Datavetenskap

Christine Andersson & Hanna Karlsson

Introduktion till webbtjänster

Examensarbete, C-nivå

2003:14

Introduktion till webbtjänster

Christine Andersson & Hanna Karlsson

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

Christine Andersson

Hanna Karlsson

Godkänd, 2003-06-03

Handledare: Robin Staxhammar

Examinator: Stefan Lindskog

Sammanfattning

Denna uppsats tar upp möjligheter för hur den nya teknik som kallas för webbtjänster kan användas och utnyttjas tillsammans med affärssystem. Diveria AB har integrerat en webbtjänst i ett affärssystem som heter Navision och söker efter fler tjänster som skulle kunna lämpa sig att utnyttjas på ett liknande sätt. Med detta som utgångspunkt undersöktes ett antal webbtjänster för att utröna huruvida de var lämpliga att integrera i affärssystemet Navision.

I sökandet efter webbtjänster fann vi att de flesta tjänster som idag finns tillgängliga är prototyper, endast skapade i syfte att undersöka möjligheterna med tekniken. Då dessa inte lämpar sig för integration, beskrivs i uppsatsen även skapandet, d v s design och implementation, av en egen prototyp. Syftet med att konstruera denna var att få ökade kunskaper om hur webbtjänster kan skapas och användas. Prototypen implementerades i C++ med utvecklingsmiljön Microsoft Visual Studio .NET.

Introduction to web services

Abstract

This report considers possibilities of how the new technique known as web services can be used and deployed in collaboration with legacy systems. Diveria AB has integrated a web service with a legacy system called Navision and they are now searching for other suitable services to integrate. Based on this, a number of web services were examined in order to determine whether or not they were suitable for integration with Navision.

During the search for web services we discovered that most of the existing services are prototypes created only to explore the possibilities with the new technique. Since these are not suitable for integration this report also describes the development of a prototype. The purpose of implementing this prototype was for us to gain understanding of how web services can be developed and deployed. The prototype was implemented using C++ and Microsoft Visual Studio .NET.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

1.1 Bakgrund

I informationssamhället, där alltmer information samlas i datorer istället för på papper, blir användandet av olika affärssystem, se avsnitt 2.2, allt vanligare. Eftersom systemen tenderar att bli mycket stora delas de ofta upp i flera mindre delsystem. Detta ger ett mer hanterbart affärssystem men det genererar också nya problem. Ett sådant problem är att inmatning av information ofta sköts manuellt av olika människor, vilket gör att det kan vara svårt att veta om alla uppgifter i systemet är korrekta.

Postens Adressregister [1], PAR, se avsnitt 2.3, tillhandahåller en databas, näringslivsregistret Parad, med aktuell och korrekt information om Sveriges alla företag. Exempel på information som finns i Parad är namn på kontaktpersoner, adresser, telefonnummer och liknande men Parad innehåller också information som omsättning och antal anställda hos företaget. Via PAR:s webbserver är det möjligt att utnyttja webbtjänsten One-by-One [2], OBO, se avsnitt 2.3, för att söka och köpa tillgänglig information om ett specifikt företag, direkt via nätet. Det är också möjligt att använda OBO gratis för att söka efter företag, men då fås inte lika mycket information.

Diveria AB [3], se avsnitt 2.1, har utvecklat en lösning, Diveria PAR Direkt, se avsnitt 2.4, som gör det möjligt att koppla ihop affärssystemet Microsoft Navision [4], med OBO. Idén går ut på att det skall vara enkelt att hålla informationen i affärssystemet uppdaterad. Denna utökning av affärssystemets funktionalitet består av ett gränssnitt, en COM-komponent[1], Component Object Model, COM, som kommunicerar med OBO och ett grafiskt användargränssnitt som är inbyggt i affärssystemet [2]. Gränssnittet i affärssystemet är utvecklat för ett specifikt affärssystem, nämligen Navision. Diveria är nu intresserade av att undersöka om det finns fler webbtjänster som affärssystemet Navision skulle kunna utnyttja med hjälp av den utökade funktionaliteten i COM-komponenten.

[1] COM, designteknik som används för att dela upp program i utbytbara moduler

1.2 Mål och syfte med uppsatsen

Ett mål med arbetet är att integrera fler webbtjänster med affärssystemet Navision med hjälp av Diveria PAR Direkt. Detta mål kan delas upp i följande delmål:

- Att hitta fler webbtjänster som är användbara i affärssystemet Navision.
- Att utreda om, och i så fall hur tjänsterna kan integreras med hjälp av Diveria PAR Direkt.
- Att integrera tjänsterna i Navision.

Ett annat mål är, om det visar sig att det inte finns några webbtjänster av intresse, att implementera en prototyp av en ny webbtjänst för att ge insikt i hur en webbtjänst konstrueras.

1.3 Uppsatsens struktur

Uppsatsen består, utöver det inledande kapitlet, av sju andra kapitel. Kapitel 2 ger en översiktlig beskrivning av de system mellan vilka Diveria PAR Direkt kommunicerar samt en kort beskrivning av hur Diveria PAR Direkt fungerar. När en överblick har getts fördjupas kunskaperna om webbtjänster i kapitel 3, vilket innehåller en utredning av begreppet webbtjänster och dess bakomliggande tekniker. Detta för att ge läsaren grundläggande kunskaper som gör det möjligt att förstå och tillgodogöra sig uppsatsens efterföljande kapitel. I kapitel 4 beskrivs förfarandet för att hitta, undersöka och integrera webbtjänster, som är användbara i affärssystemet Navision. Eftersom utkomsten av arbetet i kapitel 4 var att ingen användbar webbtjänst hittades, valdes i samråd med uppdragsgivaren att utveckla en egen webbtjänstprototyp. Designen för denna beskrivs i kapitel 5, och implementationen i kapitel 6. Resultaten av arbetet beskrivs i kapitel 7 där vi också ger rekommendationer. I kapitel 8 följer en summering av projektet med de slutsatser vi dragit. För att underlätta läsningen har en bilaga med de akronymer som används i uppsatsen lagts till, se Bilaga A.

2 Beskrivning av Diveria AB:s system

I detta kapitlet framställs hur det system som Diveria skapat ser ut. Här beskrivs de olika delarna som utgör systemet, d v s affärssystemet Navision, PAR och OBO samt Diveria PAR Direkt. Detta avser att ge läsaren en förståelse för vilka uppgifter var och en av de ingående delarna har. Kapitlet tar även upp hur en sökning i systemet utförs för att illustrera hur de olika delarna verkar tillsammans. Teknikerna som utnyttjas mellan systemen beskrivs närmare i kapitel 3.

2.1 Diveria AB

Diveria AB är ett företag i Karlstad som skräddarsyr affärssystemslösningar åt mindre och medelstora företag. Diveria arbetar med ett affärssystem som heter Navision, och är ett så kallat Navision Solutioncenter [3]. Med Navision kan företag via ett Navision Solutioncenter få ett affärssystem som är anpassat speciellt för företagets behov.

2.2 Affärssystem

I ett affärssystem lagras all information om ett företags verksamhet. Eftersom systemen lagrar stora mängder information blir de omfattande och delas ofta upp i flera mindre delsystem. Syftet med affärssystemet är att användare av systemet alltid skall ha tillgång till korrekt och aktuell information vid rätt tidpunkt. Inom ett företag finns flera avdelningar vilka ofta avspeglas i företagets affärssystem. Olika avdelningar inom företaget begränsas till att endast få tillgång till de delar av systemet som rör avdelningens verksamhet. Delar som ingår i ett affärssystem är till exempel kundregister, leverantörsregister, samt hantering av ekonomi, lager med mera. Affärssystemet kan vara ett hjälpmedel för att sammanställa uppgifter som sedan ligger till grund vid beslutsfattning.

2.2.1 Navision

Eftersom affärssystemet Navision är sammansatt av ett antal oberoende moduler, kan varje företag välja att använda bara de delar som är nödvändiga för företagets behov. Då betalas licensavgifter endast för de utvalda modulerna. Ytterligare funktionalitet eller specialanpassningar utöver vad Navisions moduler erbjuder kan i regel utvecklas av ett solutioncenter, som t ex Diveria AB.

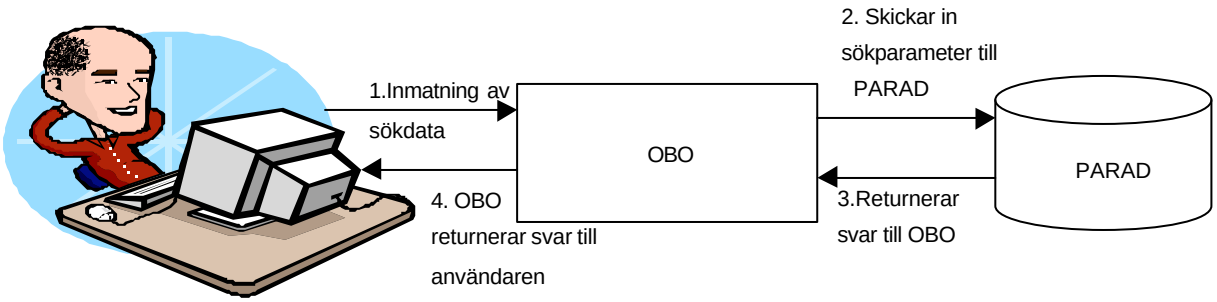
Modulerna täcker åtskilliga områden inom administration, exempel på dessa är hantering av: redovisning, kunder, order, lager, fakturering, inköp, affärsstöd, projekt, personal, resurser, fjärruppkoppling och e-handel [4].

2.3 PAR och OBO

Postens AdressRegister, PAR, är ett företag vars affärsidé bland annat går ut på att sälja aktuell och korrekt information om Sveriges alla företag. Denna information finns samlad i näringslivsregistret Parad. Företag kan, genom att införskaffa användarkonton hos PAR, låta sina anställda söka och köpa information ur PAR:s register med hjälp av webbtjänsten One-by-One, OBO. Kostnaden för varje köp beror på mängden information som köps. Betalningen görs via användarkontot, på vilket företaget i förväg betalar in en summa.

2.4 Utökning av Navision – Diveria PAR Direkt

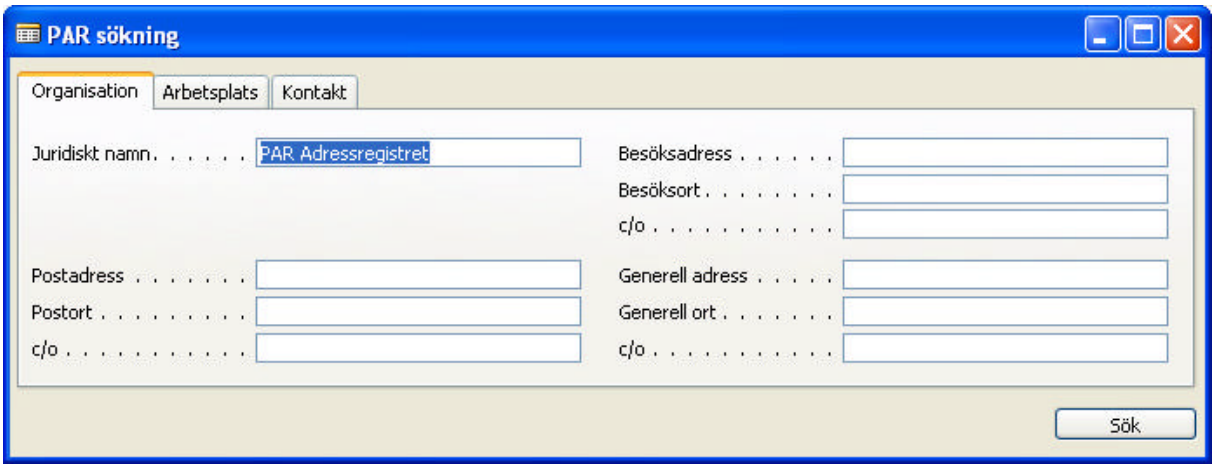
Tillägget till Navision, Diveria PAR Direkt, gör det möjligt att använda webbtjänsten OBO direkt i affärssystemet. Diveria PAR Direkt består i huvudsak av två delar, en del som sitter inbyggd i affärssystemet och en del som ligger utanför själva affärssystemet. Denna del är en COM-komponent och sköter kommunikationen med OBO. Det grafiska användargränssnittet med vilket tjänsten presenteras i affärssystemet får den att framstå som en del av systemet. Där finns alla funktioner hos OBO tillgängliga för användaren och här visas resultatet av sökningar i Parad. I Figur 1 visas händelseförloppet för en sökning.



Figur 1 En sökning med OBO

En sökning i PAR via affärssystemet Navision kan gå till på följande sätt:

a) Användaren väljer PAR från en meny i Navision och får då upp skärmbilden i Figur 2. Där anges t ex namnet på ett företag, som användaren vill söka information om.



Figur 2 En företagssökning

b) Sökinformationen vidarebefordras till COM-komponenten.

c) COM-komponenten loggar in på OBO, utför sökningen med hjälp av den inmatade informationen, returnerar resultatet och loggar ut.

d) Affärssystemsdelen av Diveria PAR Direkt presenterar alla matchningar med adress och telefonnummer för användaren, se Figur 3. Denna information är gratis.

Figur 3 Resultat av en sökning

e) Användaren kan sedan välja ut en matchning som denne vill köpa för att få all tillgänglig information lagrad i det egna systemet. I kolumnen ’Kund köpt’, längst till höger i Figur 3, visas om informationen om företaget redan innehas eller ej.

3 Den tekniska miljön kring webbtjänster

Detta kapitel avser att ge en grundläggande bild av vad webbtjänster är samt vilka tekniker som får dem att fungera. Kapitlet tar upp webbtjänster på ett mer generellt plan med bland annat syfte och funktion hos webbtjänster, men det tar även upp viktiga hörnstenar såsom eXtensible Markup Language, XML, Simple Object Access Protocol, SOAP och Web Service Description Language, WSDL. Den avslutande, mer teniskt inriktade delen av kapitlet lägger grunden inför analysen av webbtjänster i kapitel 4. Informationen i detta kapitel är, om inget annat anges, hämtad ur [5], [6], [7], [8].

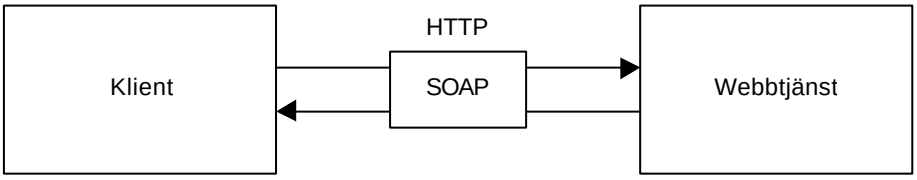
3.1 Webbtjänster

Uppfattningarna om vad som skall kallas för en webbtjänst går idag isär. Det finns en uppsjö av definitioner vilket kan leda till förvirring. Ett av skälen till detta kan vara att engelskans web service, när det direktöversätts till webbtjänst, får en något felaktig betydelse. Ordet webbtjänst för helt enkelt tankarna till en tjänst tillgänglig via Internet, vilket gör att en hel del företag säger sig ha webbtjänster och menar då tjänster som användare interagerar med via en hemsida.

Generellt kan sägas att en webbtjänst är en applikation till vilken det är möjligt att göra metodanrop, trots att applikationen inte befinner sig på lokal maskin. Detta sker med en teknik som kallas Remote Procedure Calls, RPCs [9]. Kriterierna för vad som utgör en ”riktig” webbtjänst är ännu inte bestämda, men en allmänt vedertagen standard börjar dock växa fram. För det första skall tjänsten beskrivas med Web Service Description Language [10], WSDL, se avsnitt 3.2.2. För det andra skall kommunikationsanropen till och svaren från tjänsten gå över protokollet Simple Object Access Protocol [11], SOAP. Och för det tredje skall tjänsten strukturera resultaten med XML.

Något som ofta poängteras är att tekniken för webbtjänster egentligen inte är ny. Det är tillämpningen av ett antal kända tekniker, främst då RPC, XML, och HTTP, Hyper Text Transfer Protocol, som tillsammans har gjort webbtjänster möjliga. Principen är att en användare av webbtjänsten, en klient i form av en applikation, gör ett RPC till webbtjänsten genom att med HTTP-protokollet skicka ett SOAP-meddelande till den. Svaret från tjänsten blir ett nytt SOAP-meddelande som även det returneras med hjälp av HTTP, se Figur 4. Detta

gör, eftersom HTTP är en allmänt vedertagen standard, att dessa meddelanden kan transporteras överallt på Internet.



Figur 4 Principen för en webbtjänst

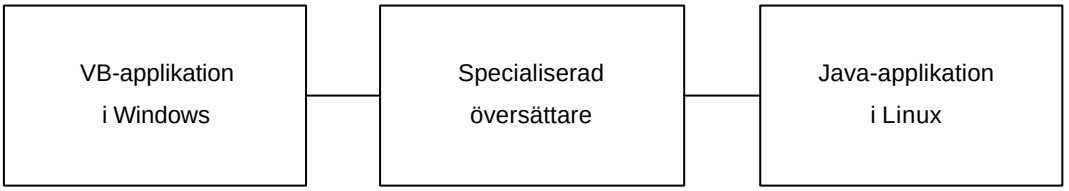
3.1.1 Vad är syftet med webbtjänster?

Eftersom det är själva programlogiken som görs tillgänglig i en webbtjänst skall tjänsten inte behöva tillhandahålla något grafiskt användargränssnitt för användarna, detta skall istället tillhandahållas av de klienter som använder tjänsten. Webbtjänster är ett koncept som redan har fått flera skilda användningsområden eftersom det är möjligt att använda dem för att lösa ett antal olika problem:

Utvecklare av hemsidor kan välja att använda webbtjänster för att ge interaktiva sidor en bättre struktur, genom att skilja presentationen av en hemsida från logiken i den. Hemsidan skulle i så fall fungera som en klient till webbtjänsten [12].

Företag eller organisationer som vill sälja eller tillhandahålla informationstjänster via Internet kan göra detta genom att konstruera en webbtjänst och sedan låta kunder själva skapa klienter för att använda tjänsten, gratis eller mot betalning, t ex i fallet med PAR och deras webbtjänst OBO. Utvecklaren tillhandahåller endast tjänsten och det är användaren som måste skapa klienten för att kunna använda tjänsten.

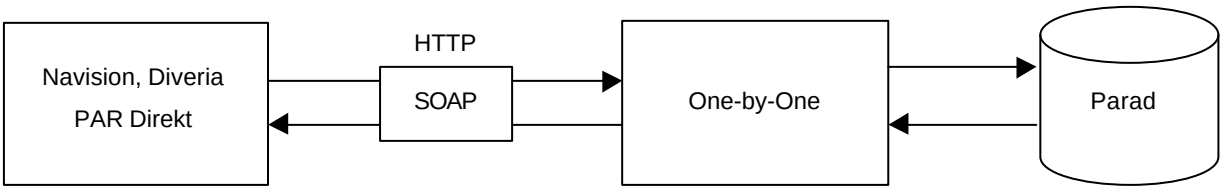
Ytterligare ett användningsområde för webbtjänster gäller samarbete, eller förmågan att samarbeta, mellan applikationer, interoperability på engelska. Utvecklare skall dra nytta av att existerande applikationer enkelt skall kunna samarbeta med varandra. Det är tänkt att webbtjänster skall användas för att skapa ett plattformsneutralt sätt för applikationer att enkelt kommunicera. Tidigare har det inneburit problem att få applikationer som är skrivna för olika system, t ex VB-applikationer för Windows och Java-applikationer för Linux, att kommunicera med varandra. Så länge applikationerna tillhör samma system är det möjligt att använda de kommunikationsverktyg som existerar för respektive system. Men om det är nödvändigt att gå utanför den egna systemmiljön är det nödvändigt att koppla in ett mellanled, en översättare, se Figur 5. Översättaren måste hantera båda applikationernas funktionalitet och de datatyper som används i de olika systemen [12].



Figur 5 Kommunikation mellan olika systemmiljöer

Åtskilliga hoppas att webbtjänster skall komma att erbjuda en möjlighet att kommunicera på applikationsnivå, oberoende av systemmiljö. Detta skulle kunna vara till stor nytta vid kommunikation mellan företagssystem, t ex vid sammanslagning av företag eller enbart för informationsutbyte mellan olika företag. Vid integration av nätverk har det tidigare varit nödvändigt att skräddarsy varje ny koppling mellan nätverken, vilket i längden gör det både kostsamt och komplicerat att ytterligare bygga ut systemet.

I fallet med Postens Adress Register, PAR, och Diveria, så vill PAR att så många kunder som möjligt skall kunna utnyttja databasen Parad oavsett vilken miljö de använder sig av. För att åstadkomma detta har PAR skapat webbtjänsten One-by-One som hjälper kunden att utföra sökningar och hämta information från Parad. Diveria har skapat en klient till webbtjänsten, med ett grafiskt användargränssnitt för att göra det möjligt att använda OBO i affärssystemet Navision, se Figur 6 för en översiktlig bild.



Figur 6 Webbservice och klient

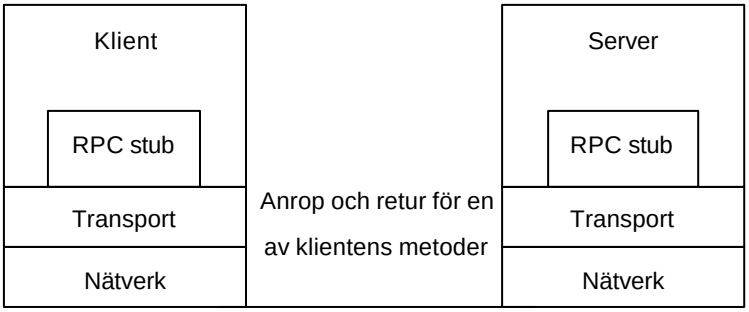
3.1.2 Hur fungerar webbtjänster?

Webbtjänster kan se ut på många olika sätt, de enklaste kanske bara returnerar ett svar, eller utför enklare beräkningar, vid anrop. En del tjänster används för att hämta information ur databaser men med de designmöjligheter som finns idag är detta långt ifrån allt som går att göra. En webbtjänst skulle t ex även i sin tur kunna använda sig av andra webbtjänster för att få tag i informationen. Genom att på detta sätt utnyttja webbtjänster likt komponenter, skulle ett antal mindre webbtjänster kunna kombineras till större och mer komplext sammansatta webbtjänster. Dessa större webbtjänster skulle på så sätt kunna presentera sina klienter ett relativt enkelt gränssnitt med avancerad underliggande logik. Detta efterliknar den metod

enligt vilken program numera konstrueras, d v s i ett antal mindre moduler istället för ett enda stort program [8].

När en användare skall skapa en klient till en webbtjänst är det inte nödvändigt för denne att känna till hur tjänsten är implementerad. För att anropa tjänsten behöver användaren endast veta vilka metoder som finns tillgängliga samt hur metodernas signatur ser ut, d v s vilka meddelanden tjänsten tar emot. Användaren behöver alltså veta hur webbtjänstens gränssnitt ser ut. All information om en webbtjänsts gränssnitt finns samlad i tjänstens Web Service Description Language-dokument, WSDL-dokument, se avsnitt 3.2.2.

RPCs gör det möjligt för applikationer att anropa programlogik som inte befinner sig på den lokala datorn, på ett sätt som får det att verka som om den gjorde det för den anropande applikationen. Anrop av den här typen används ofta i klientserver-lösningar där en klient tror sig göra ett lokalt funktionsanrop, men i själva verket anropar en server och sedan erhåller ett svar från den. Det går till så att både klienten och servern genererar var sin stub som fungerar som ett substitut för den programlogik som skall anropas. Klienten gör anropet till sin stub, som representerar den avlägset belägna logiken för klienten, och tror på så vis att logiken finns på den lokalt hos klienten. Klientens stub sköter kommunikationen med serverns stub och vidarebefordrar anropet dit. Serverns stub i sin tur, anropar den riktiga logiken i servern, och returnerar det resultat servern ger på anropet, till klientens stub. Klientens stub vidarebefordrar serverns svar till klienten och så är klienten klar med sitt anrop, se Figur 7.



Figur 7 Principen för RPC

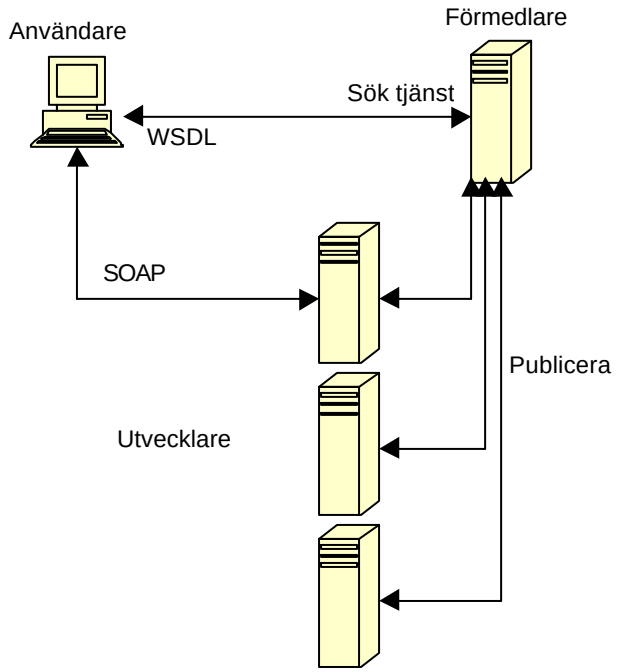
Webbtjänster lämpar sig väl för att använda RPC då klienten gör en förfrågan till tjänsten och inväntar svaret. Utifrån informationen i tjänstens beskrivning, WSDL-dokumentet, är det möjligt att generera en stub, kallad proxy eller proxy-klass. Denna proxy får det att verka som om webbtjänsten finns att anropa lokalt på datorn. Program gör då sina anrop till proxyn som i sin tur anropar webbtjänsten. Om en webbtjänst använder RPC är det även möjligt för användare att i klienter, med hjälp av informationen i WSDL-dokumentet, göra manuella anrop till tjänsten. I ett sådant fall genereras ingen proxy.

Klienter kan genom att utnyttja Hyper Text Transfer Protocol, HTTP, kommunicera med webbtjänster på tre olika sätt, antingen med ett anrop med GET-metoden[2], ett anrop med POST-metoden[3] eller genom att ett SOAP-meddelande skickas i ett anrop med POST-metoden [12].

Vid GET-kommandot läggs efterfrågad information till i slutet av URLen som skickas till servern. Vid POST-kommandot skickas data i kroppen på http-meddelandet [13]. Fördelen med SOAP är att informationen blir mer strukturerad jämfört med HTTP-kommandon.

3.1.3 Hur skall användare hitta webbtjänster?

Webbtjänster publiceras på Internet med hjälp av förmedlare. Utvecklarna registrerar sina tjänster hos olika förmedlare som publicerar information om tjänsten, i synnerhet WSDL-dokumentet. Intresserade användare letar igenom förmedlarens register efter en lämplig tjänst och får enkelt tillgång till information om tjänsten. Förmedlaren fungerar som en mellanhand mellan utvecklaren och användaren, se Figur 8.

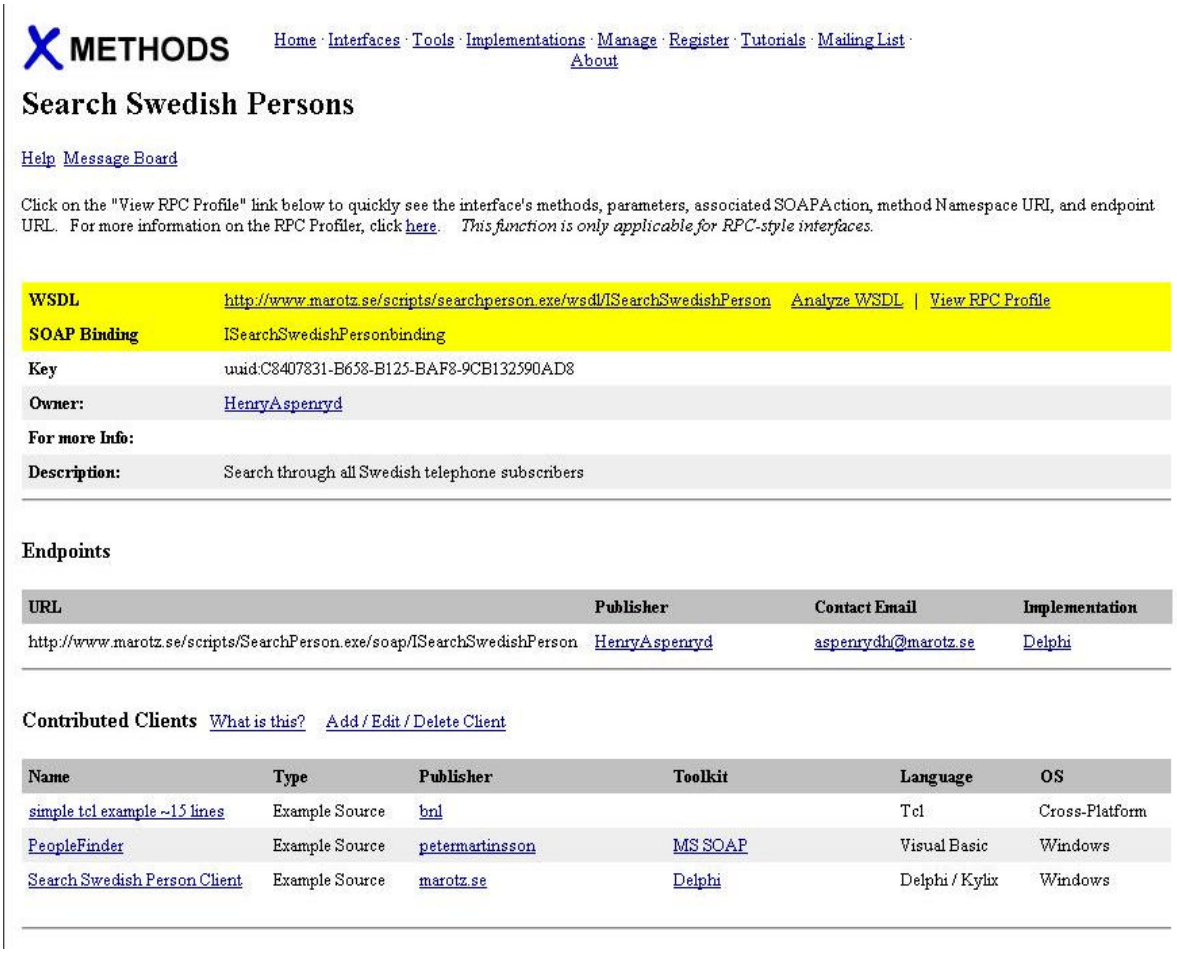


Figur 8 Förmedling av webbtjänster

Detta underlättar både för utvecklare och användare av webbtjänster att komma i kontakt med varandra. Användare hämtar WSDL-dokument från förmedlare för att få en beskrivning av hur webbtjänstens gränssnitt ser ut och hur den skall användas. Förmedlare erbjuder ibland även ytterligare funktioner; t ex att informera användare ifall tjänstens gränssnitt förändras,

[2] HTTP GET, används för att hämta data från en webbserver.

eller om tjänsten helt enkelt upphör. Detta är viktigt eftersom det påverkar klienter som använder tjänsten. När användaren hittat en tjänst som denne vill titta närmare på, kan ytterligare information om tjänsten presenteras, som i Figur 9.



XMETHODS Home · Interfaces · Tools · Implementations · Manage · Register · Tutorials · Mailing List · About

Search Swedish Persons

Help Message Board

Click on the "View RPC Profile" link below to quickly see the interface's methods, parameters, associated SOAPAction, method Namespace URI, and endpoint URL. For more information on the RPC Profiler, click here. *This function is only applicable for RPC-style interfaces.*

WSDL	http://www.marotz.se/scripts/searchperson.exe/wsdl/ISearchSwedishPerson Analyze WSDL | View RPC Profile
SOAP Binding	ISearchSwedishPersonbinding
Key	uuid:C8407831-B658-B125-BAF8-9CB132590AD8
Owner:	HenryAspenryd
For more Info:	
Description:	Search through all Swedish telephone subscribers

Endpoints

URL	Publisher	Contact Email	Implementation
http://www.marotz.se/scripts/SearchPerson.exe/soap/ISearchSwedishPerson	HenryAspenryd	aspenrydh@marotz.se	Delphi

Contributed Clients What is this? Add / Edit / Delete Client

Name	Type	Publisher	Toolkit	Language	OS
simple tcl example ~15 lines	Example Source	bnl		Tcl	Cross-Platform
PeopleFinder	Example Source	petermartinsson	MS SOAP	Visual Basic	Windows
Search Swedish Person Client	Example Source	marotz.se	Delphi	Delphi / Kylix	Windows

Figur 9 Information om tjänst från förmedlare

I Figur 9 listas information om tjänsten som; WSDL-dokumentet, adressen till tjänsten, en kort beskrivning av dess funktionalitet, vilka metoder den har mm. Här listas även information om utvecklaren som t ex hur denne vid behov kan kontaktas. Om användaren vill ha mer information om webbtjänstens metoder kan denne enkelt få fram koncentrerad information om tjänstens metoder ur tjänstens WSDL-dokument, presenterad som i Figur 10. Där beskrivs metoden ”XMLSearchAddressEx” med inparametrar och returvärden samt av vilka typer de är.

[3] HTTP POST, används för att skicka block med data till en webbserver.

Method Name	XMLSearchAddressEx	
Endpoint URL	http://www.marotz.se/scripts/SearchPerson.exe/soap/ISearchSwedishPerson	
SOAPAction	urn:SearchSwedishPersonIntf-ISearchSwedishPerson#XMLSearchAddressEx	
Method Namespace URI	urn:SearchSwedishPersonIntf-ISearchSwedishPerson	
Input Parameters	fName	string
	fNameType	TStringType
	lName	string
	lNameType	TStringType
	Address	string
	addressType	TStringType
	ZipCode	string
	City	string
Output Parameters	return	string

Figur 10 Metodinformation

När en användare har informationen i tjänstens WSDL-dokument kan denne skapa en klient för att använda tjänsten.

3.1.4 Säkerhet och webbtjänster

En fördel med webbtjänster är att de är åtkomliga via Internet. För företag som tillhandahåller tjänster är det inte alltid önskvärt att alla skall ha tillgång till dem. Det kan bero på att informationen är känslig eller att ägaren vill ta betalt för tjänsten. Att säkra en webbtjänst skiljer sig inte nämnvärt från att säkra en webbsida. Genom använda Internet Protocol Security (IPSec) och en brandvägg är det möjligt att hindra obehöriga från att få tillgång till webbtjänsten [23]. Förutsättningen för att detta skall fungera är att IP adressen är känd för de datorer som kommer att använda tjänsten. Det skulle kunna gestalta sig så att en webbtjänst som används inom ett intranät konfigureras så att den bara accepterar förfrågningar från IP-adresser inom det nätverket.

När det inte är möjligt att i förväg identifiera användarna med hjälp av ip-nummer, är det genom att använda sig av protokollet Secure Sockets Layer, SSL, möjligt att skapa säkra kanaler för kommunikation. Detta innebär att informationen som skickas är krypterad och färdas säkert över Internet. SSL verifierar också att informationen kommer från uppgiven avsändare och inte avlyssnas på vägen [23].

Ytterligare ett sätt för att åstadkomma säkra metodanrop, är att använda sig av kryptering med publika och privata nycklar. En viss nivå av säkerhet kan uppnås genom att konfigurerar webbtjänsten så att den kräver lösenord av användaren och därigenom endast acceptera vissa

användare. Något som saknas inom säkerhet vad gäller webbtjänster är stöd för transaktioner[4]. Detta är en viktig fråga som måste lösas om webbtjänster ska få stort användningsområde.

3.2 XML

eXtensible Markup Language, XML är en 5 år gammal standard framtagen av W3C [14]. Det ursprungliga målet var att anpassa markeringsspråket Standard Generalized Markup Language, SGML, för att skapa och publicera dokument på Internet. SGML är dock ett mycket komplext språk så en delmängd av det blev till XML. XML är, precis som SGML, ett metaspråk vilket betyder att det används för att beskriva data.

Syntaxmässigt liknar XML ett annat känt markeringsspråk, nämligen Hyper Text Markup Language, HTML. Den största skillnaden mellan språken är att XML används för att beskriva data och fokuserar på vad den betyder medan HTML används för att presentera data och fokuserar på hur den skall se ut när den visas upp. För att beskriva data innesluts den med så kallade taggar[5] och bildar då så kallade element. I HTML finns ett antal fördefinierade taggar att använda, medan taggarna i XML definieras av användaren efter dennes behov.

Figur 11 innehåller två korta meddelanden. Meddelandet till vänster innehåller endast information, alltså data. Meddelandet till höger innehåller förutom informationen i det första meddelandet även så kallad metadata i taggar, det vill säga XML.

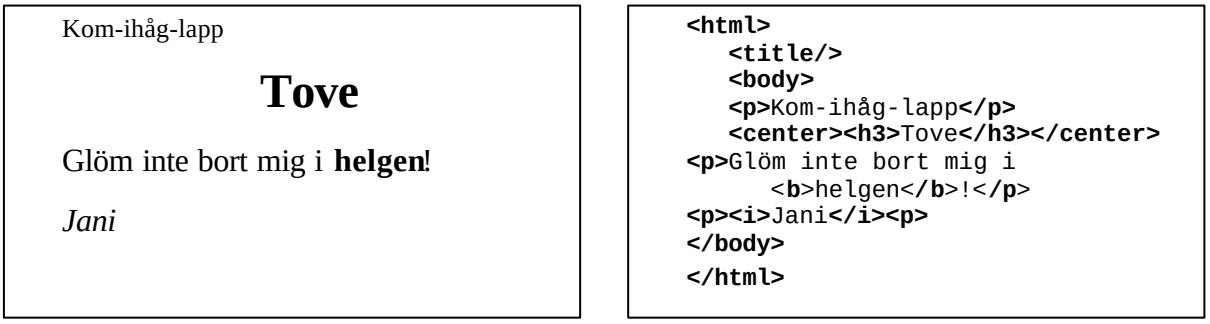
```
Kom-ihåg-lapp
Tove
Glöm inte bort mig i helgen!
Jani
```

```
<meddelande>
   <mottagare>Tove</mottagare>
   <avsändare>Jani</avsändare>
   <rubrik>Kom-ihåg-lapp</rubrik>
   <stycke>Glöm inte bort mig i
           helgen!</stycke>
</meddelande>
```

Figur 11 Användning av metadata

Samma meddelande som i Figur 11 återfinns i Figur 12 nedan som visar meddelandet formaterat med HTML istället för XML.

[4] Transaktioner – komplicerad operation som måste utföras i sin helhet, inbegriper ofta databaser.
[5] Taggar – information omsluten av < >, används för att förtydliga och gruppera.



Figur 12 Användning av HTML

Metadatan i XML-taggarna delar upp meddelandets information i olika element och strukturerar den med en trädliknande hierarki, istället för att låta informationen vara en enda sammanhängande textsträng. Element kan, som meddelandet till höger i Figur 11 visar, innehålla andra element. Element kan dessutom ha egenskaper, dessa kallas även för attribut. Användning av egenskaper för att strukturera det ursprungliga meddelandet i Figur 11 visas i Figur 13.

```
<meddelande mottagare="Tove"
            avsändare="Jani">
   <rubrik>Kom-ihåg-lapp</rubrik>
   <stycke>Glöm inte bort mig i
           helgen!</stycke>
</meddelande>
```

Figur 13 Användning av egenskaper i element

Även om XML från början var tänkt att användas för att publicera dokument på Internet, så har det under tidens gång fått allt fler användningsområden. Förutom att skapa dokument är det, eftersom XML fokuserar på att strukturera data, även möjligt att använda XML för att definiera format för data som skall lagras eller skickas. Genom att strukturera data med självdokumenterande XML-taggar blir det enklare att se vad det som skickas egentligen betyder. Detta underlättar även vid felsökning. Genom att få mer struktur på informationen blir den lättare att hantera.

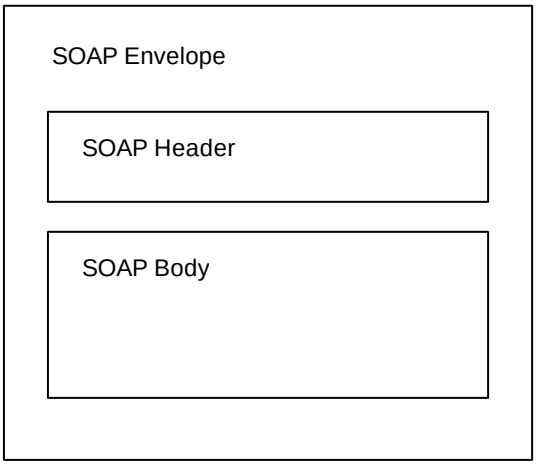
Om två företag t ex vill utbyta information mellan sina företagsnätverk behöver de ett gemensamt format för att ta emot och skicka data. En sådan gemensam meddelandestruktur för datautbyte kan definieras i XML. Antag att tidigare nämnda företag vill utbyta och dela information ur databaser. Det är då inte särskilt troligt att de har samma format på informationen i databaserna. Om informationen ur databaserna också strukturerats upp med XML innan den skickas, skulle det vara möjligt att göra en mall med hjälp av XML för att

göra översättningar mellan de olika formaten. Ytterligare ett sätt att underlätta detta vore att skapa databaser som lagrar data i XML-format.

Eftersom XML ordnar data med hjälp av ren text, metadata, kan XML-dokument skickas över Internet genom användning av HTTP på samma sätt som vanliga webbsidor. XML blir på grund av detta plattformsoberoende. Detta gör det möjligt att använda XML för att strukturera kommunikation, på applikationsnivå, mellan ändpunkter med olika miljöer.

3.2.1 SOAP

I ett försök till en gemensam standard för kommunikation mellan webbtjänster har ett XML-baserat protokoll utvecklats. Kommunikationen skall ske på applikationsnivå och protokollet heter Simple Object Access Protocol, SOAP. Med hjälp av detta protokoll skickas meddelanden, så kallade SOAP-meddelanden, skrivna i XML för att anropa webbtjänster. Strukturen för hur dessa meddelanden är uppbyggda visas i Figur 14.



Figur 14 Strukturen hos ett SOAP-kuvert

Figur 15 visar ett fullständigt SOAP-meddelande. De första 4 raderna innehåller HTTP-headern med information för HTTP och efter blankraden, som avslutar headern, följer själva SOAP-meddelandet. På rad 6 i meddelandet anges att innehållet som följer är skrivet i XML. Ytterst finns sedan taggen som börjar själva SOAP-kuvertet, `<Envelope>`. SOAP-kuvertet innehåller ett meddelande som består av elementen `<Header>` och `<Body>`. För närvarande är det inte nödvändigt att ange någon information i `<Header>`-elementet så oftast brukar det utelämnas helt. I `<Envelope>`-elemetet definieras, på raderna 8 till och med 11, ett antal namnutrymmen med hjälp av attributet `xmlns`, XML namespaces. På rad 12 `<Body>`-elemenet definieras ytterligare ett namnutrymme, ett för webbtjänsten, för att göra tjänstens metoder unika. Ofta används den URL, Uniform Resource Locator, där webbtjänsten finns, för att definiera detta namnutrymme. Härefter följer anropet till webbtjänstens metod, `MethodName`.

SOAP-meddelanden som innehåller metodanrop till webbtjänster kallas för SOAP Requests. Metodanropet består av namnet på den metod som anropas i webbtjänsten, se rad 13 i figuren, samt värdena för inparametrarna till metoden, se rad 14 till och med 16. De tre sista raderna innehåller sluttaggarna för de tidigare påbörjade elementen med `MethodName`, `Body` och `Envelope`.

```
1   POST PathToWebService HTTP/1.1
2   Host: WebServiceURL
3   Content-Type: application/soap+xml; charset=utf-8
4   Content-Length: n
5
6   <?xml version="1.0"?>
7   <soap:Envelope
8   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
9   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding"
10  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
11  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
12    <soap:Body xmlns:m="WebServiceURL">
13     <m:Methodname>
14     <m:inParamName1 xsi:type="xsd:string"> Param1Value </m:inParamName1>
15     <m:inParamName2 xsi:type="xsd:string"> Param2Value </m:inParamName2>
16     <m:inParamName3 xsi:type="xsd:string"> Param3Value </m:inParamName3>
17     </m:MethodName>
18    </soap:Body>
19  </soap:Envelope>
```

Figur 15 Generellt SOAP Request

Om metodanropet går bra, d v s tjänsten returnerar ett svar, läggs webbtjänstens svar i ett nytt SOAP-meddelande, även kallat SOAP Response, se Figur 16. Om metoden inte finns eller något annat fel har uppstått svarar webbservern, som tagit emot HTTP-meddelandet, med ett speciellt definierat SOAP meddelande, ett SOAP Fault. Ett SOAP Fault är ett meddelande som i kroppen innehåller elementet `<Fault>` för att beskriva felet som uppstått. Fel kan bl a uppkomma om metoden eller tjänsten inte hittas av webbservern som hanterar SOAP Requestet, t ex på grund av ett stavfel eller liknande i metodanropet.

```
1   HTTP/1.1 200 OK
2   Content-Type: application/soap; charset=utf-8
3   Content-Length: n
4
5   <?xml version="1.0"?>
6   <soap:Envelope
7   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
8   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding"
9   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
10  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
11    <soap:Body xmlns:m="WebServiceURL">
12     <m:MethodNameResponse>
13     <m:Response> returnValue </m:Response>
14     </m:MethodNameResponse>
15    </soap:Body>
16  </soap:Envelope>
```

Figur 16 Generellt SOAP Response

3.2.2 WSDL

WSDL eller Web Services Description Language är ett XML baserat språk som utvecklats av Microsoft och IBM, International Business Machines, och används för att beskriva webbtjänster. WSDL-dokument består huvudsakligen av sex större element [10]. Dessa är följande:

<type> - Inom dessa taggar finns definitionerna av de datatyper som används i de meddelanden som används för anrop och retur i varje metod hos webbtjänsten. För varje metod definieras ett specifikt in-meddelande och ett specifikt ut-meddelande beroende av metodens inparametrar och returvärde.

<message> - Varje meddelande som skickas till eller från en metod i webbtjänsten finns beskriven i ett `<message>`-element. Dessa element är abstrakta definitioner av meddelandena och är endast uppbyggda av logiska delar.

<portType> - Med `<portType>` definieras de operationer som finns tillgängliga i webbtjänsten. Här beskrivs vilka in- och ut-meddelanden, som definierats med `<message>`, som metoderna använder sig av.

<binding> - Under `<binding>` specificeras vilka protokoll och dataformat för de operationer som definierats med hjälp av `<portType>`, t ex vilken typ av HTTP operationen använder, HTTP GET eller HTTP POST.

<port> - Här specificeras vilken adress (URL) de olika bindningarna använder sig av, d v s här specificeras vilka ändpunkter som används för kommunikation med webbtjänsten.

<service> - Detta element innehåller `<port>`-elementen.

I Figur 17 visas en bild på hur strukturen i ett WSDL-dokument ser ut.

```
<?xml version="1.0" encoding="utf-8"?>
<definitions>
     <types>definition of data types........</types>
     <message>definition of a message....</message>
     <portType>definition of portTypes.......</portType>
     <binding>definition of portType bindings....</binding>
     <service>definitions of endpoints...
            <port>definition of endpoint....</port>
     </service>
</definitions>
```

Figur 17 WSDL-struktur

Eftersom det finns flera metoder för att anropa en webbtjänst, SOAP, HTTP-GET och HTTP-POST, innehåller WSDL-dokumentet beskrivningar för samtliga typer av anrop för samtliga metoder. Detta gör dokumenten stora och något otympliga att läsa. Något som dock

användare i regel inte behöver göra, antingen genom att automatiskt generera proxyklasser eller genom att använda verktyg för att utsortera WSDL-dokumentens väsentliga delar.

4 Analys av webbtjänster

Ett mål med uppsatsen är att finna webbtjänster som genom integration kan utnyttjas av affärssystemet Navision med hjälp av Diveria PAR Direkt. Diveria har lämnat önskemål på tjänster som de skulle vilja att vi för deras räkning letade efter. Under sökandet har vi utöver dessa även lagt till egna uppslag på webbtjänster att söka efter. I kapitlet beskriver vi tillvägagångssättet för att finna webbtjänster och vilka tjänster vi hittat. Vi undersöker även möjligheterna att integrera de funna tjänsterna i Navision.

4.1 Intressanta webbtjänster att söka efter

Vi började söka efter de webbtjänster som Diveria gav oss uppslag till:

- Kreditupplysningstjänst, med hjälp av denna tjänst ska det vara möjligt att ta reda på om en person har betalningsanmärkning.
- Valutakursupplysningstjänst, denna tjänst skall lista dagens valutakurser.
- En tjänst som gör det möjligt att hämta blanketter i elektroniskt format, som interna blanketter för t ex redovisning eller externa blanketter till leverantörer och kunder.
- e-faktureringstjänst. Denna tjänst ska göra det möjligt för fakturautställare att distribuera fakturor elektroniskt till fakturamottagare.

4.2 Metoder för att söka webbtjänster

Diverias kunder, som har köpt Navision, är mindre till medelstora företag. Merparten av dem är tjänsteföretag som har köpt in administrativa affärssystemsmoduler inom redovisning, kunder, order, lager, fakturering, inköp, personal och resurser mm.

När vi sökte efter webbtjänster använde vi oss av två angreppssätt. Den ena bestod i att uppsöka förmedlare och söka igenom deras register efter webbtjänster som skulle kunna användas tillsammans med beskrivna moduler. Med denna metod hoppades vi finna webbtjänster som kunde komplettera affärssystemet. Den andra metoden gick ut på att undersöka om företag som hade tjänster, som Diverias kunder idag redan använder via hemsidor, även finns tillgängliga som webbtjänster. De skulle i sådant fall kunna integreras i affärssystemet för att underlätta användningen.

4.3 Sökandet efter webbtjänster

Vi började med att kontakta de största kreditupplysningsföretagen; Dun & Bradstreet[6], Upplysningstjänst[7], Upplysnings Centralen[8] och United Collectors[9] för att se vilka tjänster de kunde erbjuda. Även Kronofogdemyndigheten kontaktades.

På samma sätt gick vi vidare på listan och kontaktade olika banker, växlingskontor och Stockholmsbörsen för att fråga dem om de har webbtjänster där är möjligt att få aktuella valutakurser.

Angående blankettjänsten sökte vi upp de blankettföretag vi kunde finna, Sign On[10] och CoolBAT[11], och diskuterade webbtjänster med dem.

När det var dags att titta närmare på e-fakturerningtjänsten, visste vi inte riktigt hur vi skulle gå tillväga. Vi startade att söka på nätet, och fann att Föreningssparbanken och Nordea var inblandade i e-fakturering och gick då vidare på det spåret.

Förutom att direkt kontakta företag, gick vi även igenom registren hos webbtjänstförmedlarna för att se om det fanns någon, förutom de företagen vi kontaktat, som kunde bistå med dessa eftersökta tjänster. Några av de förmedlare vars register vi besökt är: BindingPoint[12], XMethods[13], UDDI[14], WebserviceX[15] och salcentral[16].

Under sökandet lade vi även till ytterligare två tjänster att söka efter:

- Telefonnummertjänst, via denna tjänst ska användaren kunna söka efter privatpersoners telefonnummer.
- Adresstjänst, via denna tjänst ska användaren kunna söka efter privatpersoners adresser.

4.4 Sökta webbtjänster som inte hittades

Vi fann tyvärr inte någon kreditupplysningswebbtjänst. Det verkade inte som något av företagen vi varit i kontakt med upplevt någon efterfrågan efter en sådan webbtjänst och därför beslutat sig för att vänta med att tillhandahålla någon. Några av företagen uppgav även

[6] För mer information om företaget se: www.dbsverige.se
[7] För mer information om företaget se: www.upplysningstjanst.se
[8] För mer information om företaget se: www.uc.se
[9] För mer information om företaget se: www.unitedcollectors.se
[10] För mer information om företaget se: www.signon.se
[11] För mer information om företaget se: www.coolbat.com/se
[12] För mer information om företaget se: www.bindingpoint.com
[13] För mer information om företaget se: www.xmethods.com
[14] För mer information om företaget se: www.uddi.org
[15] För mer information om företaget se: www.webservicex.net

att de valt att vänta med att skaffa tjänsten på grund av att webbtjänster i dagsläget inte är nog säkra, samt att ingen standard är utvecklad. Eftersom kreditupplysningsföretag tillhandahåller känslig information vill de inte riskera att andra obehöriga kommer över den.

Ett kreditupplysningsföretag, Dun & Bradstreet, erbjuder en skräddarsydd lösning för det företag som önskar göra kreditupplysningar via sina affärssystem. De har dock ingen webbtjänst och säger sig inte ha märkt någon efterfrågan efter en sådan. Därför planerar de för närvarande inte att utveckla någon. När det finns en standard för webbtjänster tror sig dock Dun & Bradstreet att de kommer tillhandahålla en kreditupplysningswebbtjänst [15].

De flesta kreditupplysningsföretag erbjuder sina kunder tjänster via sina hemsidor där de kan beställa kreditupplysningar och få resultatet via mail, post eller direkt på hemsidan.

Vi fann heller ingen valutakurswebbtjänst. Efter att ha varit i kontakt med flera banker, fick vi veta att de i dagsläget enbart tillhandahåller information om valutakurserna via sina hemsidor.

Hos webbtjänsts-förmedlare hittade vi flera tjänster som kunde konvertera valutor. Problemet här var att det inte framgick varifrån tjänsterna hämtade sina växlingskurser och hur ofta de fick dem uppdaterade. Denna typ av tjänster var inte vad vi letade efter, utan vad vi sökte var en tjänst som gav information om aktuella växlingskurser.

När det kom till blankettwebbtjänsten, kontaktades Sign On och CoolBAT. Det fanns ingen sådan tjänst hos någon av dem och ingen hade planer på att skaffa en sådan. Båda förtagen har idag hemsidor, via vilka användare kan hämta blanketter om de är registrerade medlemmar och betalar en medlemsavgift.

e-fakturering som webbtjänst fanns heller inte att hitta. Vi fann inom e-fakturering en skräddarsydd speciallösning, d v s en lösning som inbegriper flera olika företag vars system är hårt knutna till varandra, från företaget e-faktura [16].

4.5 Webbtjänster som bedömdes intressanta

Swedish Zipcode Info – en webbtjänst för att söka efter postnummer i Sverige. Tjänsten är konstruerad av Henry Aspenryd på Marotz AB. Vi fann den hos förmedlaren XMethods[17]. Om gatunamn och stad skickas in ett metodanrop till tjänsten skall postnumret returneras.

Search Swedish Persons – en webbtjänst för att söka adress och telefonnummer till privatpersoner i Sverige. Även denna tjänst är skapad av Henry Aspenryd på Marotz AB. Den

[16] För mer information om företaget se: www.salcentral.com
[17] För mer information om företaget se: www.xmethods.com

hittades också hos XMethods. Genom att skicka in varierande kombinationer av förnamn, efternamn, gatunamn, stad och postnummer fås fullständig adress, telefonnummer och innehavare av telefonnumret ut. Tjänsten söker igenom alla listade telefonabonnenter i Sverige. Informationen den tillhandahåller är, enligt tjänstens beskrivning hos förmedlaren Xmethods, alltid uppdaterad och korrekt. Vi vet, efter kontakt med utvecklaren [17], att tjänsten hämtar informationen ur Gula Sidornas register.

4.6 Bedömningsgrunder för webbtjänster

Det är inte svårt att hitta enkla webbtjänster, för det finns mängder av dem hos olika förmedlare. Svårigheten är att finna en tjänst som uppfyller användarens alla behov eller önskemål. Innan användaren sätter sig ner för att söka efter en tjänst, bör denne tänka igenom vilka krav, med avseende på sina slutanvändare, denne bör ställa på tjänsten. Följande punkter som vi tror bör tänkas över är:

- *På vilket språk ska webbtjänsten kommunicera?*

  Vid val av webbtjänst är det värt att överväga vilket språk slutanvändare kommer att använda sig av när de använder tjänsten. Det är ju inget som går att ändra på eftersom detta är en del i själva tjänsten. Om det är möjligt är det troligen bra att välja en tjänst som använder sig av samma språk som slutanvändaren så att t ex ett felmeddelande kan tolkas av slutanvändaren. Detta leder till större användarvänlighet hos klienten.

  Vi tror att Diveria i första hand är intresserade av webbtjänster som kommunicerar med slutanvändarna på svenska eftersom de flesta av Diverias kunder är svenska företag.

- *Hur länge kommer tjänsten finnas tillgänglig?*

  Detta är en mycket central fråga som användaren bör ha i åtanke. Om det är ett företag som har utvecklat tjänsten, går det att få information om företaget och eventuellt starta ett samarbete. Det är inte önskvärt att satsa pengar på att integrera en tjänst i ett system för att sedan upptäcka att företaget som utvecklat tjänsten gått i konkurs, och att tjänsten därigenom lagts ner. Ett liknande, men mer svårbedömt, scenario är att privatpersoner som skapat tjänster, en dag bestämmer sig för att ta bort dem.

  Vi tror att det är större risk att tjänster skapade av privatpersoner har en mer oförutsägbar livslängd än tjänster skapade av företag.

- *Vem ska utveckla tjänsten?*

Vill användaren, d v s utvecklaren av klienten, använda sig av tjänster utvecklade av företag eller går det även bra med tjänster konstruerade av privatpersoner? Är tjänsten mer tillförlitlig om den är skapad av ett företag än om den är skapad av en privatperson?

Vi tror att webbtjänster utvecklade av företag är att föredra då de inger en större känsla av tillförlitlighet. Företaget lägger förmodligen ner mer tid på att skapa en väl fungerande tjänst, eftersom de önskar ett gott rykte. Dessutom har företag troligtvis mer kompetens bland sina utvecklare än vad en privatperson har. En privatperson kan vara välutbildad och ha kunskaper om utveckling av webbtjänster, men det är svårt att bedöma, vilket resulterar i att tjänster med anknytning till företag ger ett mer seriöst intryck.

- *Får tjänsten kosta pengar att utnyttja?*

  Även kostnadsfrågan för en tjänst bör has i åtanke. Hur mycket är slutanvändaren villig att betala för att utnyttja tjänsten?

  Vi tror att det kan vara värt att i förväg uppskatta eventuella pristak. Att tjänsten kostar pengar behöver inte vara ett hinder, utan kan tvärtom vara något som understyrker att tjänsten är seriös.

- *Varifrån hämtar tjänsten information?*

  En mycket viktig detalj är att ta reda på varifrån informationen, om webbtjänsten t ex använder sig av föränderlig information, hämtas. Informationen måste vara korrekt och aktuell för att användaren skall ha nytta av tjänsten. Går detta att få reda på? Vilka garantier ställer utvecklaren för att informationen är korrekt?

  Vi tror att om Diveria ska kunna integrera och sälja en tjänst som en del i ett affärssystem, är det viktigt att informationen är tillförlitlig, annars får de ingen nytta av tjänsten.

Om utvecklaren av klienten överväger frågorna ovan, kan denne förenkla sitt sökande genom att lättare kunna välja bort ointressanta tjänster.

4.7 Integrationsmöjligheter

Vid försök att anropa postnummertjänsten ’Swedish Zipcode Info’ med hjälp av Diveria PAR Direkt uppstod följande felmeddelanden, se Figur 18. Vi misstänkte att felet berodde på ett felaktigt konstruerat SOAP Request. Efter att ha varit i kontakt med Henry Aspenryd och av honom fått syntaxen till ett korrekt SOAP Request för tjänsten fick vi ändå inte kontakt med

den. Vi sökte då upp en generell SOAP-klient, Generic SOAP Client[18], på Internet. Denna klient fungerar på följande sätt; en webbtjänsts WSDL-dokument skickas till klienten som då dynamiskt genererar ett HTML-formulär med vilket det är möjligt att testa tjänstens alla metoder. Sidan gör det möjligt att mata in värden för metodernas parametrar. Klienten bygger själv SOAP-meddelandena baserade på informationen ur WSDL-dokument. Det är möjligt att undersöka både de meddelanden som skickas till tjänsten och de som returneras av tjänsten. Efter att ha testat postnummertjänsten även med denna klient och fått samma felmeddelande som tidigare, drog vi slutsatsen att något var fel med webbtjänsten. Detta var den enda svenska postnummertjänst vi hittade. Eftersom vi inte kunde anropa tjänsten avfärdade vi tanken på att försöka integrera den.

```
<?xml version="1.0" ?>
   <SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
            <faultactor />
            <faultcode>SOAP-ENV:Server</faultcode>
            <faultstring>Catastrophic failure</faultstring>
        </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
```

Figur 18 SOAP Fault från tjänsten Swedish Zipcode Info

Vi testade även att anropa vissa av Adress/Telefonnummertjänstens metoder med hjälp av Diveria PAR Direkt och det visade sig däremot fungera. För att underlätta denna testning, skapade vi en enkel klient med ett gränssnitt i form av ett formulär, se Figur 19, till webbtjänsten. I formuläret anger användaren t ex namnet på en person som denne vill söka efter samt t ex ort. Tillbaka från webbtjänsten kommer samtliga namn, på de personer som bor på adressen i fråga, samt fullständig adress och telefonnummer. I exemplet i Figur 19 söker användaren efter samtliga personer vid namn Svensson boende i Karlstad.

Figur 19 En sökning efter samtliga vid namn Svensson i Karlstad

När användaren trycker på ”Search” skapas ett SOAP Request, se Figur 20, innehållande informationen från formuläret ovan. Formulärets information hamnar mellan `<body>`-taggarna på SOAP Requestet. Svensson hamnar mellan `<lName>`-taggen och Karlstad går att finna mellan `<City>`-taggen.

```
<?xml version="1.0" encoding="utf-8" ?>
   <SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <SOAP-ENV:Body>
      <XMLSearchAddress xmlns="urn:SearchSwedishPersonIntf-ISearchSwedishPerson“
       SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <fName xsi:type="xsd:string"/>
         <lName xsi:type="xsd:string"> Svensson </lName>
         <Address xsi:type="xsd:string"/>
         <ZipCode xsi:type="xsd:string"/>
         <City xsi:type="xsd:string"> Karlstad </City>
      </XMLSearchAddress>
    </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
```

Figur 20 SOAP Request

Efter att webbtjänsten tagit emot SOAP Requestet ovan, returnerar den det SOAP Response som kan ses i Figur 21.

```
<?xml version="1.0" ?>
<SOAP-ENV:Envelope xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body SOAP-ENC:encodingStyle=
    "http://schemas.xmlsoap.org/soap/envelope/">
    <NS1:XMLSearchAddressResponse xmlns:NS1="urn:SearchSwedishPersonIntf-
     ISearchSwedishPerson">
     <return xsi:type="xsd:string">
       <?xml version="1.0" encoding="ISO-8859-1" ?>
         <Entries Count="1">
          <Entry>
             <Name>Träfflistan visar max 250 träffar. Begränsa din sökning genom
                   att fylla i fler fält.</Name>
             <Address />
             <Number />
           </Entry>
         </Entries>
     </return>
    </NS1:XMLSearchAddressResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figur 21 SOAP Response med för många träffar

Själva svaret på metodanropet finns att läsa mellan `<return>`-taggarna. I detta fallet finns blev det ”**`Träfflistan visar max 250 träffar. Begränsa din sökning genom att fylla i fler fält.`**”. Vilket betyder att det fanns fler Svensson i Karlstad än 250, detta är det maximala antalet träffar som tjänsten returnerar.

I Figur 22 visas ett annat SOAP Response. I detta fallet har användaren angivit en sökning efter en Kalle Anka i Ankeborg, vilket resulterat i en träff. Vi valde att använda ”Kalle Anka” för att konstruera ett exempel eftersom vi inte vill lämna ut uppgifter om privatpersoner i uppsatsen. Mellan `<return>`-taggarna listas alla träffar, i detta fallet en. Här kan användaren utläsa att Kalle Anka bor på Ankgatan 313, har postnummer 555 44, bor i Ankeborg och har telefonnummer 112-221122.

```
<?xml version="1.0" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
 <SOAP-ENV:Body SOAP-ENC:encodingStyle=
  "http://schemas.xmlsoap.org/soap/envelope/">
  <NS1:XMLSearchAddressResponse xmlns:NS1="urn:SearchSwedishPersonIntf-
   ISearchSwedishPerson">
   <return xsi:type="xsd:string">
   <?xml version="1.0" encoding="ISO-8859-1" ?>
     <Entries Count="1">
       <Entry>
         <Name>Kalle Anka</Name>
         <Address>Ankgatan 313, 555 44 Ankeborg</Address>
         <Number>112-221122</Number>
       </Entry>
     </Entries>
   </return>
  </NS1:XMLSearchAddressResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figur 22 SOAP Response med en träff

4.7.1 Utvärdering av SearchSwedishPersons utifrån bedömningsgrunderna

Vid en jämförelse mellan tjänsten och de bedömningsgrunder vi kommit fram till i avsnitt 4.6 anser vi att tjänsten uppfyller ett flertal av punkterna eftersom:

- ’Search Swedish Persons’ information är på svenska.
- Tjänsten har gjorts tillgänglig via hemsidan hos det företag, på vilket utvecklaren av tjänsten arbetar. Vi tycker att detta styrker tjänstens trovärdighet genom att företaget vill associeras med tjänsten.
- Tjänsten hämtar sin information Gula Sidorna.
- Den här tjänsten är gratis och det tycker vi att den ska vara eftersom den inte innehåller svåråtkomlig eller känslig information. Den fungerar som en mer lättillgänglig telefonkatalog.

Nackdelen med ’Search Swedish Persons’ är att utvecklaren inte lämnar några garantier för tjänstens livslängd. Detta anges tydligt i tjänstens beskrivning hos XMethods.

4.8 Resultat av analys

Tyvärr har Diveria inte någon användning av ’Search Swedish Persons’. Detta eftersom det i ett senare skede kom till vår kännedom att tjänsten endast är en prototyp för att testa och påvisa enkelheten i att bygga in stark funktionalitet i system med hjälp av webbtjänster [17].

På grund av detta samt att vi inte kände att det skulle tillföra något att skapa ytterligare ett grafiskt användargränssnitt åt tjänsten, i Navision den här gången, valde vi att inte gå vidare med integration av tjänsten.

Eftersom vi inte funnit några användbara webbtjänster valde vi i samråd med Diveria AB att utveckla en egen tjänst, dock endast en enkel prototyp. Denna beskrivs i kapitel 5 och 6.

4.8.1 Webbtjänster som är på gång

Enligt vad vi erfar arbetar Östgöta Enskilda Bank just nu på att utveckla en valutakurswebbtjänst [18]. När den ska vara färdig är ännu inte helt fastställt. Även Upplysnings Centralen utvecklar nu en kreditupplysningswebbstjänst, som skall vara färdig sommaren 2003 [20].

5 Konstruktion av en webbtjänst

I det här kapitlet beskrivs konstruktionslösningen för den prototyp som togs upp som ett alternativt mål för uppsatsen, detta eftersom vi inte fann någon användbar webbtjänst. Vi valde då att skapa en egen tjänst. Genom implementationen av prototypen hoppas vi få kännedom om hur webbtjänster konstrueras och få ytterligare insikt i de möjligheter som webbtjänster erbjuder applikationsutvecklare. I kapitel 6 redovisas sedan delar av prototypens implementation.

5.1 Funktionalitet hos prototypen

Eftersom vi tidigare sökt efter webbtjänster som skulle kunna vara till nytta i affärssystem, ville vi konstruera en tjänst av den typen. Vi beslöt oss då för att konstruera en webbtjänst med funktionalitet liknande en stämpelklocka. En tjänst av den här typen tror vi skulle kunna användas som ett komplement till ett affärssystem med t ex lönehantering och tidsredovisning. Det är möjligt att tidsstämpling redan finns som en modul i affärssystem. Men om stämpelklockan finns tillgänglig som en webbtjänst blir det möjligt för slutanvändaren att använda stämpelklockan även när denne inte har tillgång till affärssystemet. Slutanvändaren kan t ex vid resor rapportera sin tid genom att använda webbtjänsten via Internet. Tidsstämplingarna finns sedan lagrade i affärssystemet. Att informationen hamnar på plats direkt i affärssystemet underlättar i det här fallet för ekonomiavdelningen. Rapporteringen sker direkt.

Tjänstens funktionalitet utreddes med hänsyn till vilka slutanvändarna skulle komma att bli. Vi delar in slutanvändarna i följande grupper: vanliga medarbetare samt ekonomiansvariga. Stämpelklockans basfunktionalitet utgörs av in- och ut-stämplingar med vilka tiden för en medarbetares arbetsdag mäts. Stämpeltiden ligger till grund för beräkning av löner. In- och ut-stämpling i tjänsten är den grundläggande funktionalitet som alla användargrupper skall ha tillgång till. Beräkning av löner är en funktion som vi tror i första hand ekonomiavdelningen har nytta av.

Sammanfattning av stämpelklockans grundfunktionalitet:

- Stämpla in
- Stämpla ut
- Beräkna månadslön baserat på stämplade tider

5.2 Design

Beräkning av en månadslön baseras på medarbetarens timlön och antal instämplade timmar, detta gör det nödvändigt att kunna skilja på olika medarbetares timlöner och stämplingar. För att åstadkomma detta krävs tillgång till information om en medarbetares lön samt en möjlighet att skilja olika medarbetares stämplingar från varandra. Detta gör det nödvändigt att kunna förse tjänsten med information om medarbetarna. Beräkning av lönen kräver en sammanställning av en medarbetares totala antal arbetade timmar för en viss månad. Därför har tjänstens funktionalitet utökats till följande:

- Stämpla in medarbetare
- Stämpla ut medarbetare
- Beräkna månadslön baserat på medarbetares stämplade tider och timlön
- Spara information om ny medarbetare
- Ange timlön för medarbetare
- Hämta information om medarbetare
- Beräkna antal arbetade timmar för en viss månad och medarbetare

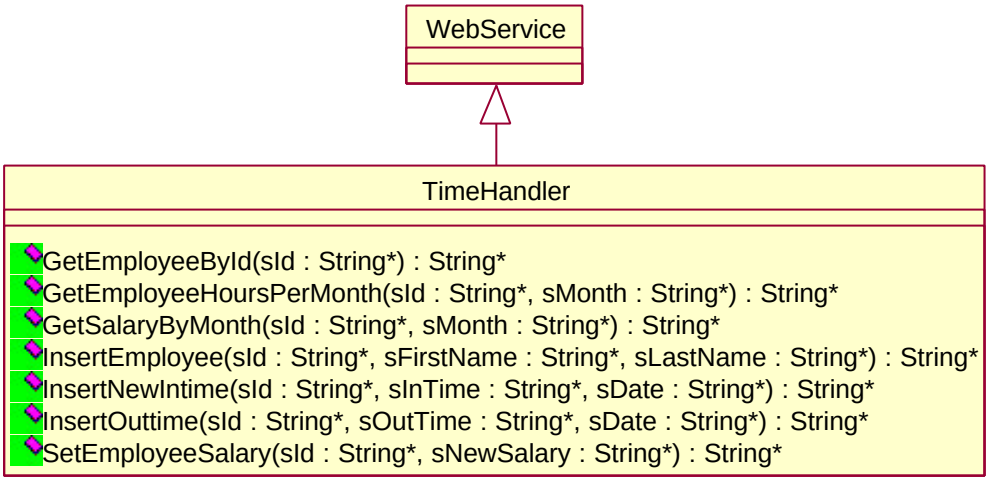
’Spara information om ny medarbetare’ är en metod som är tänkt att användas när nya medarbetare registreras för att kunna använda tjänsten. ’Ange timlön för medarbetare’ är en annan metod som används för att antingen ange timlön för ny medarbetare eller vid förändring i en befintlig medarbetares timlön. ’Hämta information om medarbetare’ skall ge möjlighet att hämta aktuella uppgifter om medarbetare.

Ett designförslag för hantering av åtkomst till metoderna vore att skapa flera olika grafiska användargränssnitt till webbtjänsten. Ett gränssnitt i affärssystemet som riktar sig till den vanliga medarbetaren samt ett gränssnitt till tjänsten via Internet. Via dessa båda gränssnitt skulle medarbetaren få tillgång till in- och utstämpling. Ekonomiavdelningen däremot får ett mer avancerat grafiskt användargränssnitt i affärssystemet med tillgång till samtliga metoder.

Tjänsten implementerades i programmeringsspråket C++ eftersom det är ett av de språk som stöds av Microsoft Visual Studio .NET, för mer information se Bilaga B.1. Ytterligare ett av våra skäl att välja C++ är vi under vår utbildning tidigare arbetat med det. Visual Studio erbjuder mycket hjälp vid konstruktion av webbtjänster, det är som tidigare nämnts möjligt att skapa webbtjänstprojekt där alla filer som behövs autogenereras. Utvecklaren behöver själv bara skapa logiken för webbtjänsten, mer om dessa autogenererade filer finns att läsa i avsnitt 6.1.1.

Prototypen döptes till TimeTicker. Det engelska språket för namngivning och dylikt i källkoden känns som ett naturligt val, eftersom både C++ och den omgivande miljön, med de autogenererade filerna och övriga tillgängliga klasser, är skrivna på engelska. Klassen TimeHandler innehåller logiken för tjänsten och implementerar ett antal metoder som utgör tjänstens gränssnitt. Vi har gjort en metod för varje punkt i listan ovan.

De egenskaper som krävs av .NET för att kunna använda klassen som en webbtjänst ärvs från .NET-klassen WebService enligt klassdiagrammet i Figur 23. I diagrammet visas även de metoder som utgör webbtjänstens gränssnitt.



Figur 23 Klassdiagram för prototypen TimeTicker

5.2.1 Metoder i klassen TimeHandler

Signaturerna för metoderna anges för varje metod i tjänstens WSDL-dokument. Här följer en kort beskrivning av metoderna i webbtjänsten TimeTickers gränssnitt och hur de är tänkta att användas.

- `GetEmployeeById`, metoden returnerar förnamn och efternamn för den medarbetare som anges.
- `GetEmployeeHoursPerMonth`, metoden returnerar antal timmar som angiven medarbetare har varit instämplad under den angivna månaden.
- `GetSalaryByMonth`, metoden returnerar den sammanlagda lönen för angiven medarbetare under angiven månad. Lönen baseras på antal timmar medarbetaren varit instämplad under den angivna månaden samt medarbetarens timlön.
- `InsertEmployee`, metoden lägger till en medarbetare.
- `InsertNewIntime`, metoden lägger till en ny instämpling.

- `InsertOuttime`, metoden uppdaterar en instämpling med en utstämplingstid.
- `SetEmployeeSalary`, metoden uppdaterar en medarbetares timlön.

5.2.2 TimeTickers databas

Webbtjänster exekveras inte kontinuerligt, som vanliga applikationer t ex ordbehandlare, utan endast under korta stunder då de anropats. På grund av detta kan tjänster, som TimeTicker, inte själv hålla kvar information utan måste istället spara den beständigt medium. Då vi ville att TimeTicker skall lagra information mellan anrop valde vi att göra detta med en databas. Genom att använda en databas fås en mer lätthanterlig struktur på informationen. Vi valde att använda en databas skapad med Microsoft Access[18] eftersom denna typ av databas lämpar sig för vår utvecklingsmiljö. Dessutom har vi sedan tidigare erfarenhet av detta databashanteringsprogram. Databasen döptes till TTDB vilket står för TimeTickerDatabase.

Till vår prototyp har vi endast gjort en enkel databasdesign bestående av två tabeller. Vi valde att dela informationen i två tabeller för att minska redundansen i databasen. De två tabellerna är ’Employee’ och ’TimeTable’. Information om medarbetare, förnamn, efternamn och timlön, lagras i en tabell som vi döpt till ’Employee’, se Figur 24. Fältet ’Id’ skapades för att fungera som primärnyckel, unik identifierare, i tabellen. Detta eftersom förnamn och efternamn tillsammans inte bildar en unik nyckel.

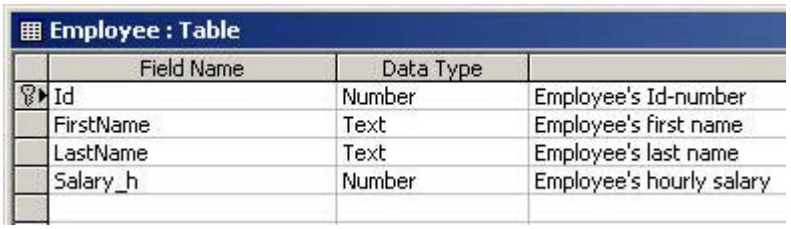
TimeTable : Table

	Field Name	Data Type	
🔑	RecordNumber	AutoNumber	Timerecord id
	Id	Number	Employee's Idnumber
	InTime	Text	Timestamp in
	OutTime	Text	Timestamp out
	WDDate	Text	Workingday date

Figur 24 Design för tabellen ’Employee’

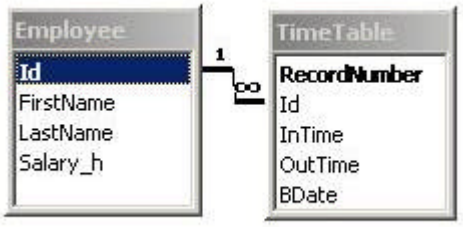
I tabellen ’TimeTable’, se Figur 25, lagras in- och utstämplingar, datum samt vem tidsstämplingen gäller. Här fanns inte heller någon riktigt bra primärnyckel eftersom den i så fall skulle bli en supernyckel, det vill säga att alla fält i tabellen tillsammans bildar primärnyckeln. Små primärnycklar underlättar identifiering av rader i tabellen. För att åstadkomma detta skapades även här ett nytt fält, ’RecordNumber’, som är en automatisk räknare.

[18] Microsoft Access – verktyg för hantering av databaser.



Figur 25 Design för tabellen 'TimeTable'

'Id' användes för att skapa en relation mellan tabellerna som gör det möjligt att sammanställa informationen ur dem, se Figur 26.



Figur 26 Relationer i prototypen TimeTickers databas

För att söka information i eller uppdatera databasen används Structured Query Language[19], SQL. Implementationen av stämpelklockan beskrivs mer detaljerat i kapitel 6.

[19] SQL – standardiserat språk för frågor med vilka information hämtas ur databaser.

6 Implementation och testning av prototypen

I detta kapitel beskrivs delar av implementationslösningen i prototypen TimeTicker. Anledningen till att endast valda stycken av lösningen beskrivs är att ett flertal av programdelarna till stor del liknar varandra. Efter detta beskrivs de tester vi utfört samt utfallen av testerna. I slutet av kapitlet diskuteras begränsningar, samt möjligheter till utökningar och förbättringar av tjänsten innan resultatet av hela arbetet sammanfattas och diskuteras i kapitel 7.

6.1 Utvecklingsmiljö

När det gäller utveckling av webbtjänster finns det flera olika utvecklingsmiljöer att använda. Några av de mest populära är Sun ONE Studio 4, Microsoft Visual Studio .NET samt Delphi Studio 7, men listan kan göras längre.

I vårt fall var valet av utvecklingsmiljö ganska enkelt. Vi valde .NET baserat på att denna miljö var lättillgänglig för oss, att vi hade mycket litteratur om miljön och att vi redan var insatta i hur webbtjänster i .NET fungerade. Om vi hade valt en annan miljö hade vi varit tvungna att först införskaffa och installera verktygen samt läsa in oss på användningen av dem, vilket hade varit kostsamt både ekonomiskt och tidsmässigt. Mer information om vår valda utvecklingsmiljö, .NET, och de delar vi använt oss av, Microsoft Visual Studio .NET och .NET Framework, finns att läsa i Bilaga B. Bilagan tar även snabbt upp den webbserver, Internet Information Service, IIS, som vi använt till prototypen.

6.1.1 Webbtjänst-projekt med Microsoft Visual Studio .NET

Vid implementation av webbtjänster i Visual Studio .NET skapas ett projekt i vilket ett antal filer autogenereras för att underlätta arbetet för utvecklaren. De mest centrala av filerna är:

- .asmx – denna fil fungerar som en proxy för webbtjänsten och är den adresserbara delen av tjänsten, d v s den fil som används vid anrop till tjänsten. Sökvägen till en webbtjänst blir: `http://servername/apppath/webservicename.asmx` där `servername` är namnet på servern på vilken webbtjänsten ligger, `appath` är namnet på den virtuella katalogen samt resten av webbtjänstens sökväg och `Webservicename.asmx` är namnet på tjänstens .asmx-fil. Adressen till vår prototyp har följande utseende:

`http://smart-ws8.cse.kau.se/TimeTicker/TimeTicker.asmx`. För att med hjälp av Visual Studio .NET göra tjänsten tillgänglig, kör vi en webbserver, IIS se Bilaga B.2, lokalt på den dator där webbtjänsten är lagrad.

- Web.config – denna fil innehåller konfigurationer för webbtjänsten, det är möjligt att göra inställningar för bl a säkerhet med hantering av inloggning, cookies med mera.
- .dll – när webbtjänstens kod, i vårt fall klassen TimeHandler, är kompilerad sparas den i en dll-fil med samma namn som webbtjänsten. När tjänsten aktiveras genom ett anrop till .asmx-filen laddas och exekveras .dll-filen.

6.2 Implementation

Webbtjänsten TimeTicker har sju metoder som antingen hämtar information ur webbtjänstens databas eller uppdaterar databasen med ny information. Samtliga metoder innehåller ett anrop till databasen och hantering av detta anrop utgör den största delen av metoderna. Detta resulterade i att källkoden för metoderna i stort sett blev densamma. Därför väljer vi att endast beskriva en metod, `TimeHandler::GetEmployeeHoursPerMonth,` mer ingående.

6.2.1 Använda klasser ur .NET Framework

För att underlätta implementationen har vi använt oss av klasser ur de klassbibliotek som följer med .NET Framework. Dessa klasser rör främst hanteringen av databasen TTDB.

`CDatabase` – hanterar kopplingen till databasen och kan användas för att exekvera SQL-uttryck i databasen..

`CRecordset` – hanterar exekvering av SQL-uttryck i databasen som ger poster i retur och används dessutom för hantering av eventuella returnerade poster.

`CString` – strängtyp som används tillsammans med `CDatabase` och `CRecordset`.

6.2.2 Detaljerad beskrivning av metoden `GetEmployeeHoursPerMonth`

Metoden GetEmployeeHoursPerMonth används för att ta reda på antalet timmar som en medarbetare har varit instämplad under en viss månad. För att demonstrera och testa webbtjänsten har vi skapat ett enkelt gränssnitt, se Figur 27. Gränssnittet används när vi anropar tjänsten med via Diveria PAR Direkt.

Figur 27 Grafiskt gränssnitt för TimeTicker

Eftersom webbtjänsten är en prototyp och metoderna i tjänsten till stor del liknar varandra har vi valt att endast demonstrera en metod. För att göra detta skapades ett gränssnitt för metoden. Det är ändå möjligt att anropa de andra metoderna via den hemsida, `http://smart-ws8.cse.kau.se/TimeTicker/TimeTicker.asmx`, som autogenereras av utvecklingsmiljön och på så vis fylla databasen med övrig information.

Gränssnittet har tre fält, två fält för inmatning av värden till inparametrarna i anropet av tjänsten och ett fält där resultatet från tjänsten visas. I Id-fältet skrivs en medarbetares idnummer och i Month-fältet skrivs ett datum in på formen ÅÅÅÅ-MM, där ÅÅÅÅ står för årtal och MM för månad. När metoden, `GetEmployeeHoursPerMonth`, anropas skapar den ett SQL-uttryck, baserat på inparametrarnas värden, samt en uppkoppling mot databasen, ett objekt av klassen `CDatabase`. Därefter skapas en hanterare för databasposter, ett objekt av klassen `CRecordset`, med vilken SQL-uttrycket exekveras i databasen. Om inga tidstämplingar finns i databasen för det angivna idnumret returnerar webbtjänsten ett svar som får gränssnittet att se ut som i Figur 28.

Figur 28 Resultat från GetEmployeeHoursPerMonth

Om det däremot finns tidsstämplingar för det angivna idnumret returnerar databasen dessa tidsstämplingar. Efter detta traverseras mängden med stämplingar och antal arbetade timmar per dag beräknas. Dessa summeras sedan till antal arbetade timmar för hela månaden. Resultatet kommer i gränssnittet att se ut som i Figur 29.

Figur 29 Resultat från GetEmployeeHoursPerMonth

Källkoden för metoden GetEmployeeHoursPerMonth finns i bilaga C. Alla de övriga metoderna beter sig på liknande sätt genom att koppla upp sig mot databasen och exekvera SQL-uttryck i den. Nedan följer en mer detaljerad beskrivning av hur några av ovanstående händelseförlopp utrycks i källkoden.

Uppkoppling mot databasen, ur TimeHandler.cpp (koden är tillrättalagd för exemplet):

```
1    CDatabase myCDatabase;
2    CString sDsn, sDriver = "MICROSOFT ACCESS DRIVER (*.mdb)";
3    CString sFile = "c:\\TimeTickerDB\\TTDB.mdb";
4    sDsn.Format("ODBC;DRIVER={%s};DSN='';DBQ=%s", sDriver, sFile);
5    TRY
6    {
7         myCDatabase.Open(NULL, false, false, sDns);
8         ...
9    }
10   CATCH(CDBException, e)
11   {
12        ...
13   }
```

Rad 1 till 4 innehåller initieringar och på rad 7 görs ett försök att med hjälp av initieringarna skapa en uppkoppling mot databasen. Eftersom det inte är säkert att detta fungerar, t ex på grund av att databasen inte är tillgänglig, ligger uppkopplingen inom ett try-catch-block, rad 5 till 13. Om något går fel i try-blocket kommer koden i catch-blocket att exekveras istället vilket gör det möjligt för oss att returnera ett felmeddelande.

Konstruktion av SQL-uttryck, ur TimeHandler.cpp (koden är tillrättalagd för exemplet):

```
1 String __gc* TimeHandler::GetSalaryByMonth(String *sId, String *sMonth)
2 {
3    int size = 0
4    CString sSqlQuery;
5    ...
6
7    sSqlQuery = "SELECT TimeTable.InTime, TimeTable.OutTime,
8              Employee.FirstName, Employee.Salary_h ";
9    sSqlQuery += "FROM TimeTable, Employee ";
10   sSqlQuery += "WHERE TimeTable.Id = Employee.Id AND TimeTable.Id = ";
11
12   size = sId->Length;
13   for(int i = 0; i < size; i++)
14        sSqlQuery += (TCHAR)sId->get_Chars(i);
15   sSqlQuery += " AND WDDate LIKE '";
16
17   size = sMonth->Length;
18   if(size > 7) // Datum kan vara både på formen ÅÅÅÅ-MM och ÅÅÅÅ-MM-DD
19        size = 7;
20   for(int i = 0; i < size; i++)
21        sSqlQuery += (TCHAR)sMonth->get_Chars(i);
22   sSqlQuery += "%'";
23   ...
24 }
```

På rad 7 inleds byggandet av SQL-uttrycket. Strängarna på raderna 7 till och med 10 talar om vilka fält ur vilka tabeller som skall returneras. På raderna 13 till och med 21 konverteras inparametrarna från String* till CString. Resultatet blir följande SQL-uttryck:

```
SELECT TimeTable.InTime, TimeTable.OutTime, Employee.FirstName,
       Employee.Salary_h
FROM   TimeTable, Employee
WHERE  TimeTable.Id = Employee.Id
AND    TimeTable.Id = <sId>
AND    TimeTable.WDDate LIKE '<sMonth>%'
```

Beräkning av antal timmar under angiven månad, ur TimeHandler.cpp (koden är tillrättalagd för exemplet):

```
...
1    CString sInTime, sOutTime, sFirstName, sHours, sRetMessage;
2    float fTimes[4], fTotal = 0.0;
3
4    if(!myCRecordset.IsEOF()) // Hämta förnamn
5         myCRecordset.GetFieldValue(2,sFirstName);
6
7    while(!myCRecordset.IsEOF()) // Räkna ut antal instämplade timmar
8    {
9         myCRecordset.GetFieldValue(0,sInTime);
10        myCRecordset.GetFieldValue(1,sOutTime);
11
12        fTimes[0] = atof(sInTime.Left(2));
13        fTimes[1] = atof(sInTime.Right(2));
14        fTimes[2] = atof(sOutTime.Left(2));
15        fTimes[3] = atof(sOutTime.Right(2));
```

```
16
17          // Räkna ut antal instämplade hela timmar och lägg till dem till
18          // totala antalet
19          fTotal += fTimes[2] - fTimes[0];
20          // Räkna ut antal delade timmar och lägg till dem till totala
21          // antalet
22          fTotal += (fTimes[3] - fTimes[1]) / 60;
23          myCRecordset.MoveNext();
24      }
25      myCRecordset.Close();
26
27      sHours.Format(_T("%.2f"), (fTotal));
28      sRetMessage = "Hours for " + sFirstName + " is : " + sHours;
...
```

`myCRecordset` innehåller en mängd tidstämplingar från databasen, utvalda med hjälp av SQL-uttrycket som skapades tidigare. Varje tidsstämpling erhålls var för sig, instämplingstid lagras i `sInTime` och utstämplingstid i `sOutTime`. Eftersom tidsvärdena i databasen är på formen TT:MM, delas strängarna upp, se rad 14 till 17, och timmar och minuter sparas undan var för sig. För att få åtkomst till timmar och minuter används metoderna `Left(int)` och `Right(int)`. Dessa metoder returnerar angivet antal tecken ur strängen från respektive håll, höger eller vänster. Tiden för instämplingen subtraheras sedan med tiden för utstämplingen och resultatet lagras i `fTotal`. Strängen `sHours` formateras med `Format(...)` för att på så sätt konvertera värdet hos `fTotal` till en sträng. `Format(...)` används för att formatera strängar på ett sätt som liknar `sprintf(...)` i C.

6.2.3 Problem under implementationen

Vid implementationen upptäckte vi att kopplingarna mellan .NET och IIS inte fungerade som de skulle. Att korrigera dessa fel försenade vår implementationsstart med en vecka eller två.

Ett annat problem var hur TimeTicker skulle kunna anropas utanför det lokala nätverket, eftersom vi vid försök hela tiden fick upp en inloggningsruta för nätverket. Det visade sig vara en inställning som gick att ändra.

Övriga problem som uppkom var mer implementationsrelaterade och löstes relativt snabbt med stöd från Internet. Microsofts produkter har åtskilliga användare som har stött på samma problem som vi och därför lagt ut lösningar via forum och hemsidor.

6.3 Testning

Vi testade TimeTickers duglighet genom att dels mata in felaktiga värden, och dels genom att undersöka hur den reagerade vid anrop från datorer belägna på olika nätverk och från olika typer av klienter.

När vi testade de parametrar som skickas in till webbtjänsten visade det sig att de värden som i databasen sparas som textsträngar tillåts innehålla vilka tecken som helst. Att förnamn, efternamn, instämplingstid, utstämplingstid och datum blir korrekt inmatade har vi i nuläget inga kontroller för. Vi är medvetna om problemet med detta eftersom vi i två metoder gör beräkningar med dessa värden, in- och utstämplingstid. Att vi valt att göra på det här sättet beror på att vårt program är en prototyp som inte har några krav på korrekthet.

Värden som i databasen sparas som heltal, t ex id och lön, fungerar bra. Om användaren matar in bokstäver eller tecken i stället för tal kastas ett undantag till webbtjänsten, vilken i sin tur returnerar ett felmeddelande till användaren för att göra denne uppmärksam på problemet.

Vi testade hur TimeTicker uppförde sig vid anrop från datorer i samma nätverk samt från datorer annat nätverk. Resultatet redovisas i Tabell 1.

Anrop	Fungerade
lokal host	Ja
annan dator i lokalt nätverk	Ja
från externt nätverk	Ja

Tabell 1 Testresultat av anrop från skilda platser

Eftersom vi försökt göra en tjänst vi tror skulle kunna vara användbar i ett affärssystem valde vi att testa vår prototyp på samma sätt som vi testade webbtjänsterna i kapitel 4. Vi testade därför också att anropa tjänsten med hjälp av två olika SOAP-klienter. Resultatet redovisas i Tabell 2.

Anrop	Fungerade
med hjälp av Diveria PAR Direkt	Nej
med hjälp av generic SOAP client	Ja

Tabell 2 Testresultat av anrop från SOAP-klienter

När vi anropade tjänsten med hjälp av Diveria PAR Direkt fick vi ingen kontakt med TimeTicker. Detta trots att vi konstruerade flera SOAP-meddelanden enligt mallen, se Bilaga D, från WSDL-dokumentet, se Bilaga E. Vi kunde dock göra HTTP-POST anrop till tjänsten från Diveria PAR Direkt. Vi använde oss senare även av Generic SOAP Client [18], som vi tidigare använt till postnummertjänsten, för att återigen testa att anropa tjänsten med ett SOAP-meddelande. Det visade sig nu fungera utmärkt. Troligtvis var det alltså inget fel på tjänsten som vi först trodde, utan felet beror antagligen på någon felaktighet i vårt SOAP-meddelande.

6.4 TimeTicker och framtiden

6.4.1 Begränsningar med TimeTicker

Eftersom TimeTicker är en prototyp behöver den inte erbjuda någon tillförlitlighet vad gäller belastning, åtkomst eller korrekthet hos innehållet. I vårt fall har vi haft problem med åtkomsten av webbtjänsten, eftersom IIS:en d v s webbservern, stängs ner med jämna mellanrum. Vad gäller belastning vet vi inte vad den klarar av. Databasen i sig har inga lås för läsning och skrivning. Det kan skapa problem när många skriver till den och läser från den, eftersom innehållet i databasen kan komma att bli inkorrekt. Ur säkerhetssynpunkt har den även en del begränsningar, som t ex att alla slutanvändare har åtkomst till samtliga metoder.

6.4.2 Möjlighet till vidareutveckling

Om TimeTicker skulle vidareutvecklas till en riktig webbtjänst finns en del förbättringar att göra. Förutom att ta hänsyn till ovanstående begränsningar vore det på sin plats att göra kontroll av inmatning av värden till TimeTicker. Under testningen upptäcktes en del möjliga fel som skulle kunna göras av användaren när denne anger in information till tjänsten. De inmatningar som sparas som strängar, instämplingstid, utstämplingstid och datum, men i grunden består av siffror, skulle kunna få ett felaktigt innehåll. Detta borde kontrolleras då en del av värdena används för matematiska beräkningar. En kontroll av att tiden skrivs in på formen TT:MM och datum ÅÅÅÅ-MM vore också lämpligt, så formateringen av värdena blir rätt eftersom detta också har betydelse i våra beräkningar.

Strukturen hos implementationen skulle kunna förbättras genom att bl a lyfta ut den kod som är gemensam för samtliga metoder, som t ex uppkoppling till databas, i privata metoder.

Vid användning av TimeTicker som en riktig webbtjänst skulle klienter till webbtjänsten kunna implementeras dels som webbsidor och dels byggas in i affärssystem. Detta skulle kunna vara ett sätt att åstadkomma olika vyer av webbtjänsten och begränsa åtkomsten till metoderna, så att t ex en medarbetare inte har rättighet att ange sin timlön. Medarbetare skulle egentligen bara ha tillgång till instämpling, utstämpling samt möjligen metoderna för att söka antalet arbetade timmar och uträkning av månadslön. De övriga metoderna, d v s där ett nytt medarbetarregister läggs in och där medarbetaren lön sätts, skulle endast vara åtkomliga för ett fåtal behöriga personer på t ex ekonomiavdelningen.

7 Resultat och rekommendationer

Från början var det främsta målet med uppsatsen att finna en, eller flera, webbtjänster som var lämpliga att integrera i affärssystemet Navision. Under arbetet framkom att det inte fanns några lämpliga tjänster.

Trots att vi inte fann någon användbar webbtjänst som vi kunde integrera fick vi kännedom om ett antal intressanta tjänster som håller på att utvecklas. Detta tyder på att det ännu är få som känner att de kan utnyttja tekniken med webbtjänster. De flesta tjänster som vi fann intressanta visade sig vara prototyper som skapats för att testa eller demonstrera möjligheterna med tekniken. Arbetet med att hitta webbtjänster som var användbara i Navision resulterade i två intressanta tjänster, en postnummertjänst och en telefonnummer- och adresstjänst. Utredningen om och i så fall hur tjänsterna kunde integreras i affärssystemet med hjälp av Diveria PAR Direkt visade att bara telefonnummer- och adresstjänsten fungerade. Dessvärre framkom att tjänsten endast var en prototyp. Detta ledde till att vi valde att inte gå vidare och integrera tjänsten i Navision.

I händelse av att vi inte hittade någon användbar webbtjänst hade vi i samråd med våra uppdragsgivare formulerat ytterligare ett mål, att öka våra kunskaper om webbtjänster genom att konstruera en egen. På så vis fick uppsatsen en ny riktning, och vi fick möjlighet att konstruera en prototypwebbtjänst, TimeTicker. På detta sätt lärde vi oss mer om XML, SOAP och webbtjänster. Vi rekommenderar andra att prova på att implementera webbtjänster. Detta eftersom det i vårt fall, med hjälp av Microsoft Visual Studio .NET, gick relativt enkelt.

Eftersom det ännu inte finns någon fastställd standard är utvecklare och företag fortfarande tveksamma till att använda tekniken. Arbetet med standardisering och utveckling av t ex transaktionshantering pågår dock vilket leder oss att tro att användning och utveckling av webbtjänster kommer expandera i framtiden. Det stora antalet prototyper som hittades visar också på att många är intresserade och överväger användningsområden. Vi föreslår att intresserade personer eller företag håller ögonen öppna eftersom vi tror att fler seriösa webbtjänster är på gång.

8 Summering av projektet

En av de viktigaste insikterna under arbetet har varit nyttan med att kunna skilja information och applikation åt med hjälp av XML och SOAP. Denna abstraktion utgör grunden för plattformsoberoende applikationer, d v s webbtjänster. Att göra ett arbete om webbtjänster har varit intressant eftersom det är ett aktuellt ämnesområde som för närvarande står under utveckling. Vi har fått lära oss hur webbtjänster fungerar samt tänkbara användningsområden för dem. Arbetet med att konstruera en egen tjänst i utvecklingsmiljön .NET har varit utmanande och intressant.

Vi trodde att det skulle finnas fler webbtjänster eftersom webbtjänster inte är helt nytt och har varit ett stort samtalsämne i datavärlden under en längre tid. Dessutom finns det gott om både verktyg och litteratur för utveckling och användning av den här typen av applikationer. Att det fanns gott om just verktyg och litteratur, och även information på Internet, underlättade utvecklandet av vår webbtjänstprototyp. Vi trodde att utvecklingen skulle kräva mer av oss som programmerare än vad den gjorde. Mycket av det vi förmodade skulle göras för hand autogenererades med hjälp av verktyg. Det som däremot krävdes av oss var att få verktygen att fungera.

Genom arbetet med uppsatsen har vi vunnit nya erfarenheter inom planering av projekt, t ex målformulering, tidsplanering, problemlösning och informationssökning. Att göra en fungerande tidsplan är något som tar tid att lära sig, vilket vi fick erfara när tidsplanen försköts bl a på grund av försenat material och problem med .NET-miljön. Detta ledde inte till några större problem, tidsplanen justerades och vi lade till några extra timmar.

Om vi skulle göra om arbetet skulle vi vilja lägga mer tid på utvecklingen av vår egen prototyp, och genom att förbättra den, öka våra kunskaper. Vi skulle bl a vilja undersöka hur säkerheten hos vår prototyp skulle kunna förbättras. Detta eftersom vi tror att oklarheten kring säkerhet och webbtjänster är ett av skälen till att tekniken ännu inte fått så stort genomslag.

Referenser

[1] Postens Adressregister, www.par.se, 2003-05-05

[2] **Eriksson**, **Anders**, Delägare av Diveria AB och handledare, muntligen 2003-01-17.

[3] Diveria AB, www.diveria.se, 2003-05-05

[4] Invid, *”Navision Attain”*, ”http://www.invid.se/tjanster/fokus_aff_2.asp?menuid=3, 2003-04-28

[5] Chappell, David A. och Jewell, Tyler, (2002) *“Java web services”*, 1:a uppl., Beijing, O’Reilly

[6] Coyle, Frank P., (2002) *“XML, web services, and the data revolution”*, 1:a uppl., Boston (Mass), Addison-Wesley

[7] Skonnard, Aaron, *“The XML Files: The Birth of Web Services”*, 2002-10, http://msdn.microsoft.com/msdnmag/issues/02/10/XMLFiles/default.aspx, 2003-01-28

[8] O’Toole, Annraí, Cape Clear Software, CEO, *“Clear Thinking with Annraí; SOAP, Web Services, Web Services: The Third Way* och *Solution to Business Integration”*, 2003, http://www.capeclear.com/clear_thinking.shtml, 2003-04-28

[9] Carnegie Mellon Software Engineering Institute, *“Remote Procedure Call – Software Technology Review”*, 2000-09-22, http://www.sei.cmu.edu/str/descriptions/rpc.html, 2003-04-28

[10] W3C, *“Web Service Description Language (WSDL)”*, 2001-03-15, http://www.w3.org/TR/wsdl, 2003-04-05

[11] Koningsberg, Paul, *“Simple Object Access Protocol”*, 2000-09-25, http://ccm.redhat.com/asj/soap/, 2003-04-08

[12] Tabor, Robert, (2001) *”Microsoft .NET XML Web Services*”, 1:a uppl. Indianapolis, Sams

[13] W3C, RFC2616 *“Hypertext Transfer Protocol – HTTP/1.1, Section 9 – Method Definitions”*, 2002-11-07, http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html, 2001-05-01

[14] W3C, *”XML Tutorial”*, http://www.w3schools.com/xml/default.asp, 2003-03-06

[15] **Rikardsson**, **Gunnar**, Marknadsavdelningen, Dun & Bradstreet Sverige AB, Sundbyberg, mail, 2003-02-20

[16] E-faktura, ”www.e-faktura.com”, 2003-05-05

[17] **Aspenryd**, **Henry**, Marotz AB, mail, 2003-03-14

[18] SQLData System, Inc., *“Generic SOAP Client”*, www.soapclient.com

[19] **Eriksson**, **Sofie**, e-master, Östgöta Enskilda Bank, Telefonsamtal 2003-03-21

[20] **Polland**, **Fredrik**, Affärsutveckling – Företagsinformation, Upplysnings Centralen, Stockholm, mail 2003-03-13

[21] Microsoft Corporation, *"Defining the Basic Elements of .NET"*, 2003-01-24, http://www.microsoft.com/net/basics/whatis.asp, 2003-05-07

[22] J. Conrad, P. Dengler, B. Francis, J. Glynn, B. Harvey, B. Hollis, R. Ramachandran, J. Schenken, S. Short och C. Ullman (2001), *"Introducing .NET*", 1:a uppl. Birmingham, Wrox Press Inc

[23] Templeman, Julian (2002), *"Microsoft Visual C++.NET steg för steg"*, 1:a uppl. Sundbyberg, Pagina

A Akronymer

ADO – ActiveX Data Object

ASP – Active Server Pages

COM – Component Object Model

CORBA – Common Object Request Broker Architecture

DLL – Dynamic Link Library

DSN – Data Source Name

HTML – Hyper Text Markup Language

HTTP – Hyper Text Transfer Protocol

IIS – Internet Information Service

OBO – One-By-One

ODBC – Open Database Connectivity

PAR – Postens AdressRegister

RMI – Remote Method Invocation

RPC – Remote Procedure Call

SMTP – Simple Mail Transfer Protocol

SOAP – Simple Object Access Protocol

SQL – Structured Query Language

SSL – Secure Sockets Language

TTDB – TimeTickerDatabase

URL – Uniform Resource Locator

WSDL – Web Service Description Language

XML – eXtended Markup Language

XSD – XML Schema Definition

B Utvecklingsmiljö

B.1 Microsoft .NET

Microsoft .NET är en samling olika verktyg för att underlätta integrering av applikationer på hög nivå. Detta skall åstadkommas med XML-webbtjänster, små återanvändningsbara program för datautbyte. Tanken är att utvecklare med hjälp av webbtjänster skall kunna sammanfoga information, människor, system och maskiner till nätverk på ett nytt och bättre sätt [21].

Med hjälp av Microsoft Visual Studio .NET, ett utvecklingsverktyg, och Microsoft .NET Framework, Microsofts nya exekveringsmiljö, skall utvecklare förhållandevis enkelt kunna skapa webbtjänster och integrera dem med andra applikationer. Utvecklare behöver inte lära sig något nytt språk, eftersom Microsoft .NET tillhandahåller en mängd kända programmeringsspråk.

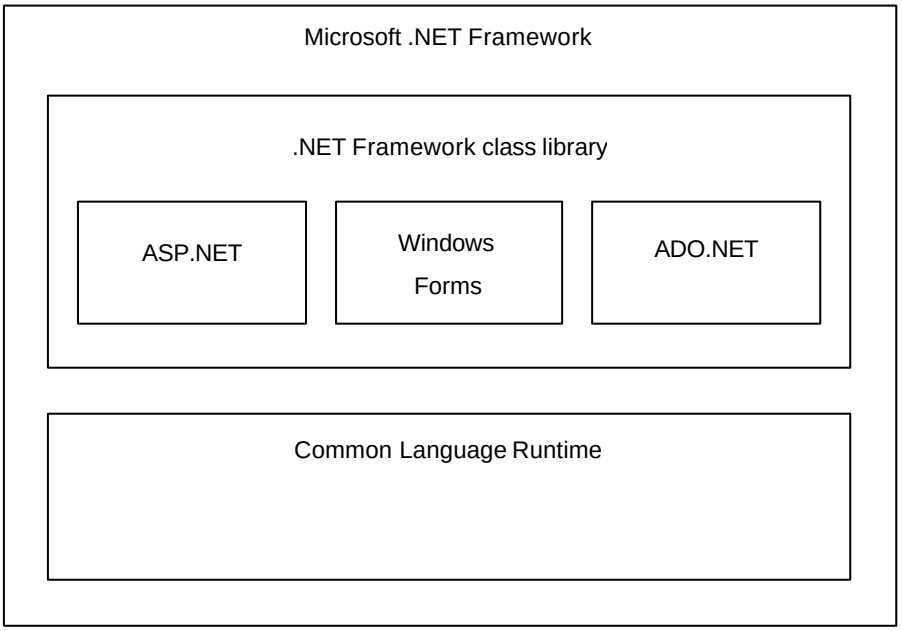
B.1.1 .NET Framework

.NET Framework är en exekveringsmiljö som bl.a. tar hand om minneshantering, tillförlitlighet och säkerhet hos de applikationer som exekveras i den. Miljön erbjuder en gemensam grund för utveckling av många olika typer av program, både vanliga fönsterprogram och webbaserade program. I huvudsak består .NET Framework av två komponenter, Common Language Runtime, CLR och .NET Framework class library.

CLR tillhandahåller de vanliga tjänsterna som program behöver så som minneshantering, dynamiskt bindning och liknande vid exekvering, och sköter även språkintegering. Det är i .NET möjligt att programmera i ett antal olika programspråk och sedan låta .NET och CLR integrera resultatet.

.NET Framework class library, som är ett stort klassbibliotek som är uppdelat i tre huvuddelar efter den funktionalitet de stödjer: ASP.NET, Windows Forms och ADO.NET. ASP.NET innehåller klasser för att konstruera webbtjänster och webbapplikationer. Windows Forms innehåller klasser som underlättar konstruktion av grafiska användargränssnitt för vanliga applikationer. ADO.NET innehåller klasser för att hantera anslutningar mellan applikationer och databaser [21].

Klassbiblioteket tillhandahåller även standardfunktionalitet så som input/output, strängmanipulation, säkerhet, nätverkskommunikation, texthantering, trådhantering mm [22].



Figur 30 .NET Frameworks beståndsdelar

B.1.2 Microsoft Visual studio .NET

Microsoft Visual Studio .NET är ett utvecklingsverktyg som bygger på .NET Framework och därigenom tillhandahåller stöd för att arbeta med XML-webbtjänster. Med detta verktyg kan kod skrivas i Visual Basic, C++ eller C#. Applikationer och webbtjänster kan skapas i ett programspråk och används av applikationer eller webbtjänster skapat i ett annat språk under förutsättning att även det språket stöds av Visual Studio .NET. Detta stärker möjligheten att använda befintliga webbtjänster och applikationer för att bygga nya lösningar.

Med Visual Studio .NET skapas automatiskt de nödvändiga XML- och SOAP-gränssnitt för att göra en applikation till en webbtjänst. Utvecklarna kan koncentrera sig på att skapa applikationen i stället för att oroa sig över hur den skall förvandlas till en webbtjänst.

B.2 Internet Information Service, IIS

Microsoft Internet Information Service, IIS, är en webbserver för Windows. Den är avsedd för företag, organisationer, och även privatpersoner som vill etablera sig på Internet. En IIS behövs för att kunna göra en webbtjänst tillgänglig via Internet, så att andra kan få åtkomst till den.

Varför IIS har blivit så populärt som det blivit, beror troligtvis på att det är gratis, men med villkoret att Windows finns installerat på datorn i fråga.
I vårt fall använder vi oss av Internet Information Service 5.0.
För att kunna publicera ett dokument, eller som i vårt fall en webbtjänst, på Internet, skall de aktuella filerna placeras i `\inetpub\wwwroot`-katalogen. I denna katalog läggs alla virtuella kataloger, d v s kataloger som skall kunnas nås från Internet. .NET skapar automatiskt en virtuell katalog åt användaren när denne utvecklar en webbtjänst. För att få tillgång till tjänsten eller dokumenten från Internet skrivs `http://servernamn/filnamn` i browsern.

C Källkod för TimeHandler::GetEmployeeHoursPerMonth

C.1 Ur TimeHandler.h:

```
// Metoden räknar ut antal instämplade timmar för angiven anställd och månad
[System::Web::Services::WebMethod]
String __gc* GetEmployeeHoursPerMonth(String *sId, String *sMonth);
```

C.2 Ur TimeHandler.cpp:

```
String __gc* TimeHandler::GetEmployeeHoursPerMonth(String *sId, String *sMonth)
{
	CDatabase myCDatabase;
	CString sSqlQuery, sGetId, sGetInTime, sGetOutTime, sDsn;
	CString sRetMessage = "GetEmployeeHoursPerMonth";
	CString sDriver = "MICROSOFT ACCESS DRIVER (*.mdb)";
	CString sFile = "c:\\TimeTickerDB\\TTDB.mdb";
	int size = 0;

	sDsn.Format("ODBC;DRIVER={%s};DSN='';DBQ=%s", sDriver, sFile);

//*****************************************************************
// Bygger strängen med SQL-frågan
// SQL: "SELECT TimeTable.InTime, TimeTable.OutTime, Employee.FirstName
// FROM TimeTable, Employee
// WHERE TimeTable.Id = Employee.Id AND TimeTable.Id = <sId> AND TimeTable.WDDate
// LIKE '<sMonth>%'"
//*****************************************************************
	sSqlQuery = "SELECT TimeTable.InTime, TimeTable.OutTime,
	Employee.FirstName ";
	sSqlQuery += "FROM TimeTable, Employee ";
	sSqlQuery += "WHERE TimeTable.Id = Employee.Id AND TimeTable.Id = ";
	size = sId->Length;
	for(int i = 0; i < size; i++)
		sSqlQuery += (TCHAR)sId->get_Chars(i);
	sSqlQuery += " AND WDDate LIKE '";

	size = sMonth->Length;
	if(size > 7)
		size = 7;
	for(int i = 0; i < size; i++)
		sSqlQuery += (TCHAR)sMonth->get_Chars(i);
	sSqlQuery += "%'";
//*****************************************************************
	sRetMessage = sSqlQuery;

	TRY
	{
		float fTotal = 0.0;

		myCDatabase.Open(NULL, false, false, sDsn);
		CRecordset myCRecordset(&myCDatabase);

	myCRecordset.Open(CRecordset::forwardOnly,sSqlQuery,CRecordset::readOnly);

		if( !myCRecordset.IsEOF() )
		{
```

```
                CString sInTime, sOutTime, sFirstName, sHours;
                float fTimes[4];

                if(!myCRecordset.IsEOF()) // Hämta förnamn
                     myCRecordset.GetFieldValue(2,sFirstName);

                // Räkna ut totalt antal instämplade timmar
                while(!myCRecordset.IsEOF())
                {
                     myCRecordset.GetFieldValue(0,sInTime);
                     myCRecordset.GetFieldValue(1,sOutTime);

                     fTimes[0] = atof(sInTime.Left(2));
                     fTimes[1] = atof(sInTime.Right(2));
                     fTimes[2] = atof(sOutTime.Left(2));
                     fTimes[3] = atof(sOutTime.Right(2));

                     // Beräkna antal hela timmar och lägg till totalt antal
                     fTotal += fTimes[2] - fTimes[0];
                     // Beräkna antal delade timmar och lägg till totala
                     fTotal += (fTimes[3] - fTimes[1]) / 60;

                     myCRecordset.MoveNext();
                }
                myCRecordset.Close();
                sHours.Format(_T("%.2f"), (fTotal));
                sRetMessage = "Hours for " + sFirstName + " is : " + sHours;
           }
           else
                sRetMessage = "No timerecords found";

           myCDatabase.Close();
     }
     CATCH(CDBException, e)
     {
           sRetMessage = "CDBExeption occured " + e->m_strError;
     }
     END_CATCH;
     return sRetMessage;
}
```

D Mallar för anrop till TimeTicker

Innehållet i denna bilaga är hämtat från de autogenererade webbsidor som skapas till varje webbtjänst som konstrueras med hjälp av Microsoft Visual Studio .NET. Varje metod i webbtjänsten kan testas via dessa sidor och de innehåller också mallar för hur de meddelanden som skall skickas till tjänsten skall se ut.

TimeHandler

Click here for a complete list of operations.

GetEmployeeHoursPerMonth

Test

To test the operation using the HTTP GET protocol, click the 'Invoke' button.

Parameter	Value
sId:	
sMonth:	

Invoke

SOAP

The following is a sample SOAP request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /TimeTicker/TimeTicker.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "TimeTickerWebService/GetEmployeeHoursPerMonth"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetEmployeeHoursPerMonth xmlns="TimeTickerWebService">
      <sId>string</sId>
      <sMonth>string</sMonth>
    </GetEmployeeHoursPerMonth>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
```

```
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetEmployeeHoursPerMonthResponse xmlns="TimeTickerWebService">

<GetEmployeeHoursPerMonthResult>string</GetEmployeeHoursPerMonthResult>
    </GetEmployeeHoursPerMonthResponse>
  </soap:Body>
</soap:Envelope>
```

HTTP GET

The following is a sample HTTP GET request and response. The **placeholders** shown need to be replaced with actual values.

```
GET
/TimeTicker/TimeTicker.asmx/GetEmployeeHoursPerMonth?sId=string&sMonth=stri
ng HTTP/1.1
Host: localhost
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="TimeTickerWebService">string</string>
```

HTTP POST

The following is a sample HTTP POST request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /TimeTicker/TimeTicker.asmx/GetEmployeeHoursPerMonth HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: length

sId=string&sMonth=string
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="TimeTickerWebService">string</string>
```

E WSDL-dokument för TimeTicker::GetEmployeeHoursPerMonth

Denna bilaga innehåller de delar av TimeTickers WSDL-dokument som berör metoden GetEmployeeHoursPerMonth för att ge läsare en inblick i hur dessa dokument ser ut.

```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:s0="TimeTickerWebService"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="TimeTickerWebService"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
- <types>
- <s:schema elementFormDefault="qualified"
    targetNamespace="TimeTickerWebService">
- <s:element name="GetEmployeeHoursPerMonth">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="sId" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="sMonth" type="s:string" />
    </s:sequence>
    </s:complexType>
    </s:element>
- <s:element name="GetEmployeeHoursPerMonthResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1"
    name="GetEmployeeHoursPerMonthResult" type="s:string" />
    </s:sequence>
    </s:complexType>
    </s:element>
  <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
    </types>
- <message name="GetEmployeeHoursPerMonthSoapIn">
  <part name="parameters" element="s0:GetEmployeeHoursPerMonth" />
    </message>
- <message name="GetEmployeeHoursPerMonthSoapOut">
  <part name="parameters" element="s0:GetEmployeeHoursPerMonthResponse" />
    </message>
- <message name="GetEmployeeHoursPerMonthHttpGetIn">
  <part name="sId" type="s:string" />
  <part name="sMonth" type="s:string" />
    </message>
```

```
- <message name="GetEmployeeHoursPerMonthHttpGetOut">
  <part name="Body" element="s0:string" />
    </message>
- <message name="GetEmployeeHoursPerMonthHttpPostIn">
  <part name="sId" type="s:string" />
  <part name="sMonth" type="s:string" />
    </message>
- <message name="GetEmployeeHoursPerMonthHttpPostOut">
  <part name="Body" element="s0:string" />
    </message>
- <portType name="TimeHandlerSoap">
- <operation name="GetEmployeeHoursPerMonth">
  <input message="s0:GetEmployeeHoursPerMonthSoapIn" />
  <output message="s0:GetEmployeeHoursPerMonthSoapOut" />
    </operation>
    </portType>
- <portType name="TimeHandlerHttpGet">
- <operation name="GetEmployeeHoursPerMonth">
  <input message="s0:GetEmployeeHoursPerMonthHttpGetIn" />
  <output message="s0:GetEmployeeHoursPerMonthHttpGetOut" />
    </operation>
    </portType>
- <portType name="TimeHandlerHttpPost">
- <operation name="GetEmployeeHoursPerMonth">
  <input message="s0:GetEmployeeHoursPerMonthHttpPostIn" />
  <output message="s0:GetEmployeeHoursPerMonthHttpPostOut" />
    </operation>
    </portType>
- <binding name="TimeHandlerSoap" type="s0:TimeHandlerSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"
    />
- <operation name="GetEmployeeHoursPerMonth">
  <soap:operation soapAction="TimeTickerWebService/GetEmployeeHoursPerMonth"
    style="document" />
- <input>
  <soap:body use="literal" />
    </input>
- <output>
  <soap:body use="literal" />
    </output>
    </operation>
    </binding>
- <binding name="TimeHandlerHttpGet" type="s0:TimeHandlerHttpGet">
  <http:binding verb="GET" />
- <operation name="GetEmployeeHoursPerMonth">
  <http:operation location="/GetEmployeeHoursPerMonth" />
- <input>
  <http:urlEncoded />
    </input>
- <output>
  <mime:mimeXml part="Body" />
    </output>
    </operation>
```

```
    </binding>
- <binding name="TimeHandlerHttpPost" type="s0:TimeHandlerHttpPost">
  <http:binding verb="POST" />
- <operation name="GetEmployeeHoursPerMonth">
  <http:operation location="/GetEmployeeHoursPerMonth" />
- <input>
  <mime:content type="application/x-www-form-urlencoded" />
    </input>
- <output>
  <mime:mimeXml part="Body" />
    </output>
    </operation>
    </binding>
- <service name="TimeHandler">
- <port name="TimeHandlerSoap" binding="s0:TimeHandlerSoap">
  <soap:address location="http://localhost/TimeTicker/TimeTicker.asmx" />
    </port>
- <port name="TimeHandlerHttpGet" binding="s0:TimeHandlerHttpGet">
  <http:address location="http://localhost/TimeTicker/TimeTicker.asmx" />
    </port>
- <port name="TimeHandlerHttpPost" binding="s0:TimeHandlerHttpPost">
  <http:address location="http://localhost/TimeTicker/TimeTicker.asmx" />
    </port>
    </service>
    </definitions>
```