Arkitekturbeskrivning av en egenutvecklad XML-webbtjänst för hantering av kalenderdata i Microsoft Exchange Server 2010

Thomas Vanhaniemi

Examensarbete
Informationsteknik
2010

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	2429
Författare:	Thomas Vanhaniemi
Arbetets namn:	Arkitekturbeskrivning av en egenutvecklad XML-webbtjänst för hantering av kalenderdata i Microsoft Exchange Server 2010
Handledare (Arcada):	Magnus Westerlund
Uppdragsgivare:	Arcada

Sammandrag:

En central kalender där alla i organisationen fyller i sina upptagna tider underlättar koordineringsarbetet dramatiskt. För att verktyget ska vara effektivt krävs att allas kalendrar är uppdaterade och korrekta. Att integrera två system kan vara en stor utmaning med många hinder. Denna rapport går igenom hur ett system, som placerar sig mellan den kommersiella kalenderlösningen Microsoft Exchange Server 2010 och klienter som vill hantera kalenderdata, är uppbyggd. Systemet förenklar klienternas arbete genom att ta hand om de tre olika tillvägagångssätt som behövs för att läsa och skriva data i centralkalendern och i kringliggande infrastruktur.

Ett designmönster är en ritning över lösningen på ett återkommande problem inom branschen. Arkitekturen i systemet som utvecklats har inspirerats av ett antal designmönster för att lösa problem, där de viktigaste designmönstren som används presenteras.

XML är ett märkspråk som används världen över för att överföra och lagra strukturerad data på ett universellt sätt. På grund av detta får XML vara bärare av data mellan det utvecklade systemet och dess klienter.

Ett system är aldrig helt färdigutvecklat; därför tar rapporten även upp de delar som ännu saknas från en komplett första version. Arkitekturen underlättar vidareutveckling; den är både flexibel och lätt att begripa ifall designmönstren som använts är bekanta.

Målsättningen med arbetet har varit att beskriva arkitekturen och dataöverföringsprotokollet i det byggda systemet. Resultatet har blivit denna rapport som beskriver arkitekturen och kopplar tillbaka till de viktigaste designmönstren som använts. Dataöverföringsprotokollet beskrivs också detaljerat för att klienter ska kunna utnyttja systemet.

Nyckelord:	Arcada, Microsoft Exchange Server, XML, designmönster, kalender, design av mjukvara, Microsoft Active Directory
Sidantal:	54
Språk:	Svenska
Datum för godkännande:	16.12.2010

DEGREE THESIS	
Arcada	
Degree Programme:	Information technology
Identification number:	2429
Author:	Thomas Vanhaniemi
Title:	Architectural description of a self-developed XML Web Service for managing calendar data in Microsoft Exchange Server 2010
Supervisor (Arcada):	Magnus Westerlund
Commissioned by:	Arcada

Abstract:

A central calendar, where everyone in the organization fills in own non-free time, helps with the coordination work dramatically. For the tool to be effective, everyone's calendar needs to be up-to-date and accurate. Integrating two systems can be a major challenge with many obstacles. This paper describes how a system, that puts itself between the commercial calendar solution Microsoft Exchange Server 2010 and clients who want to manage calendar data, is constructed. The system simplifies client work by taking care of the three different methods needed to read and write data in the central calendar and in the surrounding infrastructure.

A design pattern is a description of the solution to a recurrent problem in the industry. The developed system architecture is inspired by a number of design patterns to solve problems, where the major design patterns used are presented in this paper.

XML is a markup language used worldwide to transmit and store structured data in a universal way. Because of this, XML is chosen to be the carrier of data between the developed system and its clients.

A system is never fully developed; therefore this paper also discusses the elements still missing from a complete first version. The architecture facilitates further development: it is both flexible and easy to understand if the design patterns used are familiar.

The aim of this work has been to describe the developed architecture and the data transmission protocol. The result has been this paper that describes the architecture and shows how the main design patterns are used. The data transfer protocol is also described in detail to enable clients to use the system.

Keywords:	Arcada, Microsoft Exchange Server, XML, Design Patterns, calendar, software design, Microsoft Active Directory
Number of pages:	54
Language:	Swedish
Date of acceptance:	16.12.2010

INNEHÅLL

Figurer

Tabeller

FÖRKORTNINGAR

ISO	*Internationella standardiseringsorganisationen*
LDAP	*Lightweight Directory Access Protocol*
XML	*eXtensible Markup Language*
UTF-8	*Unicode Transformation Format-8*
URL	*Uniform Resource Locator*
HTTP	*Hypertext Transfer Protocol*
AD	*Microsoft Active Directory*
Exchange	*Microsoft Exchange Server 2010*
IP	*Internet Protocol*
API	*Application Programming Interface*

1 INLEDNING

Vi lever i en kultur där tiden är ytterst viktig och pengar är jämställt med tid. En ineffektiv hantering av tid kostar pengar, t.ex. när tiden det tar att hitta en gemensam tid för en grupp deltagare blir ett heltidsarbete. Ju fler deltagare som ska bokas in, desto svårare är det att hitta en passande tidpunkt.

En central kalender där alla användare fyller i de tider då de är upptagna underlättar koordineringsarbetet dramatiskt då man snabbt får en överblick över läget med effektiva och smarta verktyg. För att en central kalender ska fungera måste alla inom organisationen använda sig av den, och se till att den egna kalendern är uppdaterad och alltid korrekt.

Denna rapport riktar sig främst till experter inom informationsteknik. Rapporten beskriver de designbeslut som gjorts samt hur klienter kommunicerar med systemet.

1.1 Uppdragsgivare

Arcada - Nylands svenska yrkeshögskola, belägen i Helsingfors, är uppdragsgivare för detta examensarbete. Arcada har både svensk- och engelskspråkiga utbildningsprogram för en yrkeshögskoleexamen (Arcada, 2010a). Examen kan avläggas inom utbildningsområdena *Energi och materialteknik, Hälsa och välfärd* eller *Ekonomi, informationsteknik och media* (Arcada, 2010b).

1.2 Syfte

Målet med detta examensarbete har varit att underlätta integrationen av existerande system med Microsoft Active Directory och Microsoft Exchange Server 2010. Det senare systemet är en kommersiell lösning för central lagring av bland annat elektroniska kalendrar. Integrationen är viktig eftersom organisationens rumsadministration och -bokningar görs i ett internt utvecklat system, som varit i användning sedan år 2004 efter flytten till nuvarande lokaler. Den undervisande personalens kalendrar finns i huvudsak i detta rumshanteringssystem, varför överföring

av data mellan detta system och Exchange är viktigt. En centralkalender kan aldrig fungera om inte alla personers uppdaterade kalendrar finns lagrade i systemet.

Detta arbete beskriver de viktigaste designelementen i mjukvaran och hur man kommunicerar med systemet. Utvecklingen av klientapplikationen som tar hand om överföringen av data mellan Arcadas rumshanteringssystem och Exchange ingick inte i detta examensarbete, utan fokus har helt legat på att göra en fungerande webbtjänst (eng. Web Service) som gömmer undan komplexiteten för externa system. Denna webbtjänst tillhandahåller ett skräddarsytt gränssnitt som används för hantering av data i Active Directory och Exchange.

2 VERKTYG

Inom detta arbete har många olika verktyg använts. Vissa verktyg används direkt för att utveckla applikationen och andra används sedan av applikationen för att applikationen skall kunna utföra sina uppgifter. Detta kapitel tar upp de viktigaste och mest använda verktygen som används och försöker kort förklara vad det är för något.

2.1 .NET Framework

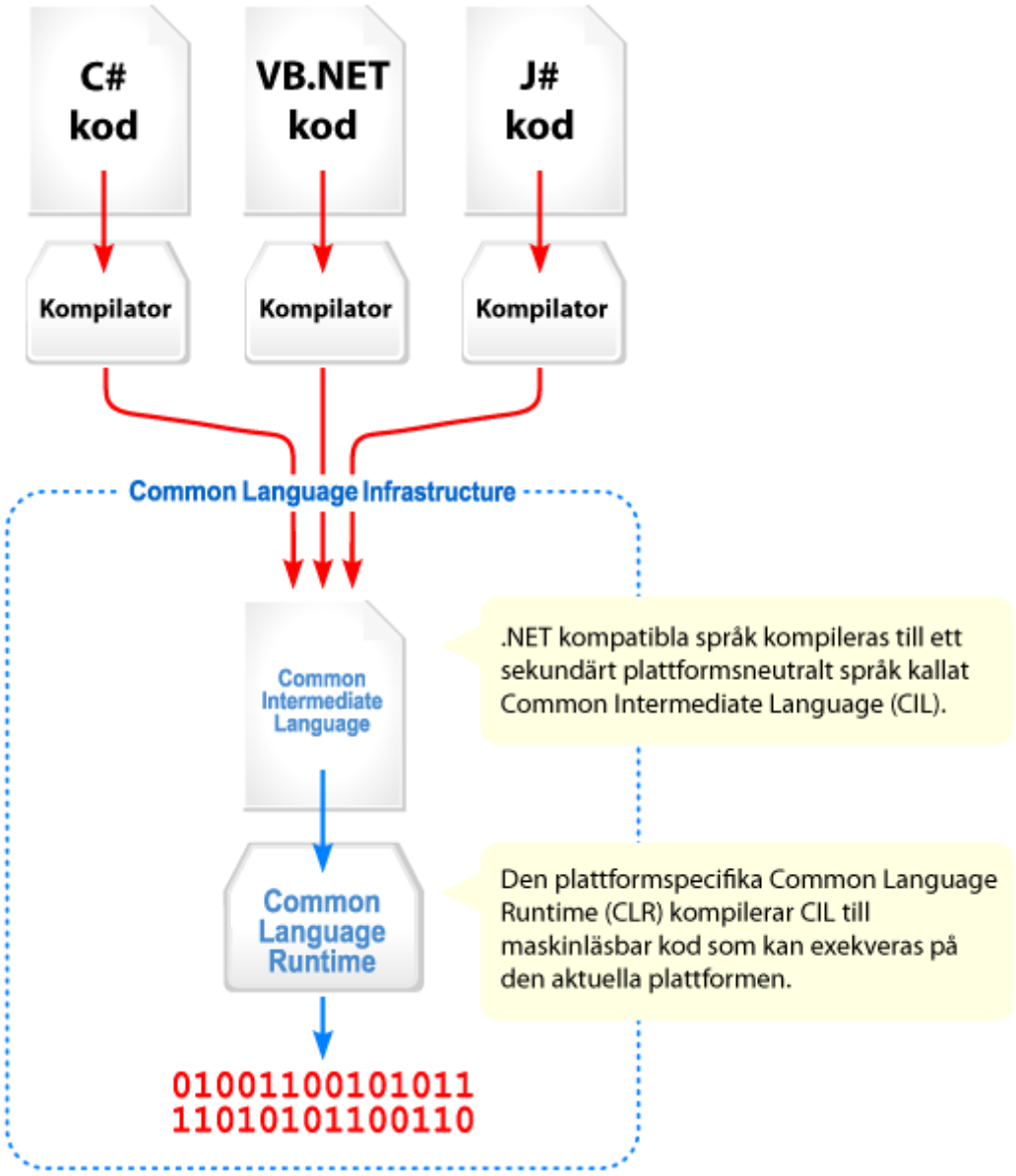
.NET Framework är ett ramverk, utvecklat av Microsoft, för utveckling av applikationer som kan användas på flertalet enheter och plattformar, t.ex. datorer, handhållna enheter och på Internet (Microsoft, 2010).

Ramverket kan installeras på datorer som kör Microsoft Windows operativsystemet, och kommer installerat på nyare versioner av Microsoft Windows. Familjen .NET Framework innehåller också två versioner riktade till mobila och inbyggda system. För enheter som kör på plattformen *Windows CE*, t.ex. *Windows Mobile*, finns *.NET Compact Framework*. Microsoft har också utvecklat *.NET Micro Framework* specifikt för enheter med begränsade resurser. (Wikipedia, 2010a)

.NET Framework är Microsofts implementering av det standardiserade *Common Language Infrastructure (CLI)*. Microsoft, tillsammans med Hewlett-Packard och Intel, arbetade i augusti 2000 för att få både CLI och programmeringsspråket C# standardiserat hos ECMA och ISO. I december 2001 godkändes båda standarderna av

ECMA, där CLI blev standardiserad under namnet *ECMA 335* och C# under namnet *ECMA 334*. Standarderna godkändes av ISO i april 2003, och aktuella ISO standarder heter *ISO/IEC 23271:2006* för CLI och *ISO/IEC 23270:2006* för C#.

Figur 1 visar en visuell översikt över Common Language Infrastructure. (Wikipedia, 2010a)



Figur 1. Visuell översikt över Common Language Infrastructure (CLI).

2.1.1 Bakgrund

Utvecklingen startade under sena 1990-talet och gick under namnet *Next Generation Windows Services*. I slutet av år 2000 publicerades första beta versionen av .NET Framework 1.0. (Wikipedia, 2010a)

2.1.2 Versioner

Tabell 1 visar alla befintliga versioner av .NET Framework till dagens datum. Version 4, som används i detta arbete, publicerades den 12 april 2010. (Wikipedia, 2010a)

Tabell 1. Lista över .NET Framework versioner.

Version	*Versionsnummer*	*Datum*
1.0	1.0.3705.0	13.2.2002
1.1	1.1.4322.573	24.4.2003
2.0	2.0.50727.42	7.11.2005
3.0	3.0.4506.30	6.11.2006
3.5	3.5.21022.8	19.11.2007
4.0	4.0.30319.1	12.4.2010

2.2 Windows Server 2008 R2

Windows Server 2008 R2 är ett operativsystem planerat för servrar och utvecklad av Microsoft. Det bygger på kärnan Windows NT 6.1, som också klientoperativsystemet Windows 7 bygger på. Förbättringar i denna version, som är den senaste, finns bland annat som ny funktionalitet i Active Directory, utgivningen av IIS 7.5 och stöd för upp till 256 logiska processorer. *Windows Server 2008 R2* är även den första versionen av Windows Server familjen som enbart finns för 64 bitars arkitektur. (Wikipedia, 2010h)

2.3 Exchange Server

Exchange Server är ett av Microsoft utvecklat system som tillhandahåller tjänster för att personer skall kunna samarbeta mer effektivt. Systemet innehåller främst tjänster för hantering av elektronisk post, kalender, kontakter och uppgifter. Systemet har även stöd för mobil- och webbaserad åtkomst till information. Den senast tillgängliga versionen är Exchange Server 2010. (Wikipedia, 2010f)

2.4 Active Directory

Active Directory är en katalogtjänst utvecklad av Microsoft som tillhandahåller en rad nätverkstjänster, bland annat LDAP, Kerberos-baserad autentisering samt en central lagringsplats för applikationsdata. I Windows Server 2008 och Windows Server 2008 R2 döptes *Active Directory* om till *Active Directory Domain Services*. *Active Directory* underlättar administrationen av en organisations användare och nätverksanslutna datorer, då inställningar kan göras på en central plats som sedan används av alla som ansluter sig till *Active Directory*. Tjänsten kan användas i både liten och stor skala, med allt från endast några få datorer, användare och skrivare anslutna mot en server till tusentals resurser anslutna mot många servrar spridda över jorden. (Wikipedia, 2010b)

2.5 Internet Information Services 7.5

Internet Information Services (IIS), tidigare under namnet *Internet Information Server*, är en webbserverapplikation med moduler som utökar funktionaliteten. *Internet Information Services* är utvecklad av Microsoft och senaste versionen som finns är version 7.5 som kommer med *Windows Server 2008 R2* och *Windows 7*. Enligt Netcraft körde, i mars 2010, 24,47% av alla webbservrar i världen på någon version av IIS, vilket gör den till den andra mest använda webbservern i världen efter Apache HTTP Server. (Wikipedia, 2010e)

För att kunna utnyttja den nya autostart-funktionen i ASP.NET 4.0 måste applikationen köra på IIS version 7.5. Autostart-funktionen innebär att IIS 7.5 automatiskt exekverar en eller flera rutiner i applikationen då IIS startar. Det finns inget grafiskt gränssnitt för att konfigurera autostart av en ASP.NET 4.0 applikation, utan manuell editering av konfigurationsfilen för IIS krävs. (Guthrie, 2009)

Den utvecklade applikationen kräver att autostart-funktionen finns och fungerar, vilket kräver IIS 7.5, och därför även Windows Server 2008 R2. Applikationen har en kö med jobb, vilken måste behandlas. Utan autostart-funktionen startas applikationen först vid första anropet till applikationen, vilket skulle göra den beroende av aktiva klienter.

2.6 ASP.NET MVC

ASP.NET MVC är ett ramverk utvecklat av Microsoft som realiserar designmönstret *model-view-controller*. Ramverket är skrivet i C# och aktuell stabil version är 2.0. Separeringen av ansvar i ramverket underlättar automatisk enhetstestning av en applikation som använder ramverket. (Wikipedia, 2010c)

2.7 Microsoft Visual Studio

Microsoft Visual Studio är en utvecklingsomgivning utvecklad av Microsoft. Omgivningen kan användas till att utveckla allt från konsolapplikationer till applikationer med grafiska användargränssnitt, t.ex. fönster- och webbapplikationer. *Visual Studio* innehåller bland annat en text editor för programkod med stöd för IntelliSense, verktyg för omstrukturering av kod samt en avlusare för kod vilken fungerar både direkt på källkoden och under körning. *Visual Studio* har stöd för många olika programmeringsspråk, bland annat *C/C++*, *C#* och *Visual Basic .NET*. Den senaste tillgängliga versionen är *Visual Studio 2010*. (Wikipedia, 2010g)

3 DESIGNMÖNSTER

Inom utveckling av programvara är designmönster (eng. *Design Patterns*) en generell lösning på ett ofta förekommande problem inom design av mjukvara. Det är inga färdigt programmerade lösningar som man direkt kan använda, utan mer en beskrivning eller mall för hur ett specifikt problem kan lösas. Designmönster kan hjälpa till att snabba upp en utvecklingsprocess då den beskriver beprövade mönster som fungerat i andra utvecklingsprojekt. Användningen kan även resultera i renare kod och göra det lättare för utvecklare att snabbare sätta sig in i befintlig kod. (Wikipedia, 2010d)

3.1 Dependency Injection

Dependency Injection, fritt översatt ”injicering av beroenden”, betyder att data ett objekt är beroende av skapas utanför objektet och ges sedan till objektet på något sätt. (Shore, 2006)

Figur 2 illustrerar med kod hur en klass själv skapar sina data den är beroende av och Figur 3 visar hur dessa data kan tillhandahållas utifrån med *Dependency Injection*.

```
public class Exempel
{
    private Dataobjekt minData;

    public Exempel()
    {
        minData = new Dataobjekt();
    }

    public void Metod()
    {
        minData.HamtaData();
    }
}

public class Dataobjekt
{
    public virtual void HamtaData()
    {
        // Hämtar någon form av data
    }
}
```

Figur 2. Dependency Injection kodexempel där klassen själv skapar data den är beroende av.

```
public class Exempel
{
    private Dataobjekt minData;

    public Exempel(Dataobjekt anvandDenna)
    {
        minData = anvandDenna;
    }

    public void Metod()
    {
        minData.HamtaData();
    }
}

public class Dataobjekt
{
    public virtual void HamtaData()
    {
        // Hämtar någon form av data
    }
}

public class TestDataobjekt : Dataobjekt
{
    public override void HamtaData()
    {
        // Hämta inte data utan gör något testbart
    }
}
```

Figur 3. Dependency Injection kodexempel där klassen får data den är beroende av utifrån.

Hur data ges åt klassen spelar ingen roll, bara det kommer utifrån. Data kan ges direkt i klassens konstruktor eller senare genom att sätta den genom publika metoder eller egenskaper. (Shore, 2006)

Dependency Injection används ofta i samband med *Service Stub* för att göra koden testbar.

3.2 Domain Model

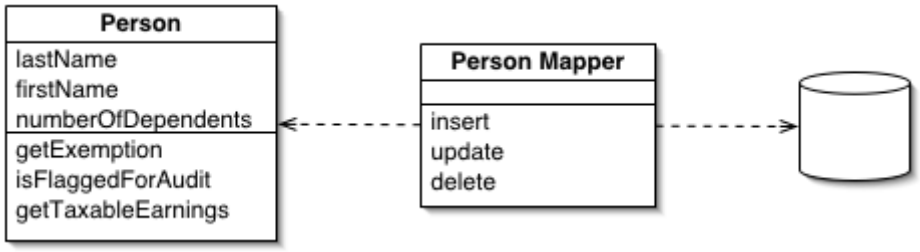
Domain Model är objekt som innehåller både logik och data. En objektorienterad *Domain Model* liknar ofta databasmodellen, men har ändå stora skillnader. En *Domain Model* kan t.ex. innehålla både data och logik, komplexa relationer mellan varandra och arv. (Fowler, 2002 s. 116)

Det är svårt att svara på när man ska använda sig av *Domain Model*, men är logiken i systemet komplicerad med regler och datakontroller tjänar man på att inkapsla denna

komplexitet i objekt. Sen beror det även på hur inkomna utvecklarna är att arbeta med *Domain Model*. De som är inkomna med *Domain Model* vill ofta inte använda något annat för mer än väldigt enkla fall. (Fowler, 2002 s. 119)

3.3 Data Mapper

Data Mapper är ett lager av mjukvara vars uppgift är att överföra data mellan objekt och databasen. Objekt och databas är ovetande om både varandra och *Data Mapper*-lagret. Figur 4 illustrerar hur en *Data Mapper* ser ut. (Fowler, 2002 s. 165)



Figur 4. Illustration över en Data Mapper (Fowler, 2002 s. 165).

En *Data Mapper* innehåller utöver metoder för att lagra data även metoder för att läsa data (Fowler, 2002 s. 168-169). Fowler benämner dessa metoder som *Finders*, troligtvis då han döper metoderna till *Find* med ett suffix som beskriver hur data ska hittas, t.ex. *FindById*.

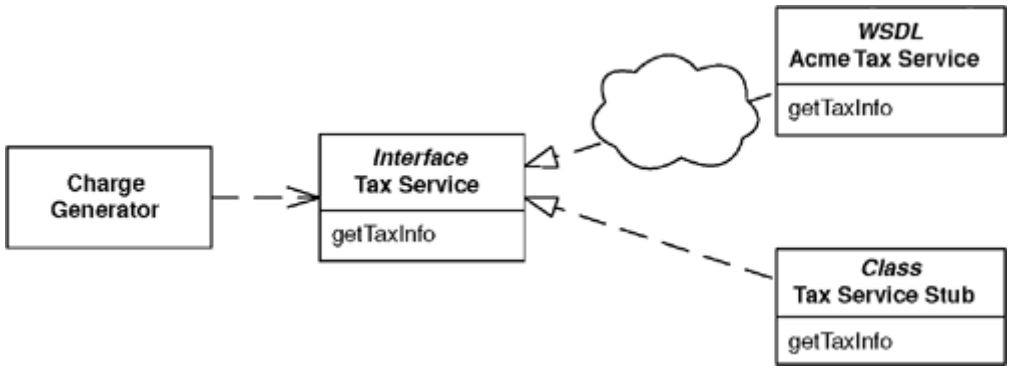
Främsta anledningen till att använda sig av en *Data Mapper* är då man vill tillåta att datalagret och dataobjekten utvecklas oberoende av varandra. *Data Mapper* passar utmärkt ihop med *Domain Model*, då isoleringen mellan datalagret och dataobjekten gör att man kan koncentrera sig på dataobjekten och deras logik. Det är även möjligt att utveckla en applikation utan att ha tillgång till det riktiga datalagret, med hjälp av *Dependency Injection* och *Service Stub*. (Fowler, 2002 s. 170)

3.4 Service Stub

System är ofta beroende av data utifrån, ofta en databas eller tredjepartssystem. Åtkomsten till dessa system kan vara långsam och data kan i många fall ändra. Att förlita sig på extern data kan i många fall göra utveckling och testning långsam. *Service Stub* avlägsnar beroendet av dessa problematiska tjänster. Användning av en *Service*

Stub, som också ofta kallas *Mock Object*, rekommenderas då ett beroende av ett annat system hindrar utveckling eller testning. (Fowler, 2002 s. 504-505)

Figur 5 visar hur en *Service Stub* fungerar. Figuren visar ett gränssnitt vid namn *Tax Service* och två klasser som realiserar detta gränssnitt, där *Acme Tax Service*-klassen arbetar över nätverket och *Tax Service Stub*-klassen är en lokal *Service Stub* man kan använda vid testning och utveckling. Denna arkitektur gör att utvecklingen inte är beroende av det externa systemet.



Figur 5. Illustration över hur Service Stub fungerar (Fowler, 2002 s. 504).

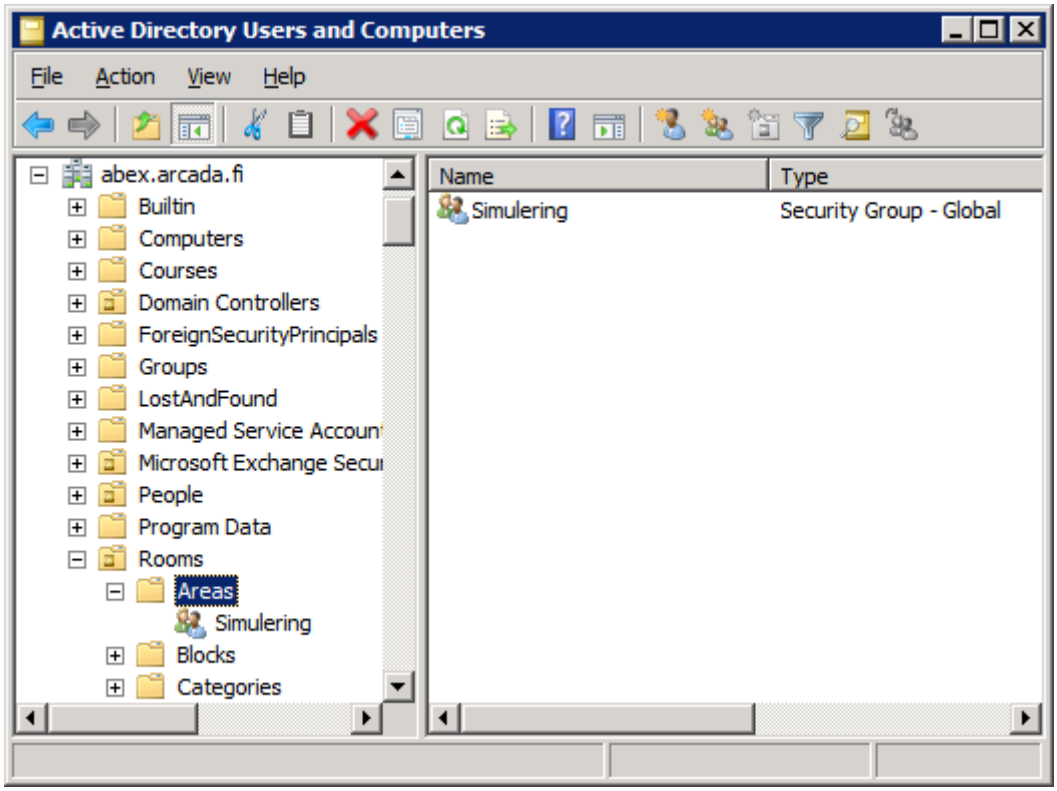
4 ARKITEKTUR

Arkitekturen består av tre olika moduler: Kärnan, ett XML-gränssnitt och en ”*ASP.NET MVC 2*”-webbapplikation som tillhandahåller åtkomst till XML-gränssnittet över HTTP för klienter. Detta kapitel går inte igenom hela arkitekturen på djupet, utan koncentrerar sig på de relevanta bitarna över vad som händer då en klient kommunicerar med systemet. Fokus ligger på realiseringen av kommunikationsprotokollet, där det visas hur ett inkommande XML-dokument blir ett dataobjekt i minnet. Detta objekt går sedan neråt i hierarkin tills det når datalagret.

4.1 Kärnan

Kärnan är ett klassbibliotek skrivet i C# för .NET Framework 4.0. Biblioteket innehåller klasser för att underlätta hanteringen av data i AD och Exchange. Tre olika sätt används för att läsa och skriva data till AD och Exchange: direkt till *AD*, med *PowerShell* och med *Exchange Web Services*. Kärnan gömmer undan denna komplexitet och tillhandahåller enklare gränssnitt att arbeta med.

Med biblioteket kan man hantera rum, personer, grupper och bokningar. Biblioteket tar även hand om databehållare, vilket kan liknas vid en katalog på filsystemet och används för att strukturera data. Rum, personer och grupper är objekt som kan placeras i en databehållare. Varje databehållare kan även placeras i en annan databehållare för att skapa en trädstruktur. Figur 6 visar databehållare i en trädstruktur på vänster sida av figuren. Databehållaren *Areas* är markerad i figuren och innehåller gruppen *Simulering*. *Areas* är placerad i databehållaren *Rooms*, som är placerad i roten på domänen *abex.arcada.fi*.



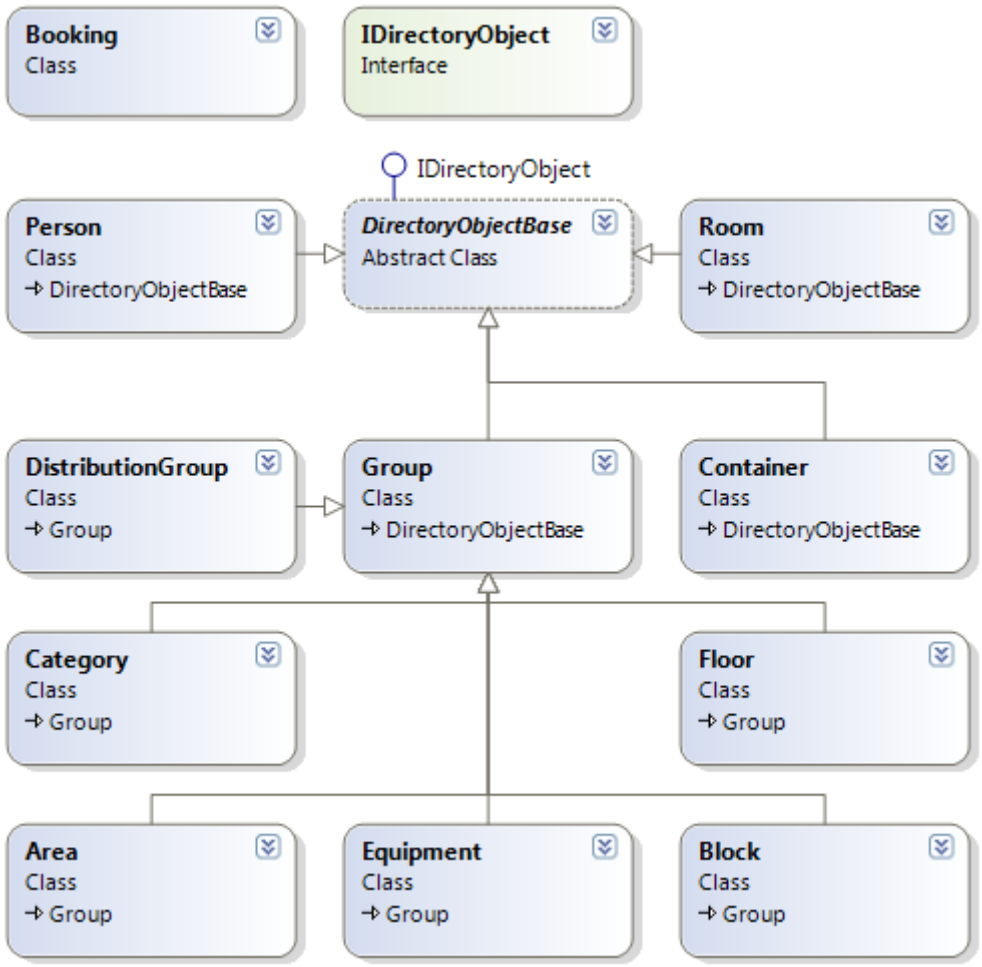
Figur 6. Databehållare i Active Directory.

Tekniskt sett är databehållaren *Rooms* inte identisk med databehållaren *Areas*, vilket indikeras av den lite annorlunda ikonen i Figur 6. *Rooms* är det som i AD kallas *”Organizational Unit”* och *Areas* kallas *Container*. Biblioteket hanterar dessa som en och samma. Om en databehållare inte existerar skapas den av biblioteket som en AD *Container*-typ.

Detta kapitel tar inte upp hela kärnans arkitektur, utan fokus ligger på att visa hur olika designmönster använts i kärnan.

4.1.1 Dataobjekt

I kärnan finns det fem olika huvudobjekt: *Person*, *Room*, *Group*, *Container* och *Booking*. Dessa objekt representerar: en person, ett rum, en grupp, en databehållare och en bokning. Alla dataobjekt utom *Booking* ärver från den abstrakta klassen *DirectoryObjectBase*, vilken i sin tur realiserar *IDirectoryObject*-gränssnittet. Figur 7 visar hela trädet med dataobjekt och hur de relaterar till varandra.



Figur 7. Dataklasserna i kärnan.

Group-objektet är det enda som har andra objekt som ärver från den. Objekten *Category*, *Floor*, *Area*, *Equipment* och *Block* används enbart av *Room*-objektet för att beskriva vilka egenskaper rummet har.

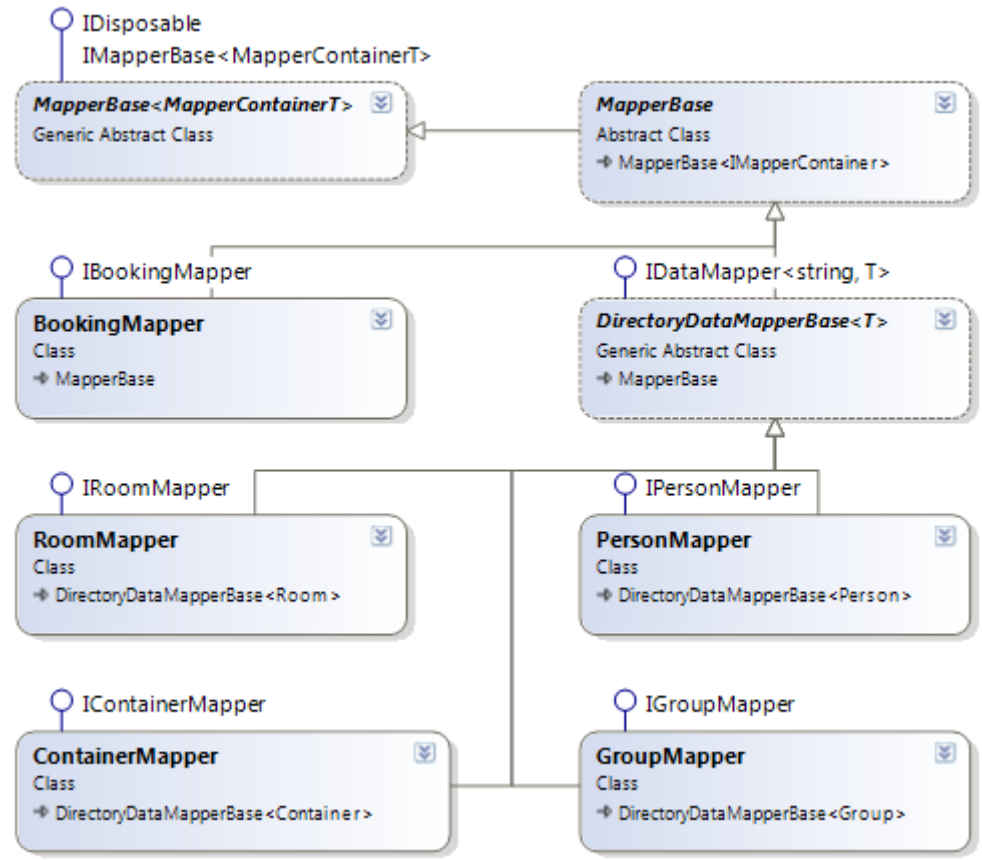
DistributionGroup ärver från *Group* och används lite annorlunda av mapper-klassen för grupper, se *Mapper-klasser* för mer information om dessa. En *DistributionGroup* är identisk med en vanlig grupp, men skiljer sig så att gruppen även är synlig i Exchange

och kan användas t.ex. i bokningar som deltagare. Tekniskt sett kan alla klasser som ärver från *DirectoryObjectBase*-klassen vara deltagare på en bokning, men praktiskt sett är det bara de objekt som syns i Exchange som kan användas.

Designmönstret *Domain Model* har använts för att hantera den logik som krävs i kärnan. Vissa objekt har relationer till andra objekt, men alla objekt är helt ovetande om datalagret och mapper-klasserna, vilket en *Domain Model* bör vara.

4.1.2 Mapper-klasser

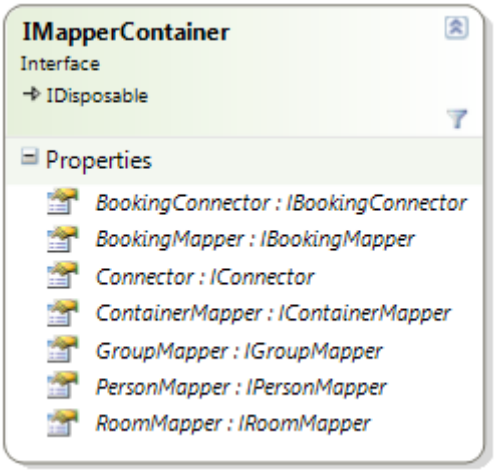
Kärnan använder sig av designmönstret *Data Mapper* för att förmedla data mellan datalagret och datamodellen. Datamodellen består av en klass per dataobjekt: *Person*, *Room*, *Group* och *Booking*. Varje dataklass har en tillhörande *Data Mapper*-klass som tillhandahåller funktioner för att spara, radera och söka efter data i de underliggande datalagren. Figur 8 visar alla mapper-klasser som finns i kärnan: *BookingMapper*, *RoomMapper*, *PersonMapper*, *ContainerMapper* och *GroupMapper*.



Figur 8. Data Mapper-klasser i kärnan.

Varje mapper-klass har metoder för att spara och radera data. För att hämta data används olika *Find* metoder, vilka i de flesta fall tar emot ett eller flera argument för att filtrera ned resultatet till enbart de objekt som önskas. Alla mapper-klasser har en *FindById* metod, vilken tar emot en sträng och returnerar ett objekt eller *null* (ingenting), beroende på objektet kunde hittas eller inte. Figur 8 visar också att alla mapper-klasserna implementerar varsitt gränssnitt, som beskriver vilka metoder och egenskaper den implementerande mapper-klassen måste ha.

Figur 9 visar *IMapperContainer*-gränssnittet som används för att konfigurera kärnan. En realiserande klass måste se till att tillhandahålla de egenskaper som figuren visar. Det är helt upp till klienten av kärnan att bestämma vilken version av en specifik mapper-klass som ska användas. Detta kallas för *Dependency Injection*, vilket innebär att den data en klass är beroende av tillhandahålls utifrån. Systemet blir flexibelt eftersom en del av systemet kan bytas ut mot en annan utan ändring eller åtkomst till kärnans källkod.



Figur 9. IMapperContainer-gränssnittet som används av kärnan.

Varje mapper-klass är även planerad så att klasserna kan testas utan ett bakomliggande datalager. Klassernas konstruktor tar emot ett objekt som realiserar *IMapperContainer*, som innehåller egenskaperna *BookingConnector* och *Connector*, vilket Figur 9 visar. Gränssnitten *IConnector* och *IBookingConnector* används av mapper-klasserna för att läsa data från och skriva data till det underliggande datalagret.

Denna arkitektur tillåter automatisk enhetstestning av mapper-klasserna då connector-klasserna kan bytas ut mot enkla objekt som sparar och läser data från minnet eller filsystemet. Dessa enkla objekt följer designmönstret *Service Stub*.

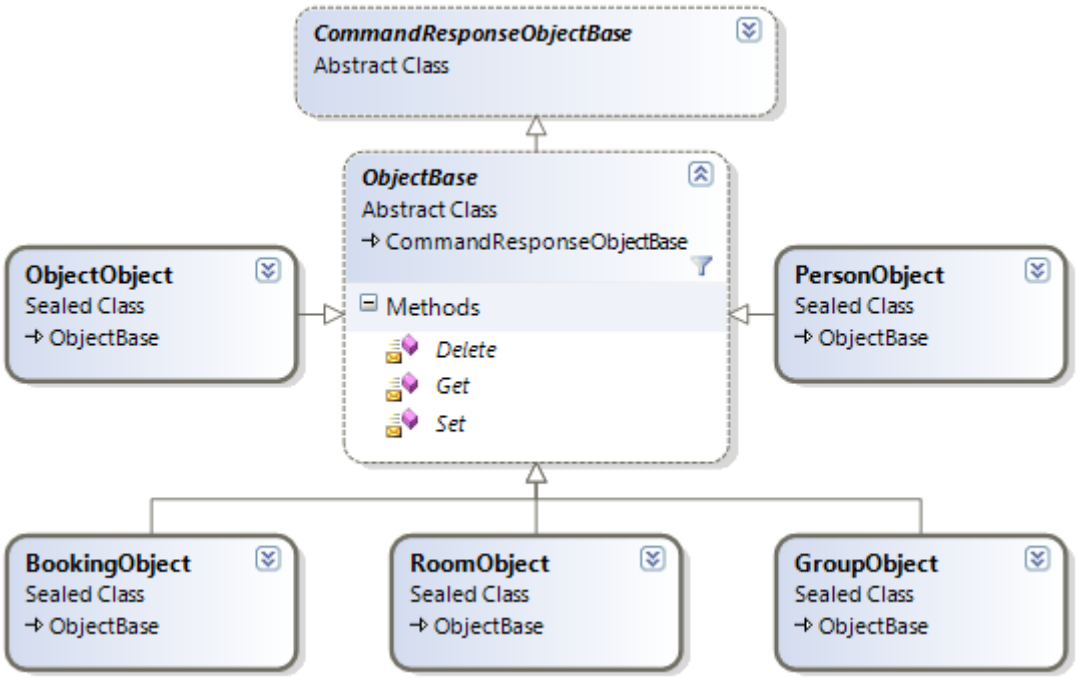
Vid testning är det även viktigt att varje test alltid utgår från samma data, vilket är möjligt om det i testet går att specificera den data som skall finnas före och efter ett test körts. Mapper-klassernas enhetstester använder detta tillvägagångssätt, där *IMapperContainer*-gränssnittets *Connector*-egenskap är en enkel *Service Stub* som lagrar och läser all data till och från minnet.

4.2 XML

En klient kan inte kommunicera direkt med kärnan över ett nätverk, därför behövs det ett lager med mjukvara ovanpå kärnan som översätter data från ett format en klient kan använda till det format som används i kärnan. Det är här XML-lagret kommer in, som är ett klassbibliotek skrivet i C# som översätter XML-data till och från det format som kärnan använder. XML-lagret innehåller även egen logik som inte erbjuds från kärnan. Detta lager har hand om att direkt erbjuda ett enkelt och bra gränssnitt mot klienterna.

4.2.1 Objekt

Data förs över med hjälp av dataobjekt. Varje dataobjekt känner enbart till egen data och hur den ska behandlas. För att stöda de olika kommandona som arbetar med data (se *Kommandon*), finns de tre metoderna *Set*, *Get* och *Delete* i alla dataobjekt. Figur 10 visar dessa metoder i den abstrakta basklassen *ObjectBase*, som ärver från *CommandResponseObjectBase* (se *Svar på kommandon* för en närmare beskrivning).



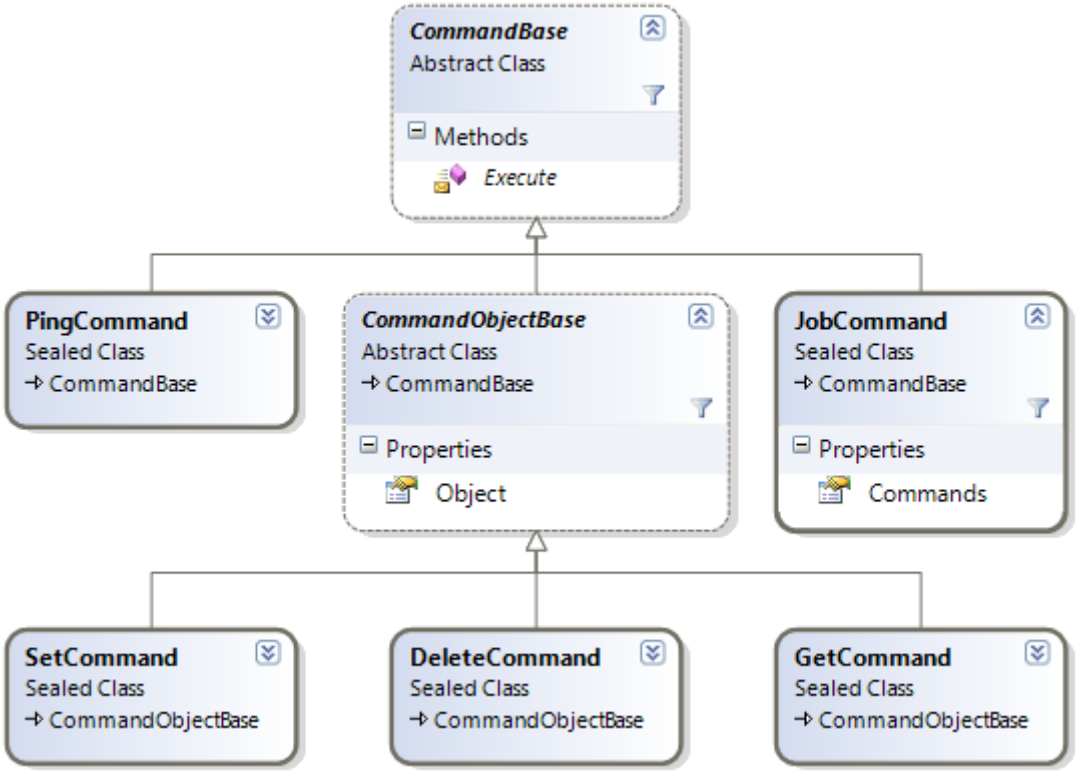
Figur 10. Objekt-klasserna i XML-protokollet.

Som Figur 10 visar finns dataobjekten *PersonObject*, *BookingObject*, *RoomObject*, *GroupObject* och *ObjectObject*. *ObjectObject*-objektet är ett generellt objekt som kan ersätta objekten *PersonObject*, *RoomObject* och *GroupObject* då kommandon som enbart behöver kunna identifiera ett objekt anropas.

Objekten kan användas både vid en förfrågan och också svar. Då en klient vill skriva ett objekt skickas XML-representationen för objektet med i förfrågan. Då en klient begär att få läsa ett befintligt objekt skickar servern hela objektet som svar. Se *Dataobjekt* för en beskrivning hur objekten ser ut i XML-format.

4.2.2 Kommandon

XML-lagret innehåller olika kommandon som klienten kan använda sig av, och som gör varsin specifik sak. Figur 11 visar alla kommandon som finns med de viktigaste egenskaperna och metoderna.



Figur 11. Kommandoklasserna i XML-protokollet.

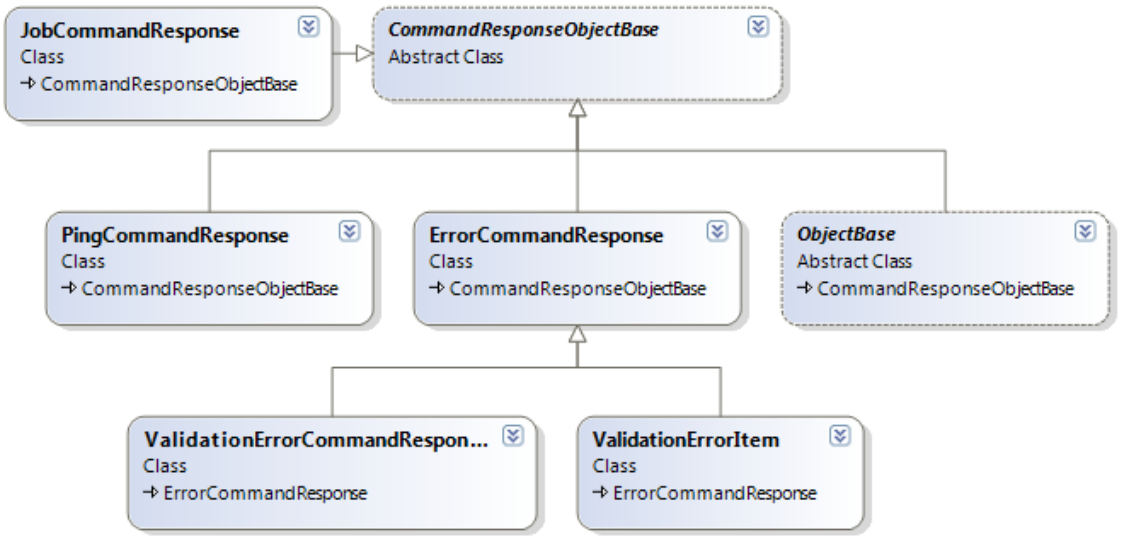
Alla kommando-klasser ärver direkt eller indirekt från den abstrakta *CommandBase*-klassen. *CommandBase*-klassen definierar en abstrakt *Execute* metod, där kommandots logik exekveras och returnerar ett resultat, som går tillbaka till den anropande klienten.

Kommandona *SetCommand*, *GetCommand* och *DeleteCommand* ärver från den abstrakta *CommandObjectBase*-klassen. *CommandObjectBase*-klassen definierar en *Object*-egenskap, vilket innebär att alla kommandon som ärver från klassen kan innehålla ett dataobjekt, se Objekt för en beskrivning av dessa.

JobCommand-klassen tillhandahåller en tjänst som inte direkt har något att göra med kärnan. Dess uppgift är att ta emot en lista med kommandon som ärver från *CommandObjectBase*-klassen och lagra dem. De lagrade kommandona körs sedan i bakgrunden. *Commands*-egenskapen i *JobCommand*-klassen innehåller de kommandon som ska köras. Om klienten önskat bli delgiven resultatet av körningen skickas svaret i XML-format till en *url* som klienten specificerat.

4.2.3 Svar på kommandon

Alla kommandon returnerar ett svar, vilket kan vara allt från en enkel bekräftelse på att kommandot kört till en lista med fel som upptäcktes och hindrade körningen av kommandot. Figur 12 visar alla svarsklasserna och hur de relaterar till varandra. *ObjectBase*-klassen är en abstrakt klass som alla dataobjekt ärver från, vilket innebär att alla dataobjekt också kan agera returdata från ett kommando, se *Objekt* för en beskrivning över dessa.



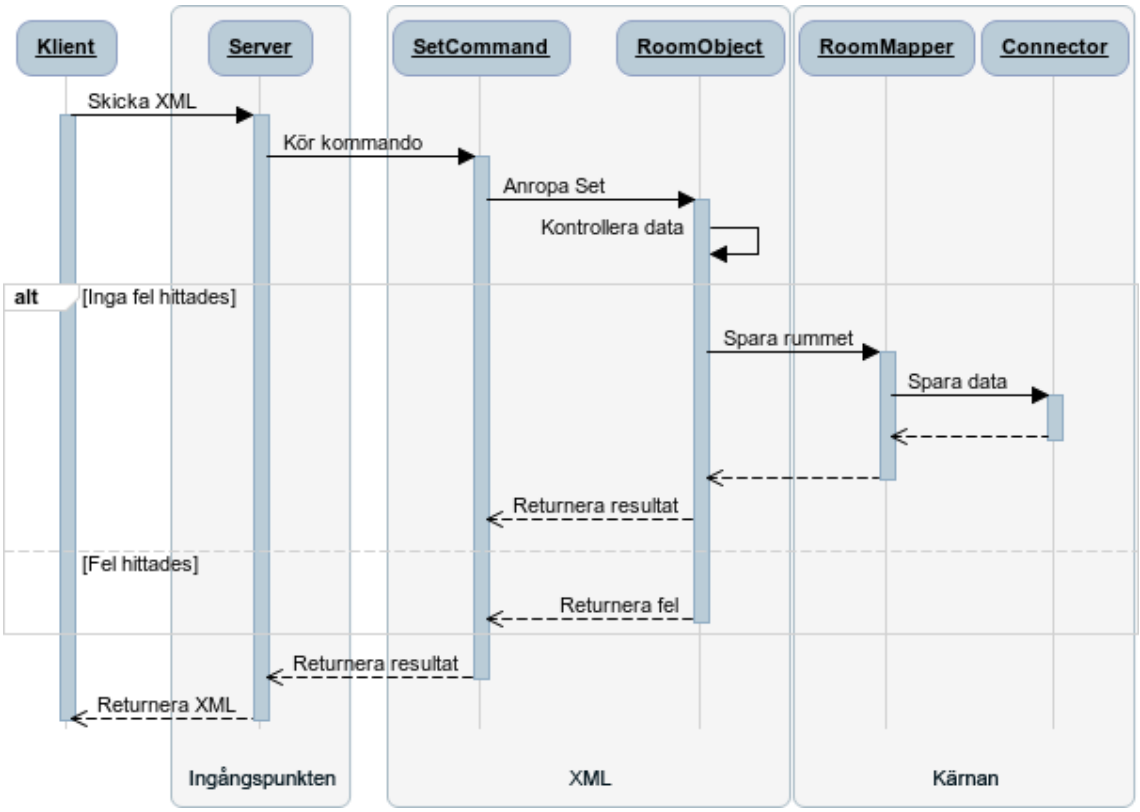
Figur 12. Svarsklasser i XML-klassbiblioteket.

4.3 Ingångspunkten

För att en klient ska få tillgång till XML-klassbiblioteket måste det finnas en ingångspunkt. Det är möjligt att tillhandahålla flera olika ingångspunkter via vilka klienter kan ta kontakt. HTTP-standarden är en globalt använd standard för kommunikation på Internet. Eftersom det är ett enkelt protokoll som är lätt att använda, oavsett plattform och utvecklingsspråk, har detta arbete fokuserat på den standarden som ingångspunkt.

Tekniskt sett är ingångspunkten en tjänst som använder sig av ramverket *ASP.NET MVC*. Tjänsten tillhandahåller en URL dit klienten skickar XML-dokument med POST-metoden i HTTP-standarden. Tjänsten avkodar XML-dokumentet från textrepresentation till instanser av de klasser som finns i XML-biblioteket. Är XML-dokumentet korrekt formaterat körs de kommandon som klienten skickat och resultatet

av körningen returneras till klienten. Figur 13 visar hur ett *Set*-kommando innehållande ett *RoomObject*-objekt går igenom de olika lagren i systemet, ända från klienten till kärnan och tillbaka igen. Hittas det fel i data når körningen aldrig kärnan utan felet går tillbaka till kommandot och därifrån vidare till klienten. Hittas däremot inga fel skickas ett *Room*-objekt till kärnan som sparar ned data, varefter *RoomObject*-objektet returnerar ett nytt *RoomObject*-objekt, innehållande rummets identifikation, tillbaka till *Set*-kommandot. Från kommandot går det direkt vidare ut till anroparen. Illustrationen är förenklad där klasser och funktionsanrop har lämnats bort för att göra flödet överskådligt.



Figur 13. Översikt av hur ett rum sätts från en klient och av flödet igenom de olika lagren i systemet.

Finns det flera kommandon kommer de att köras efter varandra tills allt har körts, varefter svaret skickas tillbaka till klienten. Ingångspunktens ansvar är att konvertera indata till objekt i XML-lagret. XML-lagret kontrollerar data, konverterar data till objekt som kärnan kan arbeta med och skickar vidare anropet dit. Slutligen går returdata tillbaka till klienten.

5 DATAÖVERFÖRINGSPROTOKOLLET

Protokollet som används för kommunikation mellan klienter och servern är uppbyggt med märkspråket XML. Språket valdes på grund av dess oberoende av utvecklingsspråk och utvecklingsmiljö, vilket innebär att man kan utveckla klienterna på vilken plattform som helst. Teckenkodningen som används är UTF-8. XML är skiftlägeskänsligt, vilket innebär att alla element- och attributnamn måste skrivas exakt som det framgår i detta dokument.

5.1 XML-dokumentet

Protokollets toppelement heter *abex* och finns alltid med i både förfrågan och svar. Elementet ser alltid likadant ut, se Figur 14. De tre punkterna indikerar var data placeras.

```
<?xml version="1.0" encoding="utf-8"?>
<abex xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
</abex>
```

Figur 14. XML-dokumentet som används i förfrågan och svar.

Varje dokument har alltid två barnelement, *Dokumenthuvud* och *Databehållare*. Se även *Bilaga 1* för ett helt dokument, både som förfrågan och svar.

5.2 Dokumenthuvud

Dokumenthuvudet finns i protokollet för att överföra data som inte direkt hör till datadelen, utan är tänkt att innehålla styrdata för protokollet. I aktuell version finns inte autentisering och auktorisering specificerat i protokollet, men det är tänkt att dessa delar ska överföras i dokumenthuvudet i framtiden. En förfrågan har således ingenting att överföra i dokumenthuvudet i den aktuella versionen. Ett svar kan däremot utnyttja huvudelementet till att returnera data om ett allmänt fel som berör hela XML-dokumentet. Felet kan till exempel vara att dokumentet av någon anledning inte kunde tolkas. Figur 15 visar dokumenthuvudet vid en förfrågan och svar utan ett dokumentövergripande fel. Figur 16 visar hur ett svar kan se ut då ett tolkningsfel uppstått, där felet representeras av ett *Error*-objekt.

```
<Head />
```

Figur 15. Dokumenthuvud använd i förfrågan och svar utan tolkningsfel.

```
<Head>
  <Error code="500">Ett fel har uppstått.</Error>
</Head>
```

Figur 16. Dokumenthuvud innehållande tolkningsfel.

5.3 Databehållare

Databehållaren är ett element vars enda uppgift är att innehålla den data som ska överföras. Elementet används i både förfrågningar och svar. I sin enklaste form är databehållaren tom, vilket Figur 17 visar.

```
<Body />
```

Figur 17. Tom databehållare.

5.4 Kommandon

Hela protokollet bygger på att kommandon förs över och exekveras på servern. Det finns fyra olika kommandon man kan använda sig av: *Set*, *Get*, *Del* och *Job*. Varje kommando är formaterat som exemplet i Figur 18 visar. Figur 19 visar det svar som alla kommandon alltid returnerar. Punkterna visar var datadelen är, se *Dataobjekt* för en beskrivning av dessa.

```
<Set id="abc">
...
</Set>
```

Figur 18. Grundstrukturen för alla kommandon som används.

```
<Response id="abc" command="Set">
...
</Response>
```

Figur 19. Svaret som returneras för varje kommando.

Returdata innehåller alltid samma id som det exekverade kommandot. Figur 18 visar att id *”abc”* skickats för kommandot *Set*, och Figur 19 innehåller samma id på *Response-*

elementet. *Response*-elementet innehåller också attributet *command* som anger vilket kommando som genererat svaret.

Id-attributet gör det möjligt för klienten att koppla ihop ett mottaget svar med ett skickat kommando. Attributet är frivilligt, men användning av det rekommenderas. Attributet bör vara unikt för klienten under kommandots exekvering tills svaret mottagits.

Alla kommandon omges av elementet *Commands*. *Commands* är det element som sätts in i databehållaren, se *Databehållare*. Figur 20 visar hur ett *Commands*-element ser ut, de tre punkterna indikerar var kommandona sätts in.

```
<Commands>
...
</Commands>
```

Figur 20. Elementet som omger alla kommandon.

Commands-elementets motsvarighet i svaret heter *Responses*. *Responses*-elementet sätts in i databehållaren, precis som *Commands*-elementet. Figur 21 visar hur ett *Responses*-element ser ut, där de tre punkterna indikerar var alla svaren på kommandona sätts in.

```
<Responses>
...
</Responses>
```

Figur 21. Elementet som omger alla svar.

5.4.1 Set

Kommandot *Set* används för att skapa eller uppdatera data och kan användas tillsammans med objekten *Booking*, *Room* och *Group*. *Person*-objektet stöds också, men en person skapas enbart om den inte redan existerar.

Figur 22 visar hur kommandot formateras och Tabell 2 visar alla attribut som används i *Set*-kommandot.

```
<Set id="abc">
...
</Set>
```

Figur 22. Kommandot Set som det ser ut i sin fulla form.

Tabell 2. Attribut som används i kommandot Set.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
id	Data som returneras tillbaka i svaret.	text	nej

5.4.2 Get

Kommandot *Get* används för att hämta data och kan användas tillsammans med objekten *Booking*, *Room*, *Group* och *Person*. Figur 23 visar hur kommandot formateras och Tabell 3 visar alla attribut som används i *Get*-kommandot.

```
<Get id="abc">
...
</Get>
```

Figur 23. Kommandot Get som det ser ut i sin fulla form.

Tabell 3. Attribut som används i kommandot Get.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
id	Data som returneras tillbaka i svaret.	text	nej

5.4.3 Del

Kommandot *Del* används för att radera data och kan användas tillsammans med objekten *Booking*, *Room* och *Group*. Figur 24 visar hur kommandot formateras och Tabell 4 visar alla attribut som används i *Del*-kommandot.

```
<Del id="abc">
...
</Del>
```

Figur 24. Kommandot Del som det ser ut i sin fulla form.

Tabell 4. Attribut som används i kommandot Del.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
id	Data som returneras tillbaka i svaret.	text	nej

5.4.4 Ping

Kommandot *Ping* används för att kontrollera om servern svarar och kan ta emot och bearbeta XML-dokument. *Ping*-kommandot kan, som alla andra kommandon, ha ett *id*-attribut. Den kan däremot inte innehålla andra data, t.ex. ett objekt. Figur 25 visar hur kommandot formateras och Tabell 5 visar alla attribut som används i *Ping*-kommandot.

```
<Ping id="abc" />
```

Figur 25. Kommandot Ping som det ser ut i sin fulla form.

Tabell 5. Attribut som används i kommandot Ping.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
id	Data som returneras tillbaka i svaret.	text	nej

Figur 26 visar svaret på *Ping*-kommandot i Figur 25.

```
<Response id="abc" command="Ping">
  <Ping />
</Response>
```

Figur 26. Svaret på ett Ping-kommando.

5.4.5 Job

Kommandot *Job* används för att skapa ett jobb på servern. Ett jobb är ett eller flera kommandon som exekveras utan en aktiv anslutning med en klient. Klienten kan med *Job*-kommandot skicka ett eller flera kommandon som sedan exekveras asynkront och rapporteras tillbaka till klienten på en angiven *url*-adress. Har inte en *url*-adress angetts körs jobbet men rapporteras inte tillbaka till klienten.

När *Job*-kommandot används kontrollerar servern alla kommandon som jobbet innehåller och returnerar resultatet av kontrollen till klienten. Går någon kontroll inte igenom sparas och exekveras inte jobbet på servern, och klienten måste då korrigera de fel som hittades och försöka på nytt. Kontrollen som görs är enbart en kontroll på att data som behövs för att exekvera kommandot finns i kommandonas dataobjekt.

Job-kommandot kan enbart innehålla kommandona *Set*, *Get* och *Del*. Figur 27 visar hur kommandot formateras och Tabell 6 visar alla attribut som används i *Job*-kommandot.

```
<Job id="abc" url="http://..." username="uid" password="lösenord">
  <Set id="set id">
  ...
  </Set>
  <Get id="get id">
  ...
  </Get>
  <Del id="del id">
  ...
  </Del>
</Job>
```

Figur 27. Kommandot Job med exempel på alla tre kommandon den kan innehålla.

Tabell 6. Attribut som används i Job kommandot.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
id	Data som returneras tillbaka i svaret.	text	nej
url	URL dit resultatet av körningen skickas genom en HTTP-POST.	text	nej
username	Användarnamnet som behöver anges för att komma åt url ovan.	text	nej
password	Lösenordet som används tillsammans med användarnamnet för att komma åt url ovan.	text	nej

Figur 28 visar ett exempel på svaret för ett *Set*-kommando med ett rum då datakontrollen gick igenom och Figur 29 visar samma men med ett fel.

```
<Response id="job id" command="Job">
  <Responses>
    <Response id="set id" command="Set">
      <Room />
    </Response>
  </Responses>
</Response>
```

Figur 28. Svar utan fel på ett Job-kommando med ett Set-kommando och Room-objekt.

```
<Response id="job id" command="Job">
  <Responses>
    <Response id="set id" command="Set">
      <Error ... />
    </Response>
  </Responses>
</Response>
```

Figur 29. Svar med fel på ett Job-kommando med ett Set-kommando.

5.5 Dataobjekt

Protokollet innehåller fyra olika dataobjekt samt ett generellt dataobjekt som kan användas i enskilda fall. Dataobjektens uppgift är att representera den data som skrivs, läses eller raderas från servern. Nedan presenteras hur dataobjekten ser ut och hur de används i de olika kommandona beskrivna i avsnittet *Kommandon*.

Det finns tre olika versioner av dataobjekt: *tom*, *kort* och *lång*. Den tomma versionen används enbart i svaret från *Job*-kommandot för att indikera att datakontrollen av objektet gick igenom utan fel. Figur 30 visar ett tomt dataobjekt, där enbart elementnamnet varierar för de olika dataobjekten: *Object*, *Booking*, *Room*, *Group* och *Person*. Den korta versionen innehåller alltid data som kan identifiera ett objekt unikt och används av servern som svar på kommandona *Set* och *Del*. Klienten kan välja att använda den korta versionen för kommandona *Get* och *Del*. Den långa versionen är en utökning av den korta versionens data. Servern använder den långa versionen för att skicka ett helt objekt som svar på kommandot *Get*. Klienten kan alltid använda den långa versionen i alla kommandon, men det är bara *Set*-kommandot som kräver det.

```
<Object />
```

Figur 30. Tomma versionen av alla dataobjekt.

5.5.1 Object

Object är ett generellt dataobjekt som kan användas i kommandona *Get* och *Del*, förutsatt att det objekt som representeras har stöd för att köra det aktuella kommandot. *Object*-elementet i XML-dokumentet formateras enligt Figur 31. Tabell 7 visar en lista över alla attribut som objektet kan ha. *Object* kan ersätta objekten *Room*, *Group* och *Person*.

Användning av objektet bör undvikas då de mer specialiserade objekten *Room*, *Group* och *Person* är tydligare då de samtidigt anger vilken typ av objekt man arbetar med.

```
<Object id="abc" />
```

Figur 31. Objektet Object som det ser ut i sin korta och långa version.

Tabell 7. Object-objektets attribut i kort och lång version.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
id	Serverns id för objektet.	text	ja

5.5.2 Booking

Booking-objektet representerar en bokning. En bokning kan skrivas, läsas och raderas med kommandona *Set*, *Get* och *Del*.

För att skapa en ny bokning lämnar man bort *id*-attributet. För att uppdatera en befintlig bokning måste *id*-attributet anges. Figur 32 visar hur en bokning skapas och Figur 33 visar hur en bokning uppdateras.

```
<Set id="abc">
  <Booking organizer="uid" ... />
</Set>
```

Figur 32. Skapa en ny bokning.

```
<Set id="abc">
  <Booking organizer="uid" id="bokningens id" ... />
</Set>
```

Figur 33. Uppdatera en befintlig bokning.

I både Figur 32 och Figur 33 finns *organizer*-attributet. Attributet är obligatoriskt och anger i vilken persons kalender bokningen ska skapas eller uppdateras. En bokning ägs alltid av en person. Vill man byta ägare på en bokning anges bokningens id och den nya ägaren i *organizer*-attributet. Systemet kommer då att radera den gamla bokningen och returnera den nya bokningens id till klienten. Den person som *organizer*-attributet anger måste existera i systemet. Figur 34 visar svaret från servern på kommandona i Figur 32 och Figur 33.

```
<Response id="abc" command="Set">
  <Booking organizer="uid" id="bokningens id" />
</Response>
```

Figur 34. Serverns svar på skapa och uppdatera bokning.

Tabell 8 visar alla attribut som kan användas på ett *Booking*-objekt i både kort och lång version. Attributen *start* och *end* är obligatoriska i den långa versionen då den används för att skriva en bokning, antingen genom att skapa en ny eller uppdatera en befintlig.

Tabell 8. Booking-objektets attribut i kort och lång version.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
organizer	Ägaren till bokningen.	text	ja
id	Serverns id för bokningen.	text	nej
title	Bokningens titel.	text	nej
description	Bokningens beskrivning.	text	nej
start	Datum och tid då bokningen startar.	datetime	lång version
end	Datum och tid då bokningen slutar. Ska vara större än *start* attributet.	datetime	lång version

I *Booking*-objektet finns två valfria element: *Attendees* och *Categories*.

Attendees-elementet är en lista över alla deltagare på bokningen, där deltagare kan vara av objekttyperna *Object*, *Room*, *Group* eller *Person*. Deltagarna måste existera då bokningen skrivs och alla deltagarobjekt i *Attendees*-elementet ska vara i kort version. Figur 35 visar en bokning med *Attendees*-elementet innehållande en lista över alla giltiga deltagarobjekt. Lämnas *Attendees*-elementet bort betyder det att bokningen inte har deltagare.

```
<Booking organizer="uid" ...>
  <Attendees>
    <Object ... />
    <Room status="abc" ... />
    <Group ... />
    <Person ... />
  </Attendees>
</Booking>
```

Figur 35. Booking-objektet med Attendees-elementet innehållande alla giltiga deltagarobjekt.

Room-objektet innehåller, utöver sina vanliga attribut, även ett *status*-attribut som anger vilken status rummet har. Tabell 9 visar alla statusvärden detta attribut kan innehålla och vad de betyder.

Tabell 9. Bokningsstatus ett rum kan ha i Booking-objektet.

Status	*Beskrivning*
Accept	En accepterad bokning.
Decline	En nekad bokning.
Tentative	En preliminär bokning.

Categories-elementet är en lista över alla kategorier som bokningen hör till. En kategori representeras av *Category*-elementet. Figur 36 visar ett *Booking*-objekt med *Categories*-elementet innehållande två *Category*-element som representerar varsin kategori: *abc* och *def*. Lämnas *Categories*-elementet bort betyder det att bokningen inte hör till någon kategori.

```
<Booking organizer="uid" ...>
  <Categories>
    <Category>abc</Category>
    <Category>def</Category>
  </Categories>
</Booking>
```

Figur 36. Booking-objektet med Categories-elementet innehållande två Category-element.

5.5.3 Room

Room-objektet representerar ett rum. Ett rum kan skrivas, läsas och raderas med kommandona *Set*, *Get* och *Del*.

Ett rum har tre olika sätt man unikt kan identifiera det med: *id*-attributet, *name*-attributet eller *block*- och *number*-attributen. Klienten bör enbart använda sig av ett identifieringssätt per *Room*-objekt. Figur 37 visar användningen av *id*-attributet, Figur 38 visar användningen av *name*-attributet och Figur 39 visar användningen av *block*- och *number*-attributen. Varje identifieringssätt representerar den korta versionen av *Room*-objektet. Tabell 10 visar alla attribut som *Room*-objektet kan ha i lång version.

```
<Room id="abc" />
```

Figur 37. Room-objektet identifierat med id-attributet.

```
<Room name="def" />
```

Figur 38. Room-objektet identifierat med name-attributet.

```
<Room block="A" number="123" />
```

Figur 39. Room-objektet identifierat med block- och number-attributen.

Tabell 10. Room-objektets attribut i lång version.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
id	Serverns id för rummet.	text	ja Figur 37
name	Namnet på rummet.	text	ja Figur 38
block	Blocket som rummet befinner sig i.	text	ja Figur 39
number	Rummets nummer.	text	ja Figur 39
floor	Våningen som rummet befinner sig på.	text	nej
area	Området som rummet befinner sig i (byggnad eller annan gruppering).	text	nej
alias	Ett alternativt namn på rummet.	text	nej
description	Beskrivning på rummet.	text	nej
notes	Noteringar på rummet.	text	nej
capacity	Rummets kapacitet, t.ex. hur många sittplatser eller datorer rummet har.	int	nej
airflow	Luftflödet mätt i antalet personer det är dimensionerat för.	int	nej
floorsize	Rummets golvyta i m^2.	int	nej
bookingwindow	Anger hur många dagar in i framtiden som bokning tillåts i rummet.	int	nej
automateprocessing	Anger om rummet automatiskt ska behandla bokningsförfrågningar.	bool	nej
hidden	Anger om rummet ska gömmas från adressboken.	bool	nej

Room-objektet innehåller sex olika element: *Categories*, *Equipments*, *BookInPolicy*, *RequestInPolicy*, *RequestOutOfPolicy* och *Delegates*.

Categories-elementet är en lista över *Category*-element. Varje *Category*-element anger en kategori som rummet hör till. Figur 40 visar *Categories*-elementet innehållande två *Category*-element: *abc* och *def*. Lämnas *Categories*-elementet bort betyder det att rummet inte hör till någon kategori.

```
<Room ...>
  <Categories>
    <Category name="abc" />
    <Category name="def" />
  </Categories>
</Room>
```

Figur 40. Categories-elementet i Room-objektet.

Tabell 11 visar de attribut som används i *Category*-elementet.

Tabell 11. Category-elementets attribut.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
name	Namnet på kategorin som rummet hör till.	text	ja

Equipments-elementet är en lista över *Equipment*-element. Varje *Equipment*-element anger en utrustningstyp som finns till förfogande i rummet. Figur 41 visar *Equipment*-elementet innehållande två *Equipment*-element: *abc* och *def*. Lämnas *Equipment*-elementet bort betyder det att rummet inte har någon utrustning.

```
<Room ...>
  <Equipments>
    <Equipment name="abc" />
    <Equipment name="def" />
  </Equipments>
</Room>
```

Figur 41. Equipments-elementet i Room-objektet.

Tabell 12 visar de attribut som används i *Equipment*-elementet.

Tabell 12. Equipment-elementets attribut.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
name	Namnet på utrustningstypen som finns tillgänglig i rummet.	text	ja

Elementen *BookInPolicy*, *RequestInPolicy* och *RequestOutOfPolicy* har samma attribut och samma typer av barnelement. Elementen används till att ställa in bokningsrättigheter på rummet. Alla tre element kan innehålla objekt av typerna *Person*,

Group och *Object*, förutsatt att *Object* anger någon av de två andra typerna. Tabell 13 visar alla attribut som används i de tre elementen.

Tabell 13. BookInPolicy-, RequestInPolicy- och RequestOutOfPolicy-elementens attribut.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
everyone	Anger om alla ingår i policyn eller enbart de som listas.	bool	ja

BookInPolicy anger de som får boka rummet förbi eventuell moderering, om bokningen följer uppsatta regler.

RequestInPolicy anger de som får boka rummet genom moderering, om bokningen följer uppsatta regler.

RequestOutOfPolicy anger de som får boka rummet genom moderering, då bokningen bryter mot uppsatta regler.

Figur 42 visar hur elementen formateras. Det enda som skiljer de tre elementen åt är deras namn.

```
<Room ...>
  <BookInPolicy everyone="false">
    <Person ... />
    <Group ... />
    <Object ... />
  </BookInPolicy>
</Room>
```

Figur 42. BookInPolicy-, RequestInPolicy- och RequestOutOfPolicy-elementens formatering.

Delegates-elementet är en lista över objekt som modererar bokning av rummet. Objekten kan vara av typerna *Person* och *Group*. *Object*-objektet kan användas, förutsatt att objektet representerar ett *Person*- eller *Group*-objekt. Tabell 14 visar de attribut som finns i *Delegates*-elementet och Figur 43 visar hur elementet används i *Room*-objektet.

Tabell 14. Attribut i Delegates-elementet.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
forward	Anger om rumsbokningsförfrågan ska vidarebefordras till moderatorerna.	bool	ja

```
<Room ...>
  <Delegates forward="true">
    <Person ... />
    <Group ... />
    <Object ... />
  </Delegates>
</Room>
```

Figur 43. Delegates-elementet i Room-objektet.

5.5.4 Group

Group-objektet representerar en grupp av personer och andra grupper. En grupp kan skrivas, läsas och raderas med kommandona *Set*, *Get* och *Del*.

En grupp har två olika sätt man unikt kan identifiera det med: *id*-attributet eller *name*- och *path*-attributen. Klienten bör enbart använda sig av ett identifieringssätt per *Group*-objekt. Figur 44 visar användningen av *id*-attributet och Figur 45 visar användningen av *name*- och *path*-attributen.

```
<Group id="abc" />
```

Figur 44. Group-objektet identifierat med id-attributet.

```
<Group name="..." path="stig/till/gruppen" />
```

Figur 45. Group-objektet identifierat med name- och path-attributen.

Figur 44 och Figur 45 visar den korta versionen av *Group*-objektet med de två olika identifieringssätten.

Varje grupp kan innehålla medlemmar av objekttyperna *Person* och *Group*. *Object*-objektet kan också användas, förutsatt att den underliggande datatypen är *Person* eller *Group*. Tabell 15 visar alla attribut som *Group*-objektet har.

Tabell 15. Group-objektets attribut.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
id	Serverns id för gruppen.	text	nej
name	Namnet på gruppen.	text	nej
path	Sökstigen till gruppen, t.ex. *”stig/till/gruppen”*.	text	nej

Group-objektet har två element som tar hand om att lägga till och ta bort medlemmar från gruppen: *AddMembers* och *RemoveMembers*. Figur 46 visar hur elementen används i *Group*-objektet.

```
<Group ...>
  <AddMembers>
    <Person ... />
    <Group ... />
    <Object ... />
  </AddMembers>
  <RemoveMembers>
    <Person ... />
    <Group ... />
    <Object ... />
  </RemoveMembers>
</Group>
```

Figur 46. Group-objektet med AddMembers- och RemoveMembers-elementen.

Group-objektet innehåller ett *Members*-element, som används för att lista alla gruppens medlemmar då servern returnerar gruppen som ett svar på ett *Get*-kommando. Figur 47 visar *Group*-objektet med *Members*-elementet.

```
<Group ...>
  <Members>
    <Person ... />
    <Group ... />
  </Members>
</Group>
```

Figur 47. Group-objektet med Members-elementet.

5.5.5 Person

Person-objektet representerar en person. En person kan skrivas och läsas med kommandona *Set* och *Get*, med restriktionen att ett *Person*-objekt enbart skrivs om det inte redan existerar. Uppdatering av befintliga *Person*-objekt är därför inte möjlig.

Ett *Person*-objekt har två olika sätt man unikt kan identifiera det med: *id*-attributet eller *username*-attributet. Klienten bör enbart använda sig av ett identifieringssätt per *Person*-objekt. Figur 48 visar användningen av *id*-attributet och Figur 49 visar användningen av *username*-attributet.

```
<Person id="abc" />
```

Figur 48. Person-objektet identifierat med id-attributet.

```
<Person username="..." />
```

Figur 49. Person-objektet identifierat med username-attributet.

Figur 48 och Figur 49 visar den korta versionen av *Person*-objektet med de två olika identifieringssätten. Tabell 16 visar alla attribut som *Person*-objektet har.

Tabell 16. Person-objektets attribut.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
id	Serverns id för personen.	text	ja Figur 48
username	Personens användarnamn.	text	ja Figur 49
firstname	Personens förnamn.	text	ja Figur 50
lastname	Personens efternamn.	text	ja Figur 50

Attributen *id* och *username* används, som tidigare nämnts, för att identifiera en person. Vid användning i *Get*-kommandot måste något av dessa attribut användas.

Set-kommandot kräver att *username*-attributet används för identifiering och att *firstname*- och *lastname*-attributen är givna. Figur 50 visar ett *Person*-objekt inne i ett *Set*-kommando.

```
<Set ...>
  <Person username="..." firstname="..." lastname="..." />
</Set>
```

Figur 50. Person-objekt i ett Set-kommando.

5.6 Felhantering

Fel som uppstår representeras av två olika objekt: *Error* och *ValidationError*. Om ett fel uppstår i ett kommando avbryts exekveringen av kommandot och nästa kommando i listan exekveras som följande.

5.6.1 Error

Error-objektet representerar ett fel. Objektet kan förekomma i dokumenthuvudet eller i *Response*-elementet som svar på ett kommando. Figur 51 visar hur objektet formateras och Tabell 17 visar de attribut som används i *Error*-objektet.

```
<Error code="500">Ett fel har uppstått.</Error>
```

Figur 51. Error-objektets formatering.

Tabell 17. Error-objektets attribut.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
code	Numerisk kod som grupperar felet enligt Tabell 18.	int	ja

Tabell 18. Värden som code-attributet i Error-objektet kan anta.

Värde	*Beskrivning*
404	Objektet kunde inte hittas.
500	Ett oväntat fel som uppstod under körningen.
505	Operationen stöds inte.

Error-objektet kan även innehålla en mer läsbar form av felet som förklarar vad som gått fel, vilket syns i Figur 51.

5.6.2 ValidationError

ValidationError-objektet indikerar att datakontrollen inte gick igenom och gör att exekveringen av kommandot avbryts. ValidationError-objektet är en behållare som kan innehålla element och objekt av typerna *ErrorItem*, *Error* och *ValidationError*. Tabell 19 visar alla attribut objektet kan ha och Figur 52 visar ett *ValidationError*-objekt med alla element och objekt den kan innehålla.

Tabell 19. ValidationError-objektets attribut.

Attribut	*Beskrivning*	*Datatyp*	*Obligatorisk*
code	Numerisk kod som grupperar felet enligt Tabell 20.	int	ja

Tabell 20. Värden som code-attributet i ValidationError-objektet kan anta.

Värde	*Beskrivning*
412	Datakontrollen hittade fel.

```
<ValidationError code="412">
  <ErrorItem>...</ErrorItem>
  <Error code="...">...</Error>
  <ValidationError code="412">
    ...
  </ValidationError>
</ValidationError>
```

Figur 52. ValidationError-objektet med de tre olika element och objekt den kan innehålla.

ErrorItem-elementet är ett element som enbart innehåller en läsbar text som beskriver ett datafel som kontrollen hittade, vilket Figur 53 visar.

```
<ErrorItem>Ett datafel hittades.</ErrorItem>
```

Figur 53. ErrorItem-elementets formatering.

6 VIDAREUTVECKLING

Ett system är aldrig helt färdigutvecklat. Alltid finns det fel eller problem som kan korrigeras eller ny funktionalitet som krävs för att göra systemet bättre. Här tas upp de saker som är mer eller mindre kritiska att utveckla innan systemet kan tas i produktion.

6.1 Prenumerationer

Det saknas ännu en hel del funktionalitet innan en fullt fungerande version är färdig. Bland annat måste klienterna kunna prenumerera på händelser som sker i olika kalendrar, främst rummens. En händelse betyder att en bokning skapas, ändras eller raderas.

6.2 Mötesinbjudningar

Exchange skickar en mötesinbjudan till alla deltagare för varje bokning som görs. Detta är i många fall praktiskt, men i många fall blir det väldigt opraktiskt då automatiserade bokningar kan orsaka att en stor mängd mötesinbjudningar skickas ut. Klienter till systemet borde kunna bestämma om bokningen är valfri eller obligatorisk för deltagarna. Är den obligatorisk tar systemet hand om att automatiskt bekräfta bokningen i varje deltagares kalender.

6.3 Grupper

Egna tester har visat att Exchange inte innehåller funktionalitet för gruppkalendrar. Det finns grupper som kan ha medlemmar, där varje bokning med gruppen som deltagare medför att varje medlem i gruppen får en egen mötesinbjudan. Nya medlemmar får inte de bokningar som tidigare gjorts och borttagna medlemmar blir inte av med de redan mottagna bokningarna. Det går inte heller att få en lista på alla bokningar som gjorts på en specifik grupp, då gruppen inte har en egen inkorg i Exchange. Då systemet kommer att användas för att ge även studenter egna kalendrar baserade på de kurser de närvaroanmält sig till måste denna grupproblematik lösas.

Förslagsvis löses detta genom att i varje grupp ha en systemhanterad medlem. Denna medlem får alla bokningar och avbokningar i sin kalender, vilket sedan fungerar som en

lista över alla bokningar där gruppen är deltagare. Denna lista kan sedan användas för att lägga till bokningar i nya medlemmars kalender och radera bokningar från borttagna medlemmar.

6.4 Personer

Stödet för att lägga till personer måste ses över om det kan finnas kvar. I värsta fall kan det orsaka problem för andra applikationer som läser och skriver konton i Active Directory.

6.5 Autentisering och auktorisering

Stöd för att autentisera en klient och ge eller neka tillgång till funktionalitet måste utvecklas, annars kan vilken klient som helst utföra vilka operationer som helst.

Autentiseringen kan lösas genom att servern har en lista på IP-adresser som klienterna ansluter sig från, och vilken klient som hör till vilken IP-adress. En annan lösning är att varje klient som får tillgång till servern måste autentisera sig i varje XML-dokument som skickas. En tredje lösning är att använda sig av HTTP-protokollets autentiseringsmekanismer med exempelvis användarnamn och lösenord. Autentiseringen kan även vara en blandning av olika alternativ.

Auktoriseringen är troligtvis svårare att utveckla då den kan gå väldigt djupt. Exempelvis kan auktorisering vara på kommandonivå, objektnivå eller till och med på objektens identifikationsnivå. T.ex. kan auktoriseringen tillåta eller neka att en klient skriver ett specifikt objekt, men tillåta att ett annat objekt av samma typ får skrivas. För att stöda denna djupa auktoriseringsmodell krävs mycket data för att systemet ska kunna göra rätta beslut.

Autentisering och auktorisering är helt skilda från systemets övriga funktionalitet. Därför skall dessa delar vara ett filtrerande lager ovanpå det fungerande systemet.

6.6 Loggning

Grundläggande loggning finns, men mycket kod loggar inte något. För att kunna följa hur systemet går i produktion och spåra eventuella fel som uppstår bör loggningen utökas till att logga rätt saker vid rätt tidpunkt.

6.7 Testning

Vissa delar av systemet har automatiska enhetstester skrivna, men många av delarna är helt otestade på detta sätt. Testerna som saknas borde skrivas för att man vid vidareutveckling skall kunna testa att befintlig logik fortsättningsvis fungerar som avsett.

6.8 Dokumentation

Systemet behöver få en ordentlig API-dokumentation av vad varje klass, metod och egenskap gör för att underlätta för någon annan i framtiden att sätta sig in i systemet och vidareutveckla det. I dagsläget finns det ytterst lite dokumentation skriven, enbart några få klasser har dokumenterats. Med hjälp av API-dokumentationen kan även utvecklingsomgivningen visa dokumentationen vid utvecklingen t.ex. då någon metod skall anropas.

7 SLUTSATSER

Slutresultatet av detta examensarbete har blivit en fungerande bas att bygga vidare på. Detta arbete har inte tagit upp någon källkod utan försökt förklara en del av den arkitektur som används och visa på kopplingen tillbaka till olika designmönster som finns och har utnyttjats. Slutresultatet har visat sig vara att ett relativt stort system känns rätt litet att utveckla vidare på, då rätt designbeslut tagits och därmed gjort systemet både lätt att underhålla och troligtvis lätt att sätta sig in i också.

Det finns många saker som däremot inte är helt färdiga med systemet då detta examensarbete skrivs, och de viktigaste punkterna som saknas har här skrivits om. Punkterna fungerar som en minneslista för den som ska utveckla systemet vidare.

Vägen att komma hit har varit lång och fylld med många fallgropar. Det har varit mycket nytt att sätta sig in i, främst hur Exchange fungerar och inte fungerar. Många gånger trodde jag att det borde fungera på ett sätt när det egentligen inte alls fungerar så. Den största ”oj då”-upplevelsen kom troligtvis då jag insåg att Exchange inte alls har någon gruppkalender, utan att allt är helt på individnivå. Denna insikt har försvårat arbetet med att få ett system som lätt kan hanteras utifrån att fungera som en gruppkalender.

Under vägen har jag även fått bekanta mig med Active Directory, PowerShell och Exchange Web Services. Dessa tre verktyg är nödvändiga för att kunna utveckla detta system. Active Directory är den katalogtjänst som används av Exchange för att lagra data om personer, rum och grupper. För att skapa dessa objekt i Active Directory så att Exchange känner till dem måste PowerShell användas. Bokningar och andra objekt sparas däremot i Exchange, och lättaste vägen att komma åt dem har varit via Exchange Web Services.

Även mina kunskaper inom design av mjukvara har fått en hård skolning och användning. Jag har tidigare satt mig in i designmönster mer på djupet och har sedan dess försökt omsätta mina kunskaper inom detta område till något praktiskt. I detta arbete har jag fått utnyttja större delen av min kunskap för att få ihop ett välfungerande system som också är väldigt flexibelt och skalbart. Visst finns det saker som har visat sig vara lite sämre beslut, men inga större saker som tagit länge att korrigera till en bättre lösning.

Mina kunskaper inom programmering har också varit till stor nytta. C# är ett väldigt bra och kraftfullt språk att utveckla i. Dessutom innehåller de senare versionerna av språket och ramverket många nya verktyg att använda sig av. Nytt i .NET Framework 4.0 ramverket är bland annat parallell programmering, vilket innebär att man delar upp programmet i flera trådar som kör på många kärnor i processorn. På detta sätt har jag kunnat optimera mycket tidskrävande operationer, men det har samtidigt orsakat många andra problem som måste lösas. Man måste lugnt sagt *hålla tungan rätt i mun* när man skriver program som gör många saker samtidigt.

Resultatet av mitt arbete kommer i alla fall att vara i flitig användning då många system inom Arcada har och kommer att ha behov av att hantera någon form av data i Exchange eller Active Directory. Det känns skönt att ett stort arbete som detta inte är gjort i onödan.

KÄLLOR

Arcada, 2010a. *Utbildning*. [www] Tillgänglig: http://www.arcada.fi/sv/utbildning Hämtad 08.03.2010.

Arcada, 2010b. *Yrkeshögskoleexamen*. [www] Tillgänglig: http://www.arcada.fi/sv/utbildning/yrkeshogskoleexamen Hämtad 08.03.2010.

Fowler, M., 2002. *Patterns of Enterprise Application Architecture*. Boston, USA: Addison-Wesley Professional.

Guthrie, S., 2009. *Auto-Start ASP.NET Applications (VS 2010 and.NET 4.0 Series)*. [www] Tillgänglig: http://weblogs.asp.net/scottgu/archive/2009/09/15/auto-start-asp-net-applications-vs-2010-and-net-4-0-series.aspx Hämtad 20.11.2010.

Microsoft, 2010..*NET Framework Conceptual Overview*. [www] Tillgänglig: http://msdn.microsoft.com/en-us/library/zw4w595w(VS.100).aspx Hämtad 10.03.2010.

Shore, J., 2006. *James Shore: Dependency Injection Demystified*. [www] Tillgänglig: http://jamesshore.com/Blog/Dependency-Injection-Demystified.html Hämtad 18.11.2010.

Wikipedia, 2010a..*NET Framework*. [www] Tillgänglig: http://en.wikipedia.org/wiki/.NET_Framework Hämtad 18.11.2010.

Wikipedia, 2010b. *Active Directory*. [www] Tillgänglig: http://en.wikipedia.org/wiki/Active_Directory Hämtad 20.11.2010.

Wikipedia, 2010c. *ASP.NET MVC Framework*. [www] Tillgänglig: http://en.wikipedia.org/wiki/ASP.NET_MVC_Framework Hämtad 20.11.2010.

Wikipedia, 2010d. *Design pattern (computer science)*. [www] Tillgänglig: http://en.wikipedia.org/wiki/Design_pattern_(computer_science) Hämtad 18.11.2010.

Wikipedia, 2010e. *Internet Information Services*. [www] Tillgänglig: http://en.wikipedia.org/wiki/Internet_Information_Services Hämtad 20.11.2010.

Wikipedia, 2010f. *Microsoft Exchange Server*. [www] Tillgänglig: http://en.wikipedia.org/wiki/Microsoft_Exchange_Server Hämtad 20.11.2010.

Wikipedia, 2010g. *Microsoft Visual Studio*. [www] Tillgänglig: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio Hämtad 21.11.2010.

Wikipedia, 2010h. *Windows Server 2008 R2*. [www] Tillgänglig: http://en.wikipedia.org/wiki/Windows_Server_2008_R2 Hämtad 20.11.2010.

BILAGA 1. XML-DOKUMENT I FÖRFRÅGAN OCH SVAR

```
<?xml version="1.0" encoding="utf-8"?>
<abex xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Head />
  <Body>
    <Commands>
      <Set id="set booking">
        <Booking organizer="uid" ...>
          ...
        </Booking>
      </Set>
      <Set id="set room">
        <Room name="def" ...>
          ...
        </Room>
      </Set>
      <Set id="set group">
        <Group name="..." path="stig/till/gruppen" />
      </Set>
      <Set id="set person">
        <Person username="..." firstname="..." lastname="..." />
      </Set>
    </Commands>
  </Body>
</abex>
```

XML-dokumentet som skickas till servern.

```
<?xml version="1.0" encoding="utf-8"?>
<abex xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Head />
  <Body>
    <Responses>
      <Response id="set booking" command="Set">
        <Booking organizer="uid" id="bokningens id" />
      </Response>
      <Response id="set room" command="Set">
        <Room id="rummets id" />
      </Response>
      <Response id="set group" command="Set">
        <Group id="gruppens id" />
      </Response>
      <Response id="set person" command="Set">
        <Person id="personens id" />
      </Response>
    </Responses>
  </Body>
</abex>
```

XML-dokumentet som returneras till klienten.