2]:

- Common Language Runtime
- Base Class Library

2.3.1 Common Language Runtime

Common Language Runtime, CLR, är en virtuell maskin som exekverar kod separat från hårdvaran. Detta gör att användaren kan vara säker på att den exekverbara filen inte får obehörig tillgång till privata filer och dylikt.

CLR är alltså gjort för att öka prestandan och säkerheten på applikationer. Den sköter till exempel minneshanteringen genom att ta hand om referenser till objekten. Detta gör att de två vanligaste mjukvaruproblemen, minnesläckor och felaktiga referenser, förhindras per automatik.

En annan smart sak som CLR medför är att en utvecklare kan skapa sina applikationer i vilket programmeringsspråk som helst och fortfarande använda sig utav CLR. Detta gör till exempel att utvecklingsprocessen går snabbare eftersom man får tillfälle att koda i det språk man behärskar bäst. Att det har stöd för olika språk gör att CLR även kommer vara användbart i framtiden [3].

2.3.2 Base Class Library

I det här biblioteket kan man hämta färdigbyggda metoder som kan ta hand om diverse uppgifter. Det kan vara till exempel stränghantering, uppkoppling mot databas eller filhantering. Att man själv inte behöver förstå hur metoden fungerar gör att man spar tid åt att lösa andra problem istället. Biblioteket bygger på att man inte ska behöva spendera timmar att försöka lösa problem som redan blivit lösta [3].

2.3.3 .NET-Framework 3.0

Efter version 1.0 släppte Microsoft både 1.1 och 2.0 men det var inte förrän 3.0 som man lade till fyra viktiga huvudkategorier som ett lager över det föregående lagret. Då till kom följande verktyg [4]:

- Windows Presentation Foundation
- Windows Communication Foundation
- Windows Workflow Foundation
- CardSpace

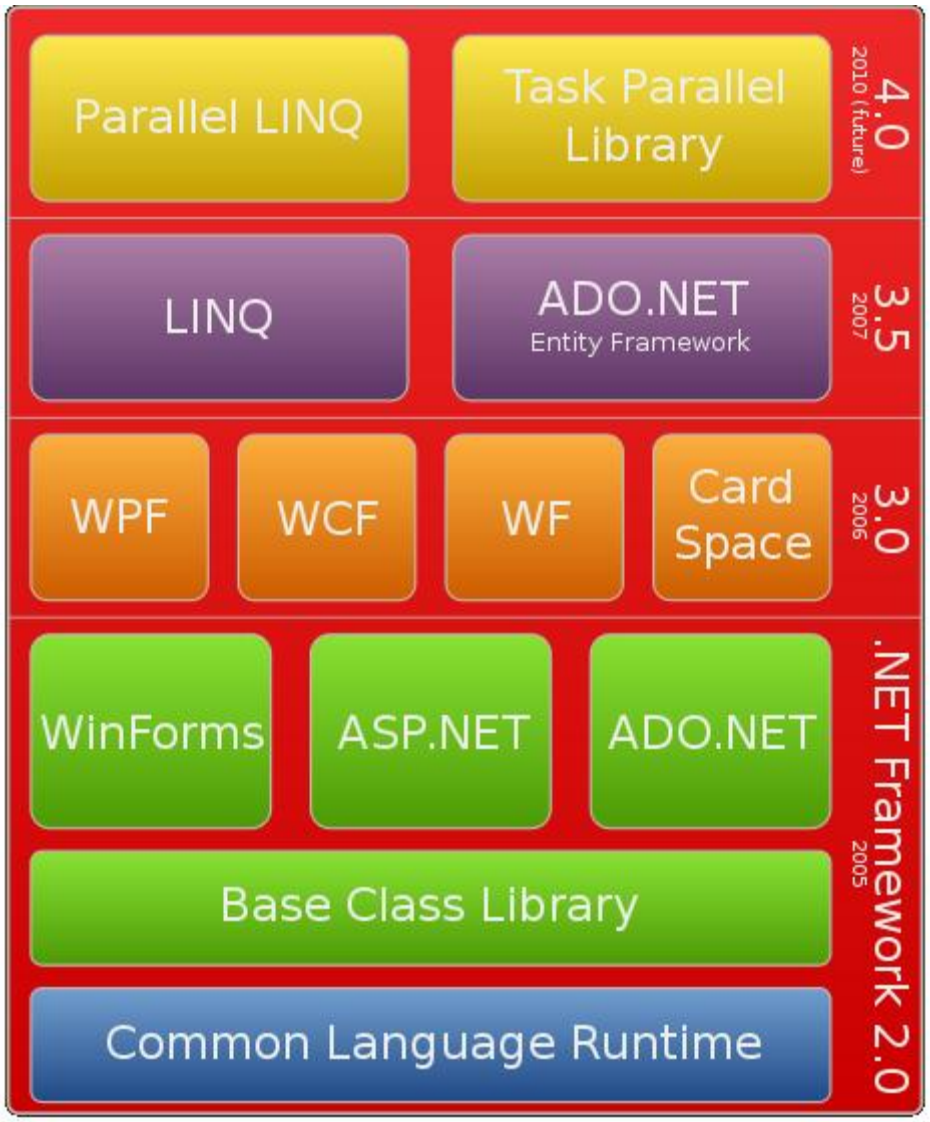
Windows Presentation Foundation har redan mer ingående behandlats tidigare i rapporten. Kort beskrivet är det Microsofts nya utvecklingsverktyg för windows- och internetapplikationer.

Windows Communication Foundation är ett verktyg som ska hjälpa företag att bygga nätbaserade applikationer som ska slussa information mellan två noder i ett system [5].

Windows Workflow Foundation hjälper användaren att visualisera sin applikation som ett flöde av information. Att arbeta med flöden gör applikationen mer flexibel och skalbar. Det här verktyget samarbetar mycket bra med WCF som nyss togs upp [6].

CardSpace är Microsofts nya lösning på hur människor ska kunna identifiera sig på internet. Till exempel har de ersatt gamla lösenordsbaserade inloggningar mot ett mer säkert sätt som använder sig av säkerhetsbevis [7].

I Figur 4 kan man lätt åskådliggöra hur .NET-Framework är uppbyggt. Där kan man även se att både version 3.5 och 4.0 har släppts.



Figur 4. Bild som visar hur .NET Framework är uppbyggt av flera lager.

2.4 SQL

SQL som står för Structured Query Language är det mest använda språket idag för att hämta och lägga in data i en databas. Det utvecklades på IBM:s forsknings–center år 1970 och nu 40 år senare används det fortfarande [10].

Antag att en databas beskriver ett litet företags verksamhet. I databasen skapas en tabell som heter Personal och innehåller kolumnerna Personnummer, Namn och Lön. När man skapar en tabell så måste man ange vilka datatyper de olika kolumnerna ska vara. Två vanliga typer är bigint och nvarchar. Bigint betyder att man bara kan lägga in heltal som värde. Detta är bra att använda om man till exempel vill göra matematiska beräkningar. Kolumnen Lön är av typen bigint. Däremot är de andra två Namn och Personnummer av typen nvarchar som är till för att innehålla strängar. Det går inte att göra några matematiska beräkningar på dessa värden. Tabellen Personal illustreras enligt Figur 5.

Personnummer	***Namn***	***Lön***
550416-2562	Rita Persson	21585
610522-2371	Gunnar Pettersson	20990
721205-4367	Nicklas Olofsson	18750
650912-1843	Eva Svensson	20550

Figur 5. Tabellen Personal som visar information om ett företags anställda.

För att hämta data från en databas använder man kommandot SELECT. Man kan precisera sin sökning genom att lägga till följande villkor:

- FROM – Väljer från vilken tabell man ska hämta data.
- WHERE – Filtrera rader efter ett visst värde eller sträng i en kolumn.
- GROUP BY – Om man vill sammanställa resultatet i grupper.
- HAVING – Filtrera grupper efter ett visst värde eller sträng.
- ORDER BY – Vilken ordning resultatet ska visas i.

Om man till exempel. vill få fram alla personer som tjänar över 21000 kr skickar man följande förfrågan till databasen:

```
SELECT Namn, Lön
FROM Personal
WHERE Lön > 21000
```

Resultatet blir enligt Figur 6.

Namn	Lön
Rita Persson	21585

Figur 6. Visar alla anställda som tjänar över 21000.

Om företaget nu anställer en ny person vill de såklart uppdatera sin databas över anställda. Då använder man sig av kommandot INSERT. Här kommer ett enkelt exempel hur man kan lägga in data i en tabell:

```
INSERT INTO Personal
VALUES ('720511-5842', 'Stig Andersson', 20175)
```

Personals nya utseende efter utförd INSERT:

Personnummer	Namn	Lön
550416-2562	Rita Persson	21585
610522-2371	Gunnar Pettersson	20990
721205-4367	Nicklas Olofsson	18750
650912-1843	Eva Svensson	20550
720511-5842	Stig Andersson	20175

Figur 7. Tabellen Personal efter inmatning av ny anställd.

Ibland kan man vilja hitta relationer mellan kolumner från två olika tabeller och sammanställa sambandet i en ny tabell. Detta görs smidigast om man använder sig av joins och de tre vanligaste är:

- Cross Join
- Inner Join
- Outer Join

Cross Join är den enklaste formen av joins eftersom den bara utförs i ett steg. Den tar nämligen bara den kartesiska produkten mellan två tabeller. Alltså alla rader i tabell A kombineras med alla rader i tabell B. Om tabell A innehåller 10 rader och tabell B 8 rader så kommer den nya sammanställda tabellen att innehålla 80 rader [10].

Inner Join är något mer komplicerad då den utför sin selektering i två steg. Första steget är samma som Cross Join men det andra steget är en slags filtrering beroende på det villkor du själv angivit [10].

Outer Join är den mest avancerade eftersom den processen utförs i tre steg. De två första stegen är likadana som Inner Join men det tredje är nytt. Innan man gör en Outer Join måste man välja ifall tabell A eller tabell B ska få vara intakt. Om man väljer tabell A så kommer resultatet innehålla alla rader från tabell A samt alla matchande rader från tabell B. I de fall där det inte finns något matchande värde kommer det bli Null [10].

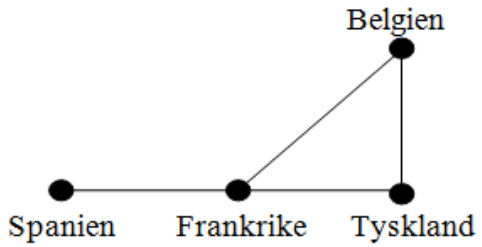
Null har en speciell roll inom SQL. I början tror de flesta att det betyder värdet noll. Men faktum är att null är odefinierat och har därför inget värde alls [10].

2.5 Grafteori

Grafteori är ett ämne som kan appliceras på många olika användningsområden. Att så många kan dra nytta av detta är för att flera av de problem man stöter på kan modelleras upp som en graf. En grafs uppgift är huvudsakligen att genom matematik visa upp relationer. Graferna kan komma i många olika skepnader och några av dessa är [11]:

- Träd
- Nätverk
- Riktade och oriktade grafer

En sak som de alla har gemensamt är att de endast består av kanter och hörn. Det är kanterna som sammankopplar hörnen och på så sätt visar en relation mellan två hörn. För att göra detta lite mer förståeligt så kan man byta ut hörnen mot människor, datorer, länder eller vadsomhelst egentligen. I Figur 8 kommer ett exempel på en enkel graf [11].

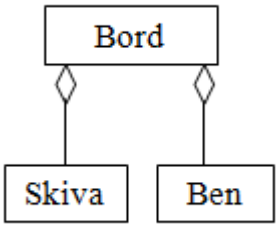


Figur 8. Enkel graf som visar länders gränser mot varandra.

I Figur 8 representerar varje hörn ett land och kanterna berättar om länderna angränsar till varandra. Av grafen kan man till exempel dra slutsatsen att Frankrike angränsar till alla länder och att Spanien endast angränsar till ett land, nämligen Frankrike.

2.6 UML

UML står för Unified Modeling Language och är ett objektorienterat språk för att modellera upp system. Man kan till exempel visa att en komponent består av mindre delkomponenter. Följande notation visar att ett bord består av skiva och ben[12]:

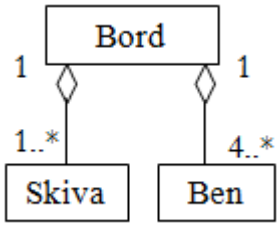


Figur 9. Bilden visar enligt UML-notation att ett bord består av skiva och ben.

Det går även att göra det mer avancerat om man till exempel vill ange hur många ben varje bord består av. I UML kallas detta för multiplicitet och de vanligaste sätten att ange det på visas i följande lista.

- Exakt en: 1
- Noll eller flera: 0..*
- Noll eller en: 0..1
- En eller flera: 1..*

Siffrorna kan bytas ut mot godtyckliga tal för att passa in i önskat system. Om man nu applicerar denna metod på bordet som användes i föregående exempel blir resultatet enligt Figur 10.



Figur 10. Bilden visar att ett bord består av en eller flera skivor och fyra eller flera ben.

3 Genomförande

I första skedet hade vi frekventa möten med System Andersson för att få en klar bild av vad de ville att produktkonfiguratorn skulle klara av, samt få tillgång till de verktyg System Andersson ville att vi skulle använda oss av, nämligen Microsoft Visual Studio 2010 beta och Microsoft SQL Server 2008 Express.

3.1 Design

3.1.1 Kompatibilitetsrelationer

Vi började fundera på hur olika artiklar skulle kunna stå i relation till varandra och vilka problem som skulle kunna uppstå där. Vår första tanke var att strukturera artiklar med följande attribut – kravartikel, startartikel eller tillvalsartikel, där varje artikel var en del av en definierad produkt. Ett exempel på detta skulle kunna vara en dator som har som startartikel ett moderkort, kravartikel en processor och ramminne och som tillvalsartikel ljudkort.

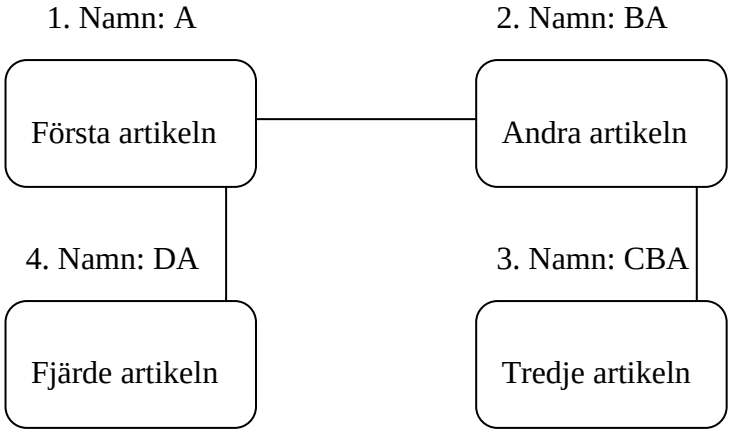
Ett problem som uppstod med detta var att alla artiklar måste definieras i alla produkter de är en del av, och det motverkar syftet med produktkonfiguratorn eftersom produkter ej ska vara fördefinierade utan kunna skapas utan att finnas tidigare. Det vill säga komponeringen av produkten sker när man sätter vilka artiklar som är kompatibla med produkten, medan syftet egentligen var att skapa en miljö som hanterar relationer mellan artiklar utan att ha fördefinierat produkten.

Förhållandet mellan artiklarna blev det som vi fokuserade på istället, men koncepten kravartikel och startartikel behöll vi om i en något anpassad form. Ett tydligt exempel på det är att vi ändrade namn på startartikel till masterartikel, det vill säga en artikel med statusen ”master”.

En annan fråga vi ställde oss var hur vi skulle hantera en situation där artikel A är kompatibel med B och B med C men A och C är inte kompatibla. Till exempel att ett moderkort är kompatibelt med en processor och en typ av ramminne, men processorn och ramminnet är inte kompatibla med varandra.

Ett sätt att komma till rätta med detta skulle kunna vara att namnge artiklarna på ett strukturerat sätt.

Varje ny artikel får en bokstav i alfabetet. Varje ny artikel ärver namnet från intilliggande kompatibel artikel men lägger till sin egen bokstav i början av namnet (se Figur 11).

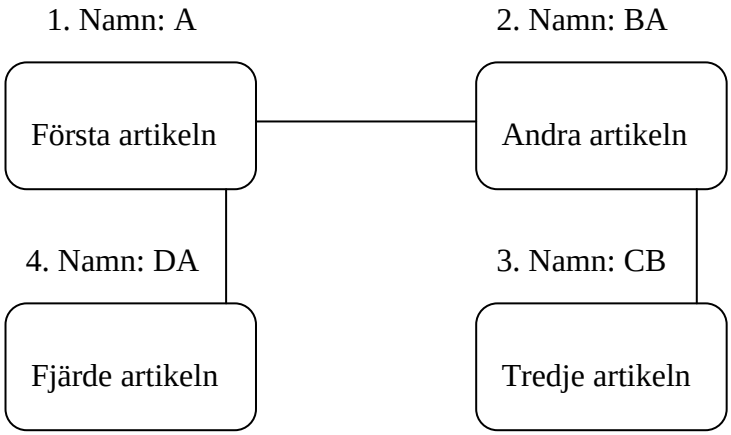


Figur 11. Artikelkompatibilitet där namn av modernod ärvs och en egen bokstav läggs till i början av namnet för en nod.

Då kan, om första artikeln är ett moderkort, andra artikeln är en processor och fjärde artikeln är ett ramminne, kompatibilitet kontrolleras genom att man för varje artikel söker efter alla andra artiklar som har ett nod-namn som innehåller någon bokstav ur artikelns nod-namn.

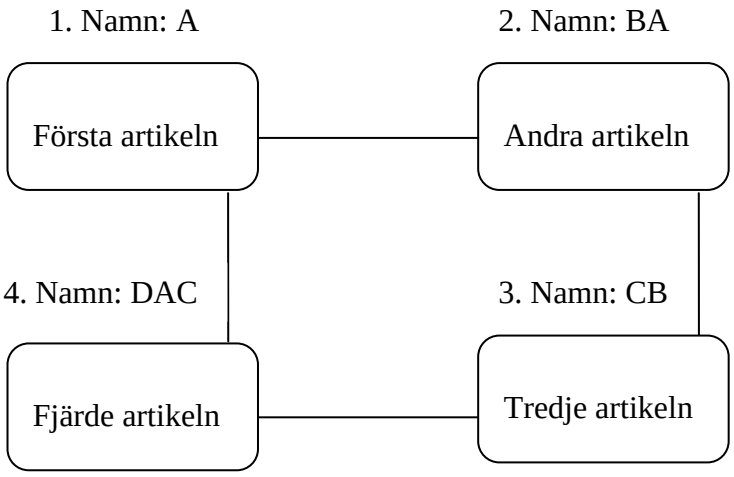
Till exempel när fjärde artikeln ska placeras ut får den namnet DA och man söker efter alla artiklar som innehåller bokstaven D eller A och jämför i kompatibilitetstabellen om de är kompatibla. Är de inte det sorteras artikeln bort och går inte att välja på den positionen.

Antag att den tredje artikeln inte spelar någon roll för fjärde artikeln i det här fallet. Om tredje artikeln är till exempel en processorfläkt spelar det inte någon roll för ramminnet. Någon sorts begränsning för hur långt namn som ärvs behövde definieras. I det här fallet skulle bara modernodens första bokstav ärvas. Man skulle då få följande struktur enligt Figur 12.



Figur 12. Artikelkompatibilitet där första bokstaven ur modernodens namn ärvs och en egen bokstav läggs till i början av namnet för en nod.

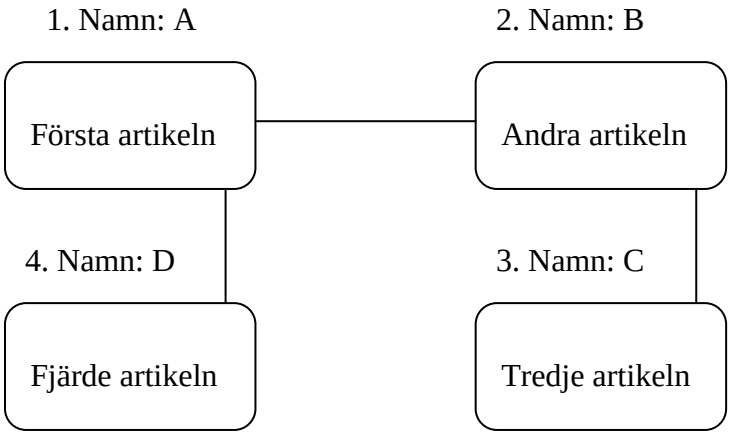
Men vad händer om det finns något fall där man har en krets, det vill säga att i det här fallet fjärde och tredje noden också är kopplade till varandra? Den fjärde noden borde få namnet DAC (se Figur 13).



Figur 13. Artikelkompatibilitet som krets där första bokstaven ur modernodens namn ärvs och en egen bokstav läggs till i början av namnet för en nod.

Detta är inte en bra lösning för alla fall. Om man inte vill sätta ihop en dator utan en cykel till exempel spelar det ju inte någon roll för sadeln vad det är för styre på ramen. Dessutom blir konfiguratorn mer krånglig att konfigurera eftersom artiklar som inte är kopplade till varandra måste få en kompatibilitet i databasen. Det vill säga att processorn inte bara var tvungen att vara kompatibel med moderkortet utan också med ramminnet. Man måste då välja att antingen skapa kompatibilitet med alla artiklar som kan tänkas ingå i samma produkt. Vilket kan ge märkliga resultat då man kan koppla ihop en processor direkt med ramminnet. Eller så får man på något sätt definiera vilka artiklar som en artikel inte är kompatibel med, vilket skulle vara om än mer omständigt.

Eftersom vi inte vill lägga in onödiga begränsningar i produktkonfiguratorn valde vi att bara kontrollera kompatibilitet mot anslutande artikel, enligt Figur 14.



Figur 14. Artikelkompatibilitet där alla noder får en egen bokstav.

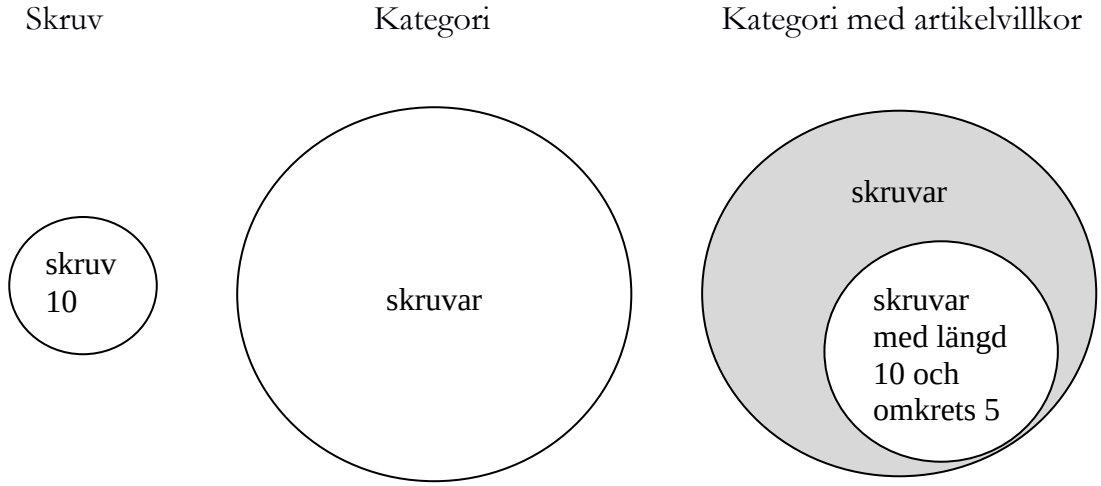
Nod B kan innehålla alla artiklar kompatibla med A, nod C kan innehålla alla artiklar kompatibla med B och nod D kan innehålla alla artiklar kompatibla med A.

3.1.2 Kompatibilitetsverktyget

För att få produktkonfiguratorn så användarvänlig som möjligt valdes tidigt att ett kompatibilitetsverktyg för att skapa kompatibilitet mellan existerande artiklar i databasen behövdes, även om detta inte preciserats i målet av System Andersson. Målet med kompatibilitetsverktyget var att lätt kunna se vilka artiklar som finns i databasen och skapa kompatibilitet mellan dem som önskas på ett smidigt sätt. Man skulle kunna välja på enskilda artiklar, artikelkategorier eller attributvillkor för artiklarna.

Först väljer man en viss artikel, en viss kategori eller en kategori och anger vilka attribut som artiklarna ur den valda kategorin får ha.

Till exempel kan man välja artikeln ”skruv 10” eller så kan man välja kategorin ”skruvar” eller kategorin ”skruvar” med attributen ”längd: 10” och ”omkrets: 5”. I första fallet väljs bara artikeln som heter ”skruv 10”. I det andra fallet väljs alla artiklar som ingår i kategorin ”skruvar”. I det tredje fallet väljs alla artiklar som ingår i kategorin ”skruvar” som har längden 10 och omkretsen 5 (se Figur 15).



Figur 10. Figuren visar olika mängder som valts med alternativen, vald artikel, vald kategori och vald kategori med artikelvillkor.

Efter att man valt en artikel, kategori eller kategori med artikelvillkor väljer man sedan den artikel, kategori eller kategori med artikelvillkor som det första valet ska vara kompatibelt med.

Man kan till exempel välja artikeln ”klassiskt cykelstyre” som första val och sedan välja att den ska vara kompatibel med kategorin ”cykelramar”. Då kommer ”klassiskt cykelstyre” bli kompatibelt med alla artiklar som ingår i kategorin ”cykelramar”.

Dessutom kan vissa artiklar kräva andra artiklar. Därför ville vi kunna sätta en kravstatus på en kompatibilitet, det vill säga att det första valet kräver någon artikel från det andra valet. Till exempel kräver en cykel en ram, hjul, sadel, trampor och styre.

Förutom den speciella statusen krav som en kompatibilitet ska kunna få, ska man kunna ge det första valet statusen ”master”. Det innebär att det är en artikel som man gärna vill utgå ifrån när man ska sätta ihop en produkt. Man kommer då bara att kunna ha den artikeln en gång under konfigureringen. Alternativt kan ”master” innebära att det är en artikel som består av andra artiklar. En ”master” i det här fallet skulle kunna vara en bilmotor i det fallet då bilmotorn ska konfigureras, men den är ingen ”master” i fallet då den är en del av en bil. Då är bil en ”master”.

3.1.2.1 Hantering av artiklar med odefinierade attributvärden

Antag att ett bord ska byggas. För att bygga bordet behövs bland annat en bordskiva och bordsben. Antalet bordsben som behövs beror på hur stort bordet är. Ett långt bord behöver fler bordsben än ett kort bord.

För att lösa detta kan behöver man hålla reda på fyra värden. Ett värde för minimum antal av den kompatibla artikeln och ett värde för antal per måttenhet av den kompatibla artikeln. De sista värdena definierar vilket attribut av artikeln som antalet för den kompatibla artikeln ska jämföras med samt värdet på detta.

Exemplet bord igen. Artikeln bordskiva är kompatibel med bordsben. Det minsta antalet bordsben för bordskivan är fyra och för varje meter som bordskiva är långt krävs ytterligare två bordsben.

Man skulle då få följande resultat:

Minimum antal är fyra. Antal per mått är två. Mått är en meter och måttattributet är längd.

3.1.3 Konfigurationsdelen

Här ville vi ha två fält till höger. En kategorilista och en artikellista. I kategorilistan listas alla kategorier och efter att man klickat på ett kategorinamn visas alla artiklar som ingår i den kategorin i en egen anslutande lista. Från artikellistan skulle man kunna dra ut med muspekaren den artikel man önskar till komponeringsytan till vänster. Om man vill koppla ihop två artiklar gör man på följande sätt.

Först markerar man den eller de artiklar som man vill ska vara kopplad till den artikel man vill lägga till, då sorteras artikellistan och visar bara de artiklar som är kompatibla med den eller de artiklar som är markerade. Tar man sedan och drar ut en artikel till komponeringsytan skapas automatiskt kopplingar mellan den eller de markerade artiklarna och den nya artikeln. Man ska också kunna lägga till

kopplingar mellan kompatibla artiklar som redan finns i komponeringsytan. Det var också tvunget att kunna ta bort artiklar och kopplingar mellan artiklar. Tas en artikel bort tas kopplingar till denna bort automatiskt.

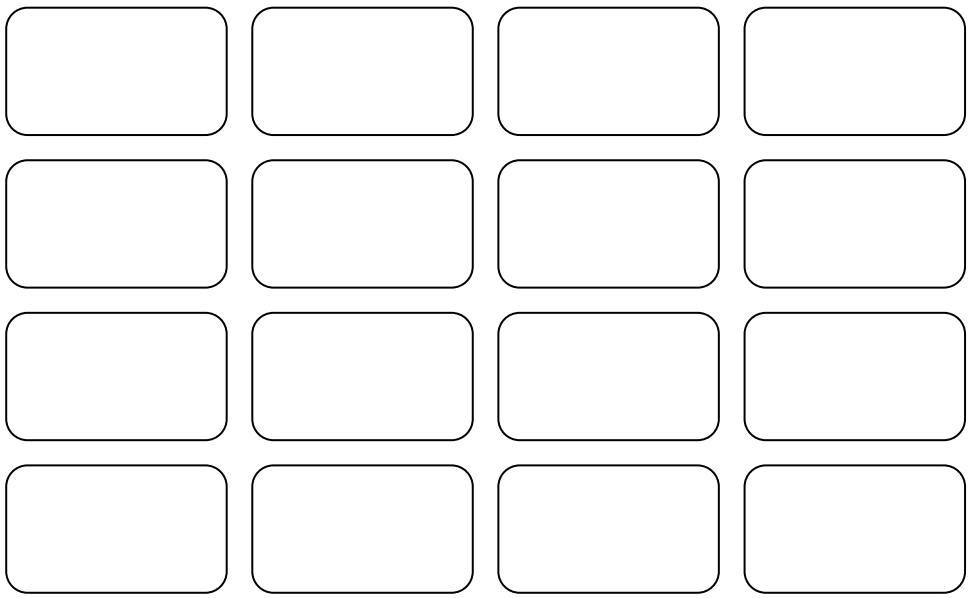
Vi bestämde att en artikel ska symboliseras med en ruta där namnet på artikeln finns representerat på något sätt tillsammans med antalet artiklar som önskas. Man kan till exempel behöva ha tolv stycken ”skruv 10”, men istället för att lägga ut tolv rutor lägger man bara ut en ruta med ”skruv 10” men ändrar antalet till tolv.

Efter ytterligare diskussion med System Andersson kom vi fram till att endast en konfiguration skulle vara tillåten åt gången. Det vill säga att bara ett konfigurationsträd ska vara möjlig för en produkt.

Till sist var konfigurationsdelen tvungen att klara av att spara och ladda konfigurationer.

3.1.3.1 Fördefinierade rutor

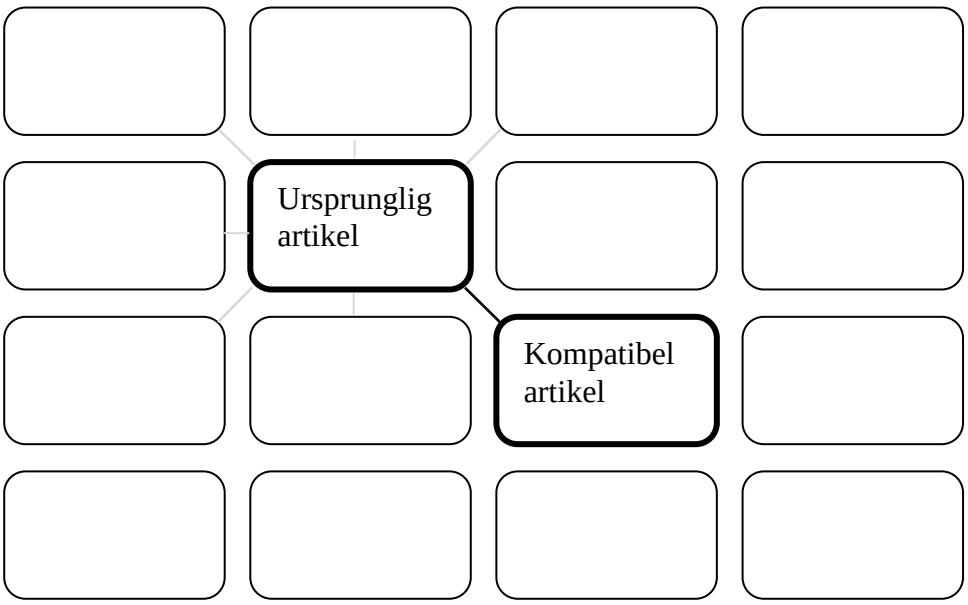
För att lösa komponeringsytan diskuterades två lösningar. För att underlätta för ”drag and drop” vid pekskärmaranvändning, och eventuellt förenklande i implementeringen, kunde flera rutor i komponeringsytan fördefinieras enligt Figur 16.



Figur 16. Fördefinierade rutor där artiklar kan placeras.

En ruta skulle då kunna vara kopplad till de åtta omgivande rutorna. Om alla rutor är tomma listas alla artiklar i databasen i artikellistan. Användaren skulle då kunna dra valfri artikel och släppa i en ruta. För att ansluta en kompatibel artikel markeras den artikel man vill koppla den till, då sorteras alla kompatibla artiklar i

artikellistan och man kan dra in en valfri artikel till en angränsande ruta till den markerade artikeln (se Figur 17).



Figur 17. Placering av kompatibla produkter i fördefinierade rutor.

Man skulle naturligtvis också kunna markera två artiklar och då få en artikellista efter vad som är kompatibelt med de två kompatibla. En sådan kompatibel artikel skulle då behöva släppas i en ruta som angränsar till båda de markerade rutorna.

Den här typen av komponeringsyta är kanske möjlig för mindre konfigurationer, men det blir opraktiskt och begränsande när mer komplexa produkter ska sättas ihop. I det här exemplet finns det fyra fält som kan ha kopplingar till åtta andra fält. Åtta fält som kan ha kopplingar med fem andra fält och fyra fält som kan ha kopplingar med tre andra fält. Antalet fördefinierade rutor är inte bestämt och kan justeras beroende på skärmupplösning under implementeringen. Oavsett antalet rutor innebär den här typen ett visst pusslande ifall att användaren önskar flytta en artikel efter att den placerats ut.

Antag att användaren placerat en artikel i någon av mittenrutorna till att börja med. Senare upptäcker man att den artikeln bara kommer att vara kompatibel med en annan artikel i konfigurationen och användaren vill använda mittenrutan som artikeln ligger i till en annan artikel med fler kopplingar. Användaren måste då flytta artikeln till en annan ruta. Men den rutan måsta vara angränsande till en artikel som är kompatibel med den artikel som ska flyttas, annars kommer konfiguratorn säga ifrån att bara ett träd får byggas åt gången.

Om användaren är välplanerad och lägger artiklar som endast är kompatibla med en enstaka artikel i kanterna skulle man nog kunna sätta ihop en större produkt. Men då förlorar produktkonfiguratorn sitt syfte eftersom användaren skulle vara tvungen att planera och delvis konfigurera sin produkt innan produktkonfiguratorn använts.

Efter kontakt med System Andersson meddelade de oss att det skulle vara ytterst få av deras kunder som skulle använda produktkonfiguratorn med pekskärm och därför behövde vi inte tänka på det.

3.1.3.2 Fri yta

Den andra lösningen innebar att vi skapar en helt öppen yta där varje artikel som släpps skapar en egen ruta som kopplas till rätt existerande ruta. Den här varianten erbjuder ett i teorin obegränsat antal kopplingar mellan artiklar och ger en annan användarvänlighet med färre begränsningar.

Utmaningen här ligger på implementeringsdelen. Eftersom området är mer dynamiskt om man jämför med alternativet ovan, måste konfiguratorn hålla koll på var olika rutor finns för att skapa kopplingar mellan dessa. Positionen av rutorna måste också sparas tillsammans med de värden rutorna har, så som namn och antal, när en konfiguration ska sparas.

Ett problem som skulle kunna uppstå är att det kan bli svårt att vara exakt i placeringen av rutorna när det gäller pekskärmar men det problemet skulle vi, enligt ovan, ignorera enligt System Andersson.

3.2 Implementering

Innan verktygen blivit tillgängliga började vi aktualisera och fördjupa kunskaperna i C# och syntaxen på SQL-satser med hjälp av Windows Visual Studio 2008 och MySql, som vi hade tillgängligt genom skolan. Vi försökte i MySql lösa problem genom SQL-satser på situationer som skulle kunna uppstå, och i Visual Studio 2008 började vi testa olika ”drag and drop” möjligheter.

Tidigt delades projektet upp i två delar. Dels kompatibilitetsdelen som skulle sköta databashanteringen och skapandet av kompatibilitet via kompatibilitetsverktyget, och dels konfigurationsdelen som skulle lösa användargränssnittet och funktionerna för själva produktkonfiguratorn. Arbetet med implementeringen av dessa delar skedde med regelbundna avbrott för att diskutera igenom det vi gjorde och för att se om det fanns bättre lösningar på problemen.

3.2.1 Kompatibilitetsdelen

Eftersom vi använde oss av MySql i början och senare bytte till Microsoft SQL Server 2008 Express var vi tvungna att ändra vissa SQL-satser, men principen för vad de skulle göra var de samma även om vi fick lösa problemen på ett alternativt sätt. Vi väljer att inte visa MySql-satserna då de inte är relevanta för redovisningen av projektet.

3.2.1.1 Databasen

Det första vi fick göra var att skapa tabeller i vår databas för att kunna hantera kompatibiliteten. Här hade vi två alternativ. En som tog lite utrymme i databasen men som krävde fler SQL-anrop för att lösa uppgiften, och en som tog mer plats i databasen men som krävde färre SQL-anrop.

Det första alternativet innebar att vi sparade SQL-strängar i ett fält tillhörande en artikel. Till exempel om artikeln cykelram är kompatibel med alla skruvar med längd 10 och omkrets 5, sparas cykelram i ett fält och i kompatibilitetsfältet sparas ”alla skruvar med längd 10 och omkrets 5” fast översatt till SQL-kod. För att hämta och visa de kompatibla artiklarna i en lista måste först kompatibilitetsmeningen hämtas. Sen måste en förfrågan där meningen ingår skickas till databasen och då får man fram en lista på alla kompatibla artiklar. För att skapa kompatibiliteten krävs en SQL-sträng. Alltså kräver den här metoden, i det här fallet, bara en rad i databasen och tre kommunikationer mellan konfiguratorn och databasen.

Med det andra alternativet skapas en rad för varje artikel som passar in i ”alla skruvar med längd 10 och omkrets 5”. Detta kan skötas med en SQL-sats. När kompatibilitet ska hämtas krävs bara en SQL-förfrågan. Den här metoden kräver en rad i databasen för varje kompatibel artikel och två kommunikationer.

Vi valde att använda det senare alternativet då vi ville hålla nätverkskommunikationen så låg som möjlig.

I vår databas använder vi tre tabeller för den här prototypen.

En artikeltabell med information om artiklarna. Följande fält ingår i tabellen som heter exArticle.

- id – primärnyckel. Datatyp: bigint.
- articleID – artikelnumret på artikeln. Datatyp: nvarchar(50).
- articleName – namnet på artikeln. Datatyp: nvarchar(50).
- categoryID – id för den kategori artikeln ingår i. Datatyp: bigint.
- length – längden för artikeln. Datatyp: bigint. Får vara NULL.
- width – bredden för artikeln. Datatyp: bigint. Får vara NULL.
- height – höjden för artikeln. Datatyp: bigint. Får vara NULL.
- circuit – omkretsen för artikeln. Datatyp: bigint. Får vara NULL.
- weight - vikten för artikeln. Datatyp: bigint. Får vara NULL.

En kategoritabell med kategorinamn. Följande fält ingår i tabellen som heter exCategory.

- id – primärnyckel. Datatyp: bigint.

- categoryName – namnet på kategorin. Datatyp: nvarchar(50)

En kompatibilitetstabell med information om förhållanden mellan artiklar och kategorier. Följande fält ingår i tabellen som heter exArticleCompatibility, och alla får vara NULL.

- articleID – id för en artikel. Datatyp: bigint.
- categoryID – id för en kategori. Datatyp: bigint.
- compatibleArticleID – id för en kompatibel artikel. Datatyp: bigint.
- compatibleCategoryID – id för en kompatibel kategori. Datayp: bigint
- necessary – sätts till ”yes” om den kompatibla komponenten är ett krav. Datatyp: nvarchar(50).
- master – sätts till ”yes” om artikeln eller kategorin är en master. Datatyp: nvarchar(50).

Från början fanns fältet ”master” i artikeltabellen, men den flyttades till kompatibilitetstabellen för att en artikel ska kunna vara master gentemot vissa artiklar men inte mot andra. Master bygger alltså på vilken eller vilka artiklar en artikel står i relation till.

Varje kompatibilitet som ska skapas blir två rader i exArticleCompatibility. Detta är på grund av fälten necessary och master. Antag att vi har artikeln bil. Bil är kompatibel med bilmotor, och kräver denna. Bilmotor är i sin tur kompatibel med bil men kräver inte denna. Vi behöver då spara relationen mellan bil och bilmotor i ett fält och relationen mellan bilmotor och bil i ett annat.

Tanken med categoryID och compatibleCategoryID är att man ska kunna sätta en hel kategori som kompatibel med en artikel för att spara utrymme i databasen.

3.2.1.2 SQL-satser

När tabellerna skapats behövdes SQL-satser för att sköta in och utmatningar i tabellerna.

Följande uppgifter valde vi att lösa med SQL-satser.

- Skapande av kompatibilitet mellan artiklar, artiklar och kategorier eller kompatibilitet mellan kategorier.
- Borttagning av kompatibilitet mellan artiklar, artiklar och kategorier eller kompatibilitet mellan kategorier.
- Hämta alla artiklar kompatibla med den valda artikeln eller de valda artiklarna.

INSERT

Skapandet av kompatibilitet sker med en INSERT-sats. För att undvika kod på flera ställen i produktkonfiguratorn med olika INSERT-satser, beroende på om det är enskilda artiklar, kategorier eller kategorier med artikelvillkor som ska kopplas till varandra, skapade vi en dynamisk INSERT-sats som fungerar oavsett vilka typer som är kompatibla med varandra. Satsen ser i princip ut som följer:

INSERT INTO exArticleCompatibility (”typA ID”, ”kompatibel typB ID”, necessary, master) VALUES (”villkorA”, ”villkorB”, varNecessary, varMaster), (”villkorB”, ”villkorA”, NULL, NULL)

Förklaring krävs. Variablerna ”typA ID” och ”kompatibel typB ID” håller reda på i vilka fält de id som följer efter VALUES ska sparas i. Det vill säga om det är i articleID- eller categoryIDfältet och compatibleArticleID- eller compatibleCategoryIDfältet som ska spara informationen. Efter VALUES följer två delar åtskilda med parenteser och kommatecken. Den första delen lagrar relationen mellan de valda artiklarna eller kategorierna, samt om det kompatibla objektet krävs eller om objekt A är en master. Den andra delen lagrar enbart kompatibiliteten mellan de båda objekten.

Variablerna ”villkorA ” och ”villkorB” definierar den eller de artiklar och kategorier som är inblandade i relationen. ”VillkorA” är det artikelID eller categoryID för den först valda artikeln, kategorin eller kategorin med artikelvillkor. Det kan alltså vara en artikel eller kategori eller flera artiklar. Om det är flera artiklar byggs satsen ovan på med ytterligare två delar av samma typ som beskrivits ovan för varje artikel. ”VillkorB” är som ”villkorA” fast med ID för det kompatibla objektet.

Variablerna ”typA ID”, ”kompatibel typB ID”, ”villkorA” och ”villkorB” skapas och översätts till SQL-syntax av den del av produktkonfiguratorn som hanterar kompatibilitetsdelen.

DELETE

Borttagning av kompatibilitet sker på liknande sätt.

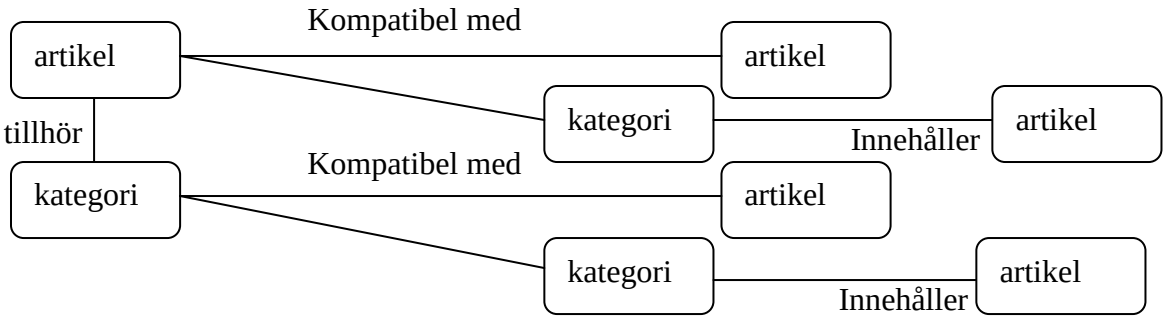
DELETE FROM exArticleCompatibility WHERE (”typA ID” IN ”villkorA” AND ”kompatibel typB ID” IN ”villkorB”) OR (”typB ID” IN ”villkorB” AND ”kompatibel typA ID” IN ”villkorA”)

Här tas de rader bort från kompatibilitetstabellen som uppfyller följande villkoren ovan. Det vill säga om ett typA id hittas i villkorA, på en rad samtidigt som ett typ B id hittas i villkorB, på samma rad tas den raden bort.

SELECT

Satsen för att hämta alla artiklar som är kompatibla med vald eller valda artiklar fungerar enligt följande princip. Om bara en artikel är vald kontrolleras

kompatibilitetstabellen efter kompatibla artiklar och efter artiklar som ingår i kategorier som artikeln är kompatibel med (se Figur 18). Dessutom söks kompatibilitetstabellen igenom efter kompatibla artiklar och kategorier till den kategori som artikeln ingår i. Om flera artiklar är markerade jämförs resultaten mellan dessa och träffar som finns hos alla artiklar hämtas.



Figur 18. Illustration som visar hur kompatibla artiklar hittas.

3.2.2 Konfigurationsverktyget

För att lösa att man ska kunna dra och släppa artiklar på den fria ytan så granskade vi en redan färdig applikation vid namn DragAndDropSmorgasbord för att hämta inspiration [13]. Nästan genast märkte vi att denna lösning var alldeles för komplex för vår situtation. Istället började vi experimentera oss fram till en relativt simpel lösning. Att man skulle kunna dra ut artiklar från listan till den fria ytan skippade vi till att börja med för att spara lite tid åt viktigare funktioner. Det blev en ”Lägg till”-knapp som gör att man kan lägga ut artiklar var man vill på den fria ytan. Muspekarens position när man klickar talar om för konfiguratorn var den nya artikeln ska hamna. Varje gång man lägger ut en artikel så stämplar man ut ett objekt från klassen myThumb. Klassen är till för att lagra information om de artiklar man använder i sin konfiguration. Där i sparas följande detaljer:

- StartLines – En lista som innehåller startpunkter på alla kopplingar till andra artiklar.
- EndLines – En lista som innehåller slutpunkter på alla kopplingar från andra artiklar.
- Antal – Hur många av artikeln som ingår i konfigurationen.
- Master – Ja/Nej om detta är en master-artikel eller inte.
- X/Y – Artikelns x- och y-position på den fria ytan.
- Kategori – Vilken kategori artikeln tillhör.
- Artikel – Namnet på artikeln.
- ArtikelID – Artikelns unika artikel-id.

För att kunna flytta på artikeln använde vi oss av en eventhandler som heter DragDelta. Den märker automatiskt när användaren tar tag i en artikel och försöker dra den över skärmen. DragDelta är unik för just objekttypen thumb som vi har använt oss av. Det är en av anledningarna till att vi använde oss av just thumb, eftersom det förenklar implementeringen av drag-and-drop.

Alla markerade artiklar sparas i en sorterad lista för att man enkelt ska kunna komma åt dem. Även alla kopplingar mellan artiklar lagras i en sorterad lista för att underlätta administrationen av kopplingar.

3.2.2.1 Sparning och Laddning av konfiguration

Om användaren är nöjd med sin konfiguration så ska man kunna spara den för att sedan kunna komma åt den vid framtida produktioner eller ordrar. Detta har vi löst genom att spara ner all nödvändig information till textdokument. Den information som sparas är alla de attribut som ingår i klassen myThumb. Dokumentet är uppdelat i två avdelningar, den första innehåller alla artiklar som finns i konfigurationen. I andra delen hittar vi alla kopplingar mellan artiklarna. Tillsammans kan de sätta ihop en konfiguration exakt som den var när den sparades till fil.

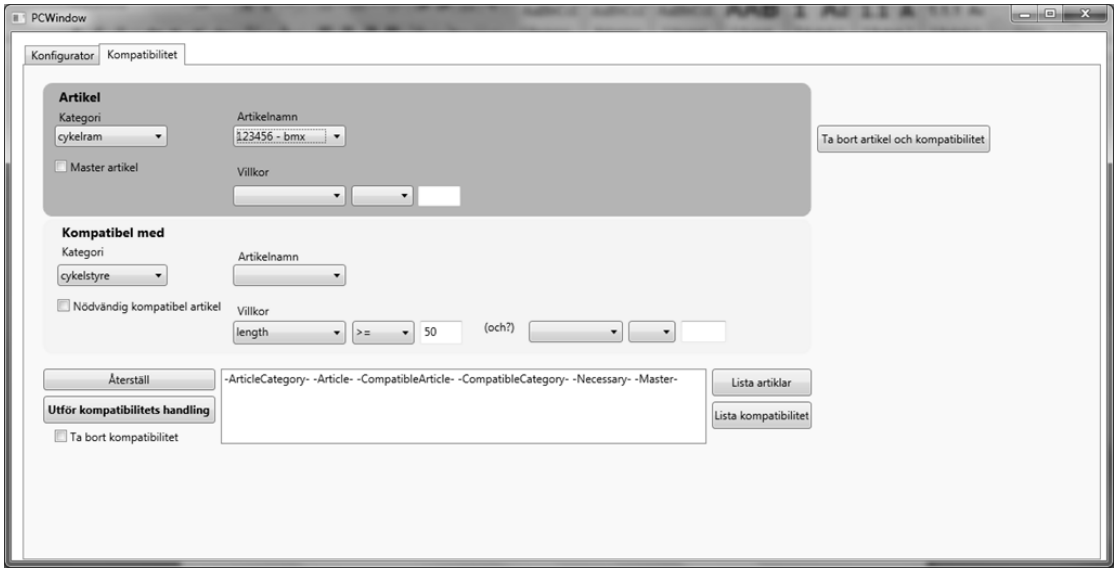
4 Resultat

Den färdiga applikation blev en produktkonfigurator som skapades i WPF med hjälp av utvecklingsmiljön Microsoft Visual Studio. Designen är uppbyggd sådan att applikationen är indelad i två flikar.

Den ena fliken innehåller själva konfiguratorn där man kan hämta artiklar från en databas och själv komponera en produkt som går att spara. Den andra fliken innehåller kompatibilitetsverktyget där man kan skapa kompatibilitet mellan redan existerande artiklar i databasen.

4.1 Kompatibilitetsverktyget

För att skapa kompatibilitet på ett enkelt sätt skapade vi konfigurationsverktyget. Det är indelat i två fält, ett artikelfält och ett kompatibilitetsfält, där en artikel, kategorin eller kategorin med artikelvillkor väljs för vardera fält (se Figur 19). När dessa är valda väljer man att utföra en uppgift, skapa eller ta bort kompatibilitet.



Figur 19. Kompatibilitetsverktyget.

4.1.1 Comboboxar och listboxar

Det finns alltså tre olika valmöjligheter på varje del. Man kan välja enbart kategori, eller först välja en kategori och sen välja en artikel som ingår i kategorin. Det tredje alternativet, kategori och artikelvillkor, får man genom att först välja kategori och sen istället för att välja artikelnamn välja något villkor på raden under.

Ett villkor innehåller tre delar. Först väljer man ett attribut från artikeltabellen det vill säga en fältrubrik från artikeltabellen förutom rubrikerna categoryID och id som inte kan väljas. Efter det väljer man ett av följande villkor:

- > - Större än.
- >= - Större eller lika med.
- = - Lika med.
- <= - Mindre eller lika med.
- < - Mindre än.
- ALLA UTOM - Alla som inte är lika med.

Till sist skriver man in det värde, siffror eller bokstäver, som man vill jämföra med i en textbox. När man valt ett villkor visas en till villkorsmöjlighet om man vill begränsa sitt val ytterligare.

Till exempel väljer man som kategori ”skruvar”. Efter det väljer man attributet ”length”, villkoret ”=” och skriver in värdet ”10” i textboxen. Då har man valt alla skruvar som har längden 10.

4.1.2 Checkboxar

I artikeldelen finns en checkbox som man bockar för om man vill ge det övre valet statusen master.

I kompatibilitetsdelen finns en checkbox som man bockar för om man vill sätta det undre valet till ett nödvändigt krav för det övre valet.

Utanför dessa båda fält finns checkboxen ”ta bort kompatibilitet”. Kryssa i denna och klicka sen på ”Utför kompatibilitetshandling” för att ta bort kompatibilitet.

4.1.3 Knappar

Det finns två knappar ”Utför kompatibilitetshandling” och ”Återställ”. ”Utför kompatibilitetshandling” skapar kompatibilitet mellan valda villkor, artiklar eller kategorier, alternativt tar bort kompatibilitet mellan dessa om checkboxen ”ta bort kompatibilitet” är ikryssad.

”Återställ” nollställer alla comboboxar, textboxar och checkboxar.

4.1.4 Kompatibilitetssituationer

Det finns några fall som kan vara bra att känna till hur kompatibilitetsverktyget hanterar, eller inte hanterar.

Antag att artikel A tillhör kategori Z.

- Om man väljer att artikel A är kompatibel med artikel B och en annan artikel ur kategori Z också är kompatibel med artikel B, och så vidare tills alla artiklar ur kategori Z sats till kompatibla med artikel B. Då kommer

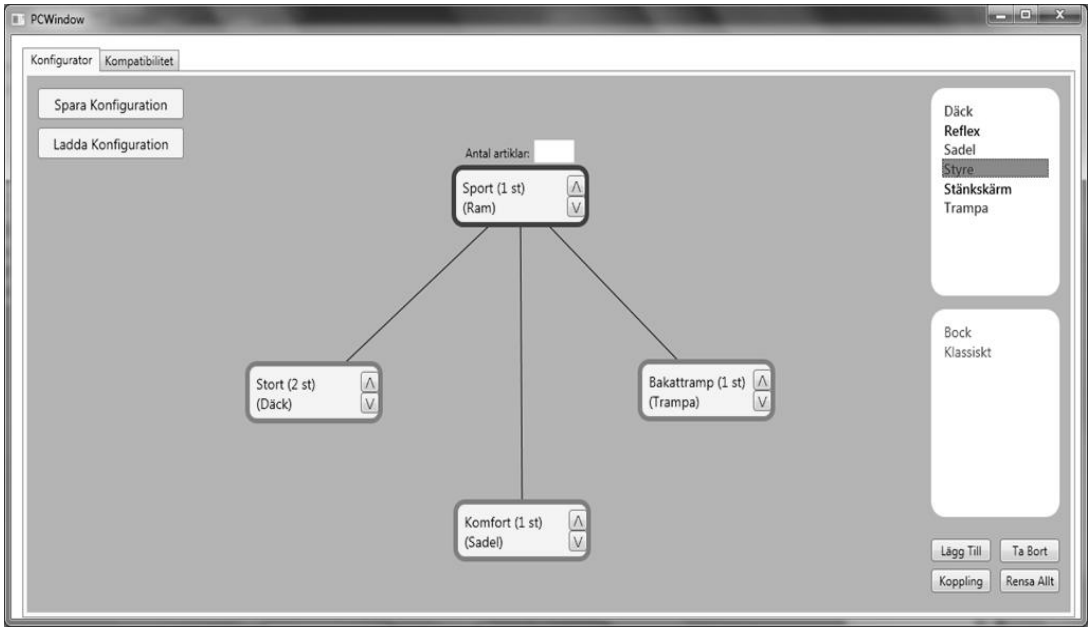
inte tabellen för kompatibilitet automatiskt uppdatera sig och sätta kategorin Z kompatibel med artikel B.

- Om man väljer att artikel A är kompatibel med artikel B och sen väljer att kategorin Z är kompatibel med artikel B kommer kompatibiliteten tas bort mellan artikel A och B för att spara plats. Om då artikel A sats till master eller att artikel B krävs av A kommer detta inte överföras till kompatibiliteten mellan kategorin Z och B, utan man måste då sätta detta i skapandet av kompatibiliteten mellan kategorin Z och artikeln B, eller på nytt skapa en kompatibilitet mellan artikel A och B där master och krav sätts som önskas.
- Om artikel A är kompatibel med artikel B och sen väljer man att ta bort kompatibiliteten mellan kategori Z och artikel B tas kompatibiliteten bort mellan artikel A och B.
- Om man väljer att kategori Z och artikel B är kompatibla, och sen vill ta bort kompatibiliteten mellan artikel A och artikel B kommer detta inte att gå eftersom kompatibiliteten är indirekt via kategorin Z.

Det är därför viktigt att tänka på i vilken ordning som kompatibilitets skapas. En hjälp kan vara att tänka stort först och sen smått. Skapa kompatibilitet där en hel kategori ska ingå först och efter det skapa specialfall mellan artiklar.

4.2 **Konfigurationsverktyget**

Största delen av konfiguratorn består av en fri yta där man kan dra och släppa artiklar. Helt enligt System Anderssons krav finner man artiklarna till höger på sidan där vi har placerat ut två listor (Se Figur 20).



Figur 20. Konfigurationsverktyget.

I första listan väljer man vilken kategori man vill utforska mer. När en kategori är markerad kommer antingen alla artiklar som tillhör den kategorin eller alla kompatibla artiklar att visas i den nedre listan beroende på om man har någon artikel markerad eller inte. Efter att man bestämt vilken artikel man vill använda klickar man på namnet på artikeln, håller musknappen intryckt och drar ut muspekaren till komponeringsytan. Då kommer muspekaren att ändra ikon till en hand för att visa användaren att man är på väg att addera en komponent till ytan. Det är helt upp till användaren var man vill placera denna artikel. För att placera artikeln släpper man musknappen.

Artikeln representeras som en rektangelformad ruta med en ram runt omkring. I rutan står namnet på artikeln, vilken kategori den tillhör och hur många av den som ingår i konfigurationen. Till höger i rutan finns två pilar för att justera antalet.

Ifall man har en och endast en artikel markerad kommer få tillgång till en inmatningsruta ovanför artikeln där man kan ange antalet. Det var en nödvändig funktion eftersom om man till exempel ska lägga till 1000 skruvar slipper man trycka på uppåtpilen 1000 gånger.

I nedre högra hörnet hittar man tre knappar:

- Ta Bort – Används för att ta bort artiklar från konfigurationen.
- Koppling – Lägger till eller tar bort en koppling mellan två artiklar.
- Rensa Allt – Tar bort allt som man gjort.

För säkerhets skull har vi lagt till en dialogruta när man tryckt på ”Ta Bort” och ”Rensa Allt” där man blir tillfrågad om man verkligen vill utföra sin handling. Eftersom de knapparna ligger lättillgängligt vore det oerhört ineffektivt om man emellanåt råkade ta bort sin konfiguration utan mening.

En artikel som är utlagd på den fria ytan kan antingen vara markerad eller avmarkerad. En markerad artikel kännetecknas av att ramen runt den är klarblå istället för den grå färgen som en omarkerad artikel har. Tanken är att den klarblå färgen ska fånga användarens uppmärksamhet eftersom den färgen är mycket mer lysande än den grå. För att markera/avmarkera en artikel räcker det med att dubbelklicka på rutan.

När man markerar en artikel kommer listorna att uppdateras. Kategorilistan fylls med alla kategorier som innehåller artiklar som är kompatibla mot den markerade artikeln. I de fall där den markerade artikeln kräver en annan artikel för att fungera kommer den kategorin att markeras med röd text i listan. Väljer man en kategori fylls den undre listan med kompatibla artiklar från den valda kategorin.

Längst uppe i det vänstra hörnet finns två knappar för att spara och ladda in konfigurationer. Trycker man på någon av dessa får man upp en dialogruta där man får visa var man vill spara sin fil alternativt var filen man vill ladda finns.

Genom hela arbetet har vi använt en färgkod som är inspirerad av System Anderssons företagsfärger, alltså blått och gult. För att öka läsbarheten på texten har vi satt all text till svart eftersom det ökar kontrasten mot den ljusa bakgrunden.

5 Diskussion och slutsatser

Det finns mycket att säga om detta projekt. Ingen av oss hade tidigare någon erfarenhet av att bygga en produktkonfigurator, men det lät som en utmanande och lärorik utmaning. Nu i efterhand är vi nöjda över att vi fick ett fungerande resultat, men det finns mycket vi upptäckt längs vägen som är värt att tänka igenom och som kunde implementerats om det funnits mer tid. Vi tänker ägna några rader åt detta här.

5.1 Databasen

Vi började vårt projekt med utgångsläget att vi i kompatibilitetstabellen behövde två rader för varje kompatibilitet eftersom det annars kunde bli fel och missuppfattas vilken artikel som var master och vilken som var necessary. Detta är inte helt nödvändigt utan beror på hur relationerna hanteras av konfiguratorn. Man kan ju välja att tolka en rad på följande sätt om necessary eller master är satt. Det är articleID eller categoryID som är master och det är compatibleArticleID eller compatibleCategoryID som krävs av articleID eller categoryID vid necessary. Detta är inte implementerat i skrivandets stund.

Istället för att ha necessary och master i kompatibilitetstabellen utan i en egen tabell skulle man kunna spara plats i databasen. Man skulle då enbart kunna visa vilka artiklar och kategorier som är kompatibla med varandra i den tabellen. Detta skulle kunna ge ytterligare fördelar vad det gäller storlek på databasen.

5.1.1 Kompatibilitetstabellen

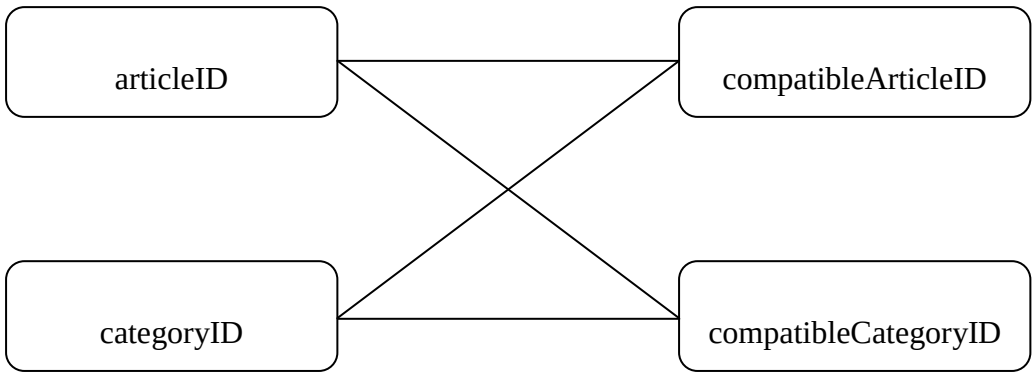
Eftersom en rad i kompatibilitetstabellen enbart ska visa vilka artiklar som är kompatibla med varandra spelar det ingen roll om en artikel står i articleID eller i compatibleArticleID, eller om en kategori står i categoryID eller i compatibleCategoryID. Då det inte spelar någon roll skulle man kunna bygga en funktion som hittar första bästa lediga plats i en rad. Antag artikel A är kompatibel med kategori Z och att man har följande rad i tabellen:

articleID	categoryID	compatibleArticleID	compatibleCategoryID
A	Z		

Och så ska man lägga in artikel B som man ger kompatibilitet mot artikel A och kategori Z, då kan man istället för att skapa en ny rad lägga in den i den existerande raden under compatibleArticleID och man får då raden:

articleID	categoryID	compatibleArticleID	compatibleCategoryID
A	Z	B	

Alternativt skulle man kunna ha någon form av distinktion mellan fälten articleID och compatibleArticleID respektive categoryID och compatibleCategoryID, så att något som står i articleID eller categoryID inte är kompatibla utan endast kompatibla med det som står i compatible-fälten. Kompatibilitetsfälten är inte heller kompatibla med varandra utan endast med articleID och categoryID. Då får man fårhållandet som visas i Figur 21.



Figur 21. Föreslaget kompatibilitetsförhållandet i kompatiblitetestabellen där varje streck symboliserar kompatibilitet.

Med detta sätt skulle det vara lättare att hitta fall som går att kombinera eftersom inte alla inblandade måste vara kompatibla. I princip kan man säga att varje rad har en artikel eller kategori som på den raden kan vara kompatibel mot både en artikel och en kategori.

Om vi då har artikel B som skall sättas som kompatibel med artikel A behöver vi bara låta programmet leta upp första lediga compatibleArticle-fält där A är articleID, eftersom vad som står i compatibleCategoryID inte kommer att påverka compatibleArticleID. Alternativt kan vi söka efter första lediga article-fält där A är compatibleArticleID. Vi har här utgått från att categoryID i första exempelfallet och compatibleCategoryID i andra exempelfallet varit tomma. Man kan sätta det som en regel, att om articleID inte är tom får inget läggas in i categoryID om både compatibleArticleID och compatibleCategoryID är satta och vice versa. Likadant gör man då för villkoren för compatibleArticleID och compatibleCategoryID om articleID och categoryID inte är tomma.

Man kan dessutom kombinera de båda förslagen ovan, och utgå från det senaste alternativet men inte låsa categoryID om articleID inte är tomt utan om man hittar en kategori som är kompatibelt med samma artikel och kategori utan då kan lägga in den. Detta är förmodligen det alternativ som är mest praktiskt då det inte försvårar kompatibilitetsmöjligheterna i det senare alternativet ovan och har större sannolikhet att inträffa än det första exemplet där alla fyra artiklarna och kategorierna måste vara kompatibla med varandra för att fylla en rad i databasen.

5.1.2 Necessary och mastertabellen

Man skulle i fallen ovan behöva skapa en ny tabell för att hantera necessary och master fallen. Om vi följer mönstret i kompatiblitetstabellen bör den innehålla sex fält.

- articleID
- categoryID
- compatibleArticleID
- compatibleCategoryID
- necessary
- master

Strukturen kan likna den vi diskuterade ovan gällande kompatiblitetstabellen, men med undantaget att necessary och master kan sättas.

Antag att vi har artikel A som är kompatibel med artikel B och att artikel A kräver artikel B då sätts A som articleID och B som compatibleArticleID, dessutom sätts naturligtvis necessary till 'yes'. Om man sedan har en kategori som också kräver artikel B kan man då sätta in den på samma rad under categoryID.

Man skulle kunna tänka sig att enbart ha den här typen av tabell för att hantera hela kompatibiliteten och lämna necessary och master tomma i vanliga fall, likande det vi gjorde i vår databas fast med effektivare hantering av tabellerna i konfiguratorn. Det finns dock ett förslag som skulle bli mycket enklare att hantera om necessary och master hanterades av en egen tabell.

5.1.3 Optimerande funktioner

Man skulle kunna skapa optimerande funktioner som ”städade” kompatibilitetstabellen.

Antag att man har lagt in alla artiklar ur kategori Z utom artikel A i databasen som kompatibla med artikel B som tillhör kategori Y. Man lägger sen till att artikel A också är kompatibel med artikel B. Då skulle en strukturerande funktion kunna skapa en kompatibilitet mellan kategori Z och artikel B och sen ta bort all kompatibilitet i kompatibilitetstabellen mellan artiklar ur kategori Z och artikel B. Speciella fall av krav och master skulle ändå sparas i necessary och mastertabellen.

En sådan funktion skulle också kunna hantera fall när en hel kategori är kompatibel med en artikel och sen tas kompatibilitet bort från en artikel tillhörande kategorin.

Antag att hela kategorin Z är kompatibel med kategorin Y. Kompatibiliteten mellan artikel A som ingår i kategori Z och kategori Y tas bort. Då skulle en strukturerande funktion skapa kompatibilitet mellan alla artiklar tillhörande kategori Z och kategorin Y, förutom artikel A. Dessutom skulle kompatibiliteten mellan kategori Z och kategori Y tas bort.

Denna princip skulle också kunna appliceras på necessary och mastertabellen.

5.1.4 Övrigt angående databasen

Vi har i vår prototyp inte konfigurerat någon låsning av databasen för skrivning, vilket man bör göra när flera användare ska använda den samtidigt.

När en artikel ur databasen tas bort bör också de rader där denna artikel finns med i kompatiblitets- och eventuella rader i necessary och mastertabellen (om man väljer att implementera dessa) tas bort.

Man skulle kunna gå vidare med den idé som vi valde bort i genomförandedelen under design. Nämligen den som skulle kunna hålla reda på om artiklar som inte är kopplade till varandra, men som ingår i samma konfiguration är kompatibla. Man skulle kunna definiera både kompatibilitet och kopplingsbarhet i databasen. På så sätt skulle man kunna undvika att en processor kan kopplas direkt till ett ramminne även om dessa är kompatibla, som i stycket mellan figur 13 och figur 14. Detta är inget ämne vi gått djupare in i och förslagen som följer bygger inte på detta förhållningssätt mellan artiklar.

5.2 Kompatibilitetsverktyget

Det finns två förbättringar som vi skulle vilja göra med kompatibilitetsverktyget. Det första gäller gränssnittet. För att förenkla skapandet av kompatibilitet mellan artiklar skulle vi vilja kunna få en lista på vilka artiklar som finns i databasen samt kunna se vilka artiklar som är kompatibla. Det smidigaste tror vi är att välja en artikel eller kategori eller en kategori med artikelvillkor i artikeldelen av kompatibilitetsverktyget och sen kunna lista kompatibla artiklar och kategorier för det valda alternativet.

En annan funktionalitet vi skulle velat implementera vore möjligheten att kunna uppdatera en kompatibilitet. Detta är endast nödvändig då status på necessary och master ändras.

5.3 Konfigurationsverktyget

Relationerna mellan artiklar är i nuläget relativt intetsägande. För att underlätta förståelsen för vad linjen mellan artiklarna betyder skulle man kunna sätta riktade pilar för att märka ut vilken relation de har. Till exempel kan man enligt de notationer som togs upp i avsnittet om UML åskådliggöra att en artikel består av en annan artikel.

Bland de saker som vi diskuterade under arbetets gång ägnade vi mycket tid åt det här med master. Det fanns många tankar och idéer på hur det skulle kunna lösas men det har inte hunnits implementeras i skrivandets stund.

Mastern skulle kunna vara en startartikel, alltså den artikel som man var tvungen att lägga ut först på den fria ytan. Om man till exempel vill bygga en cykel börjar man fördelaktligen med ramen och inte pakethållaren. Detta skulle även innebära att man endast kan ha en master åt gången i samma konfiguration.

Mastern skulle också kunna vara en artikel som består av andra artiklar. Istället för att som i första exemplet börja med en cykelram lägger man nu ut en cykel på den fria ytan som sedan består av ram, styre, däck och så vidare. Om man hade valt den här varianten hade man varit tvungen att sätta pilar på kopplingarna för att tydligt visa vad som är master. Dessutom skulle detta innebära en helt annan typ av konfiguration eftersom varje artikel skulle vara direkt kopplad till master, i det här fallet cykeln. De skulle inte ha någon inbördes koppling och man skulle då behöva använda sig av en mer komplexa typer av kompatibilitetsrelationer. Liknande det som nämns i slutet på 5.1.4 ”Övrigt angående databasen”, där en typ av koppling var i relation till mastern, men de individuella artiklarna var kompatibla med varandra.

Gemensamt för dessa alternativ till master är att man skulle kunna implementera att om man ändrar antalet på masterartikeln kommer de andra underliggande artiklarna automatiskt att ändra antal med bibehållet förhållande. Till exempel om en bil kräver fyra däck och man ändrar antalet bilar till två kommer antalet däck bli åtta.

Eftersom den fria ytan som man arbetar på har en begränsad storlek skulle det vara smidigt om man kunde minimera delar av konfigurationen. Antag att man har byggt något komplext som till exempel en bil. Bilen består bland annat av en motor som i sin tur består av många små komponenter. Motorns beståndsdelar är säkert i många fall överflödigt med information och då ska man kunna dölja motorns underliggande artiklar genom att trycka på ett minustecken. Vill man sedan utforska motorn närmre trycker man på ett plustecken för att se alla delar igen. Alla som har använt utforskaren i Windows känner igen sig hur detta fungerar.

En annan sak som vi i skrivandets stund inte riktigt har implementerat är att vi vill få den röda texten som uppstår när en artikel krävde en annan artikel att ändras. När man lagt ut den krävda artikeln vill vi att den röda texten ska bli svart igen eftersom kravet uppfyllts.

5.4 Slutats

5.4.1 System Anderssons mål

Om man ska se till de mål som skulle uppfyllas uppfyllde vi målet att skapa en konfigurator i WPF med hjälp Visual Studio 2010 och SQL-Server 2008 Express, som endast kan koppla kompatibla artiklar med varandra. Konfigureringen sker med ”drag and drop” och artiklarna listas till höger.

Vi uppfyllde dock inte målet att kunna hantera artiklar med obestämda attributvärden, och kunna räkna ut antalet av kompatibla artiklar för dessa. Även om vi redogör för vårt förslag i genomförandedelen under design hann vi aldrig implementera och pröva detta. En ändring vi skulle göra idag, efter diskussionen kring kompatibilitets- och necessary och mastertabellen, vore att placera fälten ”min antal”, ”antal/ mått”, ”måttvärde” och ”måttattribut” i necessary och

mastertabellen för att kunna hantera kompatibilitet på ett smidigt sätt (se Figur 22).

articleID	compatibleArticleID	min	add	attributevalue	attribute
Bordskiva	Bordsben	4	2	1	Length

Figur 22. Exempel på hur necessary och mastertabellen kan se ut. Fälten categoryID, compatibleCategoryID, necessary och master har valts att inte visas för att spara plats.

5.4.2 Våra mål

Hur då uppfylldes våra mål? Vi lyckades skapa en fungerande produktkonfigurator, även om det finns mycket att förbättra. Vi tycker själva att vi lyckades utforska olika relationsmöjligheter mellan artiklar. Vi kan åtminstone mycket mer idag än när vi började projektet. Vi har också lyckats skapa ett program i grafisk miljö som kan hantera ett grafiskt användargränssnitt. Visst finns det mycket som vi kunde gjort annorlunda men det är sporrande att vi upptäckte så mycket och lärde oss så mycket på vägen.

5.5 Slutord

Vi har efter en tids rapportskrivande fått en distans till designen och implementeringen och vi har sett på vissa problem på ett nytt sätt vilket tydligt kan ses här i diskussionskapitlet. Det är med blandade känslor då det är spännande att hitta förbättringar men tråkigt att inte hinna genomföra dem. Allt som allt är vi nöjda med vårt examensarbete och med vår insats. Vi är tacksamma för den hjälp vi fått av vår handledare och för möjligheten att genomföra detta arbete åt System Andersson.

6 Referenser

[1] Microsoft - Introduction to WPF (2010)
http://msdn.microsoft.com/en-us/library/aa970268.aspx
(Acc. 2010-06-24)

[2] Chappell, David (2006) *Understanding .NET, Second Edition.*
Addison-Wesley Professional, Toronto, ISBN 0-321-19404-6

[3] Microsoft - .NET Framework Conceptual Overview (2010)
http://msdn.microsoft.com/library/zw4w595w.aspx
(Acc. 2010-06-24)

[4] Microsoft - .NET Framework 3.0 Versioning and Deployment (2010)
http://msdn.microsoft.com/en-us/netframework/aa663314.aspx
(Acc. 2010-06-24)

[5] Microsoft - What is Windows Communication Foundation (2010)
http://msdn.microsoft.com/en-us/library/ms731082%28v=VS.100%29.aspx
(Acc. 2010-06-24)

[6] Microsoft - The Workflow Way: Understanding Windows Workflow Foundation (2010) http://msdn.microsoft.com/en-us/library/dd851337.aspx
(Acc. 2010-06-24)

[7] Microsoft - Introducing Windows CardSpace (2010)
http://msdn.microsoft.com/en-us/library/aa480189.aspx
(Acc. 2010-06-24)

[8] Microsoft - Officiell webbplats för Visual Studio (2010)
http://www.microsoft.com/visualstudio/sv-se/
(Acc. 2010-06-24)

[9] Microsoft Expression Blend 3 (2010)
http://www.microsoft.com/expression/products/Blend_Overview.aspx
(Acc. 2010-06-24)

[10] Kriegel, Alex; M. Trukhnov, Boris (2008) *SQL Bible, Second Edition*
John Wiley & Sons, New Jersey, ISBN 978-0-470-22906-4

[11] Buckley, Fred; Lewinter, Marty (2003) *A friendly introduction to graph theory*.
Prentice Hall, New Jersey, ISBN 0-13-066949-0

[12] Blaja, M. Rumbaugh, J. (2005) *Object-Oriented Modeling and Design with UML*.
Pearson Education Ltd., London, ISBN 0-13-196859-9

[13] Codeproject - WPF Drag-and-Drop Smorgasbord
http://www.codeproject.com/KB/WPF/WpfDragAndDropSmorgasbord.aspx
(Acc. 2010-06-24)

6.1 **Bildreferenser**

[Figur 4] http://upload.wikimedia.org/wikipedia/commons/d/d3/DotNet.svg

7 Sökord