Utveckling och integrering av kursbokningssystemet Puzzel

Ulf Hedlund

Examensarbete

Informationsteknik

2012

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	
Författare:	Ulf Hedlund
Arbetets namn:	Utveckling och integrering av kursbokningsverktyget Puzzel
Handledare (Arcada):	Göran Pulkkis
Uppdragsgivare:	Arcada - Nylands svenska yrkeshögskola

Sammandrag:
Varje år behöver utbildningsinstitutioner planera läsordningar. Rum ska bokas, resurser måste delas på och dubbelbokningar får inte ske. Processen är ofta långdragen och kan ha flera hundra personer inblandade. Kursbokningsverktyget Puzzel har utvecklats för att effektivisera bokningsprocessen för yrkeshögskolan Arcada.

Denna rapport är indelad i tre delar. I den första delen ges en överblick av de programmeringsramverk som använts vid utvecklingen av Puzzel, huvudsakligen .NET Framework, Windows Presentation Foundation och Windows Communication Foundation. Del två kartlägger befintliga bokningssystem inom yrkeshögskolan Arcada samt diskuterar hur bokningsprocessen kan effektiviseras. Del tre ger en översikt av hur Puzzel är uppbyggt samt hur integreringen mot Arcadas befintliga system har genomförts.

Nyckelord:	Arcada, C#, .NET, Windows Communication Foundation, WCF, Windows Presentation Foundation, WPF, bokningssystem, ASP .NET MVC
Sidantal:	84
Språk:	Svenska
Datum för godkännande:	14.11.2012

DEGREE THESIS	
Arcada	
Degree Programme:	Information technology
Identification number:	
Author:	Ulf Hedlund
Title:	Development and Integration of the Course Booking System Puzzel
Supervisor (Arcada):	Göran Pulkkis
Commissioned by:	Arcada University of Applied Sciences

Abstract:
Education institutions need to plan and book their course schedules every year. Rooms have to be found and booked, resources shared and double bookings should not happen. The process often takes a long time and hundreds of people can be involved. Puzzel is a course booking tool that has been built to streamline the booking process and save administrative costs.

This report consists of three different parts. In the first part an overview of the programing frameworks used in the development of Puzzel is given. Part two describes the booking systems in use within Arcada University of Applied Sciences and discusses how the booking process can be improved. Part three gives an overview of how Puzzel is built and how the integration with Arcadas systems has been made. Part three puts focus on how data is handled and exchanged, not so much on the user interface.

Keywords:	Arcada, C#, .NET, Windows Communication Foundation, Windows Presentation Foundation, WPF, booking system, ASP .NET MVC
Number of pages:	84
Language:	Swedish
Date of acceptance:	14.11.2012

INNEHÅLL

Figurer

Förkortningar

AD	Active Directory
ABEX	Arcada Booking Exchange
ARBS	Arcada RumsBokningsSystem
ASP .NET MVC 3	Active Server Page .NET Model View Controler 3
ASTA	Arcadas STudieAdministration
CIL	Common Intermediate Language
CLR	Common Language Runtime
CPU	Central Processing Unit
GPU	Graphic Processing Unit
GUID	Globaly Unique Identifier
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
LINQ	Language Integrated Query
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
WCF	Windows Communication Foundation
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

1 INLEDNING

1.1 Bakgrund

Arcada har sedan år 2004 haft ett digitalt rumsbokningssystem i bruk, kallat ARBS (Arcada Room Booking System). Under årens gång så har det visat sig att ARBS kräver en hel del från sina användare, och mycket tid och resurser läggs ner på att få alla läsordningar schemalaggda varje år. Man började fundera på vad man skulle kunna göra för att underlätta schemaläggningsprocessen, och idén om ett rumsoptimeringsprogramm föddes.

Efter att ha funderat på olika rumsoptimeringslösningar så kom man underfund med att man kanske inte behövde göra allting så komplicerat för att underlätta schemaläggningsprocessen. Det var i det här skedet som idén till Puzzel föddes - ett verktyg som hjälpte till med schemaläggning av kurser. Puzzel projektet började som en insticksmodul till Outlook, men övergick snart till en skrivbordsapplikation. Under den här tiden så var en annan student anställd för att jobba på utvecklingen, men denna student hoppade av projektet en tid efter att fokusen övergick till en skrivbordsapplikation. I det här skedet så fick jag erbjudande av Arcada om att jobba vidare på Puzzel, och det arbete som jag har gjort beskrivs i den här rapporten.

1.2 Syfte

Syftet med projektet har varit att underlätta den kursbokningsprocess som lärare går igenom varje år på yrkeshögskolan Arcada, genom att erbjuda ett verktyg, Puzzel, som förenklar processen för slutanvändaren. Hur man kunde underlätta processen var i början av projektet inte helt klart utan har kartlagts under projektets gång. Hur processen som man använder i Puzzel skiljer sig från det gamla bokningssystemet, ARBS, beskrivs i kapitlet Effektivisering av bokningsprocessen.

Puzzel har utvecklats som en slavapplikation till befintliga system som används inom Arcada, det vill säga applikationen är starkt beroende av den data som samlas in från andra system.

Detta examensarbete handlar om utvecklingen och integreringen av kursbokningsverktyget Puzzel, som beställts av och är under skrivande stund i bruk i yrkeshögskolan Arcada. Rapporten beskriver i stora drag den arkitektur som används för både klient- och serverapplikation samt ger en översikt av de olika programmeringsramverk som använts. Rapporten utgår ifrån att läsaren har kunskap inom programmering.

1.3 Avgränsningar

De programmeringsspråk som används i utvecklingen av Puzzel var från början givna, och ingen diskussion av alternativa språk kommer ingå i detta arbete.

Den metadata som samlas in vid användning av Puzzel öppnar upp många intressanta dörrar, bland annat rums- och läsordningsoptimering. Jag kommer att diskutera vidareutvecklingsmöjligheter i kapitlet Vidareutveckling, men optimeringslösningar är inte en del av detta arbete.

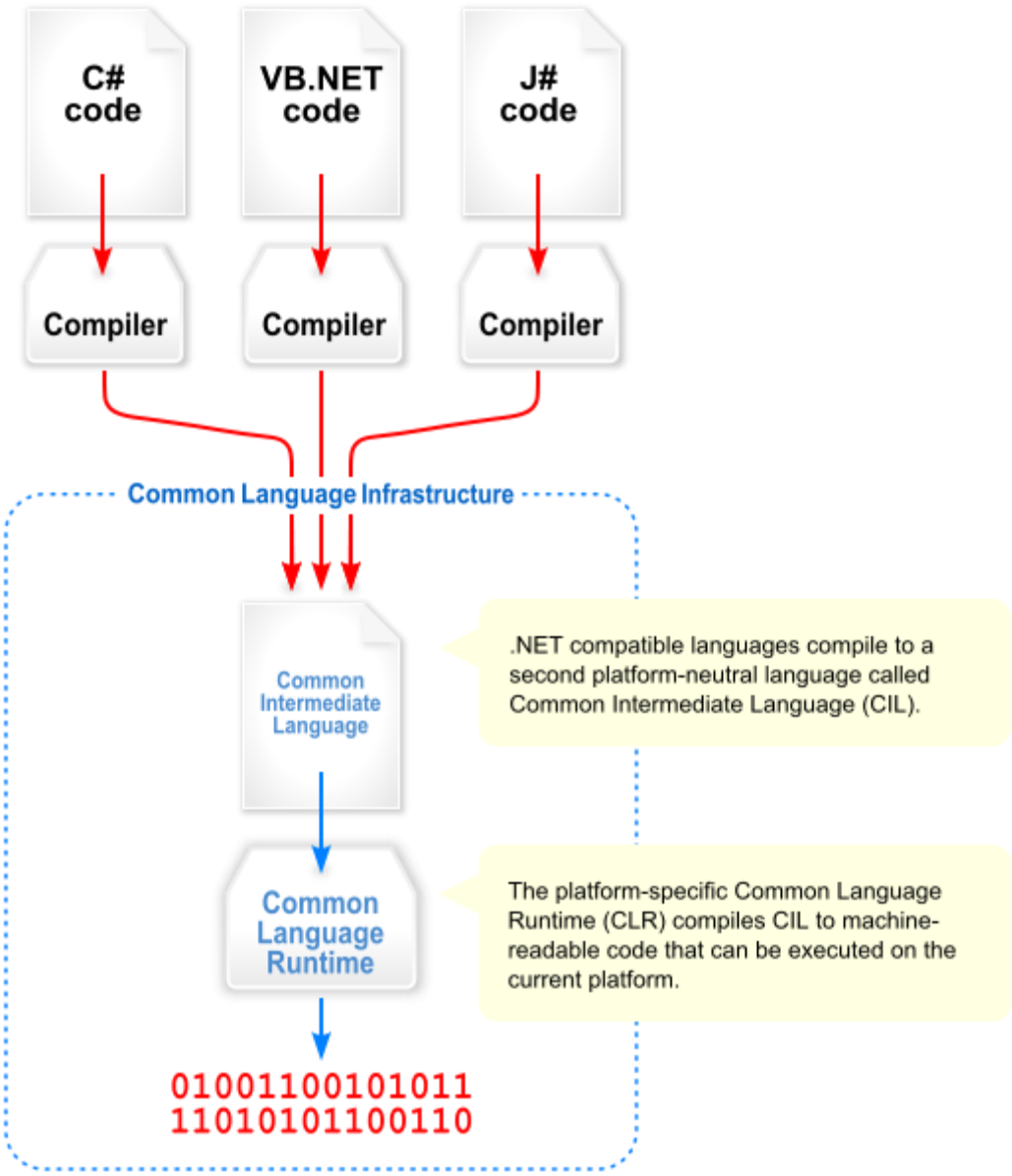
2 PROGRAMMERINGSRAMVERK

Ett flertal olika programmeringsramverk har använts under projektets gång. I detta kapitel beskrivs de olika programmeringsramverk som använts för utveckling av Puzzel.

2.1 .Net 4 Framework

.NET 4 är ett av Microsoft utvecklat ramverk som används i första hand i Windowsmiljö. I ramverket används ett flertal programmeringsspråk, bland annat C#, Visual Basic .NET, Visual C++ och J#. Vid kompilering kompileras dessa språk till ett plattformsneutralt språk kallat Common Intermediate Language(CIL). Tack vare att detta är det t.ex. fullt möjligt att använda komponenter skrivna i Visual C++ när man själv kodar i C#. Biblioteken och det slutgiltiga språket, CIL, är sist och slutligen de samma. (Microsoft, 2010c)

Program skrivna med .NET 4 exekveras i en mjukvarumiljö som går under namnet Common Language Runtime (CLR). CLR hanterar viktiga uppgifter såsom säkerhet, minneshantering och undantagshantering. (Microsoft, 2010c)



Figur 1, Visuell översikt av CIL och CLR

2.1.1 LINQ

LINQ står för Language Integrated Query och introducerades i .NET 3.5. LINQ handlar om att göra förfrågningar och få data från en datakälla. LINQ introducerar en SQL-liknande syntax som kan tillämpas på objekt såsom listor, databaskällor eller XML-dokument. Genom att använda LINQ undviker man en hel del manuellt arbete när det kommer till filtrering eller sortering av datakällor. (Microsoft, 2010a)

```
public class MyObject
{
 public String Id { get; set; }
 public String Name { get; set; }
 public String Information { get; set; }
}
public class Customer
{
 public String Id { get; set; }
 public List<String> objectIdList { get; set; }
}

.....

Customer Customer = this.getCustomer();
List<MyObject> ObjectList = this.getObjects();
IEnumerable<String> objectNames = ObjectList.Where
              (o => Customer.objectIdList.Contains(o.Id)).
              Select(o => o.Name);
```

Figur 2, Ett exempel på LINQ

I Figur 2 ses en lista på objekt och en kund. En kund har en lista på objektidentifieringssträngar, och man vill veta vad för namn dessa objekt har. Sökningen börjar med ObjectList, som innehåller alla objekt, och sedan frågar man efter alla objekt vars Id ingår i kundens lista. Nu är man inte intresserade av all information som finns i dessa objekt, utan endast av namnet. För att få ut den informationen kan man använda sig av Select, vilket gör att man kan plocka ut precis de element som man är interesserad av. Resultatet är en lista av typ IEnumerable. IEnumerable fungerar som en vanlig lista, men innehållet i listan kommer inte att skapas innan man försöker komma åt innehållet genom att t.ex. använda den i en foreach-loop eller genom att göra ytterligare en LINQ-förfrågan på listan. Notera att resultatet av en LINQ-förfrågning inte modifierar den datakälla som man gör förfrågningen mot, utan genererar en ny lista med referenser till samma objekt som finns i den ursprungliga listan. (Jon Skeet, 2011)

2.2 Windows Presentation Foundation

WPF (Windows Presentation Foundation) är ett grafiskt ramverk för Windows, och bygger på Ramverket .NET. WPF skiljer sig markant från föregångarna när det kommer till att visa grafiska användargränssnitt för Windows. Innan WPF existerade användes två Windows komponenter för att skapa grafiska användargränssnitt, User32 och

GDI/GDI+. Dessa två komponenter var ansvariga för det traditionella Windowsutseendet och beteendet, och ger tillgång till skapande och visning av knappar, fönster, textutskrift o.s.v. Det spelade ingen roll om man skrivit en applikation i Windows Forms eller Visual Basic 6, samma komponenter används. Dessa komponenter är över ett decennium gamla, och har en hel del begräsningar. WPF har frångått dessa komponenter, och använder istället DirectX för visning av grafik. (MacDonald, 2010)

Tack vare att WPF använder sig av DirectX blir grafiken hårdvaruaccelererad, det vill säga att det mesta arbetet som görs när man visar grafik sker på GPU:n (Graphic Processing Unit, grafikprocessor) istället för CPU:n (Central Processing Unit, processor). Detta i sin tur innebär att det finns möjligheter att stödja tyngre grafiska operationer, som t.ex. anti-aliasing och vektorgrafik i applikationer. (MacDonald, 2010)

2.2.1 XAML

XAML står för Extensible Application Markup Language och är det språk som används för att konstruera grafiska avändargränssnitt för WPF, Silverlight och Silverlight för Windows Phone 7. (MacDonald, 2010)

Det är fullt möjligt att skapa sina användargränssnitt genom kod, men genom att definiera användargränssnitten i XAML separerar man användargränssnitt och bakomliggande logik till två olika filer, som sedan alltid kommer som ett par (t.ex., MainWindow.xaml innehåller användargränssnittet, MainWindow.cs innehåller koden). Detta innebär att man kan utveckla funktionaliteten hos en applikation oberoende av designen. För att koppla in det grafiska användargränssnittet använder man sig av händelser (eng. events).

```
<Button Name="Knapp1"  Grid.Row="1"
Content="Klicka här!" Click="Knapp1_Click"/>
```

Figur 3, Hur man skapar en knapp i XAML

I Figur 3 har en knapp definierats i xaml-filen, och i definitionen har man angett ett namn på den funktion som ska ta emot händelsen som skapas då man trycker på knap-

pen (Click=”Knapp1_Click”). För att ta emot händelsen skapas sedan en funktion i .cs-filen, se Figur 4.

```
private void Knapp1_Click(object sender, RoutedEventArgs e)
{
    //Gör saker här
}
```

Figur 4, Funktion för mottagande av knapptryckningshändelse

Separation av användargränssnitt och kod är långt ifrån det enda nya som WPF har att komma med. Det är möjligt att på ett snabbt och enkelt sätt skapa animationer genom så kallade StoryBoards, som antingen kan fortgå hela tiden, eller startas genom att användaren gör någonting, såsom t.ex. för musen över en knapp.

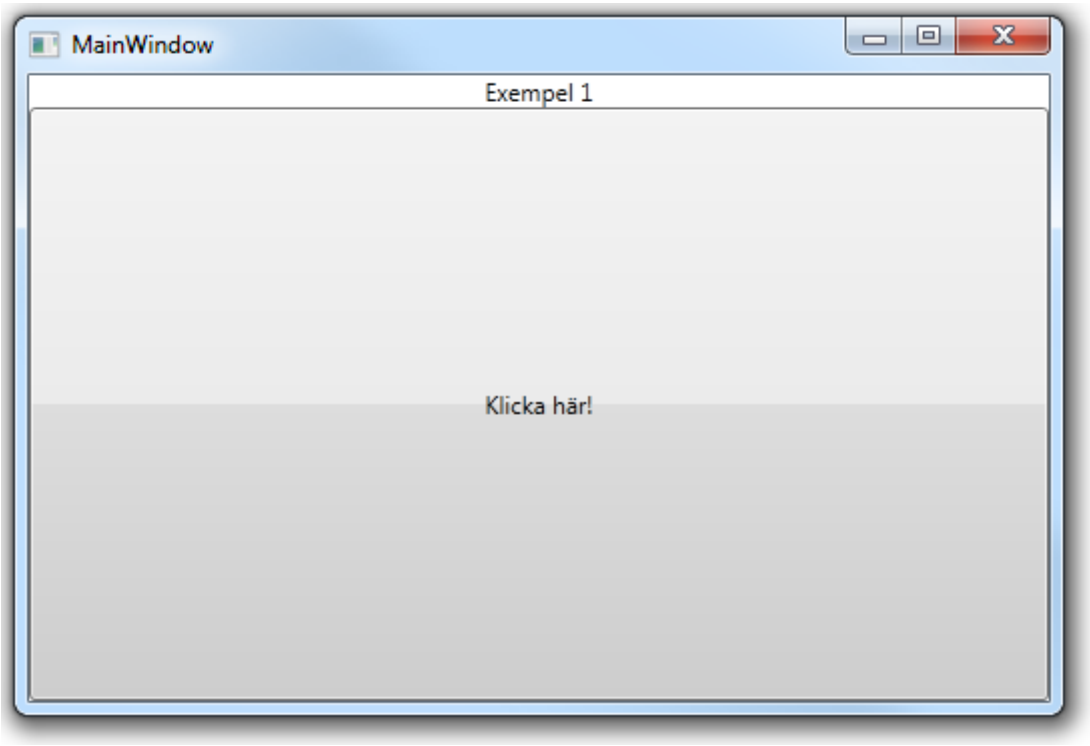
Det som alltid har varit svårt när man skapar grafiska användargränssnitt är skalbarhet. T.ex. i Windows Forms realiseras det grafiska användargränssnittet först och främst för en upplösning, då de komponenter som används inte är särskilt anpassbara. I WPF har man gjort allting mycket mer flexibelt, i och med att alla komponenter visas direkt av WPF och inte av GDI/GDI+. Därför har man full kontroll över vad som händer. Tack vare hårdvaru-accelerationen visas näst intill alla element och kontroller i vektorform, vilket gör att de kan visas i vilken storlek som helst. Det finns också väldigt flexibla ”layout”-element till förfogande, där de automatiskt kan anpassa hur mycket plats ett element ska ta upp efter vad det innehåller. (MacDonald, 2010)

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"></RowDefinition>
        <RowDefinition Height="*"></RowDefinition>
    </Grid.RowDefinitions>

    <TextBlock Grid.Row="0" Text="Exempel 1"
               HorizontalAlignment="Center"/>
    <Button Name="Knapp1"  Grid.Row="1"
               Content="Klicka här!"
               Click="Knapp1_Click"/>
</Grid>
```

Figur 5, Exempel på ett enkelt användargränssnitt i XAML

I Figur 5 ses ett exempel på ett mycket enkelt användargränssnitt som innehåller ett textblock med texten ”Exempel 1”, och en knapp. Dessa två ligger i ett Grid-element.

Man har definierat att första radens höjd är automatisk, och andra raden ska ta upp resten av platsen. Resultatet visas i Figur 6.



Figur 6, Exempel på flexibel layout med XAML

2.2.2 Bindings

Någonting som brukar vara väldigt tidskrävande när man gör ett användargränssnitt är att man presenterar sin data på ett tillfredställande sätt. Många timmar sätts på att manuellt binda ihop visuella element med den data man vill visa. Om man uppdaterar data bör ofta det visuella elementet uppdateras också.

I WPF har den här processen effektiviserats genom att skilja åt modell, visning och visningsmodell (designmönster, eng. view-model-view model). Detta innebär att den data man vill visa (modellen) hanteras skilt. När man vill visa den så ger man data till en vy, och där finns sedan en visningsmodell. Alla tre komponenterna (modell, vy, visningsmodell) behöver inte vara medvetna om varandra, men visningsmodellen bör planeras så att man får se den data man vill. Man binder ihop de tre komponenterna med en s.k. Binding, se Figur 7.

```
<TextBlock Name="MyTextBox" Text="{Binding Path=Name}"/>
```

Figur 7, Exempel på en Binding

Här definieras först i XAML ett textfält, där man säger att texten I fältet skall bindas till sökstigen Name. I det här läget vet man bara om att ett objekt med en egenskap med namn Name kommer att finnas, och när objektet kopplas till textfältet kommer Name-fältet att visas.

```
    public class Customer
    {
        public String Name { get; set; }
        public String Id { get; set; }
        public String Email { get; set; }
    }
.....
Customer customer = new Customer { Name = "Nisse", Email = "nisse@nisse.com",
                                   Id = "FAer#421"};
MyTextBox.DataContext = customer;
```

Figur 8, Hur man utnyttjar Bindings

I Figur 8 visas en klass med namn Customer, som har egenskaperna Name, Id och Email. Ett objekt i klassen Customer skapas, och sedan sätts DataContext på textfältet till det nya objektet. Resultatet är att ”Nisse” visas i textfältet. DataContext innebär det sammanhang som elementet använder för att söka efter bindningar, i detta fall hittas Name i Customer-objektet.

Det är inte bara element som man har definierat i kod som det är möjligt att göra bindningar till, det är också möjligt att kedja ihop element. Ett exempel visas I Figur 9.

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition></ColumnDefinition>
        <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <ComboBox Name="MyComboBox" DisplayMemberPath="Name"
          VerticalAlignment="Center"></ComboBox>
    <StackPanel Grid.Column="1" DataContext="{Binding ElementName=MyComboBox,
           Path=SelectedItem}">
        <TextBlock Text="{Binding Path=Name}"/>
        <TextBlock Text="{Binding Path=Id}"/>
        <TextBlock Text="{Binding Path=Email}"/>
    </StackPanel>
</Grid>
```

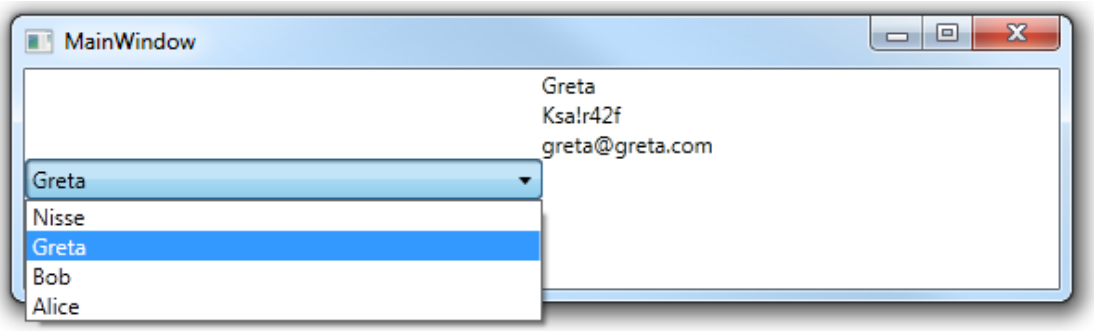
Figur 9, Definition av kedjade bindningar

Här skapas ett Grid-element med två kolumner. I första kolumnen (alltså kolumn nr. 0, detta värde behöver man inte ställa in manuellt då detta är ursprungsläget för alla element) skapas en ComboBox, en flervalsmeny, vid namn MyComboBox. Sedan definieras att MyComboBox skall visa Name som sitt visningsvärde när den blir kopplad till ett objekt.

I nästa kolumn skapas en StackPanel, det vill säga ett layout-element som kommer att stapla alla innehållande element på en rad. DataContext, sammanhanget, ska vara kopplat till MyComboBox, men sedan ska den gå in i egenskapen SelectedItem för att hitta det objekt som den ska koppla till. Resultatet blir att när man byter valt fält i MyComboBox så kommer även de fält som ligger inom StackPanel att uppdateras, då sökvägen SelectedItem pekar på ett annat objekt. Det enda som behöver styras genom kod är genereringen av data, samt att i MyComboBox skapa en referens till lisan på de element som skall genereras. Se Figur 10 och 11.

```
List<Customer> customers = new List<Customer>();
customers.Add( new Customer { Name = "Nisse", Email = "nisse@nisse.com",
                              Id = "FAer#421" });
customers.Add( new Customer { Name = "Greta", Email = "greta@greta.com",
                              Id = "Ksa!r42f" });
customers.Add( new Customer { Name = "Bob", Email = "bob@bop.com",
                               Id = "FawrF24.d" });
customers.Add( new Customer { Name = "Alice", Email = "alice@alice.com",
                              Id = "Kertsa1" });
MyComboBox.ItemsSource=customers;
```

Figur 10, Koppling av element



Figur 11, Resultat av kedjad data bindning

Det finns också möjlighet att få sitt användargränssnitt att uppdateras så fort man gör ändringar i det objekt som man har bundit element till, men detta kräver lite mer arbete i datamodellen. För att få användargränssnittet att uppdateras när man ändrar den bakomliggande datamodellen måste datamodellen realisera ett gränssnitt kallat INotifyPropertyChanged. INotifyPropertyChanged ger tillgång till händelsen PropertyChanged. PropertyChanged är en händelse som det grafiska användargränssnittet lyssnar på när en bindning har gjorts mot ett objekt. Så fort en händelse med rätta parametrar skickas iväg så kommer det grafiska gränssnittet att gå tillbaka till objektet och uppdatera det relevanta värdet. Att realisera INotifyPropertyChanged kräver lite manuellt jobb, men sparar mycket tid då man därefter inte behöver uppdatera gränssnittet manuellt. Observera att det endast lönar sig att realisera detta för de variabler som man kommer att uppdatera.

Ett exempel på den tidigare visade Customer-klassen som har realiserat INotifyPropertyChanged ses i Figur 12.

```
public class Customer : INotifyPropertyChanged
{
    private String name;
    public String Name
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
            PropertyChangedHandler("Name");
        }
    }
    private String id;
    public String Id
    {
        get
        {
            return id;
        }
        set
        {
            id = value;
            PropertyChangedHandler("Id");
        }
    }
    private String email;
    public String Email
    {
        get
        {
            return email;
        }
        set
        {
            email = value;
            PropertyChangedHandler("Email");
        }
    }

    private void PropertyChangedHandler(String property)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(property));
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
}
```

Figur 12, Realisering av INotifyPropertyChanged

2.2.3 Click once

När man väl har släppt ut sin skrivbordsapplikation till sina användare är det bekymmersamt att göra ändringar i programmet. Släpper man ut en uppdatering kan det ta väldigt lång tid tills alla användare har uppdaterat sina program. WPF erbjuder en teknik kallad Click once som ett hjälpmedel för detta. Click once skapar ett installationspaket för din applikation, och har man en Windows Server till förfogande är det möjligt att ange en sökstig som pekar på var man kan ladda ner applikationen. I Click once kan man sedan konfigurera en applikation till att alltid gå och titta på serveren för att se om det finns en ny version, antingen vid uppstart av applikationen eller när man stänger ner den. Hittas en ny version så kan applikationen antingen automatiskt uppdateras, eller fråga användaren om han/hon vill uppdatera applikationen. Tack vare detta är det möjligt att göra kontinuerliga uppdateringar för skrivbordsapplikationer.

2.3 Windows Communication Foundation

Windows Communication Foundation (WCF) är ett ramverk för serverbaserad tjänstutveckling som bygger på .NET ramverket. WCF är byggt för att stödja många meddelande-typer. En tjänst byggd med WCF kan t.ex. använda samma logik för överföring av meddelanden i SOAP-format (Simple Object Access Protocol), XML (Extensible Markup Language) eller JSON (JavaScript Object Notation). Transporten av meddelanden kan ske över många olika kanaler, såsom Http(s), Tcp(s), WebHttp(s) o.s.v. (Microsoft, 2010b)

Kodandet av WCF består av tre huvudsakliga delar, gränssnitt för din tjänst (interface), en bakomliggande klass som realiserar gränssnittet och datakontrakt.

2.3.1 WCF-gränssnitt och EndPoints

För att exponera din WCF-tjänst behöver man skapa ett gränssnitt som definierar vad tjänsten som man utvecklar gör. Gränssnittet används sedan av klienter för att ta kontakt med tjänsten. Ett exempel på ett enkelt gränssnitt ses i Figur 13.

```
[ServiceContract]
  public interface IService1
  {
      [OperationContract]
      string GetData(int value);

  }
```

Figur 13, Exempel på enkelt WCF-gränssnitt

För att WCF ska hantera gränssnittet som ett tjänste-gränssnitt behövs en extra uppmärkning av koden. Ovanför definitionen av gränssnittet återfinns texten [ServiceContract], som innebär att kommande kod ska hanteras som ett tjänstekontrakt som klienter kan använda för att kontakta tjänsten. Ovanför GetData-metoden så ses texten [OperationContract], vilket innebär att klienter kommer att kunna anropa metoden.

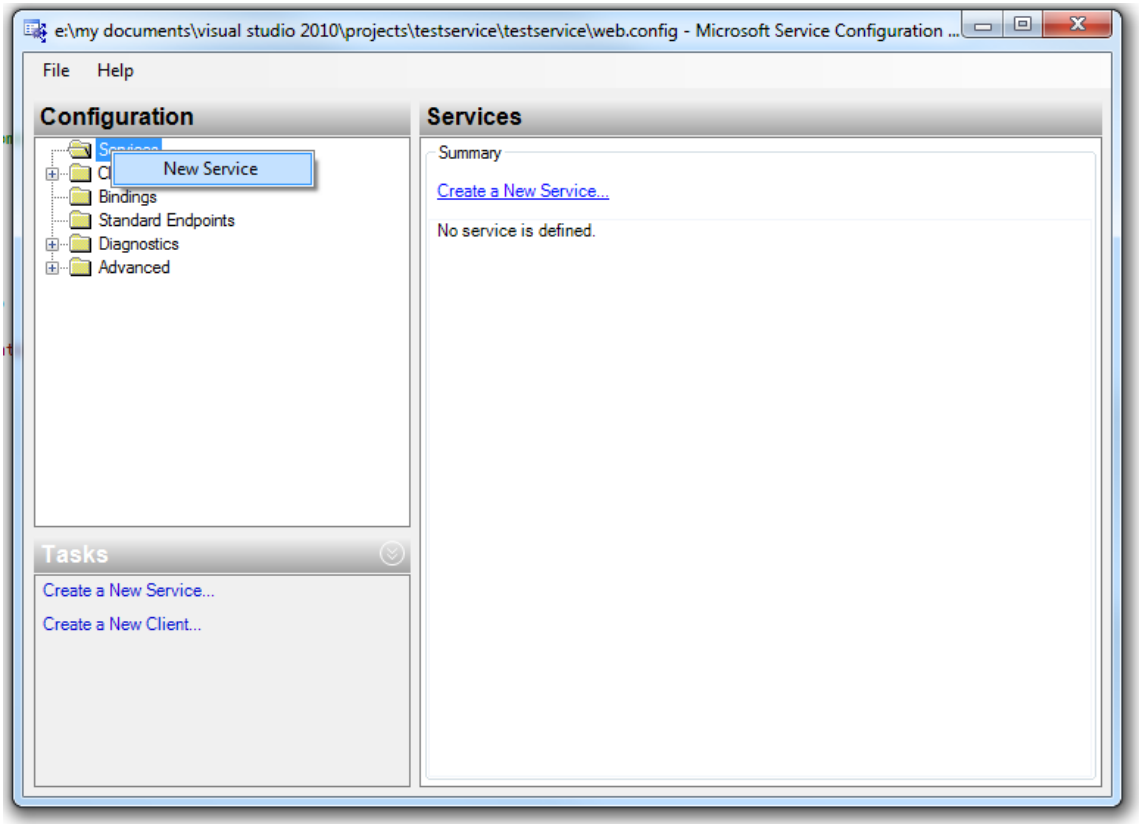
Efter detta skapas en klass som realiserar gränssnittet. Denna klass kan hanteras som vilken vanlig klass som helst, men öppnar upp möjligheter för konfigurering av tjänsten. Ett exempel på en realisering av tidigare gränssnitt ses i Figur 14.

```
public class Service1 : IService1
{
    public string GetData(int value)
    {
        return string.Format("You entered: {0}", value);
    }
}
```

Figur 14, Realisering av enkelt WCF-gränssnitt

I detta skede är den bakomliggande logiken och kodningen klar. För att exponera tjänsten måste nu en eller flera EndPoints definieras.

Först kompileras projektet så att dll-filer skapas. Efter detta kan man gå vidare på två olika sätt. Antingen skriver man en server-konfiguration för hand, eller så använder man ett inbyggt verktyg i Visual Studio 2010 kallat WCF Service Configuration Editor. Fram tills nu har det inte definierat någonting alls gällande meddelandeformat eller hur kommunikationen ska gå till, den kod som skrivits är helt neutral.



Figur 15, WCF Service Configuration Editor.

I Figur 15 ses hur WCF Service Configuration Editor ser ut när man öppnar en helt okonfigurerad tjänst. För att skapa tjänsten väljs New Service, och den dll-fil som skapades vid kompilering markeras sedan. Efter detta väljs den klass som realiserar det gränssnitt som ska exponeras, alltså Service1. Här kan man sedan lägga till EndPoints till gränssnittet där man definierar bland annat meddelandetyp, transportprotokoll, loggning, autentiseringstyp o.s.v. Vill man skapa flera versioner av EndPoints för gränssnittet så lägger man till flera stycken med olika konfigurationer.

2.3.2 Datakontrakt

För att överföra egna datatyper måste man skapa ett datakontrakt (eng. data contract). Ett datakontrakt definierar vilka variabler och datatyper i en klass som skall serialiseras vid överföring mellan tjänst och klient. Ett exempel på ett datakontrakt ses i Figur 16.

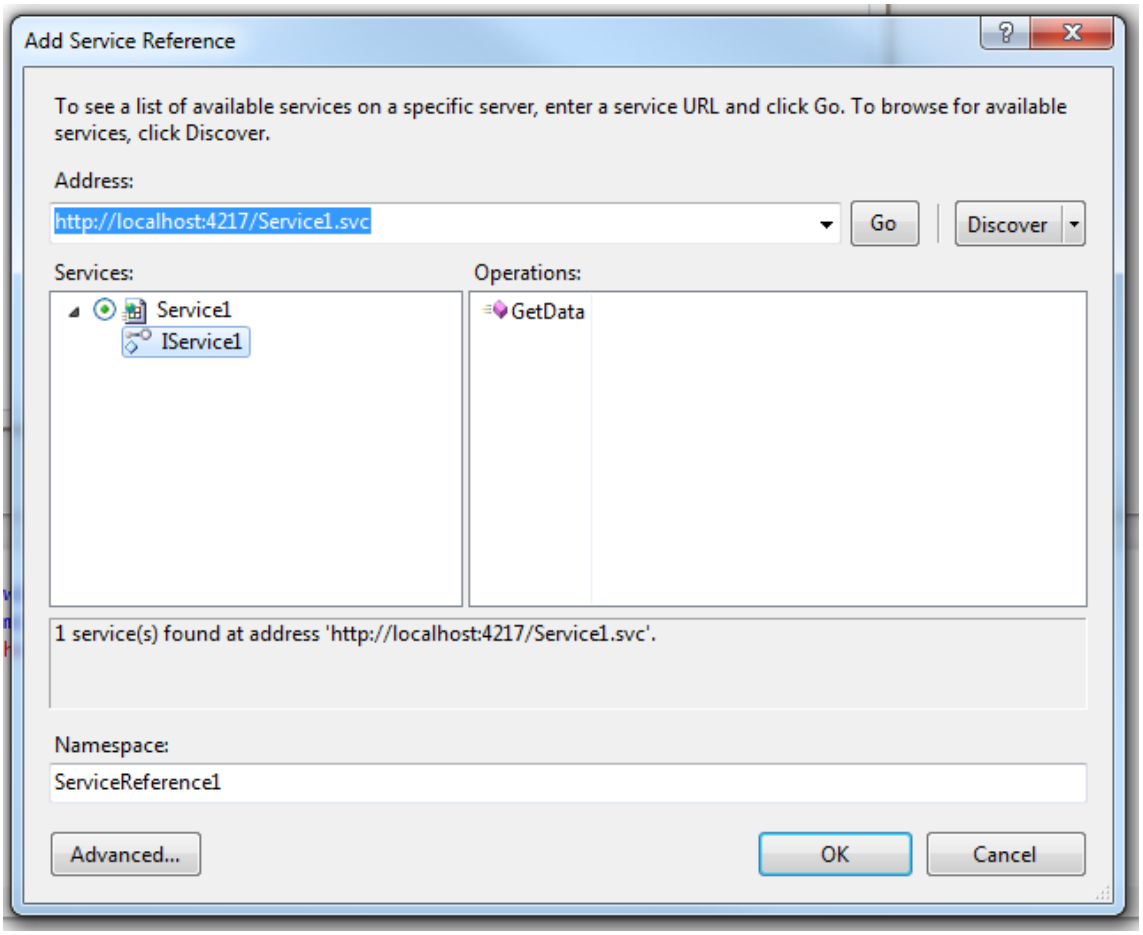
```
[DataContract]
public class CompositeType
{


    [DataMember]
    public bool BoolValue
    {
        get;
        set;
    }

    [DataMember]
    public string StringValue
    {
        get;
        set;
    }
    public bool LocalBool { get; set; }

    public void DoSomething()
    {
        //Gör någonting
    }
}
```

Figur 16, Exempel på datakontrakt

För att skapa ett datakontrakt läggs [DataContract] till ovanför klassdefinitionen. Detta markerar klassen som serialiserbar av DataContractSerialiser. För att markera vilka variabler man ska skicka vid serialiseirng lägger man till [DataMember] ovanför variabeln. Ett datakontrakt kan innehålla andra typer av datakontrakt. I det här fallet skulle BoolValue och StringValue skickas. LocalBool skickas inte vid överföring, utan används endast på servern. Om server och klient delar klassbibliotek är det möjligt att använda funktioner inbyggda i ett datakontrakt, och variabler som inte är märkta som [DataMember] kan då modifieras separat i klienten och i servern. Man bör notera att när man skickar över någonting som ett datakontrakt så skickar man det som ett värde, inte som en referens till det ursprungliga objektet.

2.3.3 Skapa en WCF-klient

När man väl har konfigurerat sin tjänst så är det lätt att ansluta andra .NET-klienter till tjänsten. Man skapar lättast en anslutning till en tjänst med Visual Studio. Välj ”Add Service Reference” från projektmenyn, och anger adressen till den tjänst man vill ansluta till.



Figur 17, Hur man lägger till en Service Reference i Visual Studio 2010

I Figur 17 ses hur den tjänst som tidigare skapats kan hittas. Efter att tjänsten markerats för tilläggning trycker man på ok, och Visual Studio kommer sedan att generera all den kod som behövs för att kommunicera med tjänsten. I Figur 18 visas hur en anslutning skapas och hur den metod som tidigare definierats anropas.

```
ServiceReference1.Service1Client client =
        new ServiceReference1.Service1Client();
client.Open();
String data = client.GetData(14);
Console.WriteLine(data);
```

Figur 18, Exempel på hur en WCF-klient ansluter och använder en WCF-tjänst

Resultatet av detta blir att programmet skriver ut ” You entered: 14” i konsolen.

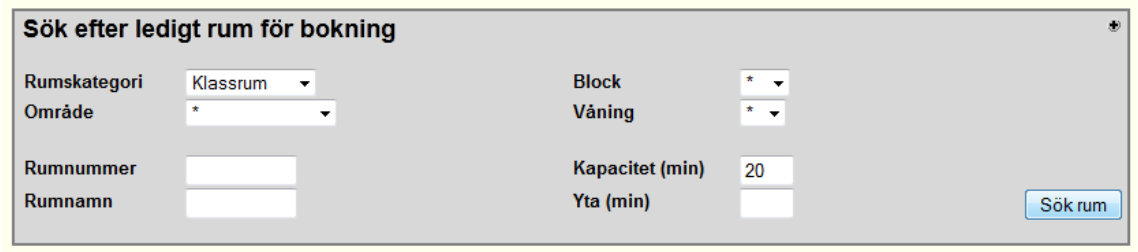
2.4 ASP .NET MVC3

ASP .NET MVC står för Active Server Pages .NET Model View Controler, och kommer hädan efter att hänvisas till som endast MVC. MVC är ett ramverk byggt på .NET-ramverket och är gjort för utveckling av webbsidor som använder sig av Modell.-Vy-Kontroller (eng Model-View-Controller, MVC) designmönstret. Modell-Vy-Kontroller bygger på en stark separation av modell - data, kontroller- logik, och vy – hur man visar data. Det finns också stöd för att på ett snabbt och enkelt sätt filtrera inkommande data, lägga till validering, autentisering och andra filter.

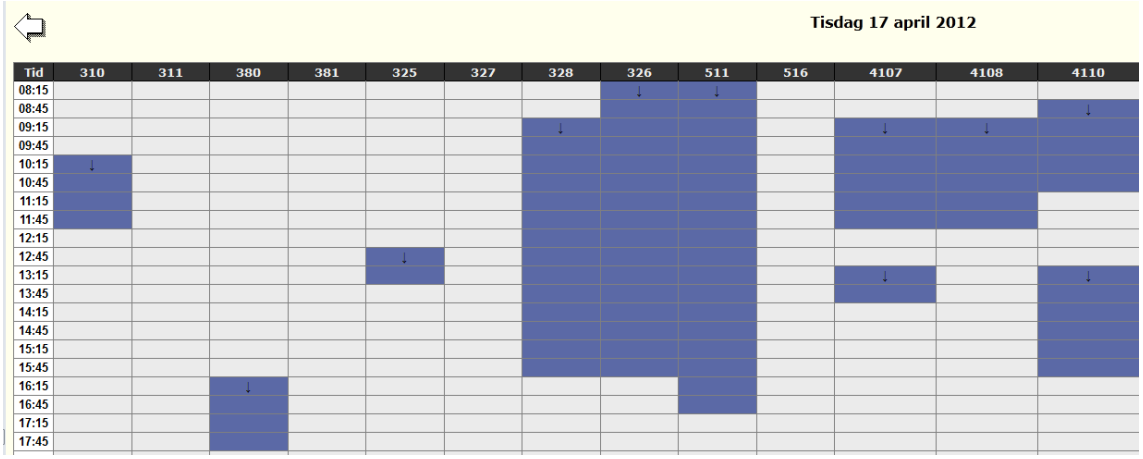
3 EFFEKTIVISERING AV BOKNINGSPROCESSEN

För att kunna effektivisera bokningsprocessen har en analys av tidigare använda bokningssystem gjorts. Nedan beskrivs den process som använts hittills.

ARBS, Arcada RumsBokningsSystem, är det bokningssystem som har används inom Arcada för att göra bokningar. Det är genom ARBS möjligt att boka föreläsare, rum, studentgrupper och kurser. ARBS är huvudsakligen ett rumsbokningssystem, det vill säga den primära fokusen ligger på att hitta ett utrymme. Hur man hittar ett rum ses i Figur 19 och Figur 20.



Figur 19, Sökning av rum i ARBS



Figur 20, Resultat av sökningen av rum I ARBS

De blå fälten visar när rummen är bokade, och de grå fälten visar när rummen är lediga. För att göra en bokning klickar man på ett grått fält, och en bokningsdialog visas då. Denna dialog har 7 steg som man ska gå igenom.

Steg 1: Här finns möjlighet att ange en kurs samt en rubrik. För att kunna koppla bokningen till en kurs måste man känna till kurskoden.

Steg 2: I steg två anger man bokningstid (start och slut), och om man vill upprepa bokningen. Upprepningar kan ske enligt vissa mönster (varje dag, varje månad, varje vecka, varannan vecka), och upprepas fram tills ett visst datum som man anger.

Steg 3: Här plockar man ut alla studentgrupper som skall bokas.

Steg 4: Här plockar man ut alla föreläsare som skall bokas.

Steg 5: Här kan man ange en ansvarig person för bokningen, om man själv inte vill vara ansvarig.

Steg 6: Här är det möjligt att lägga till ytterligare information i bokningen, samt välja vad det är för typ av bokning som man gör, och om man vill att den ska synas för allmänheten på de infotavlor som finns på Arcada.

Steg 7: Här bekräftar man alla inställningar. Efter bekräftelse görs bokningen.

Efter att dessa steg har gåtts igenom så har en bokning gjorts. Observera att det i nuläget inte finns någon kunskap om varken studentgruppernas, kursens eller föreläsarnas kalendrar – för att se hur dessa ser ut måste det göras en separat sökning på varje deltagare, och sedan manuellt jämföra kalendrarna.

Om man vill göra en bokning med samma inställningar som tidigare finns det en funktion som tillåter att man återhämtar inställningarna som man använde vid den senaste bokningen. Vill man göra ändringar får man gå igenom Steg 1 – 7 igen.

För att effektivera den här processen har tre nya koncept introducerats, bokningsmallar, automatiskt val av rum och kombinerade kalendervyer. Mer om dessa kan läsas nedan.

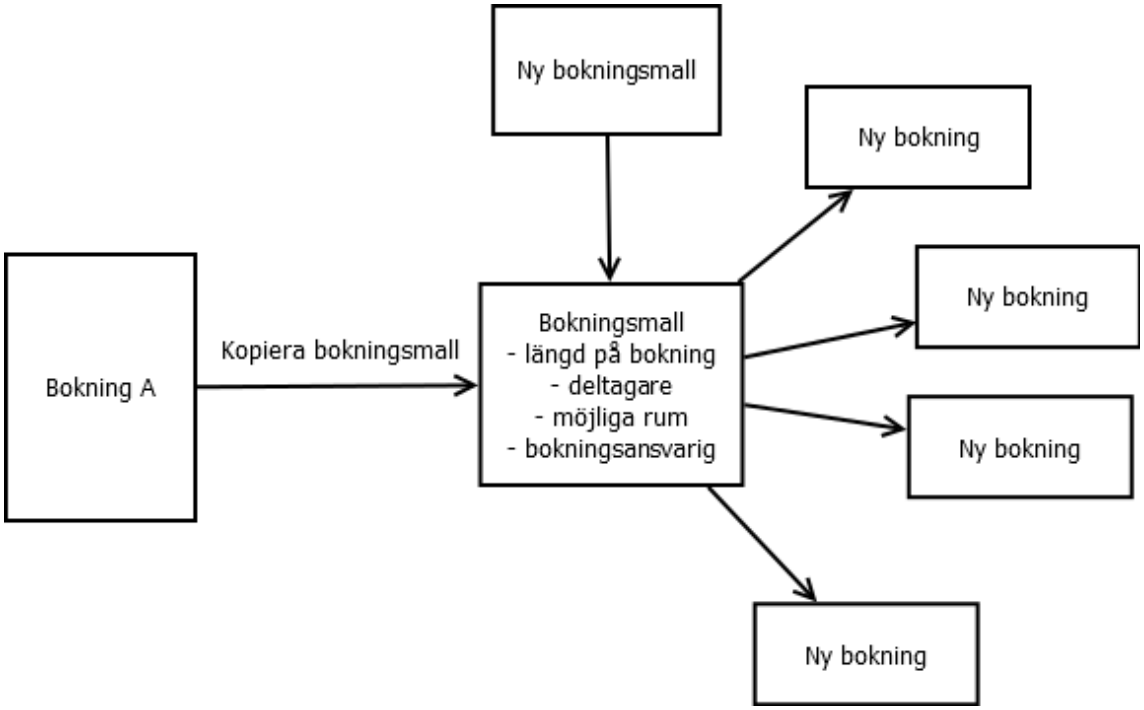
3.1 Bokningsmall

Ett nytt koncept i Puzzel är bokningsmallar. En bokningsmall är precis vad det låter som, en mall som används för att göra en bokning. Bokningsmallen innehåller alla inställningar som man behöver för att göra en ny bokning – deltagare, rum, längd på bokningen samt vem som är bokningsansvarig.

Bokningsmallen används sedan när man gör en bokning. När man väl har en bokningsmall kan man generera oändligt antal bokningar med mallens inställningar med bara ett par musklick. För en kort genomgång av hur bokningsprocessen går till, se Bilaga 1 – Snabbstartsguide till kursbokningsverktyget Puzzel.

En bokningsmall kan skapas helt från grunden, eller så kan en bokningsmall skapas utifrån en gammal bokning. Genom att välja att kopiera en bokningsmall från en tidigare gjord bokning får man alla inställningar som använts för att göra bokningen in i den aktiva bokningsmallen, och man är då direkt redo att göra en bokning med de inställningarna. Man behöver alltså inte göra alla inställningar på nytt när man väl gjort en bokning

med de inställningar man vill ha. Se Figur 21 för ett förtydligande på hur bokningsmallen fungerar.



Figur 21, Bokningsmallens funktion

3.2 Val av rum

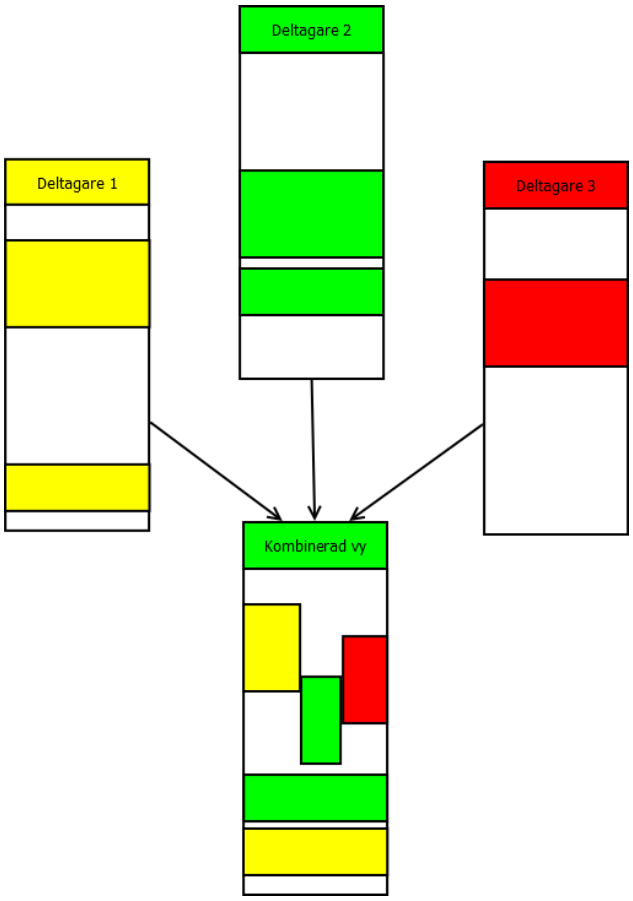
För att bokningsmallen ska vara användbar så krävs det att den är flexibel när det kommer till att boka rum. Har man endast ett rum i bokningsmallen innebär det att man måste göra om mallen om det rum man valt inte är ledigt vid den tidpunkt man vill boka.

Puzzel löser det här problemet genom att använda sig av en rumslista, som finns i bokningsmallen. En rumslista innehåller en lista på rum som man kan tänka sig vara i. När man gör en bokning kommer Puzzel att gå igenom listan, och så länge det finns ett rum som är ledigt i listan så kommer systemet automatiskt att boka ett ledigt rum.

3.3 Kombinerad kalendervy

Puzzel började utvecklas för att underlätta processen med att göra kursbokningar. Ett viktigt steg för att underlätta processen var att man skulle kunna hitta gemensamma luckor i alla deltagares kalendrar på ett snabbt och överskådligt sätt.

Puzzel har löst detta genom att kombinera kalendervyerna för alla de deltagare som man gör en bokning för. Kalendervyn är således starkt bunden till bokningsmallen – de deltagare som man har i bokningsmallen kommer få sin kalender visad i kalendervyn. Ett undantag till detta är de rum som man angett att man kan tänka sig vara i. Ett rums kalender visas vanligtvis inte i kalendervyn, då Puzzel försöker att flytta fokusen bort från rumsbokningar, men det finns möjlighet att visa kalendrar för rum i kalendervyn vid behov. Hur kalendervyn fungerar ses i Figur 22.



Figur 22, Exempel på kombinerad kalendervy

4 ALLMÄNT OM PUZZEL

Puzzel är uppbyggt av tre huvudkomponenter. Dessa är en serverdel, en klientdel samt en ensamstående klient som används för inkommande kommunikation från Exchange.

4.1 Kommunikation

Puzzel är utvecklad som en slavapplikation till ett flertal system. Autentisering och auktorisering av användare sker genom Active Directory. För att inhämta all data som behövs kombineras data från huvudsakligen två källor, ASTA och ABEX.

4.1.1 ASTA

ASTA är det system som är ansvarigt för hantering av kurser, läroplaner, vitsord och kursanmälningar. ASTA erbjuder ett flertal webbtjänster, de huvudsakliga tjänster som Puzzel använder sig av är utläsning av kurser samt läsårsdata. ASTA har ingen direkt kommunikationskanal till Puzzel. Kommunikationen med ASTA är alltså endast en envägsfunktion. För att hålla den data som kommer från ASTA uppdaterad sker en förfrågning var tionde minut, då all data synkroniseras på nytt.

4.1.2 ABEX

ABEX är det system som integrerar Exchange med Arcadas gamla bokningssystem ARBS. ABEX har två huvudsakliga funktioner. En är att övervaka valda gruppers Outlook-kalendrar och skicka ut uppdateringar till klienter som har upprättat en prenumeration på gruppens kalender när en bokning skapas, ändras eller raderas. Den andra är att möjliggöra att klienter kan modifiera Outlook-kalendrar genom att skapa och modifiera bokningar och grupper. Under utvecklingen av Puzzel vidareutvecklades ABEX-protokollet till att inkludera stöd för sökning av bokningar.

I ABEX har alla rum, kurser och studentgrupper skapats som grupper i Exchange. Genom att lägga upp prenumerationer på alla dessa grupper får man uppdateringar för alla bokningar som är relevanta för schemaläggning av kurser.

Kommunikation med ABEX kan ske både synkront och asynkront. För att möjliggöra asynkron kommunikation behövde kommunikationsprotokollet som använts för ABEX återspeglas. Detta har gjorts genom att använda samma teknik som ABEX är byggd på, ASP .NET MVC 3, och denna klient omnämns härefter som Exchange-klienten.

Kommunikationen med ABEX bygger på den integrering som redan gjorts mellan Exchange och ARBS. Detta kan man läsa mer om i (Vanhaniemi, 2010).

4.1.3 Kommunikationsflöde

Puzzel Server stödjer två kommunikationsprotokoll, https och tcps. Tack vare att http(s)-protokollet inte stödjer duplex-kommunikation sker all kommunikation mellan servern och klienterna så att klienterna hela tiden gör aktiva förfrågningar efter ny data. Klienterna är endast medvetna om Puzzel Server och har inte kontakt till några andra tjänster.

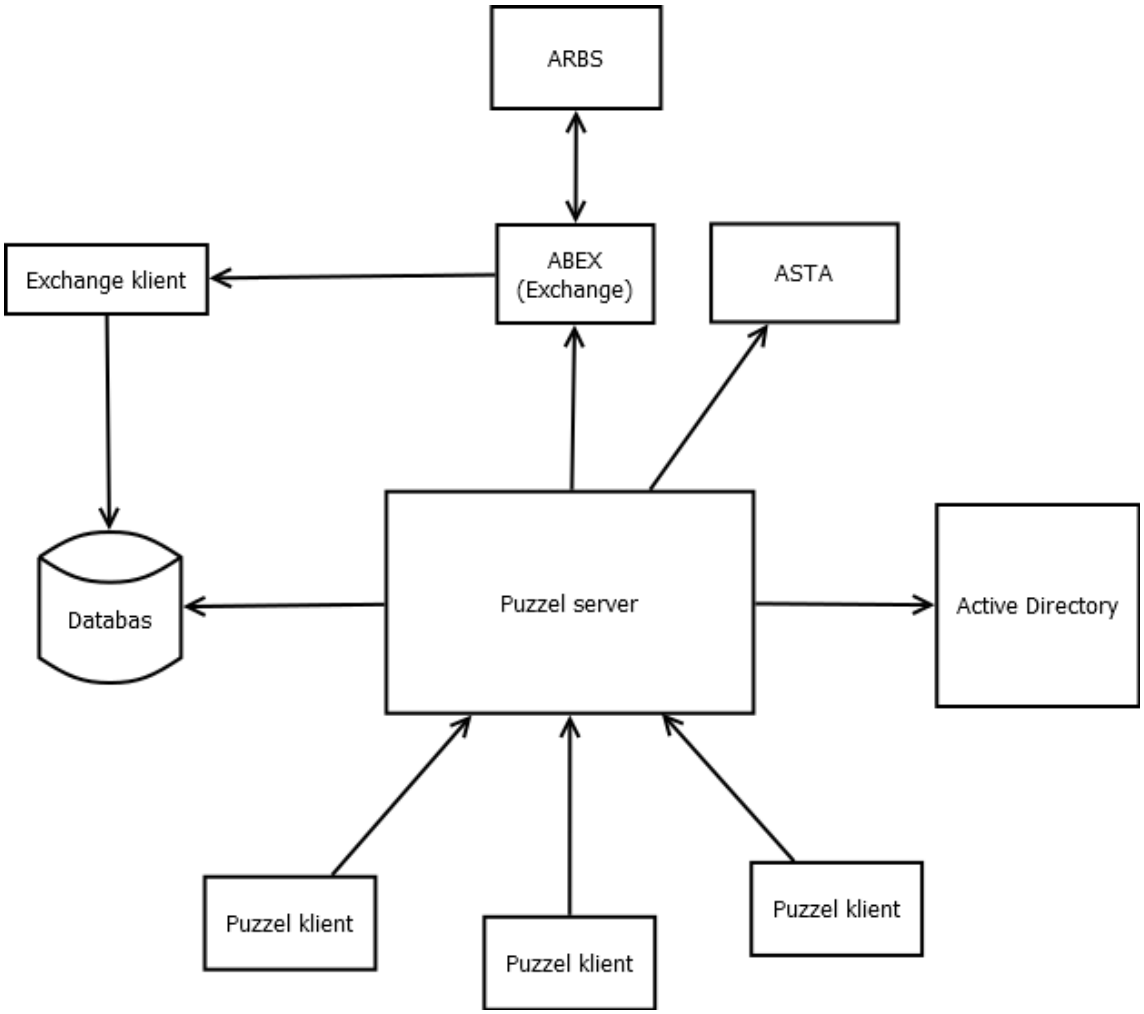
När en klient gör en bokning så kommunicerar den aktivt med Puzzel server. Allting som man gör i klienten sparas på servern. Servern i sin tur håller reda på alla ändringar som har gjorts på bokningar, och ser till att alla anslutna klienter blir meddelade om dessa ändringar vid deras nästa anrop.

För att en bokning ska skickas ut till ABEX så måste en klient välja att publicera en bokning. Puzzel server kommer då att markera bokningen med en status som innebär att den är färdig för att skickas till ABEX. Var tionde sekund kommer sedan servern att samla ihop alla bokningar som är markerade för publicering, och skicka dessa till ABEX. Tillsammans med all bokningsdata skickar man med en adress som ABEX skall rapportera tillbaka till när bokningarna blivit sparade. Denna adress är till Exchange-klienten.

När bokningarna väl blivit sparade skickas ett meddelande med varje boknings unika ID tillbaka till en angiven adress till Exchange-klienten. I adressen finns inbakad den ID som bokningen har i Puzzels databas. Denna data skrivs rakt in i en tabell i databasen

och hanteras inte alls i Exchange-klienten. Puzzel Server läser sedan denna tabell var tionde sekund och går igenom all inkommen data. Tack vare att meddelandet om en gjord bokning innehåller både Puzzels interna ID för bokningen samt ABEXs ID så kopplas de två i detta skede ihop.

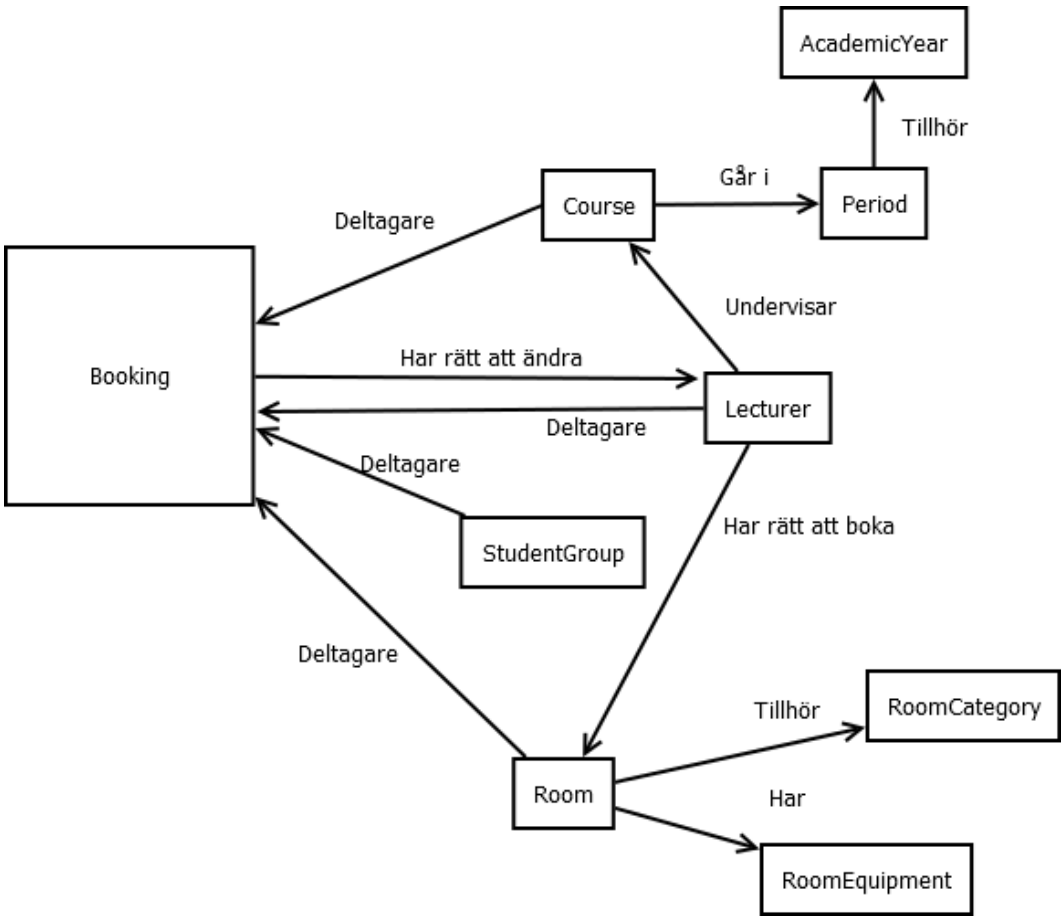
Om en bokning som skickas till ABEX innehåller grupper som man har upprättat prenumeration på så kommer det att komma ett komplett bokningsobjekt till Exchange-klienten efter några sekunder. Dessa meddelanden läses också var tionde sekund, och tack vare att man alltid kopplat bokningar gjorda i Puzzel till den ID som de fått i ABEX så kommer inte dubletter av bokningar att uppstå. Får man in en bokning vars ABEX ID finns sparad i databasen så uppdateras den bokningen. Hittas inte någon bokning med angiven ID så antas det vara en ny bokning. I och med att inläsningarna från ABEX alltid sker i en och samma tråd så undviker man kollisioner mellan trådar. Se Figur 23 för en överblick av kommunikationsflödet.



Figur 23, Kommunikationsflöde i Puzzel

4.2 Datamodell

Puzzel Server och Puzzel-klienterna använder samma datamodellbibliotek. I biblioteket finns bland annat klasserna AcademicYear, Booking, Course, Lecturer, Period, Room, RoomCategory, RoomEquipment och StudentGroup. Utöver dessa definieras klasser specifika för klienten i klienten och klasser specifika för servern i servern. Alla ovannämnda klasser har med bokningsprocessen att göra. Se hur dessa klasser är sammankopplade i Figur 24.



Figur 24, Datamodell som används av både server och klient.

Då ett bokningsobjekt innehåller många underliggande objekt är det opraktiskt att skicka över ett komplett bokningsobjekt då datamängden som överförs skulle bli alltför stor.

Varje typ av objekt kan identifieras med hjälp av ett unikt ID. Detta gör det möjligt att skicka över listor av ID-strängar som sedan kopplas mot objekt. Server och klient håller en lista på alla mer eller mindre statiska objekt i minnet under körning. Listor på objekt

av typerna Course, Period, Lecturer, StudentGroup, Room, RoomCategory och RoomEquipment läses in vid uppstart på både server- och klientsidan. I Booking-klassen finns det sedan statiska referenser till dessa listor. I och med att statiska variabler inte överförs mellan server och klient och är gemensamma för alla objekt av samma typ så kan både server och klient använda samma metoder för koppling av ID-strängar till objekt. Ett exempel på hur koppling av kursobjekt sker visas i Figur 25.

```
public class Booking
  {
      public static List<Course> AllCourses;
      . . .
     [DataMember]
          public List<String> _courses;
          private List<Course> course;
          public List<Course> Courses
          {
              get
              {
                  if (course == null || course.Count()!=_courses.Count())
                  {
                   course= AllCourses.Where
                          (c => _courses.Contains(c.AbexName)).ToList();
                  }
                  return course;
              }
              set
              {
                  _courses.Clear();
                  foreach (Course c in value)
                  {
                      _courses.Add(c.AbexName);
                  }
                  course = value;
              }
          }
   . . .
  }
```

Figur 25, Exempel på koppling från ID-sträng till objekt

Tack vare den här kopplingsmetodiken är det möjligt att hålla referenser till de objekt man vill komma åt både på klient- och serversidan. Dock innebär detta att man måste tänka på kopplingsstrukturen då man lägger till eller tar bort objekt från listorna. Att anropa booking.Courses.Add(obj) skulle t.ex. inte uppdatera listan på ID-strängar, men anropar man därefter booking.Courses=booking.Courses uppdateras listan med ID-strängar. Lägger man till en ID-sträng i _courses så kommer listan på objekt att uppdateras, då längden på listan av objekt och längden på listan av ID-strängar inte längre överensstämmer.

I klassen Room så finns det för varje objekt dels en lista på ID-strängar för föreläsare som har lov att boka rummet, en start- och en sluttid för när man får boka rummet, en flagga vid namn Hidden samt en flagga BookableByStaff. Hidden-flaggan är sann om rummet skall gömmas undan för bokningar, och BookableByStaff är sann om alla i personalkåren har rätt att boka rummet. För att ta reda på om en föreläsare, eller en lista på föreläsare, har rätt att boka ett rum finns det tre olika metoder definierade i Room-klassen, se Figur 26.

```
public bool AllowedToBook(Lecturer l, DateTime time)
{
    if (l != null)
    {
        if ((BookableByStaff || BookableBy.Contains(l.AbexId)) && !Hidden
      && ((ValidTo != null && time <= ValidTo) || ValidTo == null) &&
      ((ValidFrom != null && time >= ValidFrom) || ValidFrom == null))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    return false;
}

public bool AllowedToBook(List<Lecturer> lecturerList)
{
    List<String> idList = lecturerList.Select(l => l.AbexId).ToList();
    if ((BookableByStaff || BookableBy.Intersect(idList).Count()>0) &&
      !Hidden)
    {
        return true;
    }
    else
    {
        return false;
    }
}

public bool AllowedToBook(List<Lecturer> lecturerList, DateTime time)
{
    List<String> idList = lecturerList.Select(l => l.AbexId).ToList();
    if ((BookableByStaff || BookableBy.Intersect(idList).Count() > 0) &&
!Hidden && ((ValidTo != null && time <= ValidTo) || ValidTo == null)
&& ((ValidFrom != null && time >= ValidFrom) || ValidFrom == null))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Figur 26, Metoder för att ta reda på om en föreläsare eller en lista på föreläsare har rätt att boka ett rum.

När AllowedToBook returnerar ett sant värde så innebär det att en föreläsare har rätt att boka rummet. Då denna metod är definierad i Room-klassen så kan man med hjälp av LINQ filtrera ut rum på ett enkelt sätt. Se Figur 27 för ett exempel.

```
List<Room> AllRooms;
List<Lecturer> Lecturers;
      .  .  .
List<Room> BookableRooms = AllRooms.Where(r =>
                           r.AllowedToBook(Lecturers)).ToList();
```

Figur 27, Filtrering av bokningsbara rum efter en lista på föreläsare med hjälp av LINQ.

4.3 Autentisering och säkerhet

Autentiseringen av användare sker genom att en användare identifieras med hjälp av Windows Client Credential Token. Denna pollett innehåller bland annat användarens användarnamn och domän. Användarens lösenord används för att skapa polletten. Är man inloggad på en Windows-maskin så används en pollett med de användaruppgifter som används för det konto man är inloggad på automatiskt när man försöker autentisera sig mot en WCF-tjänst som har Windows Authentication aktivt. Det är dock möjligt att fråga användaren efter användarnamn, lösenord och domän. Med dessa kan man sedan skapa en ny pollett som används för auktorisering och autentisering.

För att auktorisera användaren ställs sedan krav på användarkontot som blivit autentiserat. I Puzzel krävs det att man är medlem i ”Active Directory”-gruppen Staff, vilket definieras i servern vid de anrop man vill skydda. Se Figur 28.

```
[PrincipalPermission(SecurityAction.Demand, Role = "staff")]
public Guid GuidConnect()
{
  . . .
}
```

Figur 28, Hur man begränsar åtkomst enligt grupptillhörighet inom en domän. Observera att servern måste vara inställd på att använda Windows Authentication.

När man ansluter sig till Puzzel så kopplar servern varje anslutning till en föreläsare med hjälp av det användarnamn som används vid anropet. Om någon mot all förmodan inte skulle kunna kopplas till en föreläsare så kan inte anslutningen realiseras. Efter att anslutningen gjorts blir man tilldelad en GUID (Globaly Unique Identifier, globalt unikt

ID) som sedan skickas med vid varje anrop. Alla modifieringar som man gör inom Puzzel kommer sedan att kontrolleras.

Skickar man ett eget modifierat anrop till Puzzel där man försöker göra ändringar som man inte har rätt till så måste man först identifieras som någon som har de rättigheterna som behövs inom den domän som servern befinner sig i, d.v.s. man behöver känna till rätt användarnamn och lösenord. Försöker man skicka ett modifierat meddelande där man själv är inloggad så kommer meddelandet att avslås, eftersom servern hela tiden kontrollerar att den föreläsare som har kopplats till anslutningen har rätt att göra de ändringar som han/hon försöker göra. Eftersom all kommunikation sker genom https och tcps är kommunikationen skyddad mot avlyssning.

5 SERVERAPPLIKATIONEN

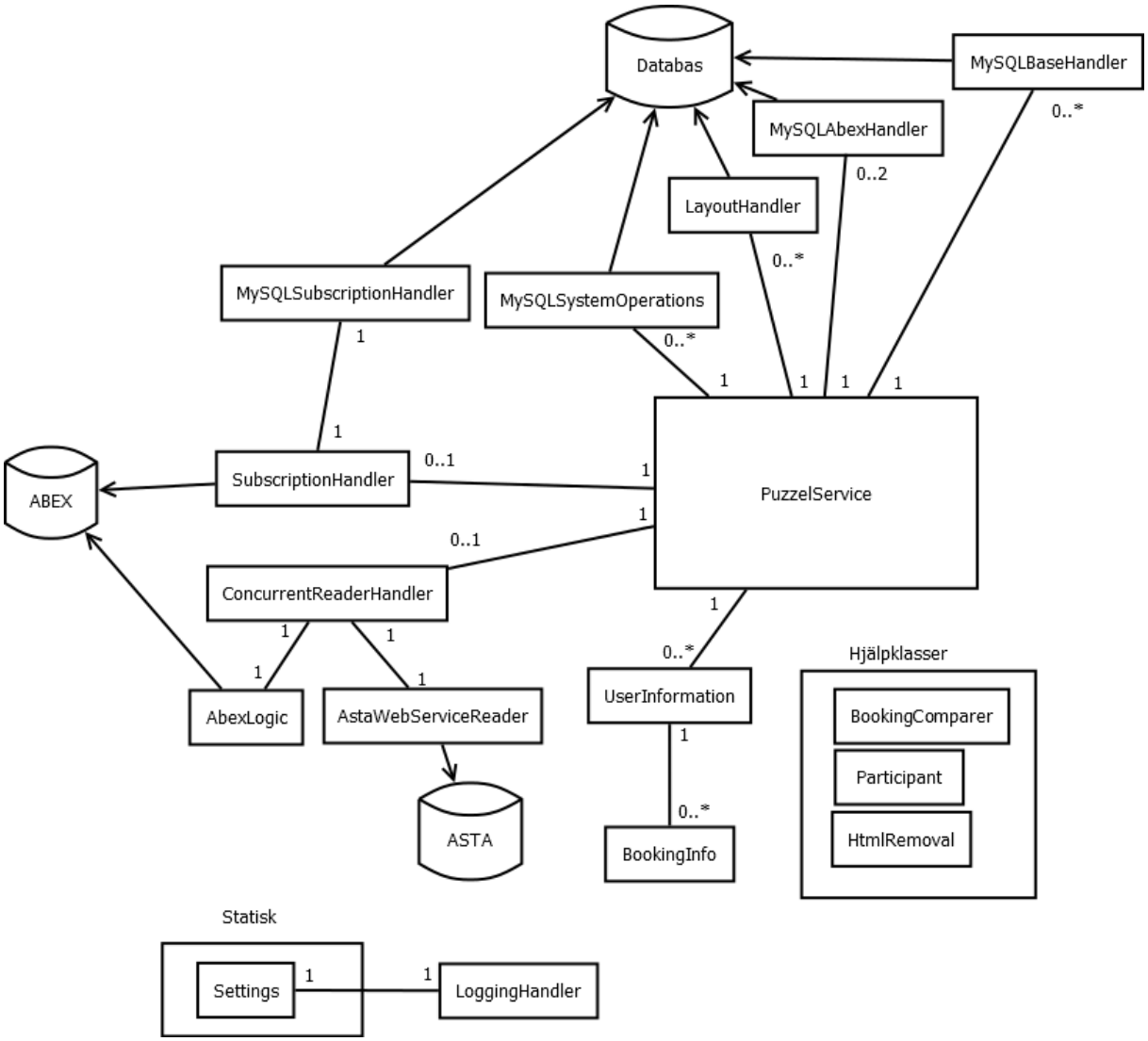
Serverapplikationen är Puzzels huvudkomponent. Servern håller reda på och läser in all data som är viktig för de operationer man genomför.

Tack vare att inläsning av data från ASTA och ABEX tar relativt lång tid (1-30 sekunder) så håller servern den informationen i minnet. För att hålla dessa data uppdaterade sker nya förfrågningar mot ASTA och ABEX var tionde minut.

Puzzel Server har ett enda objekt som tar hand om alla anrop, men tillåter att flera trådar kommer åt objektet på en och samma gång. Tack vara detta är tillgången synkroniserad till all gemensam data som flera användare kan försöka ändra på samtidigt.

Då man måste ta trådsäkerhet i beaktande så ökar komplexiteten med gemensamma datakällor en hel del. Detta är huvudanledningen till att bokningsdata som läses ut från databasen inte mellanlagras på servernivå. Då alla sökningar av bokningar som görs alltid går direkt mot databasen så kan man vara säker på att informationen som man får ut alltid är aktuell. Att läsa ut data från databasen (MySQL i detta fall) går så pass snabbt att den lilla ökningen i prestanda man kan få genom att mellanlagra bokningsdata inte bedöms vara värd den ökade komplexiteten i datalagret.

Puzzel Server körs som ett objekt som hanteras av ett flertal trådar. Kommunikationskanalerna till de datakällor som Puzzel använder sig av hålls inte öppna, utan skapas vid behov**.** En överblick av de objekt som skapas av Puzzel samt vilken källa de kommunicerar med visas i Figur 29.



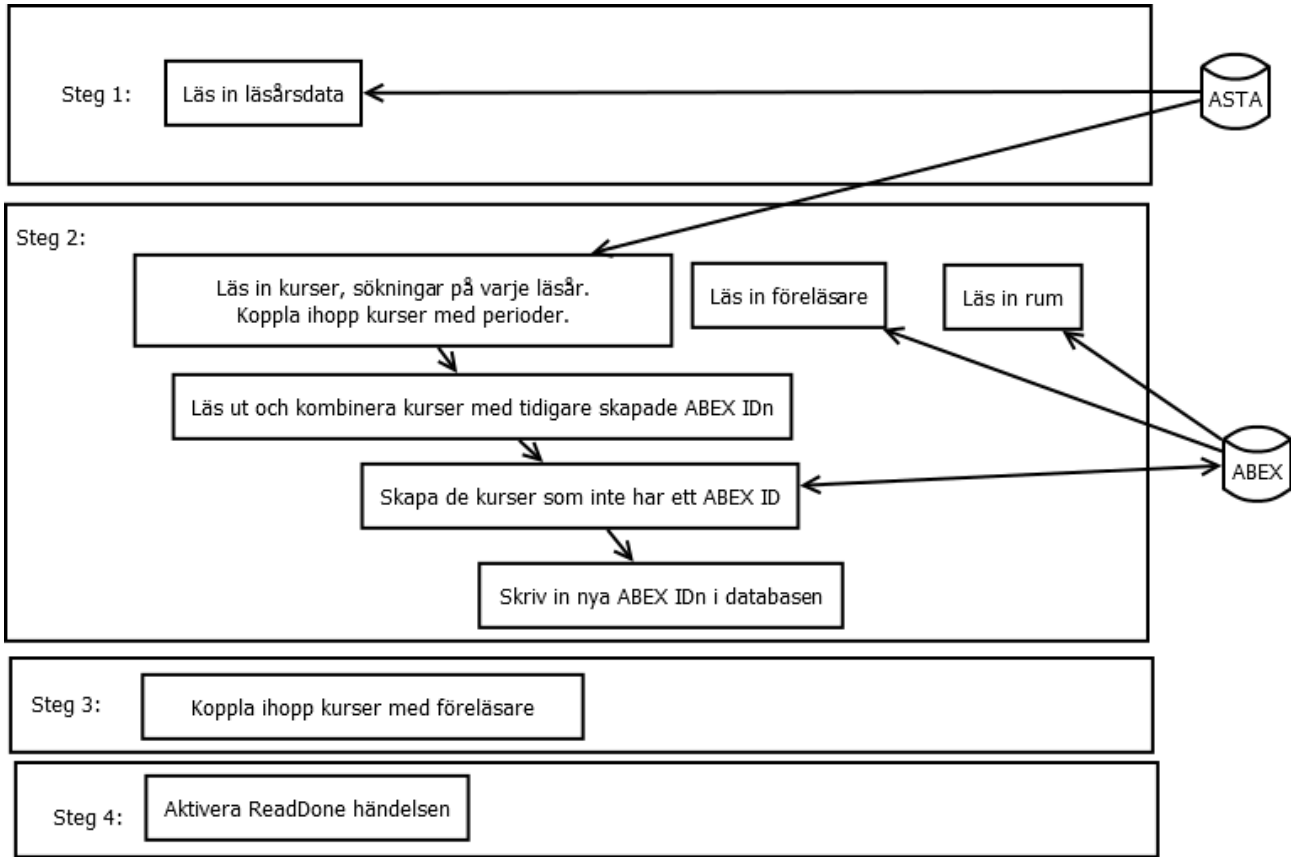
Figur 29, Överblick av objekt som skapas av Puzzel Server, samt vilka datakällor som de kommunicerar med. Siffrorna bredvid klassnamnen visar hur många objekt av klassen som kan finnas aktiva.

5.1 Extern data

5.1.1 Inläsning av grupper, kurser och rum

ConcurrentReaderHandler är en klass om samtidigt läser in data från ett flertal källor. Kurser, läsår och perioder läses in från ASTA. Rum, studiegrupper och föreläsare läses in från ABEX. Var tionde minut startas en tråd där ett objekt i klassen ConcurrentReaderHandler startas, och den skapar i sin tur objekt i klasserna AbexLogic, AstaWebServiceReader samt SubscriptionHandler.

För att Puzzel ska kunna bygga upp kompletta dataobjekt måste data från flera olika källor kombineras. Vissa sökningar behöver således ske i en viss ordning, och slutligen kombineras. Inläsningen sker i 4 olika steg, se Figur 30.



Figur 30, Överblick av inläsning av kurs-, läsårs-, föreläsar-och rumsdata

I det första steget inläses all läsårsdata från AstaWebServiceReader. Här fås information om vilka perioder som läsåret innehåller, vilka datum de börjar samt vilket ID som används för att identifiera läsåret.

I steg två skapas tre olika processer som alla är ansvariga för inläsning av en typ av data. Rumsdata och föreläsare är relativt enkla objekt och inläses direkt från ABEX genom med ett objekt i AbexLogic. Kursobjekten behöver speciell behandling då tjänsten som kursinformationen fås från inte är medveten om ABEX. Detta innebär att man inte vet hur kursen kommer identifieras i ABEX, eller om kursen finns skapad alls.

I ABEX-protokollet finns det en möjlighet för klienterna att skapa gruppobjekt. Ett gruppobjekt kan identifieras på två olika sätt, antingen med Name och Path, eller med ett ABEX ID. Name anger ett namn på ett objekt och Path en sökstig där objektet ska hittas. När man skapar ett nytt gruppobjekt anger man Name och Path, och får sedan tillbaka en ABEX ID som i fortsättningen kan användas för att identifiera kursobjektet. Försöker man skapa ett objekt som redan finns så returneras den ABEX ID som finns för objektet. Tack vare att klienterna som använder ABEX använder ett visst schema för Name och Path då man skapar nya objekt kommer aldrig ett objekt att skapas två gånger.

När kurser har lästs ut från AstaWebServiceReader försöker SubscriptionHandler koppla ihop varje kurs med en ABEX ID. Detta görs genom att läsa en tabell i databasen, där alla kurser som har skapats från Puzzel finns upptecknade med Name, Path och ABEX ID. Eftersom Name och Path räknas ut enligt ett visst schema som används av både ARBS och Puzzel-Server så är det möjligt att göra sökningen med hjälp av dessa två värden.

Om ingen ABEX ID hittas för en kurs så måste kursen skapas. Man vet inte i detta skede om kursen redan har skapats av en annan klient, eller om den inte finns alls. Eftersom det anrop som skickas till ABEX alltid ger ett ABEX ID tillbaka oavsett om kursen finns eller ej spelar detta ingen roll. Anropet för att skapa en kurs görs genom att skicka Name och Path till ABEX, och en ABEX ID sträng returneras. Dessa värden

skrivs sedan in i databasen så att de kan kopplas ihop med respektive kurs vid nästa inläsning.

I steg 3 kopplas föreläsare ihop med kurser. Den kursinformation som fås från ASTA visar kursens föreläsare med användarnamn. När informationen om föreläsare har fåtts från ABEX så kan användarnamnet kopplas mot ett komplett objekt.

I steg 4 är inläsningen klar, och händelsen ReadDone aktiveras. Händelsen tas sedan emot av PuzzelService.

5.1.2 Prenumerationer

När PuzzelService tagit emot en ReadDone-händelse så skickas listorna på inlästa rum, studentgrupper och kurser till SubscriptionHandler. SubscriptionHandler läser sedan från databasen vilka grupper som sedan tidigare har aktiva prenumerationer från ABEX. Listorna jämförs, och om nya grupper har hittats läggs först en prenumeration på gruppen upp, sedan görs en sökning på bokningar som har gruppen som deltagare. Sökningen söker två läsårsperioder bakåt i tiden, och två år framåt.

När en prenumeration har lagts upp och en sökning har gjorts skrivs den ABEX ID som identifierar gruppen in i tabellen över aktiva prenumerationer i databasen. Efter detta kommer inte sökningar eller nya prenumerationer att göras på denna grupp.

5.2 Användarhantering

När en användare ansluter sig till Puzzel Server så anroppas först en funktion vid namn GuidConnect. Om användaren har rätt att ansluta sig till servern så returneras en GUID. Funktionen GuidConnect kan ses i Figur 31.

```
[PrincipalPermission(SecurityAction.Demand, Role = "staff")]
public Guid GuidConnect()
{
   Guid id = Guid.NewGuid();
   bool added = false;
   while (!added)
   {
     if (UserData.Keys.Where(k => k == id).Count() != 0)
         id = Guid.NewGuid();
     else
     {
         lock (UserDataLock)
         {
           UserData.Add(id, new
UserInformation(Thread.CurrentPrincipal.Identity.Name));
           if (UserData[id].User == null)
           {
             UserData.Remove(id);
             throw new Sytem.ServiceModel.Security.
SecurityAccessDeniedException("Can't identify lecturer");
            }
         }
        added = true;
     }
   }
   return id;
}
```

Figur 31, Funktionen GuidConnect

När en användare anroppar GuidConnect kontrolleras först att användaren är i gruppen Staff. Sedan genereras en unik GUID för användaren, och ett objekt av typen UserInformation läggs till i Dictionary-listan UserData, mer om detta objekt senare. I konstruktorn av UserData skickas identitets-strängen på den anropande användaren. Slutligen kontrolleras om UserData kunde identifiera användaren utifrån den identitetssträng som användes då objektet skapades. Om användaren inte kunde identifieras nekas användaren tillgång och ett SecurityAccessDeniedException-objekt skickas ut. Om användaren kan identifieras returneras den GUID som användaren blev tilldelad.

Vid varje anrop som användaren gör och som har med bokningar att göra skickas sedan den tilldelade GUID:en med. GUID:en används för att komma åt UserInformation-objektet som skapades då användaren anslöt sig till Puzzel. Se Figur 32 för en överblick av klassen UserInformation.

```
public class UserInformation
{
    public DateTime LastUpdated;
    /// <summary>
    /// En lista på bokningar som skickats.
    /// </summary>
    private List<BookingInfo> _bookingSent;
    /// <summary>
    /// En lista på updaterade bokningar som ska skickas vid nästa anropp.
    /// </summary>
    private List<Booking> UpdatedBookings;
    /// <summary>
    /// Ett lås för UpdatedBookings listan.
    /// </summary>
    private object ListLock;
    /// <summary>
    /// Den aktiva användaren.
    /// </summary>
    public Lecturer User;
    /// <summary>
    /// Den aktiva bokningsmallen.
    /// </summary>
    private Booking activeTemplate;
    /// <summary>
    /// Information om skickade bokningar.
    /// </summary>
    public List<BookingInfo> BookingsSent [. . . ]
    public UserInformation(String user) [ . . . ]
    public void BookingRemoved(String id) [ . . . ]
    public void addUpdatedBooking(Booking booking) [ . . . ]
    public void AddStatusChangedBooking(Booking booking) [ . . . ]
    public void setActiveTemplate(Booking template) [ . . . ]
    public List<Booking> FilterList(List<Booking> unfiltered) [ . . . ]
    public void AddSentBooking(Booking booking) [ . . . ]
}
```

Figur 32, Definition av klassen UserInformation.

UserInformation används för att identifiera användaren, kontrollera vad användaren har rätt att göra, samt som en mellanlagringsplats och filtrering för bokningar som skickas ut till användaren.

5.2.1 Sökning av bokningar

Vid varje sökning av bokningar blir resultatet en lista av bokningar. När denna lista har lästs ut från databasen skickas den till metoden FilterList i klassen UserInformation. FilterList-funktionen kontrollerar sedan varje bokning i listan.

Först kontrolleras om en bokning med samma ID har skickats ut till användaren förut. Varje bokning som har skickats ut till användaren finns registrerad som ett BookingInfo-objekt i listan _bookingSent. Se Figur 33 för en definition av klassen BookingInfo.

```
[DataContract]
    public class BookingInfo
    {
        [DataMember]
        public String Id { get; set; }
        [DataMember]
        public DateTime Updated { get; set; }
        public BookingInfo(String id, DateTime updated) [ . . . ]
        public override bool Equals(object obj) [ . . . ]
        public override string ToString()[ . . . ]
    }
```

Figur 33, Klassen BookingInfo

BookingInfo innehåller minimal information om bokningar som har gått ut till klienten. Med hjälp av BookingInfo kan man avgöra om en bokning har fått några uppdateringar genom att jämföra fältet Updated mot bokningens senaste uppdateringstid. Om bokningen har en annan uppdateringstid än den tidpunkt som finns registrerad i BookingInfo har bokningen uppdaterats, och den läggs då till på en filtrerad lista som går ut till klienten. Om tidpunkterna är densamma innebär det att klienten redan har fått information om den här bokningen. Den filtreras då bort och behöver inte skickas ut på nytt. Alla bokningar som läggs till på den filtrerade listan läggs också till som BookingInfo-objekt på listan _bookingsSent.

Efter att alla bokningar har filtrerats går FilterList-funktionen igenom listan UpdatedBookings. UpdatedBookings innehåller alla nya eller ändrade bokningar som antingen har kommit in från ABEX eller som andra användare har gjort sedan det senaste anropet. Hur dessa bokningar filtreras och läggs till på den här listan står det mer om i sektionen ”Hantering av bokningar”. Dessa bokningar läggs till i den filtrerade listan, och UpdateBookings töms sedan. Efter detta skickas den slutgiltiga filtrerade listan ut till klienten.

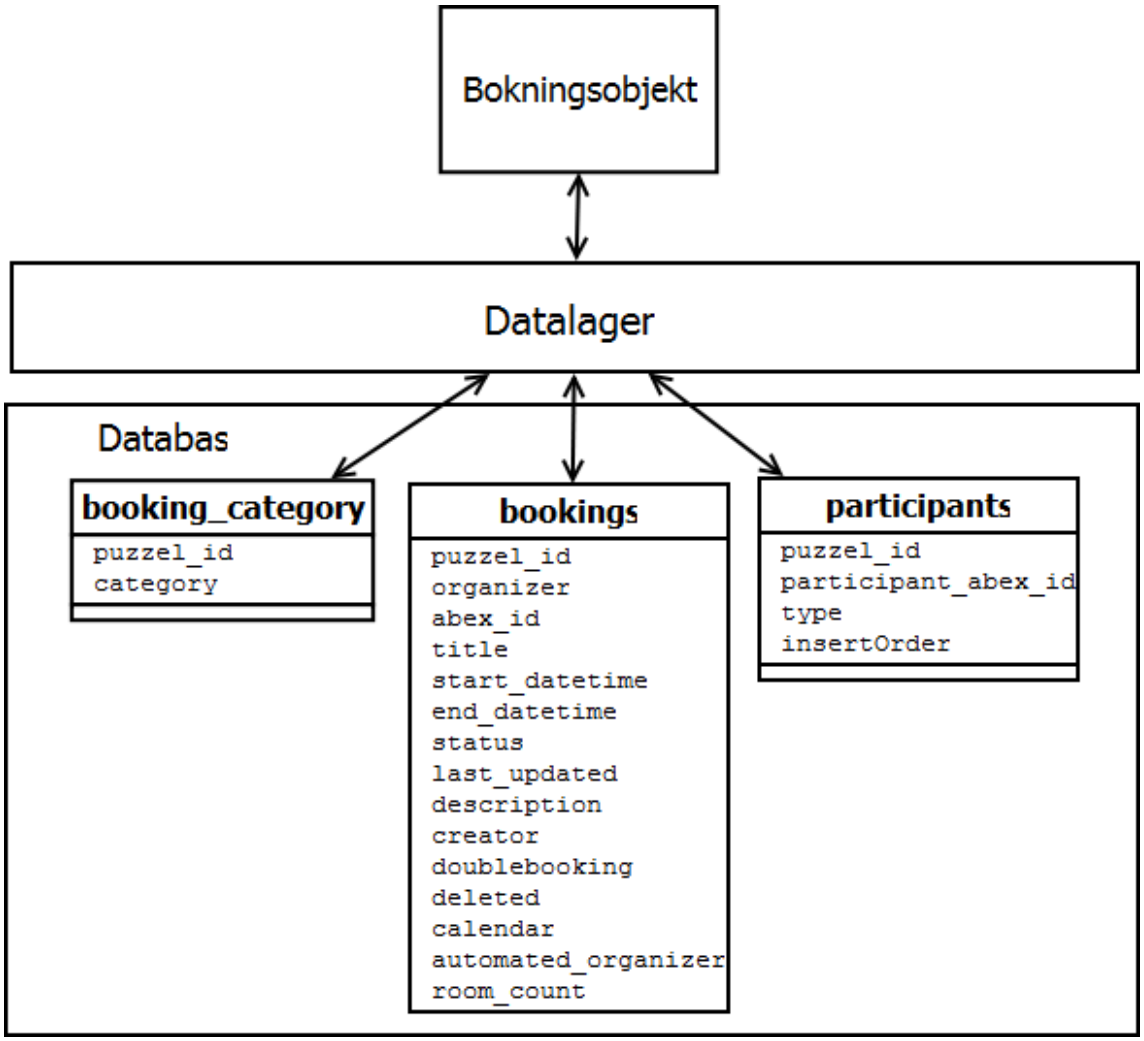
5.2.2 Modifiering av bokningar

När en användare vill ändra en bokning så inläses den gamla versionen av bokningen. Servern kontrollerar sedan vem som har rättigheter att ändra på den gamla versionen av

bokningen. Den användare som finns angiven i UserInformation-objektet som är kopplat till den GUID som skickats med vid anropet kontrolleras sedan mot dessa rättigheter. Om användaren har rätt att modifiera den gamla bokningen så går modifieringen av bokningen igenom, annars nekas den.

5.3 Hantering av bokningar

Alla bokningar som går till Puzzel Server skrivs in i en bakomliggande databas. Bokningen hålls inte i minne inom applikationen, utan återskapas från databasen vid behov. När ett objekt skrivs in i databasen utnyttjas samma förenklingsteknik som vid överföring mellan klient och server. Genom att hantera endast ID-strängar i deltagarlistor behöver inte kompletta bokningsobjekt sparas. De objekt som hålls som deltagare kan istället läsas ut och kopplas till bokningsobjektet vid behov. För en överblick av hur ett bokningsobjekt lagras i databasen, se Figur 34.



Figur 34, Överblick av lagring av bokningsobjekt i databasen.

På databasnivå delas ett bokningsobjekt upp i tre olika tabeller, booking_category, bookings och participants.

I tabellen bookings lagras all information som inte består av en lista och primärnyckeln puzzel_id insätts.

I tabellen booking_category kopplas sedan en lista av bokningskategorier som behövs för andra system inom Arcada till samma puzzel_id som insatts i bookings.

I tabellen participants inskrivs alla deltagare. En deltagare lagras med en puzzel_id, som kopplas till primärnyckeln puzzel_id i tabellen bookings. Här lagras också participant_abex_id , en unik ID sträng som gör det möjligt att koppla ihop bokningsobjektet

med en deltagare vid inläsning. För att veta vilken typ av deltagare som det har att göra med så lagras deltagartypen i variabeln type. Type innehåller ett av följande värden:

- Course – Deltagarobjektet är av typen Course. Vid inläsning av data kommer tillhörande participant_abex_id att lagras i listan _course i Booking-objektet.
- Lecturer – Deltagarobjektet är av typen Lecturer. Vid inläsning av data kommer tillhörande participant_abex_id att lagras i listan _lecturer i Booking-objektet.
- Selected_room är av typen Room. Vid inläsning av data kommer tillhörande participant_abex_id att lagras i listan _selectedRooms samt i listan _fittingRooms, båda i Booking-objektet.
- Suitable_room – Deltagarobjektet är av typen Room. Vid inläsning av data kommer tillhörande participant_abex_id att lagras i listan _fittingRooms.
- Other – Deltagarobjektet innehåller en extern deltagare. Det är möjligt att bjuda in vem som helst som har en epostadress genom Exchange, och dessa personer sparas endast med en epostadress som sitt ID.

Tabellen Participants innehåller också värdet insertOrder. InsertOrder är av typen int autoincrement, det vill säga för varje ny rad som skrivs in i tabellen så kommer insertOrder att öka med 1. Vid inläsning av ett bokningsobjekt sorteras deltagare efter insertOrder, vilket innebär att deltagare läses ut i samma ordning som de skrevs in. Tack vare detta är det möjligt att hålla sorterade rumslistor lagrade i databasen.

5.3.1 Nya eller uppdaterade bokningar

Vid kommunikation mellan Puzzel Server och Puzzel-klienter identifieras alla bokningsobjekt med hjälp av variabeln PuzzelID. När en klient gör en ny bokning skickas en bokning till servern utan PuzzelID, och servern skapar då en helt ny bokning. Finns det en PuzzelID med i anropet uppdateras bokningen. När man sparar nya bokningar eller uppdaterar gamla sker det alltid genom att klienten anroppar funktionen Save i PuzzelService-klassen.

I funktionen Save kontrolleras först att användaren har rätt att spara bokningen. Bokningar utan PuzzelID kräver inga speciella rättigheter, men för att uppdatera redan

skapade bokningar måste man antingen ha skapat dem själv, vara insatt som bokningsansvarig eller vara administrator.

Ett objekt i klassen MySqlBaseHandler skapas, och funktionen FindRoom anropas för att hitta ett ledigt rum till bokningen. Hur det går till beskrivs i sektionen ”Val av Rum”. När ett rum har hittats anropas funktionen SaveBooking i klassen MySqlBaseHandler. SaveBooking-funktionen tar reda på om man håller på att uppdatera en gammal bokning eller gör en ny. Om man gör en ny bokning skickas anropet vidare till funktionen SaveNewBooking. Om man uppdaterar en gammal bokning går det vidare till funktionen UpateBooking.

Då bokningar sparas utförs ett flertal kontroller:

- Om bokningen uppdateras inläses den gamla versionen av bokningen först. Det kontrolleras om bokningen har kollisioner genom att läsa fältet doublebooking. Om så är fallet utläses alla kolliderade bokningar ut och sparas i en lista med namnet oldCollisions. Om det är en ny bokning utförs inte detta steg.
- En sökning av bokningar görs med den uppdaterade bokningens tidpunkter och deltagare. Om några bokingar med någon gemensam deltagare inom samma tidsrymd hittas är bokningen en dubbelbokning. De bokningar som hittas vid sökningen sparas i en lista med namnet collisions.
- Bokningen sparas i databasen.
- Alla bokningar i oldCollision kontrolleras för kollisioner. Om de inte längre kolliderar med någon bokning uppdateras de i databasen och läggs till i listan updatedBookings. Listan updatedBookings innehåller bokningar vilkas enda ändring är dubbelbokning-statusen.
- Alla bokningar på listan collisions kontrolleras. Om de inte hade markerats som dubbelbokningar sedan tidigare uppdateras de i databasen och läggs sedan till i listan updatedBookings.

Efter detta är bokningen sparad i databasen och alla uppdateringar på andra bokningar som behöver göras har gjorts.

I Save-funktionen utläses det nyligen sparade bokningsobjektet, samt listan updatedBookings. Servern itererar sedan över varje objekt av typen UserInformation i Dictionary-listan UserData. På varje UserInformation-objekt anropas funktionen addUpdatedBooking, där det nya bokningsobjektet ges in. Sedan itererar servern över listan updatedBookings, och varje uppdaterat bokningsobjekt ges till varje UserInformation-objekt genom funktionen AddStatusChangedBooking.
I UserInformation-objekten filtreras sedan bokningarna enligt vad klienterna redan har utläst.

Bokningar som skickas in till funktionen AddStatusChangedBooking skickas endast ut till klienten om klienten redan känner till bokningen. På bokningarna som behandlas i denna funktion har det inte gjorts ändringar vilka kan göra att de dyker upp på sökningar som klienten redan gjort. De kan således säkert ignoreras om de inte befinner sig på listan _bookingSent.

Nya bokningar kommer till UserInformation-objektet genom addUpdatedBooking. I funktionen addUpdatedBooking kontrolleras först om bokningen redan har skickats ut. Om så är fallet skickas bokningen alltid ut på nytt. Om bokningen inte har skickats ut till klienten kontrolleras klientens senaste sökning, som finns sparad i variabeln activeTemplate. Om den nya bokningen och den senaste sökningen har någon gemensam deltagare utskickas bokningen till klienten. Om inte så är fallet ignoreras bokningen.

Efter att dessa steg har gåtts igenom är sparandet av en ny eller uppdaterad bokning färdigt.

Bokningar som kommer från ABEX hanteras på ett liknande sätt, men ett par extra steg behövs. Hantering av bokingar från ABEX utförs av ett objekt i klassen MySQLAbexHandler, som utökar klassen MySQLBaseHandler. Bokningar som kommer från ABEX har vanligtvis inte en PuzzelID, utan endast en ABEX ID. Steg ett vid sparande av en bokning från ABEX är därför att kontrollera om bokningen finns sparad sedan tidigare genom att göra en sökning på bokningsobjekt med inkommande ABEX ID. Hittas en bokning med angiven ABEX ID ges bokningen

träffens PuzzelID. Hittas ingen träff kontrolleras det om det finns några historiska ABEX ID lagrade i bokningen (en bokning kan nämligen byta ABEX ID), och sökningar görs då på dessa. Hittas bokningen kopieras det gamla bokningsobjektets lista av lämpliga rum till det nya bokningsobjektet och den nya bokningen får den gamla bokningens PuzzelID. Om inga träffar hittas efter sökningarna antas det vara en ny bokning.

En bokning från ABEX kan ha rum som deltagare. I ABEX har rum tre olika status:

- Accepted – rummet har bokats
- Tentative – rummet är ledigt, men en moderator behöver godkänna bokningen
- Declined – rummet kunde inte bokas

En bokning som kommer från ABEX och har ett rum med status Accepted har alltid förtur till rummet, och tränger undan andra bokningar som har bokat rummet. Bokningar som blivit undanträngda får sina val av rum omutvärderade. I Puzzel hanteras rum med status Tentative och Accepted på precis samma vis.

Om bokningen inte har ett accepterat rum men har gjorts från Puzzel söks ett nytt rum från listan med lämpliga rum. Om ett nytt rum hittas sparas bokningen med det nya rummet, och ett meddelande med det uppdaterade bokningsobjektet skickas direkt till ABEX.

5.3.2 Val av rum

När Puzzel väljer ett rum läses först listan FittingRooms in från bokningsobjektet. Efter detta kontrolleras om flaggan AutomatedOrganizer är sann eller osann. Om AutomatedOrganizer är sann innebär det att alla rum, som någon av de deltagande föreläsarna kan boka, kan bokas, och bokningsansvarig (Organizer i bokningsobjektet) kommer att sättas efter detta. Om AutomatedOrganizer är osann kan endast rum, som den utsatta bokningsansvarige har rätt att boka, bokas.

De rum som kan bokas lyfts ut till en ny lista. Puzzel läser sedan ut vilka rum som är bokade under den tid som bokningen behöver. Listorna jämförs och de rum som är bokade filtreras bort från rum som kan bokas.

Efter detta görs en kontroll på om bokningen har kommit in från ABEX med några rum med status Declined. Detta kan hända i sällsynta fall om t.ex. vem som har rätt att boka ett rum ändras, eller om en bokning har gjorts i Outlook eller ARBS och denna har behandlats i tiden mellan att bokningen gjordes i Puzzel och att ABEX tog emot den. Skulle Puzzel försöka välja samma rum två gånger i rad så finns det en chans att ett precis likadant bokningsobjekt skickas till ABEX två gånger. ABEX kommer då att tolka det som att bokningen redan finns och att inga uppdateringar har gjorts. Då ABEX inte behandlar bokningen fås inte heller uppdateringar om statusen på rummet. Därför lagras varje bokning som kommer in från ABEX med rum som har status Declined i en databastabell med starttid, sluttid, PuzzelID och rummets ID. De rum som finns registrerade i databasen med bokningens PuzzelID, start- och sluttid filtreras bort.

Slutligen väljs rummet på första positionen i den filtrerade listan och bokningen sparas med detta rum. Efter detta kontrolleras att endast en bokning har bokat rummet inom samma tidsperiod, eftersom det är möjligt att två trådar bokar samma rum precis samtidigt. Finns det fler än en bokning görs processen om från steg ett igen.

5.3.3 Borttagning av bokningar

Vid borttagning av bokningar sker liknande kontroller som vid uppdatering av bokningar. För att ta bort en bokning anropar klienterna funktionen Delete i klassen PuzzelService, som sedan skickar vidare anropet till funktonen Delete i klassen MySqlBaseHandler. Här görs en sökning efter kollisioner som sker med den bokning som ska tas bort. Efter detta markeras bokningen som borttagen i databasen och alla hittade kollisoner kontrolleras. Om en kolliderande bokning inte längre kolliderar med någonting uppdateras flaggan doublebooking och bokningen läggs till på listan updatedBookings. Den borttagna bokningen läggs till på listan deletedBookings.

Efter detta utläser Delete-funktionen i klassen PuzzelService listorna updatedBookings och deletedBookings. Funktionen itererar sedan över varje UserInformation-objekt som finns lagrat i Dictionary-listan UserData. På varje UserInformation-objekt ges de borttagna bokningarna till funktionen deleteBooking, där de sedan läggs till på listan Upda-

tedBookings med en flagga med namnet removed satt till sann. Då denna flagga är satt till sann tolkar klienten detta som att bokningen skall tas bort.

Alla bokningar som har fått sin doublebooking-flagga ändrad skickas sedan till varje UserInformation-objekt med funktionen AddStatusChangedBooking.

För bokningar som tas bort från ABEX sker samma kontroll, men bokningens PuzzelID kopplas först ihop med varje inkommande bokningsobjekt.

5.3.4 Sökning av bokningar

Puzzel stödjer många olika sökmetoder, men huvudsakligen används fem metoder eftersom de täcker de behov klienterna har. Mellan klient och server används dessa sökmetoder:

- **GetBookingAfterTemplate** (Guid id, Booking booking)
  Denna funktion används då en klient aktiverar en ny bokningsmall. Bokningen som skickas in som argument innehåller all data som man vill göra en sökning på. Bokningens starttid är början på den perioden man vill söka på, och sluttiden är slutet på perioden. Denna bokning sparas sedan i det UserInformation-objekt som är kopplat till klientens GUID. För sökning skickas sedan anropet vidare till FindBookingsByBookingAndTime.
- **FindBookingsByBookingAndTime**(Guid id, Booking booking, DateTime startTime, DateTime endTime)
  Denna funktion gör en sökning efter bokningar som ligger mellan start och sluttid och som har gemensamma deltagare med den bokning som skickas med som argument. Sökningen skickas visare till ett MySqlBaseHandler-objekt som sedan läser ut alla bokningar från databasen. Andra bokningars listor på lämpliga rum räknas i det här fallet inte som deltagare, men deras valda rum matchas mot alla rum som finns på bokningens lista över lämpliga rum. Efter att sökningen är gjord ges listan in till det UserInformation-objekt som hör till klienten, och listan filtreras. Alla bokningar som klienten inte läst in tidigare samt bokningar som uppdaterats läggs till på sökresultatet, och detta returneras sedan till klienten.

- **FindBookingById**(Guid userId, String id)
  Söker och läser in en bokning med ett visst PuzzelID från databasen med ett objekt i klassen MySqlBaseHandler. Bokningen markeras sedan som utläst i det objekt av klassen UserInformation, som kan kopplas till klientens GUID, och skickas sedan till klienten.
- **GetUpdates**(Guid id)
  Läser ut listan på uppdaterade bokningar från UserInformation-objektet som är kopplat till klientens GUID. Efter utläsning töms listan och resultatet returneras till klienten. GetUpdates-funktionen anropas var 5'e sekund från varje klient.
- **GetBookingsOutOfSync**(Guid id, Lecturer l)
  Läser ut bokningar som föreläsaren l gjort, vilka ännu inte blivit publicerade till ABEX. Sökningen sker genom funktionen getBookingsOutOfSync i klassen MySQLBaseHandler. Bokningarna ges sedan in till funktionen FilterList i det UserInformation-objekt som är kopplats till klientens GUID, och resultatet skickas till klienten.

5.4 Övriga klasser

5.4.1 MySQLSystemOperations

I klassen MySQLSystemOperations samlas ett flertal systemvariabler och operationer som används av systemet och behöver databastillgång. Här hanteras bland annat det ID som används då man sätter upp en ny prenumeration från ABEX, samt uppstädning av gamla opublicerade bokningar. Svenska och engelska benämningar på objekt i klasserna RoomEquipment och RoomCategory hämtas också härifrån.

5.4.2 LayoutHandler

LayoutHandler hanterar användares layouter. Puzzel-klienten har möjlighet att ange personliga layouter. LayoutHandler läser, uppdaterar och skapar dessa.

5.4.3 Settings

Settings är en statisk klass som inläser ett flertal konfigurerbara värden från serverkonfigurationsfilen (web.config) och vissa dynamiska värden från MySQLSystemOperations.

5.4.4 LoggingHandler

LoggingHandler hanterar loggning på server-nivå. Två olika typer av loggning sker. En loggning med notifiering via email används för att rapportera oväntade undantag. En loggning utan notifiering används för att övervaka vad som händer på servern. LoggingHandler kan logga i flera olika filer beroende på behov och är trådsäker.

6 KLIENTAPPLIKATIONEN

Klientapplikationen för Puzzel innehåller endast logik för att presentera data som kommer från Puzzel Server, samt logik för att generera nya bokningar. Applikationen är realiserad på ett sådant sätt att den liknar Outlook 2010. Detta är gjort med hjälp av ett inköpt bibliotek vid namn DotNetBar som innehåller grafiska moduler som liknar de som används av Microsoft Office. (devcomponents.com, 2012)

Klienten har ett flertal olika grafiska kontroller som alla arbetar mot en och samma datahanterare. Datahanteraren upprätthåller en lokal cache och har kontakt med Puzzel Server. Detta kapitel fokuserar främst på hur de olika komponenterna hanterar data och samspelet med Puzzel Server.

Klientapplikationen är byggd med hjälp av ramverket WPF och använder sig av teknologin Click-once. Klienten är konfigurerad på ett sådant sätt att den tittar efter programuppdateringar vid varje uppstart, och om det finns en ny version av klienten så uppdateras klienten automatiskt.

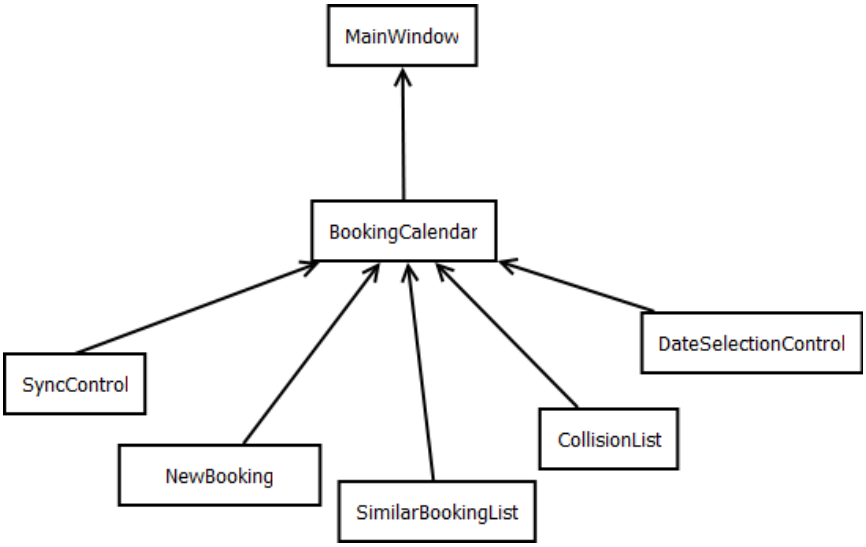
6.1 Datahantering

En Puzzel-klient har en lokal cache på all data som har hämtats från servern. Tack vare att servern håller reda på vilka bokningar som en klient har fått skickat till sig behöver inte bokningar skickas över till en klient mer än en gång. Klassen ServerMapper sköter all kommunikation med servern och upprätthåller den lokala cachen. En statisk referens till ett objekt i klassen ServerMapper finns i Puzzel-klientens rot, och all dataåtkomst och hantering sker genom detta objekt.

När klienten startar upp skapar ServerMapper-objektet en anslutning till servern med servens funktion GuidConnect. Denna funktion returnerar en GUID. Denna GUID sparas i ServerMapper-objektet och kommer att användas vid varje anrop för sökning och modifiering av bokningsdata. Efter detta anropas serverns funktion WhoAmI, och ett objekt av typen Lecturer returneras. Detta objekt kommer härefter att användas för att identifiera användaren. Om inte användaren kan identifieras skickas händelsen LogInFailed ut, och ett inloggningsfönster kommer att visas. Här kan användaren ange användarnamn och lösenord för att logga in på nytt. Då användaren angett ett nytt användarnamn och lösenord försöker ServerMapper-objektet göra en ny anslutning till servern på nytt.

När anslutningen och identifieringen har slutförts inläses all data av typerna Lecturer, Room, RoomEquipment, RoomCategory, Course, AcademicYear, Period samt StudentGroup från servern och listor på de olika objekten av varje datatyp sparas. Listorna av de olika dataobjekten används för att koppla ihop bokningsdeltagare med rätta deltagarobjekt. Hur detta går till beskrivs i kapitel 4.2, Datamodell. Efter uppstart och inläsning av data får inte klienten några uppdateringar med nya objekt av dessa typer. Detta är en liten brist i systemet, men då dessa data inte ändrar på sig ofta anses det vara acceptabelt.

När all data har lästs in är inloggningsprocessen slutförd, och applikationens huvudsakliga användargränssnitt startar upp. Det huvudsakliga gränssnittet är uppbyggt av objekt i klasserna MainWindow, BookingCalendar, NewBooking, SimilarBookingList, CollisionList samt DateSelectionControl. Vilka objekt som äger vad kan ses i Figur 35.



Figur 35, Ägande av de objekt som bygger upp det grafiska gränssnittet i Puzzel-klienten

BookingCalendar skapar av de övriga klasserna objekt som hanterar, visar eller kan modifiera bokningsdata och innehåller inställningar på den aktiva bokningsmallen. BookingCalendar är den klass som möjliggör sökningar efter bokningsobjekt utifrån en aktiv bokningsmall samt ger användaren möjlighet att ändra bokningar med drag-och-släpp (eng. Drag-and-drop) metoder. För övriga ändringar av bokningsdata används dialogfönster som öppnas vid borttagning, publicering och ändring av bokningar. Dessa dialoger skapas och hanteras med hjälpklassen BookingHandler. En statisk referens till ett objekt i klassen BookingHandler finns i roten av Puzzel-klienten.

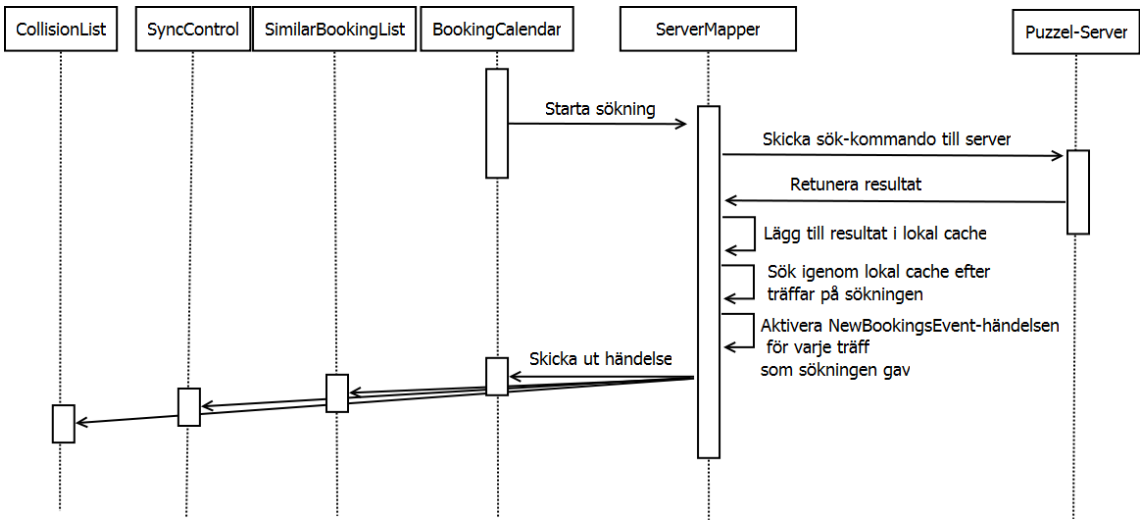
6.1.1 Sökning av bokningar

Vid sökningar av bokningar genererar BookingCalendar ett bokningsobjekt innehållande alla deltagare som finns på den aktiva bokningsmallen. Bokningsobjektet ges sedan till ServerMapper-objektets funktion startBackgroundBookingRequest. En BackgroundWorker-tråd startas sedan.

BackgroundWorker är en speciell tråd som kan rapportera tillbaka till skaparen av tråden med hjälp av fördefinierade händelser. När denna tråd exekverar visas en laddningsruta i det grafiska användargränssnittet och input spärras. För att definiera vad trå-

den skall göra binder man upp BackgroundWorker-händelsen DoWork till en händelsehanterare. När trådens arbete är slutfört aktiveras händelsen RunWorkerCompleted, och data som inhämtas hanteras. Genom att binda upp dessa två händelser till ServerMappers logik för att anropa servern och filtrera data kan bokningar läsas ut, sparas och återställas i bakgrunden.

Händelsehanteraren för DoWork har i ServerMapper ett flertal olika lägen: Load, Save, Confirm, Deltete och Restore. I Load-delen kontrolleras vilka datum användaren visar i kalendervyn, och det bokningsobjekt som används för sökning. Det kontrolleras om detta datum hör till en period inom ett läsår, och sökningen kommer då att göras inom periodens gränser. Går det inte att passa in datumet inom en period söks det två veckor bakåt i tiden och två månader framåt. Hur data från sökningen fås ut visas i Figur 36.



Figur 36, Flöde för sökning av bokningar från Puzzel-klienten

Sökningar som sker från BackgroundWorker anropar funktionen SetActiveTemplate på Puzzel Server. Då denna funktion endast används då klienten antingen navigerar sig mellan period-gränser eller byter aktiv bokningsmall måste serverns aktiva bokningsmall för klienten också uppdateras för att data skall filteraras korrekt. Efter att sökningen har gjorts införs resultatet i ServerMappers lokala bokningslista, och en ny sökning sker inom den lokala listan. Eftersom servern endast skickar ut bokningar som klienten ännu inte är medveten om är det fullt möjligt att bokningar som lästs in tidigare bör vara

med i det slutgiltiga sökresultatet. Varje bokning som hittats aktiverar sedan en NewBookingsEvent-händelse från BackgroundWorkerComplete-hanteraren. Dessa händelser hanteras i sin tur av modulerna CollisionList, SyncControl, SimilarBooking-List och BookingCalendar. Alla bokningar som hanteras av dessa komponenter fås med NewBookingsEvent-händelsen.

6.1.2 Nya eller uppdaterade bokningar

För att göra ändringar på en gammal bokning eller för att modifiera en befintlig bokning anropas funktionen Save i ServerMapper-klassen. Funktionen startar sedan Back-groundWorker-tråden i läget Save, där en lista på bokningar som skall sparas ges som ett argument.

I DoWork-hanteraren skickas sedan ett anrop vidare till Puzzel Serverns Save-funktion för varje bokning. Bokningens PuzzelID returneras när bokningen har sparats. Efter att bokningens PuzzelID har utlästs skickas till servern en förfrågan med funktionen FindBookingById där det utlästa PuzzelID:t ges som argument och bokningsobjektet returneras. Bokningen införs i den lokala cachen.

När alla bokningar har sparats och lagts i cachen aktiveras BackroundWorkerComplete-händelsen, och i hanteraren för händelsen aktiveras NewBookingsEvent-händelsen för varje sparad bokning.

6.1.3 Publicering av bokningar

När en bokning som har sparats på Puzzel Server skall publiceras i ABEX används funktionen ConfirmBookings i ServerMapper. ConfirmBookings tar emot en lista på ID-strängar som argument och också en BackgroundWorker-tråd i läget Confirm med samma lista på ID-strängar som argument. I DoWork-hanteraren skickas sedan ett anrop till Puzzel Server där bokningarna med överensstämmande PuzzelID markeras för att skickas till ABEX. Bokningarna utläses sedan på nytt med hjälp av deras PuzzelID-nummer. Dessa bokningar kommer nu ha en status som låser dem för modifiering från Puzzel-klienter.

När bokningarna har inlästs aktiveras händelsen BackgroundWorkerComplete, och hanteraren aktiverar NewBookingsEvent-händelsen för varje inläst bokning.

6.1.4 Borttagning av bokningar

Vid borttagning av bokningar används funktionen DeleteList i klassen ServerMapper. Denna tar emot en lista på bokningar som skall tas bort. En BackgroundWorker-tråd startas i läget Delete. DoWork-hanteraren skickar vidare listan av bokningsobjekt till servern, där de tas bort. Efter att anropet är klart aktiveras BackgroudWorkerComplete-händelsen, och en uppdatering av bokningsdata görs genom att aktivera uppdateringsprocessen direkt. Uppdatering av bokningsdata beskrivs i avsnittet ”Uppdatering av bokningsdata” nedan.

Om en bokning har publicerats i ABEX måste borttagningen göras i två olika steg. Först skickas ett borttagningskommando till serven. Bokningen returneras då med status ”deleted”, men räknas fortfarande som en aktiv bokning. När man sedan publicerar bokningen tas den bort från alla system.

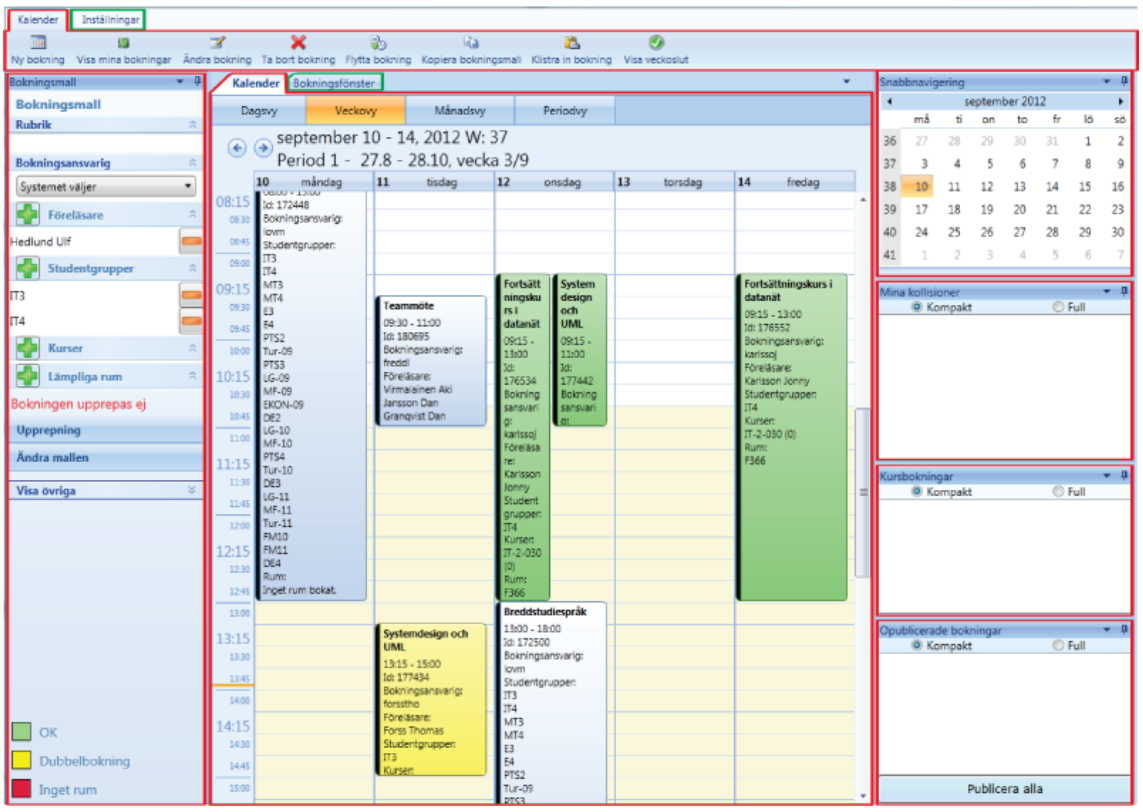
6.1.5 Uppdatering av bokningsdata

För att se till att en Puzzel-klient får alla uppdateringar som görs på bokningar görs var femte sekund ett anrop till Puzzel-Serverns funktion GetUpdates. Denna funktion returnerar alla bokningsobjekt som har blivit uppdaterade eller borttagna sedan sista anropet. Inläsningen av uppdateringar sker med en separat tråd som körs i bakgrunden. Bokningsobjekten som hittas läggs till i den lokala cachen och en NewBookingsEvent-händelse aktiveras för varje bokning.

Bokningar som har flaggan removed satt till sann hanteras på ett speciellt sätt. Dessa bokningar är markerade för borttagning. När en sådan bokning kommer in så tas den bort från den lokala cachen, och en NewBookingsEvent-händelse skickas sedan ut så att alla grafiska komponenter kan reagera på borttagningen.

6.2 Grafiska komponenter

Puzzel-klienten består av ett flertal olika grafiska moduler. Hur arbetsytan är uppdelad visas i Figur 35.

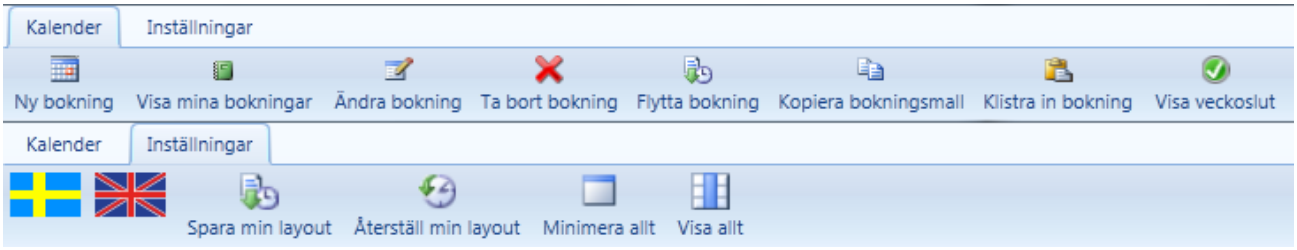


Figur 37, Uppdelning av arbetsytan i Puzzel-klienten. Röda ramar visar nu synliga komponenter, gröna ramar visar undangömda komponenter

De olika grafiska modulerna samt en kort beskrivning av dem finns nedan.

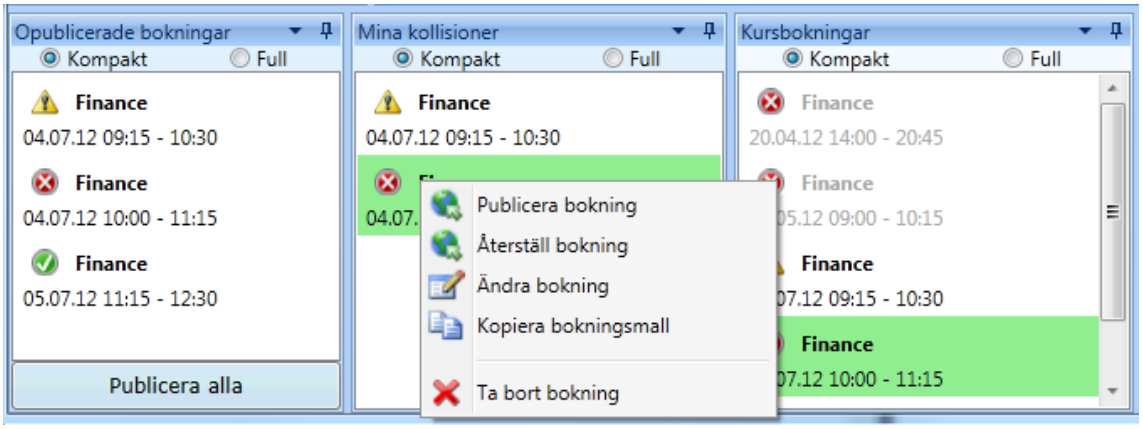
6.2.1 ”Ribbon”-menyn

Denna meny är en del av MainWindow, och innehåller funktioner för att hantera layout, byta språk samt några genvägar till BookingCalendar-funktioner. Hur Ribbon-menyn ser ut i dess två olika lägen visas i Figur 38. Ribbon-menyn använder sig av biblioteket DotNetBar.



Figur 38, "Ribbon"-menyns två olika lägen.

6.2.2 Listkomponenter



Figur 39, En överblick av de olika grafiska listkomponenterna.

De tre komponenterna CollisionList, SimilarBookingList och SyncControl fungerar enligt samma principer i grund och botten, endast de bokningar som de filtrerar ut skiljer sig. Varje listkomponent lyssnar på händelsen NewBookingsEvent från ServerMapper-objektet. Varje gång en ny bokning kommer med den händelsen går de igenom ett specifikt filter för varje listkomponent. Om man högerklickar på en bokning i en listkomponent visas en meny beroende på bokningens status. Denna meny är gemensam för de olika komponenterna. Funktionerna som utför kommandona finns i klassen BookingHandler, med undantag för Kopiera bokningsmall som utförs av klassen BookingCalendar. En överblick av hur de olika komponenterna ser ut visas i Figur 39. En kort beskrivning av de olika listkomponenterna finns nedan.

SyncControl

När en ny NewBookingsEvent-händelse är aktiverad kontrolleras om bokningen har gjorts av den aktiva användaren, och om den har ändringar som ännu inte blivit publicerade i ABEX. Detta kan ses i bokningens status. Om en bokning är gjord av den aktiva användaren och har opublicerade ändringar kommer den synas på denna lista.

SyncControl anropar funktionen GetBookingsoutOfSync på ServerMapper-objektet vid uppstart, som i sin tur läser in alla bokningar som är opublicerade och gjorda av användaren.

Denna komponent har också en extra knapp, som de övriga listkomponenterna saknar, för publicering av alla bokningar som finns registrerade i listan.

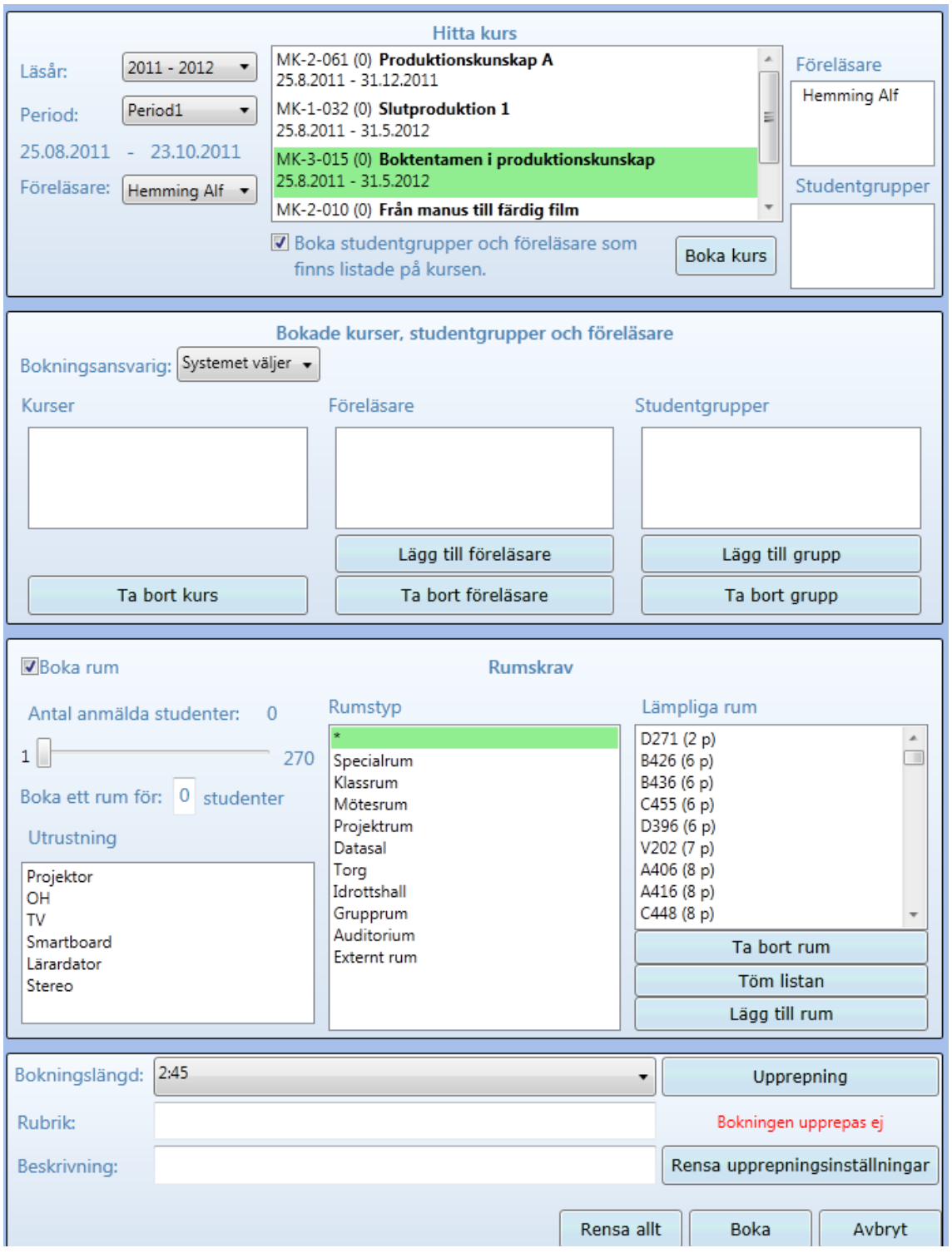
SimilarBookingList

SimilarBookingList innehåller en lista på kurser som den aktiva bokningsmallen har som deltagare. När en NewBookingEvent-händelse registreras kontrolleras om bokningen har någon av kurserna som deltagare. Om bokningen har en eller flera av de aktiva kurserna som deltagare kommer den synas på denna lista.

CollisionList

När en ny NewBookingsEvent-händelse registreras kontrolleras om bokningen är gjord av den aktiva användaren, samt om bokningen är en dubbelbokning eller om den saknar rum. Om bokningen har ett eller flera lämpliga rum på sin lista och saknar valda rum, eller om den är en dubbelbokning så kommer bokningen att visas i denna lista.

6.2.3 NewBooking



Figur 40, Översikt av NewBooking-komponenten

NewBooking-komponenten används för att generera nya bokningsmallar. Ett exempel på hur NewBooking kan se ut visas i Figur 40. Komponenten är indelad i 4 olika delar.

Del ett används för att hitta kurser enligt angivet läsår, period och föreläsare. Här kan man markera kurser samt föreläsare för kurser. När man trycker på ”Boka kurs” införs markerade kurser och föreläsare i en lista på kurser som skall bokas samt i en lista på föreläsare som skall bokas.

I del två syns alla deltagare som finns med i bokningsmallen. Här kan man lägga till föreläsare och studentgrupper, samt ställa in bokningsansvarig för bokningen. Det är också möjligt att ta bort kurser, föreläsare och studentgrupper från deltagarlistorna.

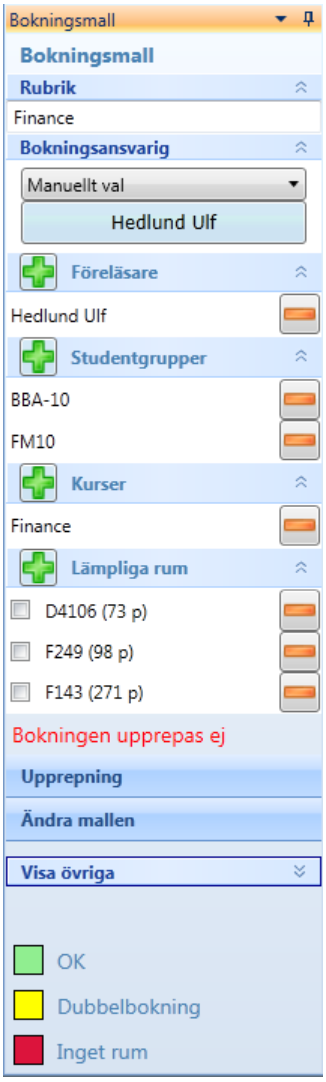
Del tre innehåller verktyg för att skapa en lista på lämpliga rum för bokningsmallen. Rummen filtreras enligt deltagarnas rättigheter samt enligt storlek, utrustning och rumstyp. Utifrån dessa filter genereras en lista på rum som uppfyller kraven. Denna lista kan sorteras om, extra rum kan läggas till och rum kan tas bort. Det är möjligt att tömma listan och ange mer specifika krav genom att trycka på ”Töm listan” och sedan lägga till de rum man vill ha genom att trycka på ”Lägg till rum”. När Puzzel gör bokningar kommer den här listan att kopieras som lämpliga rum för bokningen, och det första rummet som är ledigt kommer att bokas.

I del fyra kan man ange bokningens längd, rubrik och beskrivning. Rubriken ifylls automatiskt då man lägger till en kurs som deltagare, men kan ändras vid behov. Det är också möjligt att ställa in upprepningsinställningar för mallen.

När inställningarna är gjorda och man trycker på ”Boka” kommer en ”NewTemplate”-händelse att aktiveras. Händelsen registreras i BookingCalendar och den nya bokningsmallen utläses. En sökning kommer sedan att ske på bokningsmallen, och när sökningen är klar kommer alla bokningar relevanta för den valda bokningsmallen att visas i kalendervyn.

6.2.4 BookingTemplate

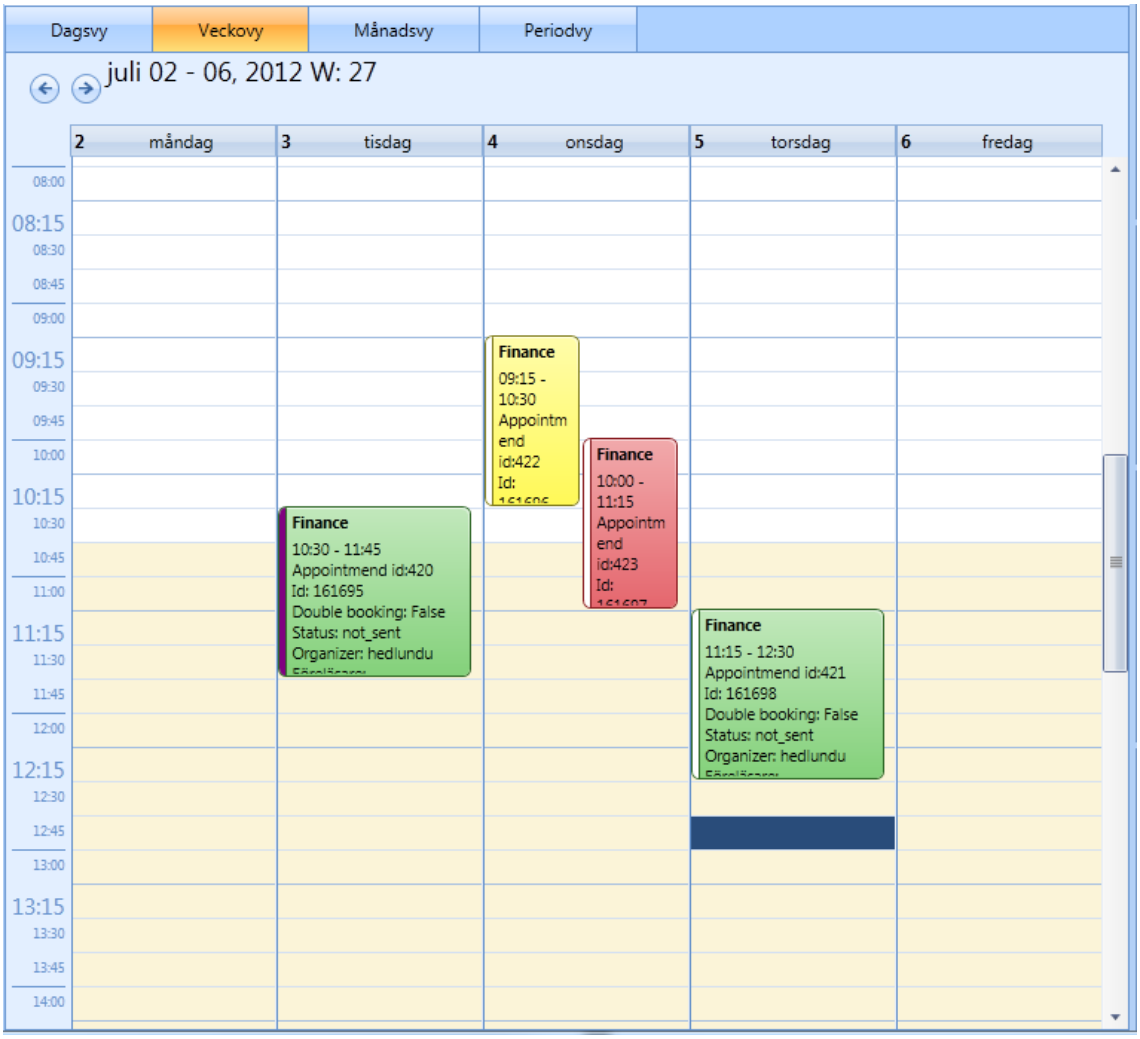
Bokningsmallen som visas i det grafiska användargränssnittet och tillåter modifieringar direkt på bokningsmallen tillkom relativt sent i Puzzel. Bokningsmallen visar data som finns i ett objekt av typen BookingTemplate. BookingTemplate finns lagrat i objektet BookingCalendar och är till skillnad från de övriga grafiska komponenterna inte fristående. Det är möjligt att utföra näst intill alla modifieringar på bokningsmallen som man kan göra i NewBooking-fönstret direkt från denna komponent. Operationerna utförs genom dialoger som skapas i BookingTemplate-klassen. Här kan man lägga till och modifiera kurser, föreläsare, studentgrupper samt lämpliga rum. Man kan dessutom ange om man vill att rumsbokningar skall visas i kalendervyn. Det är också möjligt att ändra rubrik och bokingsansvarig. Det är dock inte möjligt att ändra beskrivningen i bokningsmallen. Hur bokningsmallen illustreras i Puzzel ses i Figur 41.



Figur 41, Visuell presentation av bokningsmallen

Utöver detta är det möjligt att visa bokningar med andra deltagare än de som man har i sin bokningsmall. Här kan man lägga till föreläsare, kurser eller studentgrupper som man är interesserad av att se bokningar för i sin kalendervy, men som inte skall ingå i bokningarna som man gör med mallen.

6.2.5 BookingCalendar



Figur 42, Kalenderkomponenten i BookingCalendar samt visning av bokningar.

BookingCalendar-klassen innehåller en kalenderkomponent från biblioteket DotNetBar. Hur kalenderkomponenten ser ut visas i Figur 42. För att kunna använda Puzzels datamodell vid generering av bokningar har den klass utvidgata som används för att visa bokningar i kalendervyn. BookingCalendar sköter om konverteringen av de olika dataobjekten. Den klass som används för visning av bokningsdata kallas PuzzelAppointment, och är en utvidgad version av klassen Appointment som används av DotNetBarkomponenten. PuzzelAppontment kan ta emot ett bokningsobjekt av typen Booking och sedan generera bokningens innehåll och visuella utseende utifrån objektet. Det är möjligt att utifrån ett PuzzelAppointment-objekt få ut det urspringliga Booking-objektet.

BookingCalendar-klassens huvudsakliga uppgift är att koppla ihop bokningsmallen med sökning av bokningar, och sedan införa sökresultatet i kalendern. BookingCalendar är också ansvarig för att förutspå resultatet av nya bokningar då man går in i bokningsläge. Då man gör en bokning i Puzzel får man en ”skugga” av en bokning som följer med musen då man rör sig i kalendervyn. Denna skugga visar med hjälp av färgkoder om bokningen kolliderar med en annan bokning samt om det finns lediga rum. Detta görs genom att skapa temporära bokningsobjekt, och sedan utvärdera dessa mot vad som finns i den lokala cachen.

Puzzel-klienten behöver veta vilken tidsperiod användaren behöver söka bokningar från. För att ta reda på detta lyssnar BookingCalendar på kalenderkomponentens händelse UserDataChange som aktiveras då användaren navigerar till en ny datumperiod. BookingCalendar utläser datumperioden och jämför sedan dessa datum mot den lista av perioder som finns i ServerMapper-objektet. Passar datumen in inom en period sätts start- och sluttid för sökningar in i listan currentTimeSpan, som finns i applikationens rot. På position ett i listan finns sökningens starttid, på position två sökningens sluttid. Om start- eller sluttid ändras kommer BookingCalendar att göra en ny sökning med den nya tidsperioden.

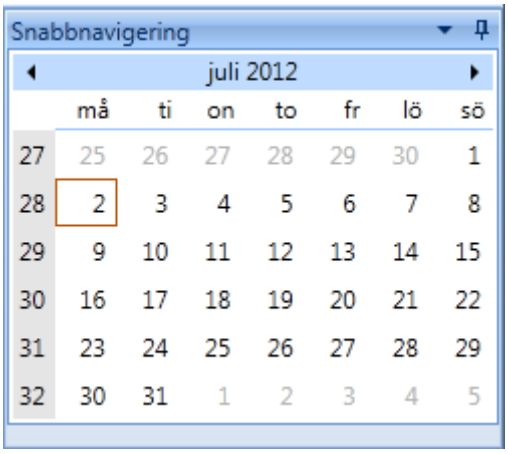
Då man ändrar bokningsmallen kommer alla bokningar att tömmas från kalenderkomponenten. BookingCalendar genererar sedan ett bokningsobjekt utifrån bokningsmallen, och skickar bokningsobjektet till ServerMapper-funktionen startBackgroundBookingRequest.

BookingCalendar lyssnar på händelsen NewBookingsEvent från ServerMapper. När en ny händelse aktiveras utläses bokningsobjektet från händelsen. Bokningsobjektets deltagare jämförs sedan med bokningsmallens deltagare, men alla rum som inte har markerats för visning ignoreras. Om bokningsobjektet och bokningsmallen har gemensamma deltagare visas bokningen i kalenderkomponenten.

BookingCalendar är också ansvarig för att meddela ändringar av bokningsmallen till listkomponenten SimilarBookingList, som alltid behöver veta vilka kurser som finns i bokningsmallen.

6.2.6 DateSelectionControl

Denna komponent innehåller en datumnavigeringskomponent från biblioteket DotNetBar. Komponenten används för snabbt kunna navigera mellan olika datum. När man klickar på ett datum i datumnavigeringskomponenten så aktiveras händelsen SelectedDateChanged. DateSelectionControl lyssnar på denna händelse. När ett datum byts utläser DateSelectionControl det nya datumet och skickar det vidare till BookingCalendarobjektet, som i sin kommer att byta datum. DateSelectionControl visass i Figur 43.



Figur 43, DateSelectionControl

7 EXCHANGE-KLIENTEN

Exchange-klientens enda uppgift är att ta emot inkommande data från ABEX och skriva in den i en databastabell. Tabellen läses sedan av PuzzelServer och bokningsdata hanteras där. Exchange-klienten är byggd med hjälp av ramverket ASP .NET MVC 3.

Data som kommer från ABEX kommer som post-data på en webbadress. Det finns två olika funktionsadresser som ABEX använder:

UpdateBooking
Hit skickas alla bokningsprenumerationer från ABEX samt sökresultat.

ConfirmBooking/{id}
Hit skickas en s.k. ”callback” när en bokning har utskickats till ABEX från Puzzel. Denna adress innehåller ett ID-nummer. Denna ID är den skickade bokningens Puzzel-ID. I den callback som kommer från ABEX finns endast en ABEX ID. Med hjälp av detta meddelande kan en ABEX ID kopplas ihop med en PuzzelID.

Kommunikationen mellan Exchange-klienten och ABEX sker med HTTPS. För att säkra kommunikationen används samma autentiseringsprinciper som när man kommunicerar med ABEX. I huvudet för varje meddelande skickas ett användarnamn och lösenord kodat i Base64-format. Dessa värden avkodas och jämförs sedan med förinställda värden som finns i web.config-filen. Endast om de överensstämmer blir meddelandet accepterat.

8 VIDAREUTVECKLING

Bokningar gjorda med hjälp av Puzzel innehåller en del extra data i jämförelse med andra bokningssystem. En sådan enkel sak som att ha en lista på flera rum som bokningen kan göras i öppnar upp flera vidareutvecklingsmöjligheter. Några av dessa möjligheter diskuteras nedan.

8.1 Rumsoptimering

Det ursprungliga syftet med Puzzel var att göra ett rumsoptimeringssystem, d.v.s. ett system som försöker att optimera mängden utrymmen som är tillgängliga vid varje tidpunkt, och som t.ex. kan lämna rum tomma under längre perioder om möjligt. Om ett rum inte används under en period minskar man på städkostnader.

Tack vare att Puzzel samlar in data om vilka rum som en bokning kan befinna sig i så skulle det vara möjligt att låta Puzzel optimera användingen av rum. Systemet för optimering borde vara väldigt flexibelt, eftersom kraven på rum skiljer sig mycket från bokning till bokning. Det skulle också kunna finnas möjlighet för Puzzel att försöka boka om andra bokningar om en specifik bokning behöver ett speciellt rum som är upptaget.

8.2 Personliga kalendrar

En brist som finns i Puzzel är att personliga kalendrar från Exchange inte inläses. Orsaken är att ABEX endast tillåter prenumerationer på gruppkalendrar, inte på personliga kalendrar. Om en bokning görs i en personlig kalender som innehåller ett gruppobjekt så kommer prenumerationen för gruppobjektet att skickas ut. Saknas ett gruppobjekt så sker inte någon prenumerationsutskickning.

Ifall ABEX-protokollet vidareutvecklas på ett sådant vis att prenumerationer kan läggas upp på personliga kalendrar skulle Puzzel kunna öppnas upp för andra sorters bokingar än endast kursbokningar. I dagsläget finns det en spärr i Puzzel-klienten som endast tillåter att man gör bokningar som innehåller minst en kurs som deltagare. Orsaken är att man inte skall försöka boka möten med personer vilkas personliga kalendrar inte syns i Puzzel.

Det finns möjligheter att göra sökningar på personliga kalendrar i ABEX. En möjlighet att realisera personliga kalendrar i Puzzel skulle vara att inläsa alla personliga kalendrar t.ex. en gång om dagen. Detta skulle dock innebära att man för sökperioden en gång om dagen inläser all bokningsdata som finns i hela systemet. Detta är möjligt att genomföra, men datamängden som överförs blir väldigt stor, och det skulle inte vara möjligt att hålla personliga kalendrar uppdaterade i realtid. Att upptäcka personliga bokningar som har tagits bort mellan sökningarna utan prenumerationer på dem är också en utmaning.

8.3 Automatisk schemaläggning

För att förenkla schemaläggningen av kurser ytterligare skulle man kunna låta systemet själv schemalägga undervisningstimmar. En användare skulle kunna skapa en bok-

ningsmall som vanligt, och sedan ställa in hur många timmar per vecka som ska bokas, samt under vilka dagar och tidpunkter bokningarna kan göras. Om användarna inte ställer allt för stränga krav på när de vill ha lektioner skulle en automatisering som denna kunna spara mycket tid.

Hela planeringsprocessen skulle i detta fall kunna ske på klientnivå. Saker som t.ex. lunchpaus för studenter måste beaktas för att en automatisk schemaläggningsprocess ska bli användbar.

9 SLUTSATSER

Slutresultatet av det här projektet är ett fullt fungerande kursbokningssystem som har integrerats med Arcadas äldre bokningssystem. Puzzel har i skrivandets stund varit i bruk i över tre månader. Tack vare att oväntade undantag rapporteras från både klient och server så har påträffade buggar snabbt kunnat åtgärdas.

I och med att Puzzel existerar vid sidan om två andra bokningssystem i Arcada så har användargraden inte varit fullt så hög som jag hoppats. Användarna kan sedan förr använda de äldre systemen, och att lära sig ett nytt system har avskräckt många från att ge Puzzel en chans. Många föredrar att fortsätta jobba på samma vis som de har gjort tidigare istället för att lära sig nya tillvägagångssätt. Att jobba med bokningsmallar istället för med enskilda bokningar har varit ett svårt steg att ta för några, då det innebär ett helt nytt sätt att tänka.

Den respons som jag har fått från aktiva användare av Puzzel har varit väldigt positiv och jag tror att användargraden av Puzzel kommer att öka då flera personer får information om vad Puzzel är för någonting. Jag har haft ett flertal undervisningstillfällen för Arcadas personal där jag har gått igenom hur Puzzel används, tyvärr har inte många från personalen deltagit på dessa tillfällen.

Puzzel-projektet var från början inte särskilt väldefinierat. Allt som var bestämt var att Puzzel skulle bli ett nytt verktyg som skulle underlätta kursbokningsprocessen. Det har varit en utmaning att analysera och försöka förbättra sättet som kursbokningar görs på,

framförallt då jag själv har inte haft någon erfarenhet i hur kursbokningsprocessen gått till innan jag började jobba med Puzzel. Att se på det hela från ett perspektiv där jag inte riktigt varit medveten om hur bokningsprocessen går till har säkert haft fördelar, då jag inte redan var ”inkörd” på ett visst sätt att jobba.

De plattformer som Puzzel bygger på, WCF och WPF, var redan bestämda då jag började jobba på projektet. En påbörjad version av Puzzel klienten fanns redan då jag började. Denna gjordes dock helt om från grunden då det var tänkt att den påbörjade versionen skulle kommunicera direkt med ABEX och andra datakällor.

Ursprungligen var min arbetsuppgift att bygga upp Puzzel-klienten, men jag fick sedan fortsätta att bygga upp Puzzel-servern också. Jag har inte haft någon tidigare erfarenhet av varken WCF eller WPF, och serverprogrammering var ganska nytt för mig på det stora hela, lite erfarenhet hade jag dock inom C#. Många dagar har spenderats på att läsa böcker om WCF och WPF. Jag är väldigt nöjd med valet av plattformar. Jag blev framförallt väldigt positivt överaskad över hur lätt och snabbt man kan bygga upp användargränssnitt med WPF. I denna rapport nämns inte mycket om användargränssnittet som Puzzel har då det lätt skulle ha fyllt ut ytterligare 20 sidor, men jag garanterar att WPFs flexibilitet och klara syntax har hjälpt massvis. Projektet har varit oerhört lärorikt, och jag har fått mycket bättre förståelse för nätverksprogrammeringens problematik.

KÄLLOR

Devcomponents.com, 2012. DotNetBar [www] Tillgänglig: http://www.devcomponents.com/dotnetbar/ Hämtad: 4.07.2012

Jon Skeet, 2011 *C# in Depth, Second Edition*

MacDonald Matthew. 2010, *Pro WPF in C# 2010 - Windows Presentation Foundation in .NET 4*

Microsoft, 2010a, LINQ (Language-Integrated-Query) [www] Tillgänglig: http://msdn.microsoft.com/en-us/library/bb397926 Hämtad 10.05.2012

Microsoft, 2010b, Windows Communication Foundation [www] Tillgänglig: http://msdn.microsoft.com/en-us/library/ms735119 Hämtad 12.05.2012

Microsoft, 2010c, .NET Framework 4 [www] Tillgänglig: http://msdn.microsoft.com/en-us/library/w0x726c2 Hämtad 12.05.2012

Vanhaniemi Thomas. 2010, Arkitekturbeskrivning av egenutvecklad XML-webbtjänst för hantering av kalenderdata i Microsoft Exchange Server 2010 [www] Ingenjörsarbete, Arcada – Nylands Svenska Yrkeshögskola, Helsingfors, Finland. Tillgänglig: https://publications.theseus.fi/handle/10024/24081 Hämtad 10.06.2012

BILAGA:

SNABBSTARTGUIDE FÖR KURSBOKNINGSVERKTYGET PUZZEL

ARCADA

PUZZEL

Snabbstartguide för kursbokningsverktyget Puzzel
Version 1.0

KORT OM PUZZEL

Puzzel är ett kursbokningsverktyg utvecklat av Arcadas IT-avdelning. Syftet med verktyget är att underlätta bokning av kurser, där kursens och personalens kalendrar kommer i första hand. I det här häftet får du en kort introduktion till hur verktyget används.

Puzzel är ett program som måste installeras och kommer endast att fungera på Windows-datorer. Du kan installera och köra programmet på vilken Internetansluten Windows-dator som helst, antingen hemma eller i Arcada.

Puzzel hittar du på följande adress:

puzzel.arcada.fi

NYA KONCEPT

I Puzzel används ett par nya koncept som du bör vara medveten om innan du börjar boka.

Bokningsmall

En bokningsmall innehåller alla inställningar som behövs för att göra en bokning. Du kan skapa en bokningsmall från grunden eller återanvända en bokningsmall från en befintlig bokning.

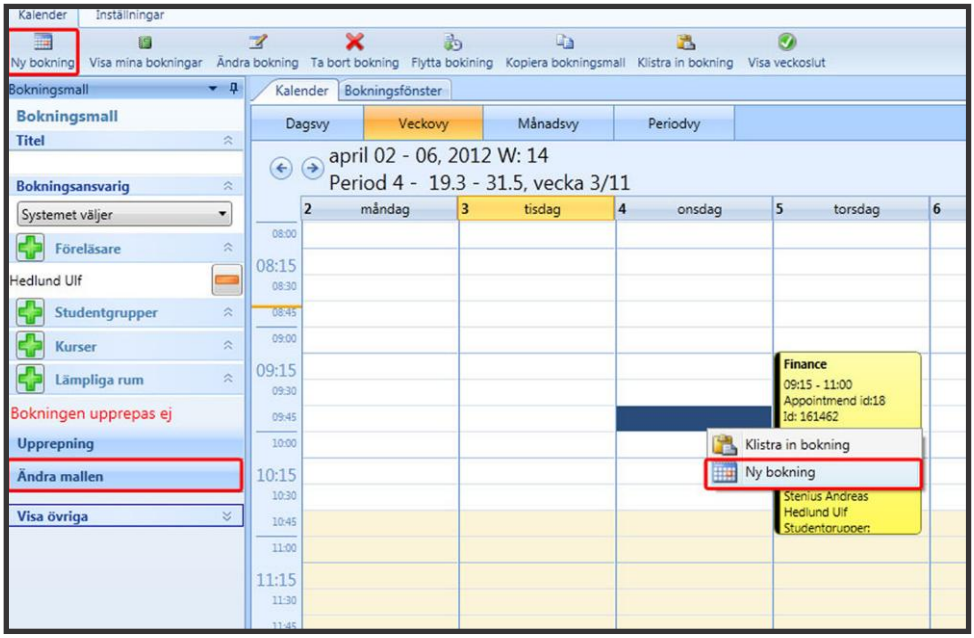
Sammanslagen kalendervy

Den kalendervy som du ser i Puzzel är alltid anpassad till den bokningsmall som är aktiv. Vyn kombinerar alla deltagarnas kalendrar till en överskådlig vy, där du snabbt får en överblick för att lättare hitta tidpunkter som passar alla. Har du till exempel valt en kurs, en studiegrupp och en föreläsare i din bokningsmall, visas deras kalendrar i en och samma kalendervy.

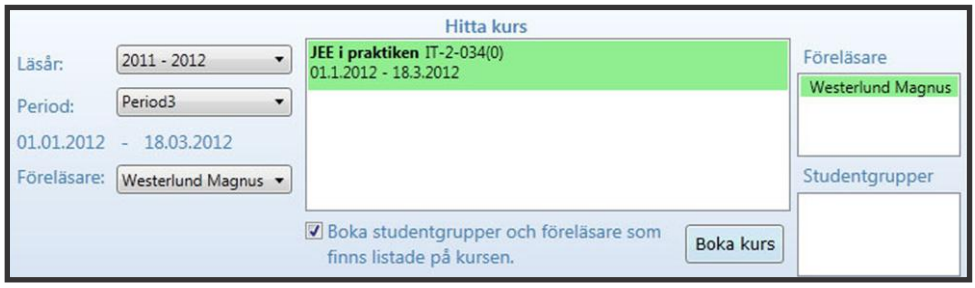
Rummet på andra plats

Till skillnad från ARBS, som är ett rumsbokningssystem, behöver du i Puzzel inte först hitta ett ledigt rum att boka. Du anger istället hur stort rum du vill ha, vilken sorts rum du kan vara i samt om du behöver någon extra utrustning i rummet, t.ex. en projektor. Dessa kriterier sållar genast fram alla rum som passar, och Puzzel tar sedan hand om att hitta ett ledigt rum åt dig med hjälp av denna lista.

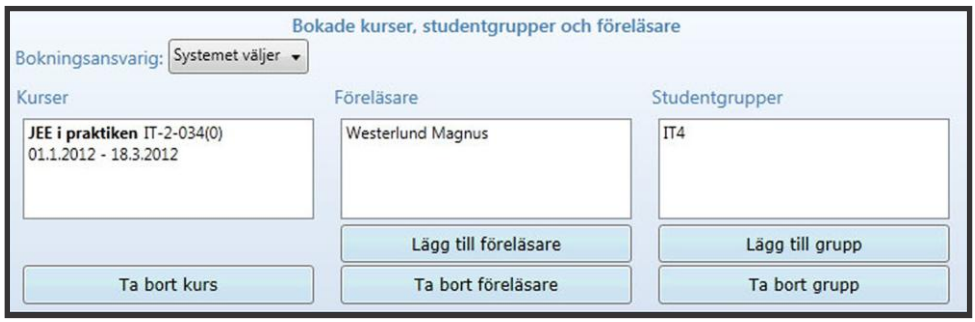
KOM IGÅNG MED PUZZEL



För att göra en bokning kan du kopiera bokningsmallen från en befintlig bokning, eller skapa en helt ny bokningsmall. Välj något av de rödmarkerade områdena på bilden ovan för att komma igång. Rutan som syns ovanpå bokningen får du fram genom att högerklicka med musen i kalendervyn.

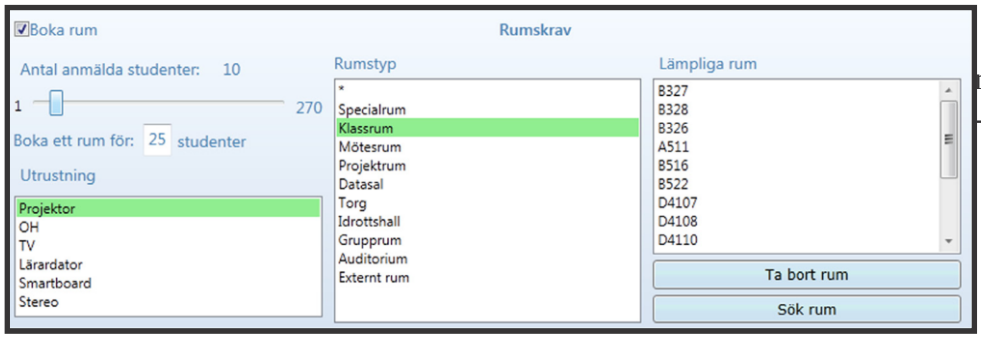


Detta tar dig till bokningsfönstret, där du hittar den kurs du vill boka. Alla kurser hämtas från ASTA. Finns inte den kurs du vill boka i listan betyder det högst sannolikt att kursen inte är publicerad i ASTA, och är då det första du bör kontrollera. Kurser visas enligt läsår, period och föreläsare. När du loggar in hittar du dina kurser som går i den innevarande perioden. Vill du boka andra kurser är det bara att välja rätt läsår, period och föreläsare för att hitta de kurser du söker. Du kan markera föreläsare som finns registrerade som lärare på kursen för att även boka dem.

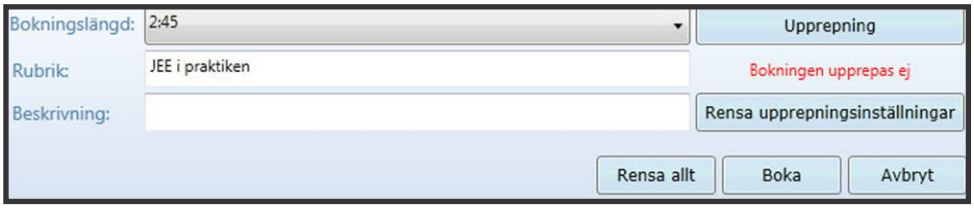


Nedanför ”Hitta kurs” rutan ser du vilka grupper, föreläsare och kurser du kommer att boka. Studentgrupper måste i dagsläget läggas till manuellt. Du lägger till studentgrupper genom att välja ”Lägg till grupp”, och föreläsare genom att välja ”Lägg till föreläsare”. Här kan du också välja vem som skall vara

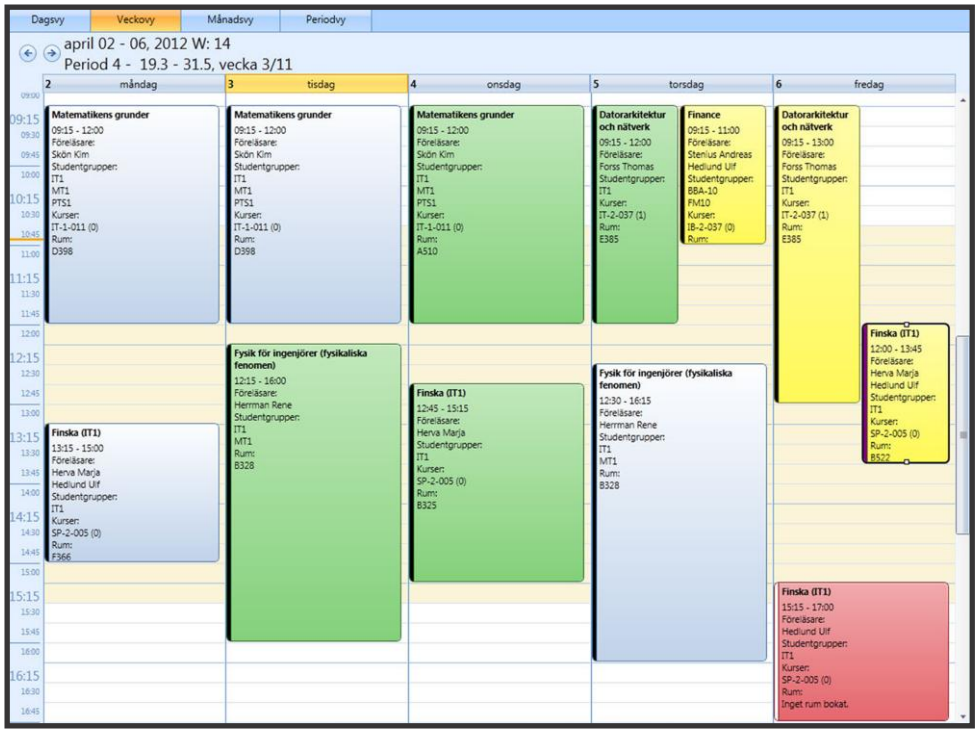
ansvarig för bokningen. Den ansvariga för bokningen blir ägare till bokningen i Outlook och kan därmed ändra bokningen den vägen också. Väljer du inte någon i denna lista kommer Puzzel att välja en lämplig bokningsansvarig.



I nästa steg anger du dina krav på rum. Om kursen du valt redan har anmälda studenter kommer detta antal att visas här. Ange hur många studenter du vill att rummet ska kunna rymma genom att antingen dra på skalan eller genom att skriva in ett antal i textrutan. Välj därefter vilken utrustning du behöver och vad för rumstyper du vill ha. Baserat på dina val kommer en lista med lämpliga rum att presenteras på höger sida. Vanligtvis är du klar med rumsvalet i detta skede, men du kan nu också lägga till extra rum i listan genom att välja ”Sök rum” och sortera rumslistan. Puzzel försöker hitta ett passande rum i den ordning som angivits här, med början uppifrån och ner. Endast rum som den bokningsansvarige har rätt att boka kommer att synas i listan.



Till sist kan du ange en rubrik, kort beskrivning och längden på din bokning. Rubriken fylls automatiskt i med den valda kursens namn, men kan här fritt ändras. En bokning måste ha en rubrik och en längd, men beskrivningen är helt valfri. Du kan också ange om bokningen ska upprepas enligt ett visst mönster. Välj ”Boka” för att fortsätta.



Då du valt en ny bokningsmall läses alla relevanta bokningar in, vilket kan ta några sekunder. Puzzel färgkodar bokningar för att du lättare ska få en överblick över hur situationen ser ut.

- Grått betyder att du inte kan ändra på bokningen.
- Grönt betyder att allting är ok.
- Gult betyder att någon av deltagarna i bokningen blivit dubbelbokad.
- Rött betyder att inget av de valda rummen är ledigt vid den valda tidpunkten.

När musen förs över kalendervyn kommer du att se en ”skugga” av bokningen. För att placera en preliminär bokning klickar du med vänster musknapp i kalendervyn på den plats du vill boka. Klickar du på höger musknapp avbryter du bokningsprocessen, men bokningsmallen stannar kvar. För att fortsätta bokningsprocessen i detta skede väljer du ”Klistra in bokning”, antingen genom att klicka i balken högst upp i fönstret, eller i menyn som visas då du högerklickar i kalendervyn.

Vill du göra en bokning som fyller ut ett tomrum i kalendern kan du ”färga” ett valfritt område och välja ”Klistra in bokning”. Bokningslängden kommer då att uppdateras i bokningsmallen till att vara densamma som det färgade området visar.

Alla bokningar som inte är gråfärgade kan du ”dra och släppa” för att ändra tiden då bokningen äger rum. Vill du ändra längden på en befintlig bokning gör du det lätt genom att dra den i antingen början eller slutet. När du placerat ut alla bokningar som du vill ha dem är det dags att publicera ändringarna till andra system i Arcada. Detta gör du enklast genom att välja ”Publicera alla” i publiceringsfönstret. Du kan också välja att publicera enstaka bokningar genom att högerklicka på dem och välja ”Publicera bokning”.

Efter att en bokning blivit publicerad tar det en kort stund innan den blir synlig i ARBS och Outlook.

Vi på IT-avdelningen hoppas att detta verktyg kommer att underlätta ert kursbokningsarbete och medföra bättre läsordningar för studenter och personal.

En mer detaljerad guide och svar på vanliga frågor hittar du på Puzzels hemsida puzzel.arcada.fi

Har du frågor eller förslag på förbättringar, skicka gärna mail till:

puzzel@arcada.fi

PUZZEL

ARCADA

PUZZEL
puzzel.arcada.fi