YRKESHÖGSKOLAN NOVIA

Program för kontroll av inträdesbiljetter till evenemang

Utbytbara hårdvaruspecifika implementationer genom Domain Oriented N-Layered Architecture

Tomas Ingo

Examensarbete för ingenjör (YH)-examen

Utbildningsprogrammet för informationsteknik

Vasa 2013

EXAMENSARBETE

Författare:	Tomas Ingo
Utbildningsprogram och ort:	Informationsteknik, Vasa
Handledare:	Kaj Wikman

Titel: *Program för kontroll av inträdesbiljetter till evenemang*

Datum 19.11.2013 Sidantal 19

Abstrakt

Detta examensarbete behandlar utvecklingen av ett program för kontroll av inträdesbiljetter till olika kulturevenemang. Programmet används på handdatorer av olika märken och behöver därför på ett smidigt sätt kunna stöda olika implementationer för olika streckkodsläsare. Detta har lösts med Domain Oriented N-Layered Architecture.

Programmet har utvecklats för Windows Mobile 6 med .NET Compact Framework 3.5 och kommunicerar med en extern serverprogramvara över Internet via en web service.

Språk: svenska Nyckelord: arkitektur

OPINNÄYTETYÖ

Tekijä:	Tomas Ingo
Koulutusohjelma ja paikkakunta:	Tietotekniikka, Vaasa
Ohjaaja:	Kaj Wikman

Nimike: *Ohjelma tapahtumapääsylippujen tarkistamiseen*

Päivämäärä 19.11.2013 Sivumäärä 19

Tiivistelmä

Tämä opinnäytetyö käsittelee kulttuuritapahtumien pääsylippuja tarkistavan ohjelman kehittämistä. Ohjelmaa käytetään eri merkkisillä kämmentietokoneilla, ja tämän vuoksi on tarpeellista saada se tukemaan eri toteutuksia eri viivakoodinlukijoita varten. Tämä on saatu aikaiseksi Domain Oriented N-Layered Architecturen avulla.

Ohjelma on kehitetty Windows Mobile 6:a varten .NET Compact Framework 3.5:llä, ja se kommunikoi ulkoisen palvelinohjelmiston kanssa Internetin kautta käyttämällä web serviceä.

Kieli: ruotsi Avainsanat: arkkitehtuuri

BACHELOR'S THESIS

Author:	Tomas Ingo
Degree programme:	Information Technology, Vaasa
Supervisor:	Kaj Wikman

Title: *Application for validation of admission tickets to events*

Date 19.11.2013 Number of pages 19

Abstract

This thesis deals with the development of an application for the validation of admission tickets to different cultural events. The application is used on handheld computers of different brands, and thus a way of supporting different implementations for different barcode readers is needed. This has been solved with Domain Oriented N-Layered Architecture.

The application has been developed for Windows Mobile 6 using .NET Compact Framework 3.5, and it communicates with an external server software over the Internet through a web service.

Language: Swedish Key words: architecture

Innehållsförteckning

Ordförklaringar

Domän	I detta examensarbete används ordet domän för att syfta på den domän som ett program görs för. Om man till exempel utvecklar ett bankprogram syftar ordet domän i detta sammanhang på själva bankväsendet.
Gränssnitt	I detta examensarbete används ordet gränssnitt för att syfta på en uppsättning metoder. En klass uppfyller ett gränssnitt om den implementerar alla metoder som definieras av gränssnittet.
SDK	Ett SDK (Software Development Kit) är en samling verktyg för programvaruutveckling för exempelvis en viss plattform eller ett visst ramverk.
XML	XML (eXtensible Markup Language) är ett textbaserat format för hierarkiskt strukturerad data. Ett exempel på en XML-fil finns i kodexempel 1.

1 Inledning

Arbetet gick ut på att utveckla ett program för avläsning av inträdesbiljetter till olika kulturevenemang. Programmet skulle ha följande egenskaper:

- Programmet används på en handdator med streckkodsläsare.
- Handdatorer av två olika märken används av uppdragsgivaren, och båda skall stödas.
- Det skall även vara lätt att i framtiden bygga stöd för andra enheter.
- Programmet hämtar information om evenemang och biljetter sålda till dessa från en web service (se kapitel 2.3).
- När streckkoden för en biljett har avlästs skall programmet meddela användaren om biljetten är giltig, om biljetten är en rabattbiljett eller om biljetten eventuellt redan har använts.

1.1 Uppdragsgivare

Uppdragsgivare är Oy NetTicket Finland Ab. Företaget grundades år 1995 och hette ursprungligen Oy Soft & Micro Help – Jonny Mandell Ab. År 2011 togs det nuvarande företagsnamnet i bruk. Företaget har fyra heltidsanställda och säljer inträdesbiljetter åt evenemangsarrangörer runtom i landet. Biljettförsäljningssystemet Felix som utvecklas inom företaget används också av Vasa stadsteater och Vasa stadsorkester. Wasa Teater använde Felix fram till år 2013, varpå deras biljettförsäljning helt övergick till NetTicket Finland. Till företagets verksamhet hör även konsultering och skolning inom IT. Ägaren Jonny Mandell fungerar som IT-stödperson i ett antal företag.

1.2 Bakgrund

Uppdragsgivaren har tidigare använt ett liknande program för kontroll av biljetter. Detta hade utvecklats av uppdragsgivaren för handdatorer av märket Symbol. Eftersom uppdragsgivaren även hade tillgång till handdatorer av märket Intermec, var det önskvärt att även kunna använda dessa på samma sätt. Dessutom hade det från kunderna kommit feedback om att programmet var långsamt eftersom statusen för en avläst biljett

kontrollerades i realtid från web servicen (se kapitel 2.3). Detta ledde till en fördröjning på upp till några sekunder från att streckkoden för en biljett hade avlästs till att användaren meddelades om biljettens status. Därför ville uppdragsgivaren att programmet skulle lagra statusinformation om biljetter lokalt i arbetsminnet och vid avläsning asynkront meddela web servicen om att en biljett hade förbrukats.

1.3 Uppdrag

Uppdraget gick ut på att planera och utveckla ett program som skall komma att användas av biljettkontrollanter vid entrén till ett evenemang. När användaren har läst av streckkoden på en besökares inträdesbiljett kontrollerar och meddelar programmet om biljetten är giltig.

Programmet skall kunna användas på svenska, finska och engelska. När programmet startar ombeds användaren välja språk, varpå information om dagens föreställningar hämtas på det valda språket från biljettförsäljningssystmet över en web service (se kapitel 2.3).

Uppdragsgivaren hade sedan tidigare ett sådant program, men det programmet hade två huvudsakliga problem. För det första var programmet utvecklat för handdatorer av märket Symbol, och nu ville man även ha stöd för andra enheter. För det andra använde det synkrona web service-anrop vid kontroll av biljetter, vilket ledde till en fördröjning på upp till några sekunder från att streckkoden för en biljett hade avlästs till att användaren meddelades om biljettens status. Detta examensarbete behandlar lösningen av det första problemet.

Det andra problemet med det befintliga programmet löstes genom att programmet hämtar biljettinformationen från web servicen vid start och och lagrar denna i primärminnet. Detta för att informationen för en biljett så snabbt som möjligt skall kunna kontrolleras och meddelas åt användaren när streckkoden för en biljett har avlästs. När en giltig biljett har lästs av, markeras den som använd i den lokala biljettinformationen. Samtidigt skickas ett meddelande asynkront till web servicen om att biljetten har använts, så att biljettförsäljningssystemet uppdateras med informationen. Ifall en Internet-uppkoppling inte är tillgänglig skrivs i stället biljettkoden till en fil som lagras lokalt på apparaten. På så

vis kan programmet användas i ”Offline-läge” (se kapitel 3.3) då möjlighet till ständig uppkoppling saknas. Följande gång programmet startas ombeds användaren uppdatera web servicen med dessa offline-förbrukade biljettkoder.

Ett problem med lösningen att använda biljettinformation lagrad i primärminnet uppstår ifall man använder flera apparater. Då finns risken att personer med kopierade biljetter går förbi varsin biljettkontrollant. Då kan programmet i den ena apparaten inte veta om en biljett redan har förbrukats i den andra apparaten. Detta löstes genom att man i biljettförsäljningssystemet lagrar information om antalet gånger en biljett har förbrukats. Ifall en biljett förbrukas mer än en gång vet man att missbruk har skett och behövliga åtgärder kan vidtas.

I kapitel 3.1 finns en utredning för huruvida det hade varit värt att bygga vidare på det befintliga programmet, eller om det behövde planeras och byggas om från grunden. Web servicen tillhandahålls av uppdragsgivarens serverprogramvara och utvecklades separat från detta arbete. Detsamma gäller de ändringar och tillägg som behövdes i själva biljettförsäljningssystemet.

2 Tekniker

Programmet skulle utvecklas för handdatorer med Windows Mobile 6 som operativsystem, vilket ledde till vissa givna tekniker vad beträffar ramverk och SDK:er. Eftersom handdatorer av två (och i framtiden eventuellt fler) olika märken med olika SDK:er skulle stödas, behövdes en lösning för att göra hårdvaruspecifika implementationer utbytbara beroende på vilken apparat programmet körs på.

2.1 .NET Framework

.NET Framework är ett ramverk som utvecklas av Microsoft. Ramverket erbjuder bland annat ett stort klassbibliotek för utveckling av programvara för Windows-baserade plattformer samt den exekveringsmiljö som behövs för att kunna köra dessa program. Till .NET Framework hör även programmeringsspråket C# som specifikt utvecklats för användning med ramverket. (Mössenböck, Beer, Birngruber & Wöß 2004, s. 2).

2.2 .NET Compact Framework 3.5

.NET Compact Framework är en nedbantad version av .NET Framework (se kapitel 2.1). Ramverket utvecklas av Microsoft för att användas vid programvaruutveckling för handdatorer, mobiltelefoner och dylika enheter. Ramverket saknar en del av de moduler som finns i .NET Framework, och det innehåller vissa moduler specifika för mobila system. Ett för detta arbete relevant exempel på skillnader mellan ramverken är att stöd för byte av språk för användargränssnittet genom System.Threading.Thread.CurrentUICulture saknas i .NET Compact Framework. Därför behövde en egen lösning för språkhantering implementeras (se kapitel 3.2.2). (Differences Between the .NET Compact Framework and the .NET Framework, u.å.).

2.3 Web services

Web services är en del av .NET Framework som möjliggör metodanrop över Internet med hjälp av det XML-baserade kommunikationsprotokollet SOAP. Utvecklaren anropar en web service som om den vore en vanlig metod tillhörande en vanlig klass, men i själva verket bildas ett SOAP-meddelande som skickas till en extern server. Servern utför anropet och skickar tillbaka resultatet som ett nytt SOAP-meddelande. (Mössenböck m.fl. 2004, s. 8–9).

2.4 Windows Mobile 6 SDK Refresh

Windows Mobile 6 SDK Refresh är Microsofts SDK som används med programmeringsmiljön Microsoft Visual Studio för att utveckla program för plattformen Windows Mobile 6. Förutom klassbibliotek och dokumentation ingår även emulatorer som gör att man kan testa sin programvara utan tillgång till en fysisk apparat. (Download Windows Mobile 6, u.å.).

2.5 EMDK for .NET v2.0

EMDK (Enterprise Mobility Developer Kit) är Motorolas SDK för utveckling av program för deras apparater. Den används även för handdatorer av märket Symbol. SDK:n hette tidigare SMDK (Symbol Mobility Developer Kit). (EMDK for .NET v2.0, 2008).

Version 2.0 av EMDK är den sista versionen som stöder modellen Symbol MC50, och eftersom detta är den modell som uppdragsgivaren använder kunde ingen nyare version av SDK:n användas. (Release Notes - EMDK for .NET v2.3, 2010).

2.6 IDL Resource Kit - Data Collection

IDL (Intermec Developer Library) Resource Kit – Data Collection är ett av Intermecs SDK:er för utveckling av program för deras apparater. Detta paket innehåller bland annat de programklasser som behövs för användning av streckkodsläsare. (Intermec Applications & Software, u.å.).

2.7 Domain Oriented N-Layered Architecture

När man utvecklar programvara lönar det sig att dela upp programkoden i separata, utbytbara delar. Detta underlättar bland annat underhåll och vidareutveckling. När man gör ändringar i en del av programmet skall detta ha så liten inverkan som möjligt på andra delar av programmet. Ett sätt är att dela upp programmet i olika lager med olika ansvarsområden som kommunicerar med varandra genom abstrakta gränssnitt. Ett exempel på detta är Domain Oriented N-Layered Architecture. Denna arkitektur kretsar kring ett centralt lager där logiken för domänen implementeras (se kapitel 2.7.1) medan allt annat sköts av andra lager. Arkitekturen baserar sig på Domain Driven Design som beskrivs i Eric Evans bok ”Domain Driven Design – Tackling Complexity in the Heart of Software”. (Torre de la, Zorrilla, Calvarro & Ramos 2011, s. 43, 51–52, 55).

2.7.1 Domain-lagret

Domain-lagret är det centrala lagret inom Domain Oriented N-Layered Architecture. Här implementeras logiken för själva domänen utan att man tar ställning till tekniska detaljer såsom databaser och grafiska användargränssnitt. Exempelvis beskriver programkoden i Domain-lagret för ett bankprogram hur bankväsendet fungerar så likt som möjligt hur en banktjänsteman skulle beskriva det med egna ord. Huvudklasserna i Domain-lagret kallas entiteter och innehåller både den data och logik som hör till det verkliga ting de representerar. Exempelvis innehåller klassen som representerar ett bankkonto information om kontots saldo samt logik för att göra insättningar och uttag. (Torre de la m.fl. 2011, s. 59).

Domain-koden är den mest beständiga delen av en programvara eftersom den inte behöver ändras om inte ändringar sker i själva domänen, det vill säga bankväsendet i föregående exempel. Andra delar av programmet, såsom grafiska användargränssnitt, är föränderliga och fullt utbytbara. (Torre de la m.fl. 2011, s. 52).

2.7.2 Infrastructure-lagret

I Infrastructure-lagret implementeras logiken för specifika teknologier såsom databaser och hårdvaruenheter. För dataåtkomst definierar Domain-lagret gränssnitt för så kallade Repositories. Ett Repository representerar en generell förvaringsplats och används för att läsa och skriva data. Exempelvis kan Domain-lagret definiera gränssnittet för ett CustomerRepository med metoder såsom GetAllCustomers, FindCustomersByName, UpdateCustomer och så vidare. Infrastructure-lagrets uppgift är att implementera dessa gränssnitt så att de kan användas. När man på Application-nivå (se kapitel 2.7.3) i programmet anropar metoden GetAllCustomers sker det på Infrastructure-nivå ett databasanrop, en läsning ur en fil eller motsvarande. Detta resulterar i att man på Application-nivån får en lista över alla kunder utan att man behöver veta varifrån informationen kommit. På så vis kan man exempelvis byta ut databasleverantören som används i samband med ett bankprogram och göra de ändringar som behövs i Infrastructure-lagret utan att behöva göra några som helst ändringar i Domain-lagret eftersom det ju inte har skett några som helst ändringar i hur saker och ting fungerar i själva bankväsendet. (Torre de la m.fl. 2011, s. 56, 61).

Till Infrastructure-lagret hör även så kallade Cross-cutting-klasser. Dessa klasser används på flera olika håll i programmet, i alla lager. Därför är det vanligt att man behandlar dessa klasser som ett skilt lager (eller flera) vid sidan av de andra lagren i en arkitektur (se figur 1). De är hur som helst en del av Infrastructure-lagret eftersom de hanterar olika teknologispecifika aspekter såsom inversion of control (se kapitel 2.7.5), cache-minne, konfigurationshantering, logghantering och autentisering. (Torre de la m.fl. 2011, s. 56, 394, 397).

2.7.3 Application-lagret

Application-lagret koordinerar uppgifterna för de olika klasserna och gränssnitten i Domain-lagret samt utför transaktioner och annat som behövs för själva programmet snarare än för domänlogiken. Till exempel är det Application-lagret som anropar CustomerRepositoryns GetAllCustomers-metod och serverar den erhållna informationen till Presentation-lagret när användaren klickar fram en lista över alla kunder. Man kan säga att Application-lagret fungerar som en fasad (se kapitel 2.8.2) för Domain-lagret. (Torre de la m.fl. 2011, s. 57).

2.7.4 Presentation-lagret

Presentation-lagrets uppgift är att presentera data och ta emot kommandon från användaren, vanligen genom ett grafiskt användargränssnitt. (Torre de la m.fl. 2011, s. 56).

2.7.5 Inversion of control

Inversion of Control (IoC) är en teknik som med fördel kan användas i samband med Domain Oriented N-Layered Architecture för att separera gränssnitt och implementation exempelvis då det gäller Repositories som definieras i Domain-lagret och implementeras i Infrastructure-lagret. Eftersom man i Application-lagret inte ska behöva veta eller ta ställning till vilken implementation av ett visst gränssnitt som används, delegeras detta till en så kallad IoC-behållare. En IoC-behållare är en klass som känner till vilka implementationer som finns tillgängliga till vilka gränssnitt och är konfigurerbar exempelvis genom en konfigurationsfil. Om man på Application-nivå ber en IoC-behållare om en implementation av CustomerRepository (se kapitel 2.7.2), så returnerar IoC-

behållaren exempelvis en databashanterarklass som implementerar de metoder som definierats för detta gränssnitt. (Torre de la m.fl. 2011, s. 66–68).

2.8 Designmönster

Designmönster är abstrakta beskrivningar av typexempel på lösningar på vanliga problem inom programutveckling. (Gamma, Helm, Johnson & Vlissides 1994, s. 2–4).

2.8.1 Singleton

Singleton är ett designmönster som används för att garantera att endast en instans av en viss klass existerar i ett program. Detta löses huvudsakligen med en skyddad konstruktor för att förhindra att klassen instansieras utifrån, samt en instansvariabel som lagras statiskt i klassen. Instansvariabeln kommer man åt genom ett metodanrop som returnerar en referens till denna. (Gamma m.fl. 1994, s. 127–134).

2.8.2 Fasad

Fasad är ett designmönster som innebär att man ger ett enhetligt gränssnitt utåt för att göra ett komplicerat system lättare att använda. Ett vardagligt exempel på detta är frontpanelen på en enkel tvättmaskin. I stället för att ge tvättmaskinen en lång serie detaljerade instruktioner som att ta in vatten, applicera tvättmedel, rotera trumman och så vidare, väljer man helt enkelt ett tvättprogram och trycker på en knapp för att starta. (Gamma m.fl. 1994, s. 185–193).

3 Utförande

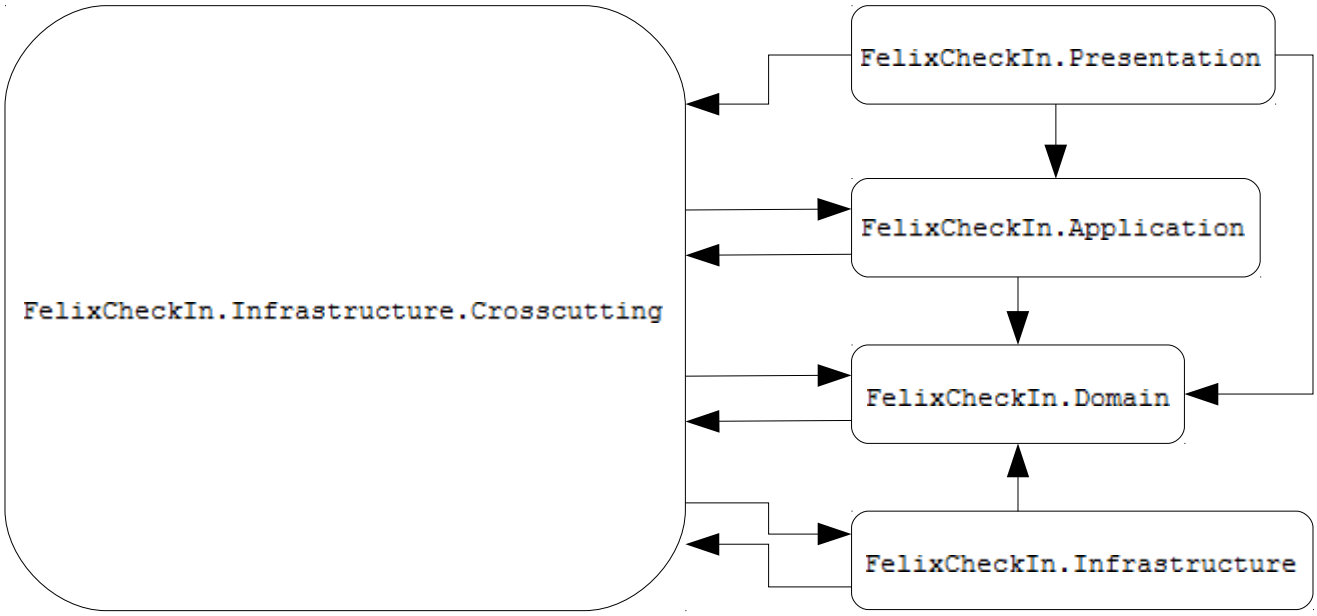
Eftersom uppdragsgivaren redan hade ett motsvarande program fanns det färdigt en uppfattning om hur det skulle se ut och fungera ur användarens synvinkel. Dessutom fanns det färdiga web service-anrop som kunde återanvändas. Det viktigaste beslutet att ta var hur arkitekturen i det nya programmet skulle se ut.

3.1 Val av arkitektur

När det gällde val av arkitektur var det naturligt att överväga huruvida den som redan fanns i det befintliga programmet kunde användas och byggas vidare på. Det befintliga programmet hade en enkel och monolitisk uppbyggnad som var gjord för att fungera på handdatorer av märket Symbol, utan att ta hänsyn till eventuell framtida utveckling och stöd för andra apparater. Detta innebar att programkod specifik för just denna apparat vad beträffar streckkodsläsare och ljudenhet fanns utspridd på olika ställen i programmet. Detta innebar i sin tur att det skulle bli väldigt klumpigt och felbenäget att bygga stöd för apparater av andra märken, eftersom det på flera olika ställen skulle behövas en kontroll för vilken apparat som för tillfället används, för att sedan köra rätt hårdvaruspecifik programkod. Dessutom skulle programkoden bli svår att både läsa och underhålla i framtiden. Därför bedömdes det som ett bra val att i stället bygga om programmet från grunden och dela in programkoden i olika lager enligt principerna för Domain Oriented N-Layered Architecture för att isolera den hårdvaruspecifika programkoden från den domänspecifika. Detta skulle leda till att det blir lätt att använda programmet på olika plattformer och att i framtiden bygga stöd för nya hårdvaruenheter.

3.2 Programmets arkitektur

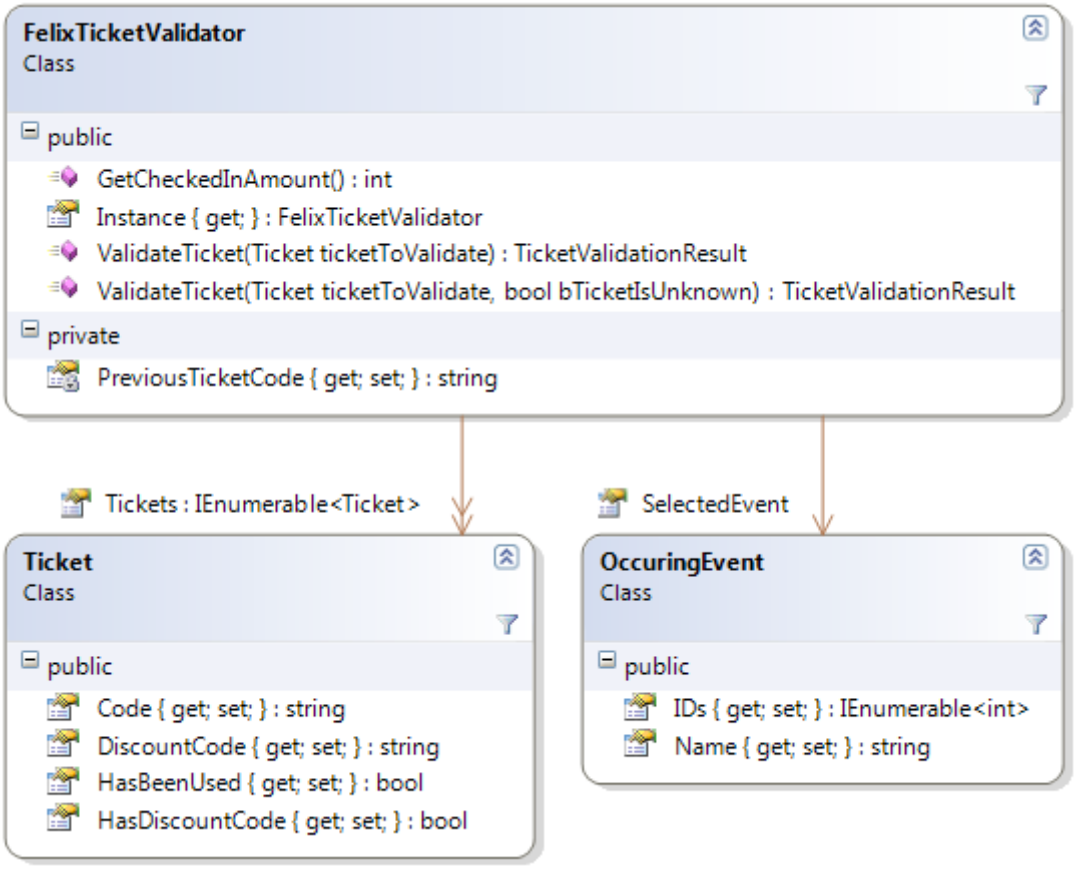
Programkoden delades in i ett antal lager enligt principerna för Domain Oriented N-Layered Architecture (se kapitel 2.7), vilket illustreras i figur 1.



Figur 1. Diagram över programmets arkitektur. Lagren representeras av sina huvudnamnrymder och pilarna visar hur de olika lagren använder sig av varandra. Notera att Domain-lagret fungerar självständigt utan att använda sig av de andra lagren, undantag Cross-cutting.

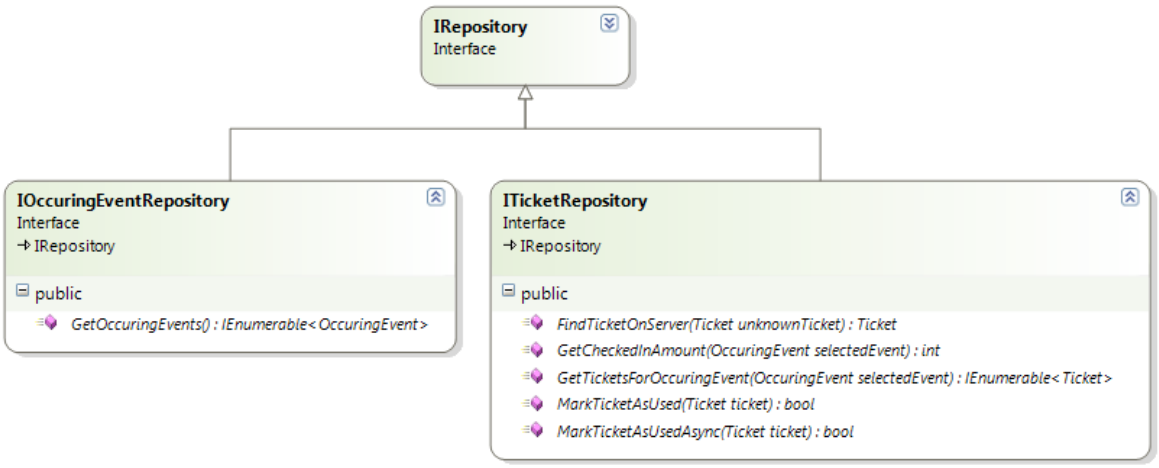
3.2.1 Domain-lagret

Kärnan i domänkoden gäller kontroll av biljetter. I figur 2 visas ett diagram över de viktigaste klasserna i Domain-lagret. Biljettkontrollen har lösts med en biljettkontrollantklass som känner till vilka biljetter som har sålts till en viss föreställning. Genom att använda den kan man kontrollera om en viss biljett är giltig, om den är en rabattbiljett och om den eventuellt redan har använts.



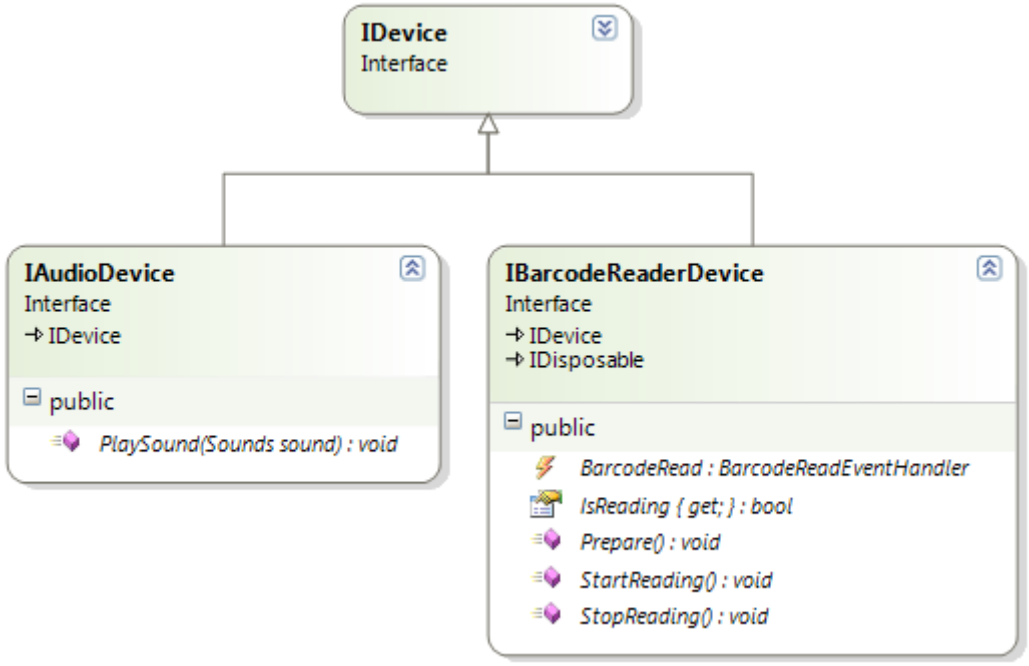
Figur 2. Klasserna i Domain-lagret.

Information om biljetter och föreställningar fås från Repositories vars gränssnitt definieras i Domain-lagret (se figur 3) och implementeras i Infrastructure-lagret (se kapitel 3.2.2). På så vis är datakällan utbytbar ifall den för tillfället använda web servicen (se kapitel 2.3) behöver bytas ut i framtiden.



Figur 3. De Repository-gränssnitt som definierats i Domain-lagret.

För detta program behöver även en streckkodsläsare och en ljudenhet användas. Dessa enheter måste vara utbytbara då programmet ju kommer att köras på apparater av olika märken, så för dem definierades gränssnitt i Domain-lagret (se figur 4) som implementeras i Infrastructure-lagret. Streckkodsläsaren läser av biljetter som ska kontrolleras, och efter en lyckad kontroll spelas ett ljud upp beroende på resultatet av kontrollen.

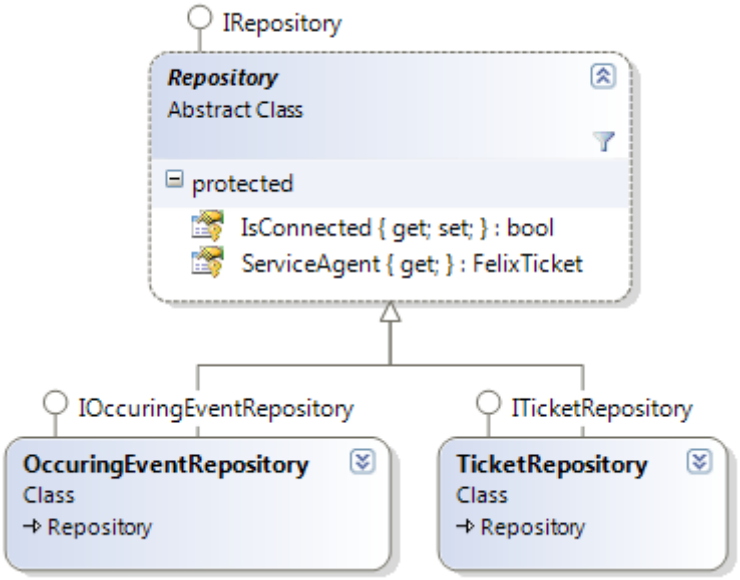


Figur 4. Gränssnitt för de olika hårdvaruenheter som används i samband med programmet. Ljudenhetsgränssnittet erbjuder en metod för att spela upp ett ljud och streckkodsläsargränssnittet tillhandahåller metoder för att slå på och av streckkodsläsaren samt en händelsenotifikation som kan kopplas till en händelseprocedur för när en streckkod har lästs in.

3.2.2 Infrastructure-lagret

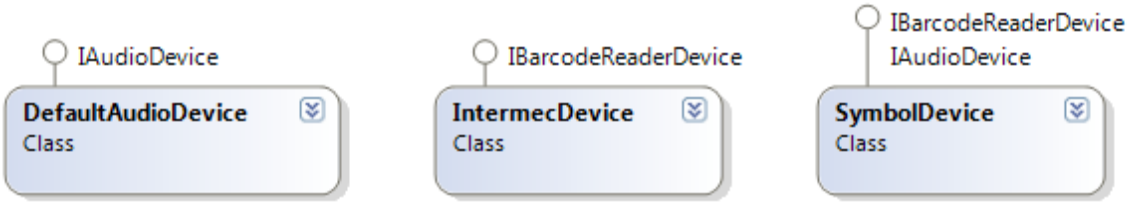
I Infrastructure-lagret implementeras de olika gränssnitten för Repositories och hårdvaruenheter som definierats i Domain-lagret (se kapitel 3.2.1).

I figur 5 visas ett digram över klasserna som implementerar de Repository-gränssnitt som definierats i Domain-lagret. Eftersom de båda använder samma web service-instans (se kapitel 2.3), placerades denna i en basklass som de båda ärver.



Figur 5. Repository-klasserna.

I figur 6 visas ett diagram över klasserna som implementerar de gränssnitt för ljud- och streckkodsläsarenheter som definierats i Domain-lagret. Klassen IntermecDevice implementerar streckkodsavläsning för handdatorer av märket Intermec med tillverkarens egen SDK. Klassen DefaultAudioDevice använder inbyggda klasser i .NET Compact Framework (se kapitel 2.2) för uppspelning av ljud. Dessa kunde inte användas för ljuduppspelning på handdatorer av märket Symbol, så klassen SymbolDevice implementerar både streckkodsavläsning och ljuduppspelning med tillverkarens egen SDK. Stöd för nya hårdvaruenheter implementeras genom att skapa nya klasser för dessa som implementerar gränssnitten IBarcodeReaderDevice och/eller IAudioDevice.



Figur 6. Implementationer för de olika hårdvaruenheterna.

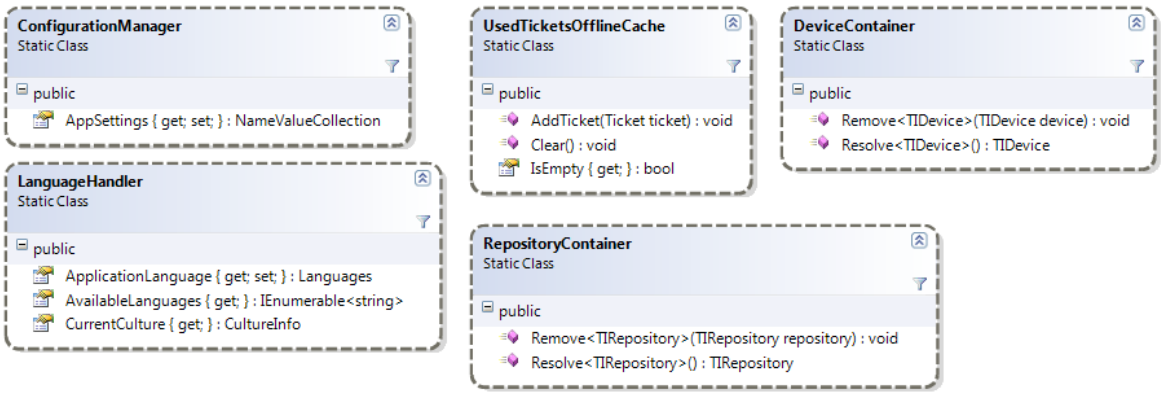
I figur 7 visas ett diagram över Cross-cutting-klasserna i programmet. Hit hör bland annat IoC-behållarna (se kapitel 2.7.5) DeviceContainer och RepositoryContainer. Klassen DeviceContainer förser användare av IDevice-gränssnitten (se figur 4) med implementationer från Infrastructure-lagret (se figur 6). Vilken implementation som ska

användas bestäms av programmets konfigurationsfil (se kodexempel 1). Eftersom klassen System.Configuration.ConfigurationManager saknas i .NET Compact Framework behövde en egen klass för för hantering av konfigurationsfilen implementeras.

Kodexempel 1. Programmets konfigurationsfil, där man bland annat anger vilken apparat programmet körs på.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <appSettings>
    <add key="ServerURL" value="http://a.b.c.d:port" />
    <add key="BarcodeReaderDevice" value="Intermec"/>
    <!--<add key="BarcodeReaderDevice" value="Symbol" />-->
    <add key ="BarcodeReaderTimeout" value="1000"/>
    <add key="AudioDevice" value="Default"/>
    <!--<add key="AudioDevice" value="Symbol"/>-->
  </appSettings>
</configuration>
```

Det konstaterades att stöd för byte av språk för användargränssnittet genom System.Threading.Thread.CurrentUICulture saknas i .NET Compact Framework. Därför behövdes en egen lösning för språkhantering, vilket implementerades i klassen LanguageHandler.

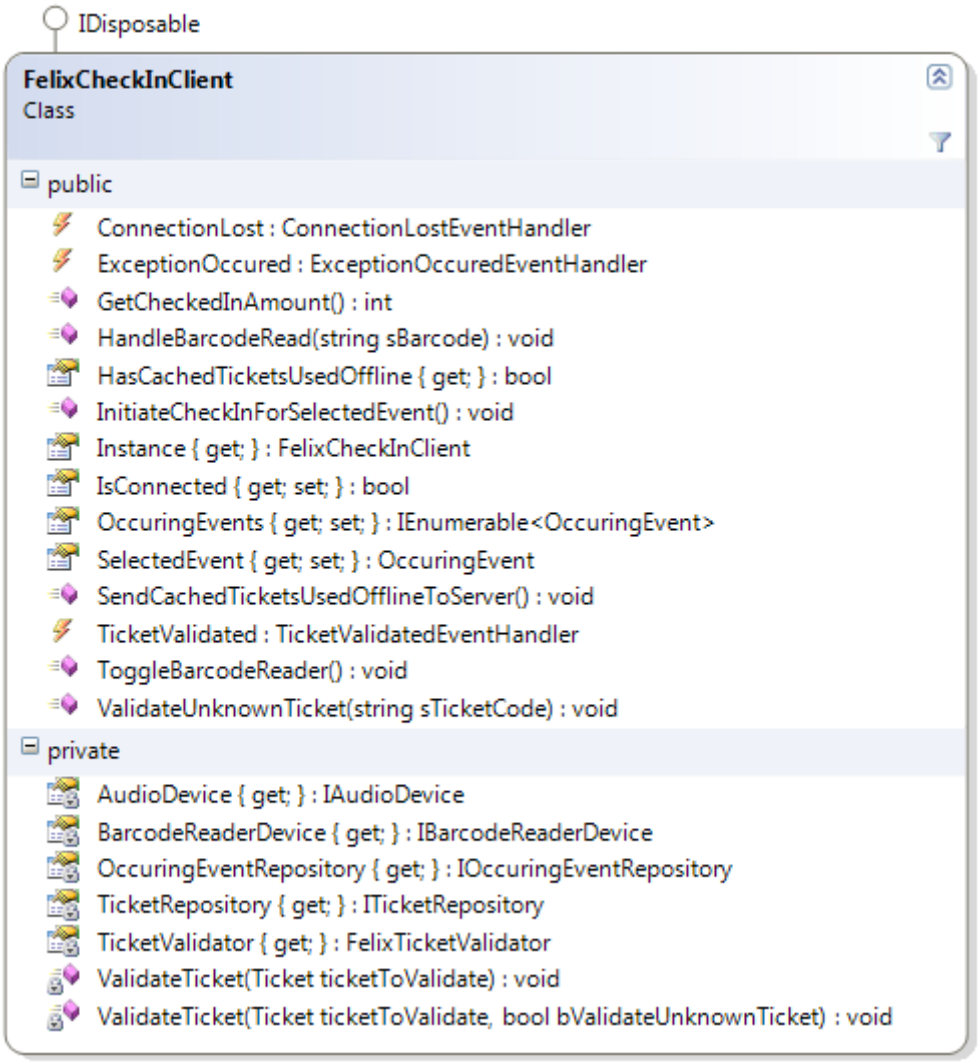


Figur 7. Cross-cutting-klasserna i detta program.

3.2.3 Application-lagret

Application-lagret består av en enda klass (se figur 8) som fungerar som en fasad (se kapitel 2.8.2) mot resten av programmets arkitektur. Denna har implementerats som en singleton (se kapitel 2.8.1) för att garantera att endast en instans av klassen existerar under körningen av programmet. Klassen tillhandahåller metoder som Presentation-lagret

behöver, såsom en metod för att slå på och av streckkodsläsaren och en metod för att kontrollera en manuellt inmatad biljettkod. Dessutom notifierar den om händelser, såsom när ett en biljett har blivit kontrollerad eller när ett fel har inträffat, vilket hanteras av händelseprocedurer i Presentation-lagret.



Figur 8. Klassen FelixCheckInClient, en fasad mot resten av programarkitekturen.

3.2.4 Presentation-lagret

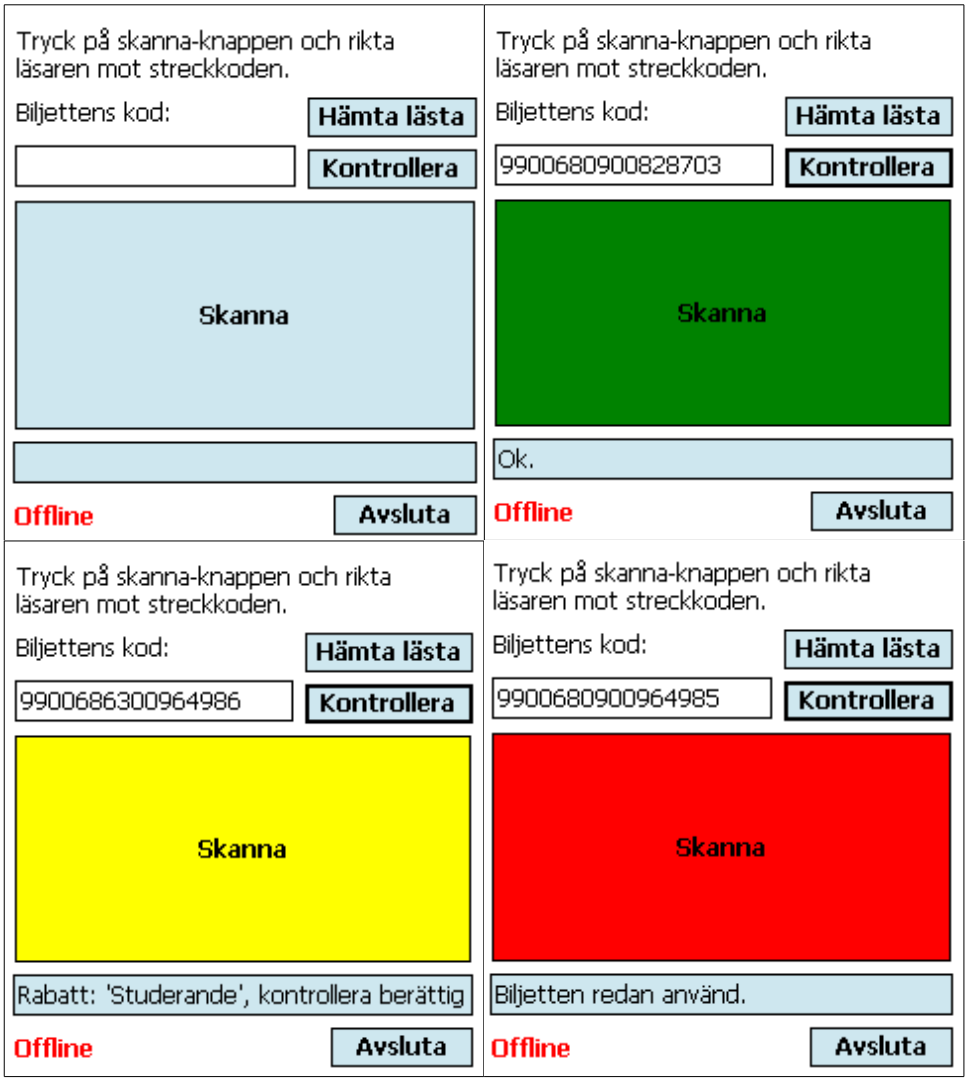
Presentation-lagret implementerades som en Windows Phone 6-applikation med ett lämpligt grafiskt användargränssnitt för handdatorer med detta operativsystem. Programmet delades upp i två binärmoduler på så vis att Presentation-lagret placerades i en EXE-fil, medan resten av programmet placerades i en DLL-fil som denna använder sig av.

På så vis är användargränssnittet utbytbart. Vill man exempelvis bygga stöd för att köra programmet på en PC-dator med tillhörande streckkodsläsare behöver man bara utveckla en ändamålsenlig användargränssnittsmodul som använder sig av DLL-filen med den egentliga programlogiken.

Användargränssnittet består av tre vyer. När programmet startar väljer man först språk (se figur 9). Därefter hämtas en lista på dagens föreställningar från web servicen (se kapitel 2.3) och man får välja vilken som ska användas (se figur 10). Här kan man även välja att använda programmet i Offline-läge (se kaptel 3.3). Till slut hämtas information om de biljetter som sålts till den valda föreställningen och programmets huvudvy visas (se figur 11). I huvudvyn kan man slå på och av streckkodsläsaren, och vid behov även mata in biljettkoder för hand. Dessutom finns en knapp för att få reda på hur många biljetter som hittills har blivit inlästa. När en biljett har kontrollerats visas resultatet med färg och text samt med en ljudsignal.

Figur 9. Vy för val av språk.

Figur 10. Vy för val av evenemang.



Figur 11. Programmets huvudvy samt exempel på de viktigaste resultaten av en biljettkontroll.

3.3 Offline-läge

Programmet kan användas utan Internet-uppkoppling efter att all evenemangs- och biljettinformation har hämtats från web servicen. Offline-läget kan aktiveras manuellt när programmet startar, och ifall programmet av någon anledning tappar kontakten till web servicen (se kapitel 2.3) under programkörningen aktiveras Offline-läget automatiskt. När biljetter läses in i Offline-läget skrivs deras biljettkoder till en fil med hjälp av klassen UsedTicketsOfflineCache (se figur 7). Nästa gång programmet körs ombeds användaren uppdatera web servicen med den lokalt lagrade informationen om förbrukade biljetter.

4 Resultat och diskussion

Resultatet av arbetet är ett program för kontroll av inträdesbiljetter till olika kulturevenemang som fungerar på handdatorer av märkena Symbol och Intermec. Därtill går det smidigt att i framtiden även stöda andra enheter med Windows-baserade operativsystem. Programmet kan användas på svenska, finska och engelska, och stöd för fler språk kan tilläggas vid behov. Användningen av lokal biljettinformation och asynkrona web service-anrop (se kapitel 2.3) har ökat programmets användbarhet betydligt, och Offline-funktionaliteten (se kapitel 3.3) gör att programmet kan användas även på platser med begränsad Internet-åtkomst. Programmet har varit i produktionsbruk i drygt ett års tid, och det har använts i ett tiotal av Studio Tickets evenemang samt vid Seinäjoen kaupunginteatteri.

Utvecklingen av detta program har gett mig en ökad förståelse för Domain Oriented N-Layered Architecture, samt för de problem man stöter på när man följer principerna hos arkitekturen, såsom Inversion of Control. En sak jag efteråt konstaterat är att lösningen för Repository-klasserna i Infrastructure-lagret (se figur 5) är onödigt klumpig. I stället för att ha två olika klasser som implementerar de olika Repository-gränssnitten (se figur 3) med en basklass som håller reda på web servicen som de kommunicerar med, hade man kunnat ha en enda klass för hantering av web servicen som implementerar båda Repository-gränssnitten. Detta anser jag att vore en smidigare och snyggare lösning eftersom det representerar vad som händer i verkligheten, nämligen att det finns en web service som tillhandahåller metoder för att hämta information om föreställningar och sålda biljetter till

evenemang, det vill säga syftet med de två olika Repository-gränssnitten. Detta kan jämföras med att klassen SymbolDevice (se figur 6) implementerar både gränssnittet för ljudenheter och gränssnittet för streckkodsläsare, eftersom den är en apparat som uppfyller båda funktionerna. För övrigt tycker jag att detta arbete resulterade i en lyckad lösning på de problem som fanns i uppdragsgivarens tidigare program för biljettkontroll, och samtidigt är det ett bra exempel på vad man kan åstadkomma genom användning av Domain Oriented N-Layer Architecture.

Källförteckning

Differences Between the .NET Compact Framework and the .NET Framework. (u.å.). http://msdn.microsoft.com/en-us/library/2weec7k5%28v=vs.90%29.aspx (hämtat: 12.7.2013).

Download Windows Mobile 6. (u.å.). *Download Windows Mobile 6 Professional and Standard Software Development Kits Refresh from Official Microsoft Download Center.* http://www.microsoft.com/en-us/download/details.aspx?id=6135 (hämtat: 11.7.2013).

EMDK for .NET v2.0. (2008). https://docs.symbol.com/KanisaPlatform/Publishing/903/11777_f.html (hämtat: 28.6.2013).

Evans, E. (2004) *Domain Driven Design – Tackling Complexity in the Heart of Software.* Boston (MA): Addison-Wesley.

Gamma, E., Helm R., Johnson R. & Vlissides J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading (MA): Addison-Wesley.

Intermec Applications & Software. (u.å.). *Intermec Applications & Software: Development Tools: Developer Resource Kits.* http://www.intermec.com/products/devresourcekit/downloads.aspx (hämtat: 11.7.2013).

Mössenböck, H., Beer, W., Birngruber, D. & Wöß, A. (2004) *.NET Application Development with C#, ADO.NET, ASP.NET and Web Services.* New York: Addison-Wesley.

Release Notes - EMDK for .NET v2.3. (2010). https://docs.symbol.com/ReleaseNotes/Release%20Notes%20-%20EMDK-M-020305-UP1D.htm (hämtat: 2.7.2013).

Torre de la, C., Zorrilla, U., Calvarro, J. & Ramos, M. (2011) *N-Layered Domain-Oriented Architecture Guide with .NET 4.0.* Krasis Press u.o.