Avdelning för datavetenskap

Daniel Jansson och Mikael Jansson

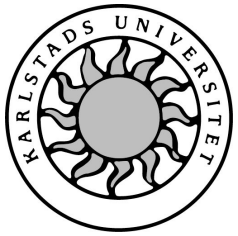
Utvecklingen av en Instant Messaging klient som en språkwrapper

The development of an Instant Messaging client as a language wrapper

Datavetenskap
D-uppsats (20p)

Datum:	07-06-07
Handledare:	Donald F. Ross
Examinator:	Martin Blom
Löpnummer:	D2006:nn

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se



Datavetenskap

Daniel Jansson och Mikael Jansson

Utvecklingen av en Instant Messaging klient som en språkwrapper

---

Magisteruppsats

2007:xx

Utvecklingen av en Instant Messaging klient som en språkwrapper

Daniel Jansson och Mikael Jansson

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

Daniel Jansson

Mikael Jansson

Godkänd, 2007-06-07

Opponent: Thijs Holleboom

Handledare: Donald F. Ross

Examinator: Martin Blom

Sammanfattning

Denna rapport beskriver utvecklingen av en Instant Messaging klient som implementerats som en språkwrapper. Klienten använder ett känt öppen källkods-bibliotek vid namn libGaim för att få tillgång till Instant Messaging tjänster. Klienten fungerar som en språkwrapper till detta bibliotek, och låter andra applikationer nyttja biblioteket genom ett databasgränssnitt. Arbetet har utförts åt och hos Evolve i Karlstad. Systemet som utvecklats klarar att hantera vanliga Instant Messaging-funktioner så som uppkoppling av konto, nedkoppling, ändring av status, visa kompisikoner, och hämtning av information angående kompisar.

I beskrivandet av arbetet med detta system ges bakgrundsinformation angående wrappers och Instant Messaging, och dessutom angående de verktyg och metoder som använts likt byggverktyget SCons och utvecklingsmetoden eXtreme Programming. Valet av lösning och design presenteras.

Som en del av arbetet undersöks språkwrappers med hjälp av av ett antal tester. Dessa tester visade på vilka skillnader som kan väntas beroende på valet av språkwrapper. En språkgenerisk språkwrapper (som skapar ett gränssnitt mot flera språk) behöver inte utgöra någon större prestandaoverhead jämfört med en språkstatisk språkwrapper (som skapar ett gränssnitt mot *ett* språk).

Resultatet av arbetet blev ett system som Evolve blev nöjda med och som uppfyllde alla krav som ställdes, antingen fullständigt eller delvis.

The development of an Instant Messaging client as a language wrapper

This report describes the development of an Instant Messaging client that is implemented as a language wrapper. The client uses a well known open source library, known as libGaim, to access Instant Messaging services. The client works as a language wrapper for this library, and allows other applications to use the library by means of a database interface. The work has been carried out at and for Evolve in Karlstad. The system that has been developed can handle common Instant Messaging operations such as connecting a user account, disconnecting, changing an accounts status, showing buddy icons, and collecting information regarding buddies.

In the description of the work with this system, background information is given regarding wrappers and Instant Messaging, and also regarding the tools and methods used, such as the build tool SCons and the development method eXtreme Programming. The choice of solution and design is presented.

As a part of the work, language wrappers are evaluated using a number of tests. These tests show the differences that can be expected depending on the choice of language wrapper. A language generic language wrapper (which creates an interface for several languages) need not imply any significant performance overhead compared to a language static language wrapper (which creates an interface towards *one* language).

The result of the work we have done was a system that Evolve was pleased with and

that satisfies all the requirements, either completely or partially.

Innehåll

1 Inledning **1**

1.1 Introduktion 1

1.2 Disposition 3

2 Bakgrund **5**

2.1 Introduktion 5

2.2 Wrappers 6

2.2.1 Vad är en wrapper? 6

2.2.2 Wrappers historia 9

2.2.3 Användningsområden 10

2.2.4 Wrappers i denna rapport 11

2.2.5 Språkwrappers 13

2.2.6 Är wrappers en bra lösning? 16

2.2.7 Alternativ 17

2.3 Instant Messaging 18

2.3.1 Vad är Instant Messaging? 18

2.3.2 Protokoll 21

2.3.3 Gaim och libGaim 22

2.4 Sammanfattning 23

3 Metoder och verktyg **25**

3.1 Introduktion 25

3.2 Utvecklingsmetod 26

3.3 MySQL 27

3.4 Språk 27

3.4.1 C och C++ 27

3.4.2 PHP 29

3.5 Byggverktyg 30

3.5.1 GNU make 30

3.5.2 GNU Autotools 32

3.5.3 SCons 36

3.6 Doxygen 38

3.7 Versionshantering 39

3.8 Sammanfattning 40

4 Analys av möjliga lösningar **41**

4.1 Introduktion 41

4.2 Kravinsamling och specifikation 41

4.3 Lösningsförslag 42

4.3.1 PHP-extension till libGaim 43

4.3.2 Jabber-klient 51

4.3.3 libGaim-wrapper 56

4.4 Sammanfattning 59

5 Prototyp av libGaim-wrapper **60**

5.1 Introduktion 60

5.2 libGaim 61

5.2.1 Struktur 61

5.2.2 Kompilering och länkning av libGaim 68
5.2.3 Kritik mot libGaim 70
5.3 Öppen källkodsutveckling och Gaim 70
5.4 Prestandatest av språkwrappers 72
5.4.1 Utförande 72
5.4.2 Resultat 74
5.4.3 Utvärdering 74
5.5 Design 75
5.5.1 Systemöversikt 75
5.5.2 Meddelandehantering 78
5.5.3 Databas 80
5.6 Gaim blir Pidgin 81
5.7 Sammanfattning 82

6 Utvärdering av libGaim-wrapper **83**
6.1 Introduktion 83
6.2 Val av lösning 83
6.3 Val av design 84
6.4 Resultat 86
6.5 Sammanfattning 89

7 Sammanfattning och slutsats **90**
7.1 Introduktion 90
7.2 Utvärdering 90
7.3 Framtida arbete 94
7.3.1 libGaim-wrappern 94
7.4 Sammanfattning 95

Referenser **97**

A Definitioner och förkortningar **103**

A.1 Defintioner 103

A.2 Förkortningar 105

Figurer

1.1 Systemöverblick 2

2.1 Generell wrapper beskrivning 8
2.2 Jämförelse mellan filter och wrapper 12
2.3 Språkgenerisk språkwrapper 15
2.4 En talk session utförd över PuTTY i Windows XP 19
2.5 ICQ meddelande med IM klienten Gaim i Windows Vista 20

3.1 Arv i C 29
3.2 Hur Autotools verktyg interagerar 35
3.3 Doxygen dokumentation för funktionen setName 39

4.1 Första designen 44
4.2 Version 2 av vår PHP-extension 45
4.3 Version 3 av vår PHP-extension 51
4.4 Systemet implementerat med Jabber 54
4.5 a) PHP-extension-design och b) C++-wrapper-design 57

5.1 libGaim's viktigaste delar 62
5.2 Kommunikationsdiagram över uppkoppling med libGaim 66
5.3 libGaim's uppdelning av logik och gränssnitt 67
5.4 Strukturen av kompislistan 69

5.5 Klassdiagram över systemet 77
5.6 E/R-diagram 80

7.1 Tidsplanen i början av projektet 91
7.2 Den faktiska tidsplanen som blev resultatet 92

Kapitel 1

Inledning

1.1 Introduktion

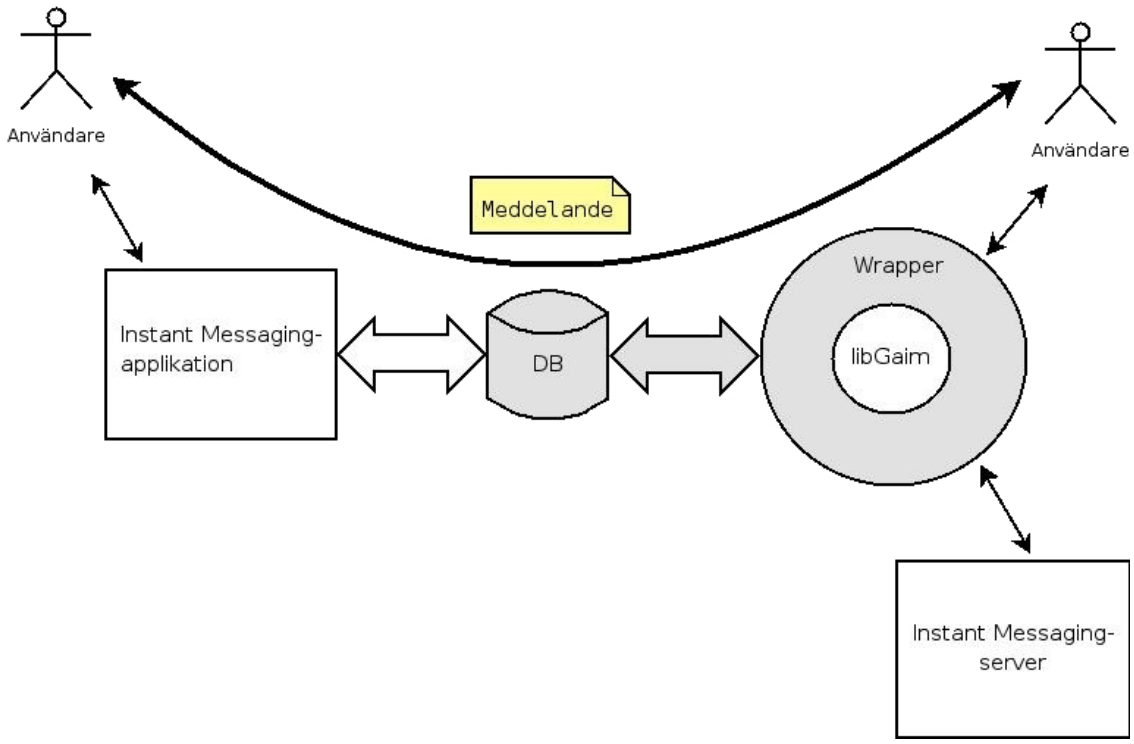
I takt med att Instant Messaging eller textmeddelanden blir mer och mer populärt ökar även mängden Instant Messaging-protokoll. Gaim-projektet är ett öppen källkods-projekt som innehåller ett bibliotek (libGaim) som försöker generalisera användningen av olika protokoll för att samma bibliotek ska kunna användas till att använda flera olika IM-tjänster som ICQ och MSN.

Evolve AB i Karlstad sökte en möjlighet att använda Instant Messaging (som inte var knutet till endast ett specifikt protokoll) i skriptspråk, och då huvudsakligen PHP. Man kände till C-biblioteket libGaim som möjliggör åtkomst åt Instant Messaging-funktioner där stöd för olika IM-protokoll ges genom att koppla in nya insticksmoduler. Detta blev grunden till problemområdet för detta arbete – att skapa en koppling mellan libGaim och PHP genom att skapa ett skal (en wrapper) till libGaim.

Syftet med arbetet som beskrivs i denna rapport är att designa och utveckla ett system som gör det möjligt för skriptspråk (huvudsakligen PHP) att använda Instant Messaging-tjänster. Som en del av arbetet undersöks språkwrappers, som använts till att koppla samman IM-biblioteket libGaim och den klientkod som använder det. I figur 1.1

visas hela systemet, där delarna i grått är de som implementerats i detta arbete. LibGaim är kärnan i systemet, och runt det ligger en språkwrapper som hanterar meddelanden till och från klienten genom ett databasgränssnitt. En klientapplikation kan alltså skicka in meddelanden till databasen i form av poster vilka sedan hämtas ut av libGaim-wrappern och hanteras av libGaim.

Användaren på vänster sida i figuren använder en klient som utnyttjar IM-tjänster genom libGaim-wrappern (och når denna genom ett databasgränssnitt). Användaren på höger sida däremot använder antingen en annan IM-klient, till exempel Gaim eller Meebo, eller wrapper-klienten för att kommunicera med andra IM-användare.



Figur 1.1: Systemöverblick

Rapporten beskriver hur detta arbete fortskred, vilka problem som uppstod, och hur de löstes. Dessutom ges en redovisning av wrappers och språkwrappers inom

datavetenskap. Språkwrappers undersöks vidare i rapporten genom ett antal tester som jämför prestandan hos ett antal språkwrapper-implementationer. Anledningen till att wrappers tas upp i denna rapport har sitt ursprung i vårt första möte med vår handlerade på Evolve då vi diskuterade problemområdet. Begreppet *wrapper* togs upp och vi började fundera på vad som innefattas i detta begrepp och varifrån det kommer. Då wrapper är ett väldigt brett begrepp fann vi det mer specialiserade begreppet *språkwrapper* vilket mer tydligt definierar den typ av mjukvara vi utvecklat. Vi fann att språkwrappers kunde delas in i två grupper – språkstatiska och språkgeneriska – och för att få en överblick över prestandaskillnaden mellan dessa två grupper utfördes ett antal tester som presenteras i avsnitt 5.4.

Den slutgiltiga produkten har lämnats över till Evolve för vidare utveckling och testning. All respons vi mottagit från Evolve angående systemet har hittills varit mycket positiv och de är mycket nöjda med vår insats.

1.2 Disposition

Resterande del av rapporten har följande disposition:

Kapitel 2 innehåller bakgrundsinformation angående projektet. I detta ingår ett delkapitel om wrappers, där begreppet presenteras och definieras. Historien bakom begreppet wrappers ges, och 5 olika syften för wrappers ges. Dessutom ges en ny definition av wrappers för användning i denna rapport. Därefter presenteras språkwrappers, som är en wrapper för att göra det möjligt att använda en kodmodul skriven i ett språk att användas i ett annat språk. I detta kapitel ges även en presentation av Instant Messaging; vad det är och vilken som är historien bakom det. En kort beskrivning av Gaim och libGaim ges även.

I kapitel 3 presenteras de verktyg och utvecklingsmetoder som använts i projektet. De delar av eXtreme Programming som använts beskrivs kortfattat. Utöver det motiveras

valet av databassystemet MySQL och språken C++ och PHP. Valet av byggverktyg diskuteras djupgående. Dokumenteringsverktyg och versionshanteringsverktyg presenteras kortfattat.

Kapitel 4 beskriver resultatet av analysfasen av utvecklingen. Här presenteras kravspecifikationen och tre olika lösningar och varför de valdes eller inte valdes.

I kapitel 5 ges en beskrivning av slutprodukten. Dessutom ges en presentation av strukturen av libGaim och hur det fungerar. I detta kapitel presenteras även resultatet av ett antal tester för att jämföra prestandaskillnaden mellan språkwrappers.

Kapitel 6 innehåller en utvärdering av systemet som utvecklats. Här motiveras val som gjorts under utvecklingen, både val av lösning och val av en design för den lösningen. Till sist sammanställs en lista av hur väl kraven i kravspecifikationen uppfylls av systemet.

I det sista kapitlet, kapitel 7, sammanfattas och utvärderas projektet som helhet. Tidsaspekter tas upp. Utöver det presenteras även en lista av framtida arbete med systemet – vad som kan vidareutvecklas.

Kapitel 2

Bakgrund

2.1 Introduktion

Ett viktigt mjukvarumönster som används inom så gott som all utveckling idag[47] är wrappers. En wrapper beskrivs i Free On-Line Dictionary of Computing som *kod som kombineras med en annan del av kod för att bestämma hur den koden ska exekveras*[12]. En wrapper fungerar på detta sätt som ett gränssnitt mellan anroparen och den wrappade koden (i avsnitt 2.2.1 beskrivs mer ingående vad som menas med begreppet wrapper).

Som en del av denna rapport undersöks språkwrappers (se avsnitt 2.2.5 för en detaljerad beskrivning av språkwarappers). Ett exempel på en språkwrapper är när ett mjukvarubibliotek ska användas av en viss applikation, men detta bibliotek är skrivet i ett språk som skiljer sig från applikationens språk. För att applikationen ska kunna använda biblioteket krävs en wrapper som skapar kopplingen mellan de två språken.

I detta kapitel ges bakgrundsinformation till generella wrappers och språkwrappers. Generella wrappers ges en definition genom att sammanställa en samling definitioner. Dessutom ges en historisk bakgrund till wrappers, där begreppets ursprung försöker finnas.

Eftersom applikationen som utvecklats är en Instant Messaging-klient ges i detta

kapitel bakgrundsinformation till Instant Messaging för att ge en djupare förståelse för problemområdet.

Dessutom introduceras libGaim som är Instant Messaging-biblioteket som arbetet har kretsat runt att skapa en wrapper till.

2.2 Wrappers

2.2.1 Vad är en wrapper?

Ordet wrapper betyder omslag, det vill säga något som skalar av ett visst objekt från omvärlden. En wrapper inom mjukvaruutveckling är just ett skal som hanterar input och/eller output till/från objektet inne i skalet.

En exakt definition för wrappers finns inte då detta skrivs. De online-uppslagsverk som finns har alla egna och ofta divergerande uppgifter om wrappers innebörd, och vetenskapliga rapporter ansätter vanligtvis egna definitioner angående wrappers. Detta är inte underligt då wrappers betydelse är fast knuten till dess sammanhang. Till exempel, inom Pattern Recognition[22] används begreppet wrapper till att betyda en viss utvärderingsmetod för att sortera vilka egenskaper hos en mängd data som ska användas i en inlärningsalgoritm. Denna typ av sorteringsmetod använder inlärningsalgoritmen själv för att sortera ut egenskaperna; med andra ord görs ett skal till inlärningsalgoritmen som utnyttjar denna.

På Wikipedia[63] finns två olika beskrivningar där begreppet wrapper används inom datavetenskapen. I den första beskrivningen används wrapper för att beteckna ett objektorienterat designmönster[14] med syfte att låta klasser fungera ihop som annars inte skulle kunna göra det. Beskrivningen utvecklas sedan till att wrappers kan ha syftet att öka säkerheten och addera ett lager abstraktion. I den andra beskrivningen används wrappers för att beteckna en viss sorts klass inom Java[1] som används för att skapa ett skal runt en primitiv datatyp, så att exempelvis en variabel av typen integer kan

användas som ett objekt.

Inom nätkommunikation används begreppet wrapper ofta till att betyda den data som sätts framför eller runt ett paket för att utöka informationen eller gömma innehållet från andra än mottagaren[51].

I sin rapport om hur COTS-mjukvara ska kunna inkorporeras i ett system utan att riskera säkerheten[47], definierar Jeffrey Voas wrappers som *ett mjukvaruskal som sitter runt komponenten och begränsar vad komponenten kan göra med avseende på dess omgivning*, och att det finns en annan typ av wrapper som begränsar vad omgivningen kan göra med komponenten. Detta ger upphov till Voas två wrapper-typer:

- Input wrappers – En wrapper som begränsar vad som kan skickas in till det som wrappas.
- Output wrappers – En wrapper som begränsar vad som kan skickas ut från det som wrappas.

Boris Chidlovskii börjar sin rapport *Wrapper Generation via Grammar Induction*[6] med en diskussion kring metasökmotorer, vilket är en tjänst som utnyttjar andra sökmotorer för att söka istället för att vara en sökmotor själv, och beskriver där wrappers som program som hanterar en specifik källa genom att transformera användarens söksträng så att den passar just den sökmotorn.

Den grundläggande boken inom designmönster, *Design Patterns - Elements of Reuable Object-Oriented Software*[14], beskriver två wrappermönster – Adapter och Decorator. Designmönster är lösningar på kända problem i mjukvarudesign; det är mallar för hur ofta förekommande problem kan lösas. De två wrappermönstrena definieras i korthet på följande sätt:

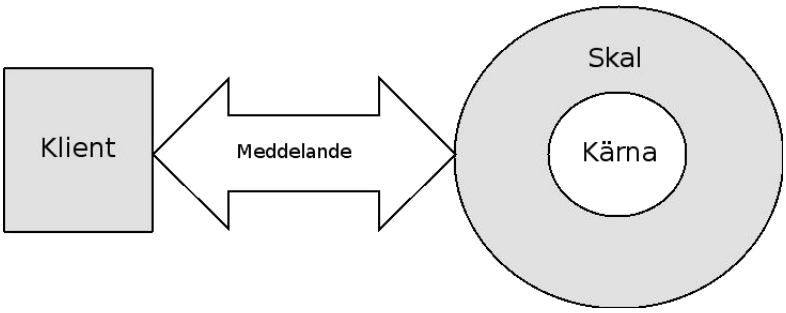
Adapter – Mönstret beskriver hur en klass med inkompatibelt gränssnitt kan användas genom att sätta ett skal runtom objektet som använder ett gränssnitt utåt som systemet är kompatibelt med.

Decorator – Mönstret beskriver hur ett objekt kan ges ytterligare funktionalitet dynamiskt utan att gränssnittet ändras. Notera att här är wrapperns primära syfte att lägga till ny funktionalitet.

En sammanställning av alla dessa olika användningar av begreppet wrapper ger oss följande generella definition av en wrapper:

```
En wrapper är en mjukvarumodul, det vill säga kod, som använder
en annan mjukvarumodul i syfe att antingen utvidga denna, öka
kompatibiliteten utåt, förstärka säkerheten, öka enkelheten, eller
addera en vy. Wrappern begränsar hur kärnan (det som wrappas)
hanteras, och hur omgivningen hanterar den.
```

I figur 2.1 visas en generell bild av wrappers. En klient i det här fallet är en entitet som använder skalet istället för kärnan, och som till exempel skulle kunna vara en människa som skriver in text i ett formulär, eller en C++-klass som använder en annan klass.



Figur 2.1: Generell wrapper beskrivning

En wrapper är ofta en relativt enkel mjukvarukomponent, då den ofta inte tillför funktionalitet som det som wrappas inte redan har. Men en wrapper till en komplex mjukvarukomponent kan behöva vara relativt komplex själv, och även en wrapper till en väldigt enkel mjukvarukomponent kan behöva vara komplex. Som exempel kan tas ett

enkelt program som kräver ett avancerat grafiskt gränssnitt. I de fall en mjukvarukomponent utökas med hjälp av en wrapper kan denna utökning vara en avancerad åtgärd och kräva att wrappern blir relativt komplex.

2.2.2 Wrappers historia

Då wrappers är ett mångfacetterat begrepp inom datavetenskapen och såpass allmänt använt är det mycket svårt att spåra dess användning genom historien. Detta är en kort diskussion om var wrappers ursprung kan ligga.

Om man antar att wrappers historia grundar sig i designmönster[14], så går wrappers historia bak till 1987, då Ward Cunningham och Kent Beck skrev uppsatsen *Using Pattern Languages for Object-Oriented Programs*[9]. Efter denna uppsats började mönster skapas överallt inom datavetenskap, vilket gör det svårt att bestämma någon startpunkt för wrappers. *Design Patterns*[14] som gavs ut första gången 1991 innehöll två mönster med beteckningen wrapper. Huruvida begreppet användes inom mönster här först är dock oklart.

Det verkar dock osannolikt att begreppet wrapper ursprungligen bottnar i uppkomsten av designmönster, då detta kom tämligen sent i datavetenskapens historia. Assembly-språk, det vill säga språk som använder enkla kommandon för att representera maskinkodsinstruktioner, kom redan på början av 1950-talet[49], därför borde wrappers ha uppkommit någon gång inom tidsperioden 1950-1987. När uppstod problem där en wrapper var en önskvärd lösning? Rimligtvis när kod började modulariseras, och återanvändas genom modularisering. Troligtvis har det funnits wrappers innan begreppet existerade.

1968 presenterades begreppet *Modular Programming*[7] vid the National Symposium on Modular Programming. Modular Programming innebar att program skulle ha en logisk struktur; uppdelad i kodmoduler. Det verkar då rimligt att anta att wrappers, i varje fall i någon bredare utsträckning, började användas runt tiden 1968-1987.

2.2.3 Användningsområden

Som tidigare nämnts kan wrappers fylla fem syften: enkelhet, kompatibilitet, säkerhet, addering av vy, och funktionalitet. Här följer en lista på exempel av wrappers för var och ett av dessa syften:

Enkelhet – En klass som används för att koppla upp sig mot MySQL-databaser innehåller en mängd generella funktioner. I ett system som alltid ska kopplas upp mot en specifik databas, och utföra en eller flera av en mängd fördefinierade SQL-frågor, kan man definera en wrapperklass till den första klassen. I denna klass kan tänkas att uppkoppling endast sker med connect(), utan att några parametrar behöver anges eftersom detta är definierat i wrapperklassen. Detta är ett exempel på en wrapper som ökar enkelheten.

Kompatibilitet – Ett system som byter ut ett bibliotek mot ett annat, men har kvar samma gränssnitt mot biblioteket, måste skapa en wrapper. Denna wrapper har då gränssnittet av det gamla biblioteket och vidarebefordrar meddelanden till det nya biblioteket. Detta är ett exempel på en wrapper som ökar kompatibiliteten.

Säkerhet – Design by Contract[29] är en mjukvarudesignmetod som bygger på användandet av kontrakt för att specificera vilka skyldigheter och ansvar som varje mjukvarumodul har. Ett begrepp som används här är starka och svaga kontrakt. En funktion som har ett starkt kontrakt har definierat alla de villkor som måste vara uppfyllda när funktionen anropas, och förutsätter därmed att den som anropar funktionen uppfyller dessa villkor. En funktion som har ett svagt kontrakt har inga förvillkor, och tillåter alltså att vem som helst anropar den. Svaga kontrakt används vanligtvis till gränssnitt utåt mot klienter där input inte kan förutses, som till exempel text som skrivs in av en användare. Dessa funktioner hanterar input och ser till att vissa villkor uppfylls, för att sedan anropa funktioner med starka kontrakt. Dessa gränssnittsfunktioner fungerar som wrappers för funktioner med

starka kontrakt. Detta är ett exempel på en wrapper som ökar säkerheten.

Addering av vy – Flera applikationer som använder någon form av grafiskt användargränssnitt är strukturerade på ett sådant sätt att logiken är skild från gränssnittet i koden. Gränssnittetsdelen representerar en addering av vy på logiken, som då är en wrapper till logiken.

Funktionalitet – För att läsa I/O-strömmar i Java har Sun valt att använda designmönstret Decorator (som ibland betecknas wrapper). För att läsa från en fil skapas först en FileInputStream:

```
InputStream in = new FileInputStream( "minFil" );
```

För att sedan effektivisera inläsningen kan en decorator appliceras på strömobjektet på följande sätt:

```
in = new BufferedInputStream( in );
```

Java tillhandahåller ett flertal andra I/O-decorators som bland annat AudioInputStream och InflaterInputStream. På detta sätt går det enkelt att dynamiskt definiera vilken typ av inläsning som ska användas, och det går även enkelt att skriva sin egen decorator. Dessa strömhanteringsklasser är exempel på wrappers som ökar funktionaliteten.

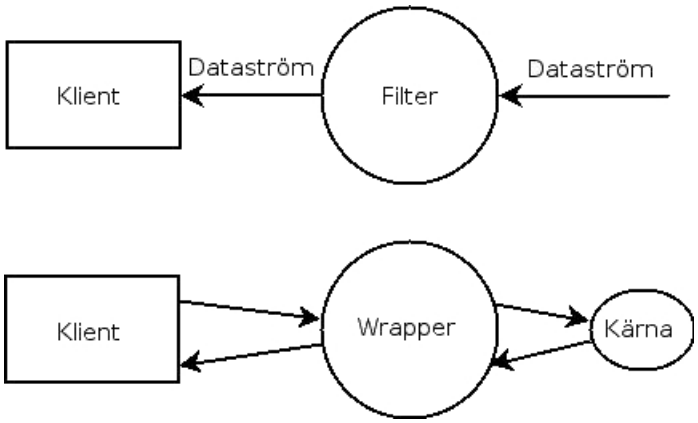
2.2.4 Wrappers i denna rapport

I denna rapport kommer wrapperdefinitionen från 2.2.1 att användas, med en liten modifikation, och vidare kommer det som en undertyp till wrappers definieras även *språkwrappers* i avsnitt 2.2.5. Skillnaden i definitionen av wrappers som används i denna rapport från den allmänna definitionen som definierades i avsnitt 2.2.1, är att en wrapper i denna rapport inte lägger till ny funktionalitet som leder till en för stor ändring av

cohesion. En wrapper ska alltså inte lägga till funktionalitet som gör att den nya mjukvarumodulen (wrappern plus det som wrappas) avviker för mycket från det ursprungliga syftet med det som wrappas.

Förutom detta måste även gälla att: Det som wrappas är den viktigaste och huvudsakliga kärnkomponenten inom wrappern – allt annat är sekundärt.

För att ytterligare förtydliga wrappers innebörd kan wrappers jämföras med filter. Figur 2.2 visar en jämförelse mellan dessa två. Medan ett filter endast hanterar en ström av data, arbetar en wrapper mot en kodmodul. Ett exempel på ett filter är kod, förslagsvis en funktion eller en klass, som gör om HTML-kod till mer läslig text så som den visas i webbläsare. Ett exempel på en wrapper är ett program som använder filtret från filterexemplet för att visa en hemsida på ett annat sätt, kanske genom att rensa bort alla bilder så att dessa inte visas för användaren.



Figur 2.2: Jämförelse mellan filter och wrapper

I den wrapper som utvecklats som en del av detta arbete lagras meddelandena, som fås med biblioteket som wrappas, i en databas. Innebär detta ny funktionalitet som har för stor ändring av bibliotekets syfte? Svaret är nej, eftersom databasen är bara ett annat gränssnitt för output från skalets kärna. Det lägger inte till någon funktionalitet som inte redan existerar i biblioteket, utan ger endast ett annat gränssnitt mot bibliotekets

funktioner.

Syftet med libGaim är *att tillhandahålla funktionalitet för att tillåta utnyttjande av IM-tjänster* och syftet med den wrapper som utvecklats som en del av detta arbete är *att tillhandahålla funktionalitet för att tillåta utnyttjande av IM-tjänster, via ett databasgränssnitt*. Enligt vår uppfattning är denna ändring i syfte tillräckligt liten för att applikationen kan betraktas som en wrapper, då det som skiljer dem åt är att vår applikation är en addering av vy (se avsnitt 2.2.3).

För att sammanfatta denna rapports wrapperdefinition; en wrapper är:

- En mjukvarumodul, det vill säga kod, som använder en annan mjukvarumodul i syfe att antingen utvidga denna, öka kompatibiliteten utåt, förstärka säkerheten, öka enkelheten, eller addera en vy, utan att den nya mjukvarumodulen (wrappern + det som wrappas) avviker för mycket från syftet hos det som wrappas.

- Ett gränssnitt som begränsar hur omgivningen hanterar det som wrappas, och hur det som wrappas hanterar omgivningen.

- En mjukvarumodul som använder en annan mjukvarumodul som sin huvudsakliga komponent; den del som är viktigast och allt bygger på.

2.2.5 Språkwrappers

Ett av wrapperns syften är kompatibilitet, att göra det möjligt för en mjukvarumodul att använda en annan som den vanligtvis inte skulle kunna använda. I denna rapport används begreppet språkwrapper att betyda en wrapper där wrappern skapar ett skal runt en kodmodul i ett språk för att föra över användandet av denna till ett annat språk.

Språkwrappers kan användas till att bland annat göra det möjligt för en applikation skriven i språk A att använda ett bibliotek skrivet i ett annat språk B. Detta innebär dock inte att en språkwrapper alltid är nödvändig. Till exempel kan ett kompilerat bibliotek skrivet i programspråket Fortran användas i C utan någon språkwrapper. Men

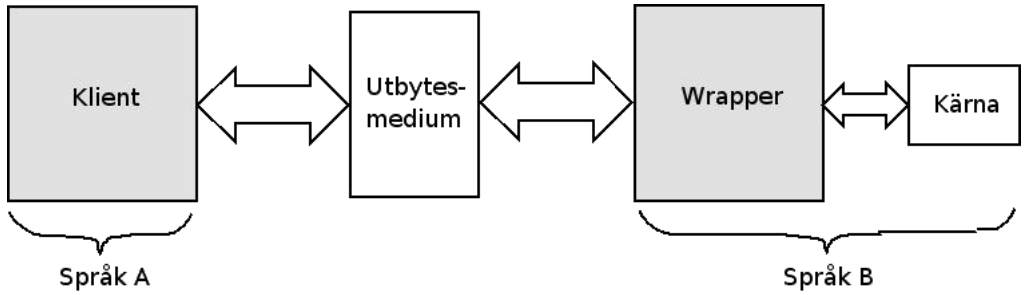
vid de tillfällen en kodmodul inte kan användas direkt av ett annat språk krävs en språkwrapper. Ett sådant tillfälle är exempelvis användningen av ett C-bibliotek i PHP.

I denna rapport har det valts att dela in språkwrappers i två undertyper:

1. Språkstatisk – Den språkstatiska varianten kopplas direkt in i koden, och gör att vanliga funktionsanrop är möjliga till den wrappade kodmodulen. Detta löses olika beroende på vilket språk som används. Ett Perl-bibliotek som ska användas i PHP kan användas genom att Perl-interpretatorn wrappas och meddelanden till och från Perl omvandlas av denna[42]. Ett C-bibliotek som ska användas i Tcl kan användas genom att skapa ett paket av de nya kommandon som behövs, och sedan registrera detta i en ny Tcl-interpretator.

2. Språkgenerisk – Den språkgeneriska varianten skickar meddelanden genom ett externt medium för att kommunicera med klienten (se figur 2.3). Detta gör att klienten kan vara skriven i valfritt språk som kan hantera mediet.
   Förutsättningarna är att wrappern är väl dokumenterad för att utvecklaren av klienten ska veta vilka meddelanden som ska skickas in och vilka meddelanden som kan väntas som svar. Då alla meddelanden måste skickas genom ett förbindande medium istället för direkt med lokala funktionsanrop verkar det rimligt att denna typ av språkwrapper blir långsammare än den andra. Huruvida detta är ett problem eller inte beror helt på vilket system som utvecklas. Detta är den typ av wrapper som utvecklats i arbetet som ligger som grund till denna rapport.

Denna rapport redovisar en undersökning i kapitel 5 där prestandaskillnaden mellan dessa två typer av språkwrappers undersöks med hjälp av ett antal olika wrappers och ett test utan wrapper för att jämföra med.



Figur 2.3: Språkgenerisk språkwrapper

För att underlätta skapandet av språkwrappers finns ett flertal verktyg som mer eller mindre automatiskt skapar wrappers. Här följer några exempel på denna typ av verktyg:

SWIG – Simplified Wrapper and Interface Generator[43], är ett verktyg för att generera wrappers från C/C++-kod till bland annat Perl, Python, Tcl, Ruby, Guilde, eller Java. För att exempelvis skapa en wrapper från C till Perl skrivs först en gränssnittsfil där ANSI-C-prototyperna för varje funktion som ska nås av Perl definieras. SWIG tar denna fil som input och genererar en fil med C-wrapperfunktioner, och en Perl-fil som används för att läsa in modulen. Wrapperfilen kompileras och länkas sedan till ett "shared library", vilket sedan används av den genererade Perl-filen. Nu går det enkelt att ladda in modulen genom att använda:

```
use myWrapper;
```

Där myWrapper är den modul som skapats[2].

Matwrap – Matwrap[19] är en wrappergenererare för matrisspråk som Matlab. Matwrap är ett perlskript som tar en C-headerfil som input och genererar en wrapper. En C-funktion

```
int my_silly_function(int num)
```

som wrappas med Matwrap kan då anropas i Matlab på följande sätt:

```
>>result = my_silly_function(47);
```

eller som en vektor

```
>>result = my_silly_function([12;7]);
```

vilket gör att funktionen anropas en gång per element och resultatet blir en ny vektor.

Denna typ av genererade wrappers är dock specialiserade och restriktiva. De mappar endast kodmodulerna exakt som de är från språk A till språk B. Om wrapperns syfte inte bara är att öka kompatibilitet genom att koppla samman språk, utan kanske även att lägga till en nivå av abstraktion, räcker det inte att generera en wrapper av denna typ.

Den wrapper som utvecklats i detta arbete krävde mer än vad denna typ av wrappergenererare tillhandahåller, då den använder ett bibliotek skrivet i C och adderar ett abstraktionslager för att kommunicera med biblioteket.

2.2.6 Är wrappers en bra lösning?

Wrappers är en form av kodåteranvändning. Tack vare wrappers kan en kodmodul skriven för ett annat system återanvändas. Bibliotek skrivna i ett språk kan återanvändas i ett annat. Ett fullständigt system kan byggas ihop av komponenter som ursprungligen inte är gjorda för att användas tillsammans. En stor del av all kod som skrivs idag finns redan, därför skulle en stor del av all kodutveckling kunna vara wrapperutveckling[47].

Oundvikligen hamnar utvecklaren av ett mjukvarusystem i frågeställningen om det är motiverat att använda wrappers istället för att skriva om allt från början. Finns det några nackdelar med kodåteranvändning? Ger kodåteranvändning verkligen bara positiva effekter, eller för det med sig svårigheter?

Kodåteranvändning i sig självt är ett positivt begrepp, då det minskar tid och energi som annars läggs på redundant arbete. Vissa nackdelar kan dock erhållas då kodåteranvändning inte används på ett korrekt sätt. Att enbart söka återanvända kod utan att se på designen och cohesion vore att öka sannolikheten för att erhålla dessa nackdelar. Ett system som är "hopklistrat" av kodmoduler enbart för kodåteranvändandets skull utan att se till vad komponenterna har för syfte blir svårförstått, kan tappa cohesion, och riskerar att minska i effektivitet.

På grund av dess enkelhet och möjlighet att på ett snabbt sätt koppla in avancerade komponenter i ett mjukvarusystem, kan wrappers bli ett lättvindigt sätt att lösa buggar och kompatibilitetsproblem i mjukvara utan att utvecklaren ser till djupare designfel. Till exempel skulle en kodkomponent kunna användas till att mäta tid genom att den används genom en wrapper, när dess syfte var att hantera debugmeddelanden. Detta kan leda till onödig kod, ökad komplexitet, och förlorad cohesion. Notera att detta skulle bryta mot denna rapports wrapperdefinition, då wrapperns syfte skiljer sig mycket från den wrappade kodmodulens. Genom att använda wrappers som följer denna rapports definition kan man alltså skydda sig mot denna typ av felsteg.

Innan en wrapper utvecklas bör utvecklaren fråga sig: *Hur mycket lättare är det att utveckla denna wrapper mot att skriva om det som ska wrappas?* Om wrappern visar sig vara mer komplex och svårare att utveckla än det skulle vara att skriva om det som ska wrappas försvinner användningsvärdet av wrappern.

2.2.7 Alternativ

Då wrappers är en kodåteranvändningsmetod är alternativet till den att skriva om koden för det som ska wrappas. Effektiviteten kan öka när ny kod skrivs, men det gör även tidsåtgången – ofta väsentligt. Genom att skriva om koden kan man minska cohesion och enkelheten, till exempel vid användargränssnitt. Om varje gränssnitt skulle ha sin egen logik, som dessutom var hårt kopplad i gränssnittskoden, skulle cohesion brytas och en

stor mängd redundant kod skulle finnas.

2.3 Instant Messaging

2.3.1 Vad är Instant Messaging?

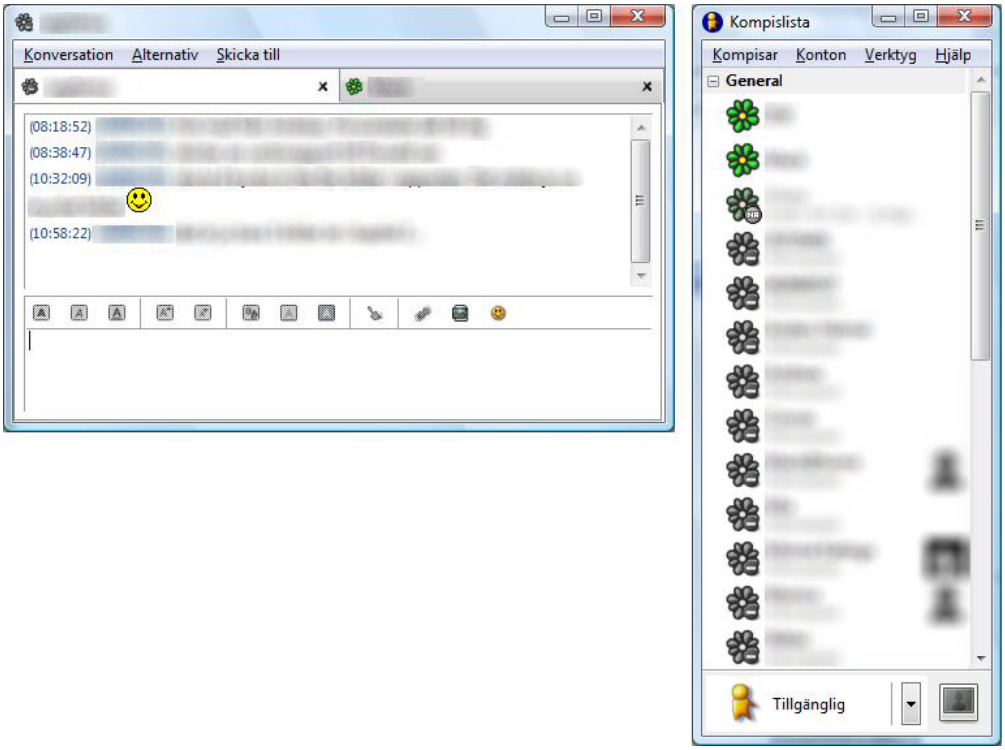
Instant Messaging är en term för att beskriva en typ av kommunikationssystem, som e-post eller samtal. Denna typ av system innebär synkron kommunikation i textform mellan två eller flera användare.

Det första i ledet synkrona meddelandesystem var Unix-programmet *talk*(se figur 2.4) som utvecklades redan 1973. Från början användes talk för att låta användare på en multianvändardator kommunicera, men utvecklades eventuellt för att möjliggöra kommunikation mellan olika maskiner. Det som användaren skrev in i talkklienten skickades direkt till mottagaren för varje nytt tecken. En användare som skickade ett meddelande med talk fick göra det utan vetskap om mottagaren var tillgänglig, eftersom talk saknade denna funktionalitet[15].

Figur 2.4: En talk session utförd över PuTTY i Windows XP

1988, på ett universitet i Finland, utvecklades protokollet IRC[34] (Internet Relay Chat) för synkront utbyte av textmeddelanden. IRC använder sig av kanaler (ett ID som definierar ett virtuellt rum) där flera användare kan kommunicera med varandra, ofta runt ett gemensamt ämne. IRC har även möjlighet till ett-till-ett kommunikation genom privata meddelanden. Användare kopplar upp sig mot en IRC-server med en klient och väljer sedan en kanal att chatta i.

1996 släppte företaget Mirabilis Ltd. det första systemet för Instant Messaging[15]. De kallade systemet ICQ[56] (I seek you). Varje användare har en klientapplikation som vanligtvis innehåller en lista av alla användarens kontakter, och fönster dedikerade till konversationer med dessa kontakter används för att skriva in och skicka meddelanden. Meddelanden skickas inte per inslaget tecken som i talk, utan endast när användaren väljer att skicka (genom att till exempel klicka på en skicka-knapp). En annan skillnad mot talk är att användaren har möjlighet att se när dennes kontakter är tillgängliga. Ett exempel på en ICQ klient visas i figur 2.5.



Figur 2.5: ICQ meddelande med IM klienten Gaim i Windows Vista

En stor skillnad mellan konferens-chat (IRC-typen) och ett-till-ett-chat (ICQ-typen), är att i den förstnämnda samlas ett flertal användare som ofta aldrig träffats i verkligheten runt ett gemensamt ämne som till exempel “Bond-filmer”, medan användare av den sistnämnda typen mer frekvent känner de som de skriver till[16].

Dagens Instant Messaging fungerar i regel på följande sätt:

- IM klienten kopplar upp mot en server med användarnamn och lösenord.
- Klientens IP-nummer och port skickas till servern.
- Servern skickar tillbaks en lista på användarens alla kontakter.
- En temporär fil med kontaktinformation skapas på servern.

- Servern kontrollerar statusen för användarens kontakter och skickas detta till klienten.

- Beroende på protokoll och inställningar har klienten fått IP-numret till sina kontakter. När ett meddelande ska skickas till en kontakt används IP-adressen och portnumret som nu är lagrat i klienten, vilket gör att servern inte behövs användas alls i detta skede.

2.3.2 Protokoll

Alla IM-tjänster utnyttjar någon form av protokoll. Ett IM-protokoll är en samling regler som definierar hur IM-meddelanden är strukturerade innan de skickas.

Det finns idag en stor mängd olika IM-protokoll. Här följer en lista på några av de vanligaste:

QQ – Utvecklat av Tencent i Kina och används huvudsakligen av klienten Tencent QQ[20]. Tjänsten öppnade 1999 under namnet OICQ (Open ICQ), men bytte sedan till QQ för att undvika märkesintrång och för att varken klienten eller protokollet är öppna. Idag används QQ av 160 miljoner aktiva medlemmar. Användarna är nästan uteslutande i Asien.

OSCAR – Open System for CommunicAtion in Realtime[60] används av båda Instant Messaging-tjänsterna ICQ[56] och AIM[52], och skapades av den amerikanska internetleverantören AOL[53]. OSCAR är ett stängt protokoll, det vill säga att AOL inte givit ut några specifikationer för det, vilket gjort att den information som är känt om det kommer från reverse engineering, och det är endast nyligen som öppna implementationer fått stöd för filöverföring och text till borta-meddelanden.

MSNP – Microsoft MSN's IM protokoll[59]. Microsofts senaste version av detta protokoll är MSNP15. MSNP är ett stängt protokoll.

XMPP – Extensible Messaging and Presence Protocol[38] är ett XML-baserat protokoll som används av Instant Messaging-tjänsten Jabber[21]. XMPP är en IETF(en standardiseringsgrupp som utvecklar och främjar Internet-standarder)[57] standard som ursprungligen utvecklades av Jeremie Miller, skaparen av Jabber, och utvecklades ytterligare av XMPP Working Group (en grupp sammansatt för att förbättra och standardisera protokollet) för att bli en IETF-standard.

2.3.3 Gaim och libGaim

1998 började Mark Spencer utveckla ett program för att lära sig utveckla GTK-applikationer i C[41][10]. Programmet var en Instant Messaging-klient för AOL's Instant Messaging tjänst och Spencer kallade programmet för Gaim. Ursprungligen tänkt som ett tvåveckorsprojekt har Gaim vuxit till ett av de mest aktiva öppen källkods-projekten på SourceForge.net[24] (en webbplats som tillhandahåller verktyg för att sköta denna typ av projekt) och är dessutom inkluderat i nästan alla stora distributioner av Linux. Spencer skrev bara kod för projektet under en tid av 2-3 månader, varefter han lämnade över det till en grupp andra utvecklare.

Rob Flynn, som var en av de som tog över efter Spencer, minns i en intervju[27] att han gick med i projektet 1998 för att han behövde en AIM-klient till Linux (vilket AOL vid denna tidpunkt inte tillhandahöll[10]). Vad som började som ett lärningsprojekt fyllde snart ett behov hos flera Instant Messaging-användare som behövde en klient. Gaim var ursprungligen endast tänkt som en klient för AOL och hade detta stöd inbyggt. År 2000 introducerades protokoll i form av insticksmoduler, vilket betydde att ett protokoll kunde skrivas som ett självständigt delat bibliotek och sedan dynamiskt laddas in i Gaim. År 2002 portades Gaim till Windows.

Idag har Gaim stöd för flera olika protokoll, bland annat QQ, OSCAR, och MSNP. Gaim gör det möjligt för en användare att vara inloggad på flera IM-tjänster samtidigt, och har stöd för de vanligaste Instant Messaging-funktionerna.

Gaims användargränssnitt och kärna var länge hårt kopplade, vilket gjorde det svårt att använda dem oberoende av varandra. Efter en tid (innan version 0.60 enligt en av utvecklarna vi frågat, vilket skulle innebära innan april 2003) började dock arbetet att dela upp Gaim i två delar: gränssnittet (gtk), och kärnan (libGaim). Ett utdrag från samtalet med Gaim-utvecklaren Kevin "SimGuy" Strange angående tidpunkten för påbörjandet av libGaim:

> *"När jag ser tillbaks på koden skulle jag säga att 0.60 var början. Koden i 0.59 och innan det verkar inte varit separerad på något sätt."*

LibGaim separerades fullt från gränssnittskoden i version 2.0.0beta4, och blev därmed ett eget bibliotek. När detta skrivs är libGaim dock fortfarande inte ren logik, då det bland annat krävs ett gränssnitt för att kunna använda libGaim.

I och med denna separation av logik och gränssnitt har ett antal nya Instant Messaging-klienter som använder libGaim skapats. Exempel på klienter som använder libGaim är Fire[11] och WengoPhone[50].

En viktig funktion i libGaim, och kanske den stora anledningen till att det blivit populärt, är dess möjlighet att dynamiskt koppla in stöd för Instant Messaging-protokoll. Om det är något protokoll som saknas kan den som vill skriva en insticksmodul och koppla in det i libGaim. Dokumentationen som finns är något bättre för hur man skriver en insticksmodul än för hur man skriver en egen klient (se avsnitt 5.2.3).

I arbetet som utförts som grund till denna rapport har libGaim använts som kärnan till en wrapper. Hur libGaim fungerar och något om dess fördelar och nackdelar beskrivs vidare i kapitel 5.

2.4 Sammanfattning

I detta kapitel har relevant bakgrundsinformation till arbetet presenterats för att ge läsaren en inblick i problemområdet. Begreppet wrapper har presenterats och definierats.

Wrapper har även givits en mer tydlig definition inom sträckningen av denna rapport.

Språkwrappers har beskrivits som en undertyp till wrappers. Denna typ av wrapper undersöks med tester i kapitel 5.

Instant Messaging är en viktig komponent inom arbetet som utförts, varför detta presenterats tillsammans med Instant Messaging-klienten Gaim och dess bibliotek libGaim. Arbetet som utförts använder libGaim-biblioteket och skapar en wrapper runt detta.

I nästa kapitel beskrivs de metoder och verktyg som använts vid utvecklingen av libGaim wrappern.

Kapitel 3

Metoder och verktyg

3.1 Introduktion

I detta kapitel presenteras vilka utvecklingsmetoder och verktyg som använts under utvecklingen av libGaim-wrappern, för att ge en vidare bakgrund till projektet och för att ge inledande detaljer kring hur projektet utförts. Detta kapitel är speciellt intressant för den som har för avsikt att göra ett liknande arbete.

Först beskrivs utvecklingsmetoden – vilken struktur arbetet följt.

Därefter beskrivs mediet som använts för att koppla samman wrappern som utvecklats under projektet och klienten. Mediet är en MySQL-databas[32]. Av denna anledning ges här en kort presentation av MySQL. En beskrivning av hur databasen använts i detta projekt står att läsa i avsnitt 5.5.3.

LibGaim, IM-biblioteket som är kärnan i den wrapper som utvecklats, är skrivet i C. Själva wrappern har dock utvecklats i C++. Anledningen till detta diskuteras i detta kapitel.

En kort bakgrund till skriptspråket PHP[8] och motivering till valet av PHP ges då detta använts i flera av testerna i kapitel 5.

För de som är ovana av att använda byggverktyg, och då i huvudsak Autotools[48] och

SCons[25], innehåller detta kapitel en genomgång av de tre byggverktygen GNU make[30], GNU Autotools, och SCons. I början av ett liknande projekt kan det ta lång tid att sätta sig in i denna typ av verktyg, varför detta kapitel ger en överblick och försöker peka på några skillnader mellan dem. De tre byggverktyg som presenterats har alla använts under olika delar av projektet.

För att generera dokumentation till projektet har doxygen[17] använts, och här ges en kort beskrivning av detta verktyg.

Avslutandes tas versionshanteringssystemet SVN[62] upp som använts för att hantera olika versioner av källfiler i projektet.

3.2 Utvecklingsmetod

Delar av XP(eXtreme Programming)[3] har inkorporerats i utvecklingen av libGaim-wrappern. För att använda XP i utvecklingen av ett projekt behöver man bara använda de metoder som är lämpliga för projektet. Här följer en listning av de delar av XP som använts:

- Applikationen har utvecklats i iterationer.
- Applikationen har utvecklats under tänkandet "bara tillräckligt för att det ska fungera". Det vill säga att varje ny funktionalitet har implementerats enklast möjligt och innehållit bara det som krävts.
- Kontinuerliga upptaktsmöten med kunden. Då kunden i detta fall var handledaren på Evolve var detta naturligt.

Däremot har inte *Test-driven development* eller *parprogrammering*, som båda är viktiga delar av XP-processen, använts under utvecklingen.

Test-Driven Development, eller TDD, är en utvecklingsteknik där testfall skrivs innan kodimplementation sker. Denna implementation innehåller bara vad som krävs för att

testfallen ska lyckas. Under utvecklingen av libGaim-wrappern togs en stor del av tiden upp till att förstå hur libGaim fungerar. Under denna inlärningsprocess utvecklades ett antal prototyper av wrappern, för att se hur libGaim fungerar och hur alla problem skulle lösas. Här hade TDD enligt vår mening endast tagit tid från utvecklingen och inte tillfört något. När det sedan var dags att utveckla den riktiga wrappern användes dock tester mot databasen för att kontrollera att alla funktionaliteter fungerade.

Parprogrammering kunde ha använts i projektet men vi valde att inte använda det, av den anledning att vi upplevde det effektivare att kunna dela upp arbetsuppgifter som bland annat utveckling av wrappern, utveckling av tester, hantering av databasen, utförande av tester, och research om bland annat libGaim.

3.3 MySQL

Till kopplingen mellan libGaim-wrappern och dess klient har databassystemet MySQL[32] använts. Anledningen till varför ett databassystem valdes och ingen annan form av interprocesskommunikation står att läsa i avsnitt 4.3.3. Anledningen till varför just databassystemet MySQL valdes, och inte till exempel PostgreSQL[37] är flerfaldigt. För det första användes redan MySQL av uppdragsgivaren till detta arbete (Evolve) vilket gjorde att utvecklarna på Evolve har kunskap om MySQL. Dessutom hade vi använt MySQL vid flera tidigare tillfällen, bland annat i ett examensarbete på C-nivå.

3.4 Språk

3.4.1 C och C++

Instant Messaging-biblioteket libGaim (se avsnitt 2.3.3 och 5.2), och även själva klienten Gaim, är skrivet i programspråket C. C är ett procedurellt programspråk, men enligt Sean Egan[10] är Gaim utvecklat enligt en objektorienterad paradigm. Han menar att

detta uppnås genom att använda dynamiskt allokerade struct:ar som representerar objekt, och semantiska simuleringar av objektorienterade egenskaper. Här följer en lista över hur objektorienterad programmering simulerats i Gaim:

- **Objekt** – Ett objekt inom objektorienterad programmering är en instans av en klass, där en klass är mallen som beskriver strukturen av en datatyp. Dynamiskt allokerade struct:ar används för att representera objekt. Objektorienterad syntax kan även användas när *typedef* används på en struct för att skapa ett alias.

- **Publikt och privat** – För att skydda medlemmar av ett objekt i objektorienterad programmering definieras att dessa är privata (i C++ med nyckelordet *private*). Eftersom nyckelordet *private* inte finns i C sätts underscore (_) för att ange att en variabel, metod, eller struct är privat. Detta ger dessa endast den semantiska egenskapen att vara privata, och det finns alltså inget inbyggt skydd mot att en utvecklare använder dessa i alla fall.

- **Konstruktorer och andra medlemsmetoder** – Inom objektorienterad programmering används vanligtvis ett nyckelord som *new* för att instansiera ett nytt objekt, varpå objektets konstruktor anropas. För att lösa detta i C måste egna konstruktorer skrivas som anropas explicit som till exempel:

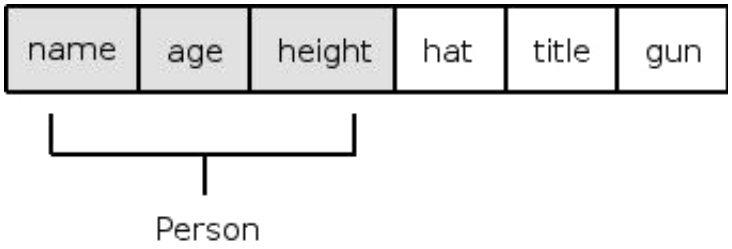
  ```
  GaimBlistNode *n = gaim_blist_node_new();
  ```

  Detta skapar en ny nod i kompislistan.

  På liknande sätt måste alla funktioner deklareras med ett prefix för att utvecklaren ska veta vilken typ eller simulerad klass de tillhör.

- **Arv** – Arv är ett nyckelbegrepp inom objektorienterad programmering. En klass som ärver av en annan klass har kvar alla publika medlemmar som sin förälder har. Då klasser i C representeras av struct:ar, och en struct är en platt struktur där

varje element följer direkt efter ett annat, löses detta genom att subklassen (eller substruct:en) innehåller den ärvda struct:en som sitt första element. Figur 3.1 visar en struct Police som är en subklass/substruct till Person.



Figur 3.1: Arv i C

En klient som bara söker använda ett objekt av föräldraklassen kan tack vare denna metod göra det, då den första delen av struct:en är likadan som föräldraklassen.

Istället för att simulera dessa objektorienterade egenskaper har C++ valts till detta projekt, vilket har inbyggt stöd för alla dessa delar av objektorienterad programmering genom klasser.

Eftersom C++ är bakåtkompatibelt med C är det heller inga problem att använda libGaim, som är ett C-bibliotek, i C++ .

3.4.2 PHP

Skriptspråket PHP[8] har använts i tre av testerna som redovisas i avsnitt 5.4. Anledningen till varför PHP valdes, och inte något annat skriptspråk som till exempel Perl, var flerfaldigt. För det första ansågs det rimligt att använda PHP eftersom detta är det skriptspråk som wrappern som utvecklas ska arbeta emot. För det andra tillhandahåller PHP väldokumenterade verktyg för att utveckla PHP-extensions (se avsnitt 4.3.1), vilket används i ett av testerna. För det tredje har PHP inbyggt stöd för

MySQL[32] som låter programmeraren utnyttja en MySQL-databas med bara några få rader kod, till exempel på följande sätt:

```
<?php
     mysql_connect(localhost,"användarnamn","lösenord");
     @mysql_select_db("databasen") or die("Kunde inte välja databasen");
     mysql_query("INSERT INTO min_tabell VALUES ('7406036218','Pelle','polis')");
     mysql_close();
?>
```

Detta exempel kopplar upp mot ett databassystem på den dator som skriptet exekveras på, därefter väljs vilken databas som ska användas, och sedan utförs en SQL-fråga som sätter in en ny rad i tabellen. Till sist stängs uppkopplingen. Allt detta gjordes på fyra rader. Detta är ingen ovanlig företeelse inom skriptspråk, som vanligen arbetar på en högre abstraktionsnivå än till exempel C. I Perl hade motsvarande kunnat utförts med tre rader men vi ansåg att PHP var mer lättläst om kodbitar skulle komma att användas i test-avsnittet av rapporten.

3.5 Byggverktyg

3.5.1 GNU make

I de ursprungliga testerna som gjordes för att undersöka hur svårt det var att koppla upp mot en IM-server och skicka ett meddelande med hjälp av libGaim, användes GNU make som kompileringsverktyg. Anledningen till att make valdes var helt enkelt att det är relativt enkelt, det finns vanligtvis installerat i Unix/Linux, och tidigare erfarenhet av att använda GNU make fanns. Detta fungerade även utmärkt till en början.

När make-kommandot körs söker make efter en fil med namnet GNUmakefile, makefile eller Makefile[30]. Denna fil används sedan för att bestämma vad som ska utföras. En

Makefile-fil är strukturerad på följande sätt:

```
mål:      beroenden
          kommando 1
          kommando 2
             ...
          kommando n
```

Varje grupp *mål*, *beroenden* och *kommandon* kan betraktas som en regel[10]. Detta betyder att en Makefile-fil kan betraktas som en samling regler. En regel definierar hur en fil (målet) ska byggas. Beroendena anger vad som krävs för att kunna bygga målet, och kommandona anger hur målet ska byggas. Här följer ett exempel på hur en Makefile-fil kan se ut:

```
mitt_program:     test.o main.o
                  gcc -o mitt_program test.o main.o
test.o:           test.c
                  gcc -o test.o -c test.c
main.o:           main.c main.h
                  gcc -o main.o -c main.c
```

GNU Make exekverar endast en regel om mål-filen inte existerar eller om mål-filen är äldre än någon av beroende-filerna. Detta leder till att onödiga byggningar av ett projekt kan undvikas, och dessutom kompileras endast det minsta som behövs.

Anledningen till varför GNU make ensamt inte passade bra som byggverktyg till projektet var att det saknar funktionen att kontrollera om byggmiljön är korrekt uppsatt, det vill säga undersöka om de bibliotek som behövs är installerade och sätta upp systemspecifika flaggor. Om Make används tvingas utvecklaren skriva shellskript med en mängd villkorliga uttryck. Detta blir snabbt svårläst och tar lång tid att skriva, och dessutom blir det krångligt att expandera projektet.

3.5.2 GNU Autotools

Det är relativt enkelt att skriva en Makefile-fil för ett visst system om man i förhand vet den exakta konfigureringen av byggmiljön. Ett exempel är ett hemmaprojekt där man vet vilka bibliotek som finns och var dessa är installerade. Om man däremot gör ett projekt där man inte vet hur systemet det ska köras på kommer att vara konfigurerat behöver man verktyg som automatiskt kan ta reda på konfigureringen av systemet. Autotools[48] är ett sådant verktyg.

Gaimutvecklarna har valt att använda Autotools som byggverktyg för Gaim[10], varför detta till en början kändes naturligt att använda som byggverktyg för libGaim-wrappern. Det skulle dock visa sig att Autotools är ett verktyg som kräver relativt lång tid att sätta sig in i för den som inte använt det förut, varför det efter en tid bestämdes att detta projekt skulle använda SCons[25] (se avsnitt 3.5.3). Detta avsnitt ger en sammanfattande inblick i hur Autotools fungerar, dels för att det är en viktig del av Gaim-projektet, men även för att läsaren ska lättare se anledningen till varför det valdes bort i detta projekt.

Det vanliga tillvägagångssättet vid kompilering och installation av ett program som använder Autotools är

1. ./configure
2. make
3. make install

./configure undersöker systemet för att se att allt som krävs för att kunna kompilera programmet finns. **Make** bygger programmet från källfilerna, och **make install** installerar programmet.

Autotools är ett sammlingsnamn för ett antal fristående automatiseringsverktyg där kärnan av autotools består av Autoconf, Automake och Libtools. Dessa tre applikationer arbetar tillsammans för att skapa ett configure-skript som i sin tur genererar relevanta

Makefile-filer för ett givet system när det körs. Automake skapar filer kallade Makefile.in från filer kallade Makefile.am.

Makefile.am-filerna beskriver mycket generellt hur projektet ska byggas. Makefile.am-filer har samma syntax som Makefile-filer och med några undantag kommer allt som skrivs i Makefile.am att kopieras till den slutgiltiga Makefile-filen. Detta gör att man kan skapa sina egna make-targets direkt i Makefile.am-filen som om man skrev en Makefile-fil. Det som skiljer Makefile.am-filer från Makefile-filer är att vissa variabler har en speciell mening för Automake. En viktig variabel är bin_PROGRAMS som är en lista av program som ska skapas av projektet.

Den delen av ett variabelnamn som är skrivet i versaler kallas för *primary*, och dessa har alla en speciell mening för Automake. Den delen av variabelnamnet som består av gemener som är prefix till primary:n kvalificerar den. bin_PROGRAMS antyder att en lista av program ska installeras i standard-sökvägen för exekverbara filer, typiskt /usr/local/bin i Linux. Beskrivningen för resterande variabler för att kunna bygga projektet följer samma mönster. Varje prefix har flera primary:s, fast de viktigaste är SOURCE och HEADERS. SOURCE är en lista av source-filer, medan HEADERS är en en lista av header-filer. Eftersom make inte tillåter andra tecken än alfanumeriska symboler och underscore i variabelnamn måste namn som innehåller andra symboler konverteras till underscore. Detta är vad som behövs för att Automake ska kunna skapa korrekta Make-filer som bygger applikationen i projektet på ett korrekt sätt. En mycket simpel Makefile.am-fil kan se ut så här:

```
bin_PROGRAMS = test
test_SOURCES = file1.c file2.c
test_HEADERS = file2.h
```

Det är Autoconf som skapar configure-skriptet. Skriptet tar reda på alla systemspecifika detaljer om systemmiljön som projektet ska byggas på. Detta innefattar

vilken kompilator som ska användas, vilka argument som ska skickas till kompilatorn, och var biblioteksberoendena finns installerade. Man kan också skicka kommandoprompts-argument till configure-scriptet som påverkar hur projektet byggs.

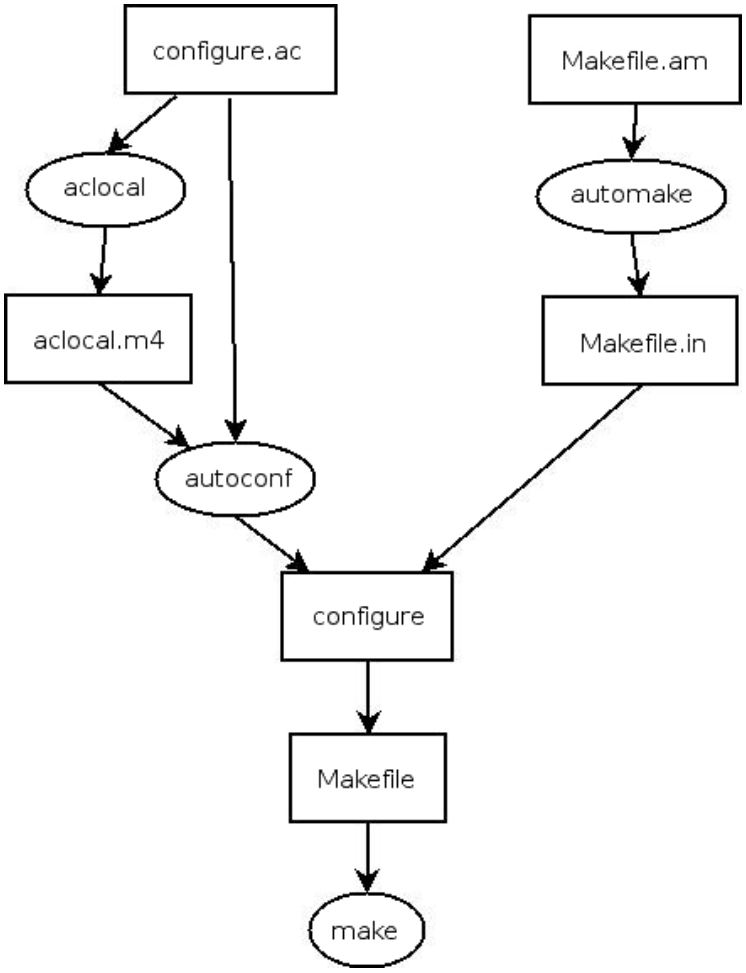
Det slutgiltiga målet för configure-skriptet är att skapa Makefile-filer från Makefile.in-filerna. Makefile.in-filer påminner om Make-filer genom att förklara hur programmet ska byggas, fast de gör det mer generellt så att configure-skriptet kan generera Makefile-filer från Makefile.in-filerna genom att substituera de generella parametrarna med systemspecifika parametrar.

Autotools använder sig av en fil configure.ac som input. Configure.ac är ett shellscript som innehåller ett antal macron som är skrivna i macrospråket m4[13]. Autoconf tillhandahåller ett antal fördefinerade macron vars uppgift är att kontrollera olika systemegenskaper. En mycket enkel configure.ac-fil kan se ut så här:

```
AC_INIT(hello.cc)
AM_INIT_AUTOMAKE(hello,0.1)
AC_PROG_CC
AC_PROG_INSTALL
AC_OUTPUT(Makefile)
```

AC_INIT makrot kontrollerar att källfilen finns. AM_INIT_AUTOMAKE utför ett antal standard tester. AC_PROG_CC anger att koden är skriven i C. AC_PROG_INSTALL genererar ett installationsmål, så att användaren kan installera programmet med *make install*. AC_OUTPUT sätter namnet på Makefile-filen som ska genereras.

Libtool används för att abstrahera detaljerna involverade för att skapa ett delat bibliotek och används därför inte lika ofta som Autoconf och Automake i projekt. Figur 3.2 visar hur Automake, Autoconf och deras respektive input- och outputfiler interagerar med varandra.



Figur 3.2: Hur Autotools verktyg interagerar

Sean Egan skriver i *Open Source Messaging Application Development: Building and Extending Gaim* att:

> *"Autotools är mycket användbart, men notoriskt svåranvänt för större projekt. Detta problem ökar med det faktum att olika versioner av Autotools uppför sig avsevärt olika. Det som fungerade fint i en version kan misslyckas fullständigt i en annan."*

Att det är svåranvänt beror även till stor del på att verktygen är fristående från varandra och att configure.ac-filen har en tendens att bli mycket stor.

Bootstrapping kallas termen för att köra autotools som skapar configure-scriptet och makefile-filerna. För att kunna köra autotools brukar man skapa en autogen.sh-fil som automatiserar denna process. En enkel autogen.sh-fil kan se ut på följande sätt:

```
aclocal || exit;
autoconf || exit;
automake --add-missing --copy || exit;
./configure $@
```

Aclocal[36] är ett fristående program som har till uppgift att kopiera de användbara makron som Autoconf och Automake tillhandahåller till projektet, vilket medför att detta inte behöver göras manuellt. Genom att inkludera dessa makron i projektet behöver de inte vara installerade på det system där applikationen ska installeras.

Flaggorna till automake (-- add-missing och -- copy) är där därför att Automake kräver att ett antal filer existera i projektet, och om inte dessa filer finns måste man kopiera över standardversionerna av de som Automake tilhandahåller. Dessa filer är NEWS, AUTHORS, ChangeLog och README.

3.5.3 SCons

SCons[25] är ett byggverktyg för mjukvara likt Autotools. Men till skillnad från Autotools är configurationsfilerna för SCons alla skrivna i samma språk, nämligen skriptspråket Python. Tack vare detta är det lätt att sätta sig in i för den som använt Python förut, till skillnad från många andra verktyg som ofta har egenutvecklade språk eller använder flera olika (Autotools använder bland annat språket m4).

Det klassiska “Hello, World” programmet följer:

```
int main() {
     printf("Hello, world!\n");
}
```

För att kompilera detta (om koden antas ligga i en fil *hello.c*) räcker det att skriva en rad i SCons inputfil, SConstruct:

```
Program('hello.c')
```

Denna enkla rad talar om för SCons att en exekverbar fil ska byggas från källfilen hello.c.

SConstruct är SCons motsvarighet till make's Makefile. Låt säga att vi vill bygga ett projekt som kräver att pkg-config[18] finns installerat och har version >= 0.15.0. Följande listning ur en SConstruct fil är ett exempel på detta:

```
def CheckPKGConfig(context, version):
    context.Message( 'Checking for pkg-config... ' )
    ret = context.TryAction('pkg-config --atleast-pkgconfig-version=%s' % version)[0]
    context.Result( ret )
    return ret

env = Environment()
conf = Configure(env, custom_tests = {'CheckPKGConfig' : CheckPKGConfig})
if not conf.CheckPKGConfig('0.15.0'):
    print 'pkg-config >= 0.15.0 not found.'
    Exit(1)
env = conf.Finish()

Program('test.c')
```

I denna SConstruct-fil anropas funktionen CheckPKGConfig för att kontrollera om pkg-config är installerat och har version >= 0.15.0. Om det inte är fallet skrivs ett felmeddelande ut och byggprocessen avbryts. I annat fall kompileras och länkas test.c (notera dock att pkg-config inte alls behövs för att kunna kompilera och länka test.c,

utan detta är endast för att illustrera hur det kan se ut då ett projekt har som förvillkor för en korrekt byggning att vissa bibliotek finns installerade innan byggning kan ske).

Den stora fördelen med SCons gentemot make och Autotools är att SCons använder ett och samma språk som dessutom är ett välkänt, stort programspråk med hög abstraktionsnivå. Till skillnad från Autotools använder SCons dessutom bara vanligen en konfigureringsfil – SConstruct (flera konfigureringsfiler kan dock användas).

SCons blev därför till slut det byggverktyg som valdes till detta projekt; för att det är betydligt enklare att sätta sig in i, arbetar på en mycket högre abstraktionsnivå än Autotools, och för att det använder ett enda språk som är välkänt och relativt enkelt att lära sig.

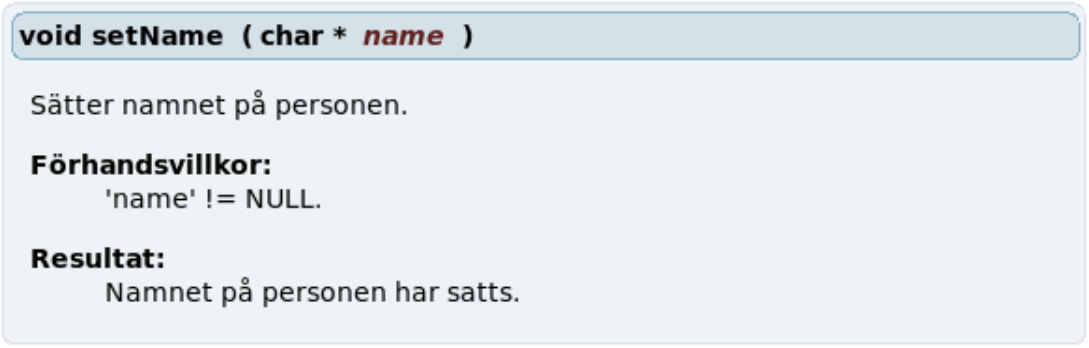
3.6 Doxygen

Doxygen[17] är ett verktyg för att generera formaterad dokumentation från strukturerad koddokumentation. Källfilerna i ett projekt läses in av Doxygen, varefter dokumentationen som är skriven i koden hämtas ut och används i genereringen av självständig dokumentation. Doxygen stödjer en mängd språk; bland annat C, C++, och Java, och output formaten är bland annat HTML[55], LaTeX[58], och PDF[61].

Här är ett exempel på dokumentation av en funktion, som sätter namnet på en abstrakt person, skriven i en källfil:

```
/**
 * Sätter namnet på personen.
 *
 * @pre 'name'!= NULL.
 * @post Namnet på personen har satts.
 */
void setName(char *name);
```

Delen i HTML dokumentationen som genereras från denna funktionsdokumentation med Doxygen kan ses i figur 3.3.



Figur 3.3: Doxygen dokumentation för funktionen setName

För att öka sannolikheten att dokumentationen beskriver den senaste versionen av ett system bör all dokumentation på modulnivå finnas direkt i koden[29], för att med lämpligt verktyg hämtas ut och göras mer lättläst och att all dokumentation är samlad på ett ställe. Av denna anledning har moduldokumentation skrivits i källfilerna enligt en struktur som Doxygen känner igen, för att sedan kunna extraheras med Doxygen och samlas i HTML-filer.

3.7 Versionshantering

För att kontrollera vilka ändringar som gjorts vid varje ny version, för att kunna gå tillbaks till gamla versioner på ett enkelt sätt, och för att helt enkelt lagra all kod på ett lättillgängligt sätt, användes SVN[62]. SVN designades för att ersätta CVS[54], och är ett versionshanteringssystem. Ett versionshanteringssystem är ett system för att hantera flera versioner av en samling information, till exempel källkod. Genom att använda SVN kan man enkelt gå tillbaks till en äldre version av systemet om det gått snett någonstans på vägen.

Det gör det även möjligt att på ett enkelt sätt låta flera användare arbeta med samma kod samtidigt.

3.8 Sammanfattning

I detta kapitel har de olika metoder och verktyg som använts under projektet presenterats, tillsammans med motivering till varför de använts.

eXtreme Programming (XP) har till viss del inkorporerats i utvecklingen.

De programspråk som använts har presenterats. C++ valdes över C för att få ett språk med inbyggt stöd för objektorienterad programmering, istället för att försöka simulera detta med dynamiska struct:ar och semantik. Dessutom gavs en kort introduktion till skriptspråket PHP då det använts i testerna i avsnitt 5.4.

Anledningen till att SCons valdes som byggverktyg till projektet har redogjorts tillsammans med en presentation av de två första alternativen som undersöktes – GNU make och GNU Autotools.

Databassystemet MySQL har använts som gränssnitt mellan Instant Messaging-wrappern och wrapperns klient, och en motivering till detta har presenterats.

Doxygen har använts som dokumenteringsverktyg, som genererar välstrukturerad dokumentation i form av bland annat HTML-sidor. Doxygen tar källfiler som input där dokumentationen är skriven tillsammans med koden.

Eftersom versionshantering har använts i projektet har en kort redogörelse gjorts för detta.

I nästa kapitel presenteras kraven som ställts på det system som utvecklats, och de olika lösningsförslag som tagits fram.

Kapitel 4

Analys av möjliga lösningar

4.1 Introduktion

Detta kapitel börjar med att introducera kravspecifikationen som ligger till grund för systemet som utvecklats. I resterande del av kapitlet presenteras tre olika förslag på hur systemet kunnat designats utifrån kravspecifikationen, och motivering till varför den sista lösningen – en C++-wrapper till libGaim – valdes och inte någon av de två andra.

I kapitlet ingår en genomgång av PHP-extensions och olika sätt att implementera dessa, och en kort presentation av IM-tjänsten Jabber.

4.2 Kravinsamling och specifikation

Ingen formell kravspecifikation har funnits under detta arbete. Detta är istället en sammanställning av de muntliga krav som erhållits och diskuterats med vår handledare på Evolve, Andreas Hassellöf. Kravinsamlingen har skett i samtal med Andreas, varefter detta har undersökts vidare för att se om detta varit den bästa lösningen, och om det över huvud taget varit en möjlig lösning.

Härnäst följer systemkraven som ställdes på systemet:

- Systemet som ska utvecklas är en Instant Messaging-klient som gör det möjligt att använda Instant Messaging-tjänster med ett skriptspråk, huvudsakligen PHP, men det är fördelaktigt om det även går att använda andra skriptspråk.
- IM-klienten ska arbeta i bakgrunden, det vill säga vara en *daemon*, och klara av att hantera ett stort antal användare på samma gång.
- IM-protokollet MSNP[59] ska stödjas av systemet. Andra protokoll har inte lika hög prioritet men det vore önskvärt med stöd för Jabber[21] och OSCAR[60], och även andra IM-protokoll.
- Skulle systemet krascha av någon anledning ska det gå att starta om det utan att detta märks i klienten.
- Det ska inte gå att krascha systemet eller utnyttja det för att få ut känslig information genom att en användare skickar in ett IM-meddelande innehållandes speciellt utformad text.
- Gränssnittet till IM-klienten ska vara enkelt; det vill säga att komplexiteten inte ska ligga hos klienten, utifall att klienten kan behövas skrivas om i framtiden.
- Systemet ska fungera i operativsystemet Linux.

4.3 Lösningsförslag

Här presenteras tre olika lösningsförslag för att utveckla ett system som uppfyller kraven från 4.2. Det ursprungliga förslaget var att libGaim[10] skulle utnyttjas av systemet för att hantera IM-tjänster. Utifrån detta, och kraven, sammanställdes de tre förslag som presenteras här.

Det sista förslaget som presenteras – en libGaim-wrapper i C++ – är det system som till slut valdes.

4.3.1 PHP-extension till libGaim

Eftersom PHP[8] skulle användas och IM-biblioteket libGaim är skrivet i C, var det första förslaget att göra en PHP-extension till libGaim. Denna extension skulle då mappa de nödvändiga metoder från libGaim till PHP.

En PHP-extension är kod som är designad för att addera funktionalitet till PHP. Till exempel finns extensions för att låta PHP arbeta mot MySQL-databaser[31], och extensions för att låta PHP dynamisk generera bilder[45]. En PHP-extension är ett exempel av en språkstatisk språkwrapper (se avsnitt 2.2.5).

Det finns tre olika typer av PHP-extensions:

Zend-extension – Zend[33][65] är den motor som parsar, översätter och exekverar alla PHP-skript. En zend-extension ändrar hur själva zend fungerar, till exempel hur villkorsuttryck utvärderas.

Inbyggd extension – Inbyggda extensions kompileras direkt in i PHP, vilket gör att de laddas automatiskt av PHP och behöver alltså inte laddas manuellt för att kunna användas. Med denna typ av extension behövs heller inga extension-filer, då den kompileras in i PHP-interpretatorn. Detta kräver dock att hela PHP-interpretatorn kompileras om varje gång någon ändring görs i extensionen.

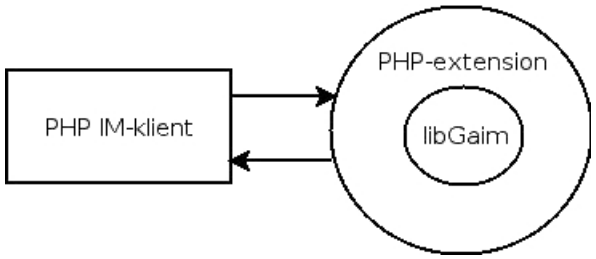
Extern extension – Externa extensions, till skillnad från inbyggda extensions, måste laddas manuellt under exekvering. Detta gör att exekveringen tar något längre tid.

Här är några exempel på användningsområden för PHP-extensions:

- Om det skulle finnas funktionalitet som inte går att implementera i PHP kan en PHP-extension skapas för att tillgodose detta behov.
- Om man vill att själva PHP ska uppföra sig på ett annorlunda sätt än det gör som standard kan en PHP-extension skapas.

- Om man vill öka prestandan för kod skriven i PHP kan en PHP-extension användas som använder kompilerad C-kod istället.
- Om kod är skriven som ska säljas till kund, men koden får inte visas för kunden, kan en PHP-extension användas.

I figur 4.1 presenteras ett diagram över hur den ursprungliga lösningen var tänkt.



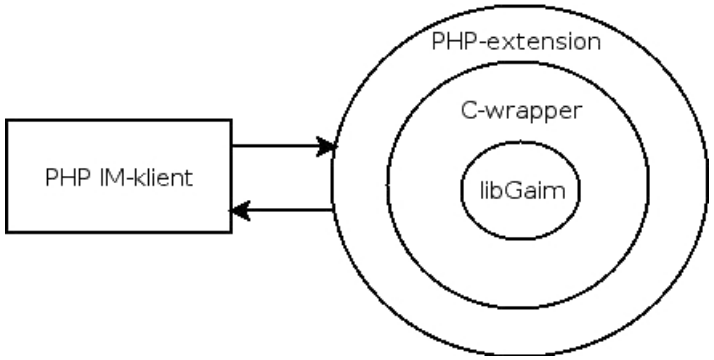
Figur 4.1: Första designen

I denna design antas att det går att skapa en PHP-extension direkt till libGaim, men efter en djupare efterforskning i hur libGaim fungerar kunde det konstateras att detta inte skulle fungera. För att motivera detta krävs först en kort genomgång om hur libGaim hanterar inkomna meddelanden.

När libGaim tar emot ett meddelande från nätverket undersöks detta för att se vilken typ av meddelande det är, och skickar meddelandet vidare till en funktion som kan hantera det. Denna funktion, efter att ha hanterat det på ett protokollspecifikt sätt, skickar ut en signal. En signal är i själva verket bara ett funktionsanrop som sker genom att funktionen gaim_signal_emit anropas med en sträng som anger vilken signal det är. gaim_signal_emit söker då upp i en tabell med hjälp av signalsträngen vilken funktion som ska anropas.

Det viktiga att observera i allt detta är att en funktionspekare måste lagras i en tabell för varje signal som ska kunna skickas ut. Om nu en PHP-extension skulle utvecklas till libGaim skulle det betyda att ett PHP-skript som använder denna extension måste

koppla in PHP-funktioner som signalfunktioner. Detta fungerar helt enkelt inte. Det finns inget inbyggt stöd i C för att anropa PHP-funktioner. Det finns däremot C-bibliotek som kan göra detta möjligt, men detta skulle kräva att det tillkommer ännu en wrapper, ovanpå libGaim men innanför PHP-extensionen (se figur 4.2).



Figur 4.2: Version 2 av vår PHP-extension

Denna C-wrapper använder ett PHP API för att göra det möjligt att koppla in PHP-funktioner som signalmottagare.

I de efterföljande tre avsnitten visas tre olika metoder för att skapa en PHP-extension.

Standardmetoden

För att skapa en PHP-extension tillhandahåller PHP ett skript, ext_skel, som genererar de filer som behövs. Skriptet exekveras från ext-biblioteket i PHP's källkodskatalogstruktur på följande sätt:

```
./ext_skel --extname=min_extension
```

Flaggan *extname* anger att extensionen ska ha namnet *min_extension*. Vid detta skrivs följande ut:

```
Creating directory min_extension
Creating basic files: config.m4 config.w32 .cvsignore
```

```
min_extension.c php_min_extension.h CREDITS EXPERIMENTAL
tests/001.phpt min_extension.php [done].

To use your new extension, you will have to execute the following steps:

1.  $ cd ..
2.  $ vi ext/php_wrapper/config.m4
3.  $ ./buildconf
4.  $ ./configure --[with|enable]-min_extension
5.  $ make
6.  $ ./php -f ext/min_extension/min_extension.php
7.  $ vi ext/min_extension/min_extension.c
8.  $ make

Repeat steps 3-6 until you are satisfied with ext/min_extension/config.m4 and
step 6 confirms that your module is compiled into PHP. Then, start writing
code and repeat the last two steps as often as necessary.
```

Instruktionerna som skrivs ut gäller dock bara om PHP-extensionen ska kompileras in i PHP (intern extension). Då en PHP-extension ska göras i form av en extern extension används vanligtvis följande sekvens av kommandon i extensionens katalog efter att ext_skel-skriptet exekverats:

```
$ phpize
$ ./configure --enable-min_extension
$ make
$ make install
```

Innan phpize exekveras måste filen config.m4 (som skapades av ext_skel-skriptet) editeras. ext_skel har skapat grunderna, så för en enkel extension räcker det att ta bort

kommentartecken på två rader så att det står antingen

```
PHP_ARG_WITH(min_extension, for min_extension support,
[  --with-min_extension             Include min_extension support])
```

eller

```
PHP_ARG_ENABLE(min_extension, whether to enable min_extension support,
[  --enable-min_extension           Enable min_extension support])
```

Den första används när extensionen behöver använda ett existerande C-bibliotek, annars används den andra. Om det dessutom är fler än en källfil som används vid kompilering av extensionen (vilket ofta är fallet om befintlig C-kod ska wrappas till PHP), måste den sista raden i filen ändras. Denna rad ser ursprungligen ut på följande sätt:

```
PHP_NEW_EXTENSION(min_extension, min_extension.c, $ext_shared)
```

Ytterliggare källfiler läggs till bredvid *min_ extension.c*, med endast mellanslag emellan.

PEAR

PEAR[46] (PHP Extension and Application Repository) är ett projekt som har som mål att skapa ett strukturerat bibliotek av PHP-kod som är lätt att distribuera i form av paket och med hjälp av en pakethanterare. PEAR främjar återanvändandet av ofta använd PHP-kod.

PEAR tillhandahåller ett pakethanteringssystem som förenklar distribueringen och hanteringen av paket. Om man till exempel vill installera ett paket *pear_ paket* som inte redan finns installerat används följande kommando:

```
pear install --alldeps pear_paket
```

install anger att ett paket ska installeras. Flaggan *--alldeps* anger att alla paket som det paketet som ska installeras beror på ska installeras de också. *pear_ paket* är namnet på

det paket som ska installeras. Alla paket som krävs laddas då ned från PEAR's server och installeras.

Ett paket inom PEAR är CodeGen_PECL[44]. Detta paket innehåller skriptet pecl-gen som, likt ext_skel, används till att generera PHP-extensions. Pecl-gen genererar dock inte bara ett skelett för en PHP-extension och låter användaren vidareutveckla detta, utan pecl-gen läser in en XML-fil och genererar färdig extension-kod utifrån denna (som sedan kan användas med phpize på samma sätt som i det tidigare avsnittet). Skriptet exekveras på följande sätt:

```
pecl-gen min_extension.xml
```

pecl-gen har stöd för de flesta vanliga delar av en PHP-extension-konfiguration, som stöd för användning av externa bibliotek (vilket anges med <lib>-taggen), inkludering av header-filer (vilket anges med <header>-taggen), och villkorlig kompilering. Även om det inte står dokumenterat i manualen för CodeGen_PECL finns det stöd för att lägga till ytterligare källfiler till kompileringen (vilket med ext_skel var tvunget att anges direkt i config.m4-filen). Detta görs med <file>-taggen.

En enkel input-fil kan se ut på följande sätt:

```
<?xml version="1.0"?>
<extension name="min_extension">
  <function name="min_funktion">
   <proto>string min_funktion(void)</proto>
   <summary>En enkel funktion</summary>
   <description>
    Denna funktion returnerar en statisk sträng.
   </description>
   <code>
    RETURN_STRING("test", 1);
```

```
   </code>
  </function>
</extension>
```

Detta skapar en extension med namnet min_extension som innehåller funktionen min_funktion.

I det förra avsnittet visades hur config.m4-filen ändrades för att antingen använda externa C-bibliotek, eller inte använda externa C-bibliotek. I pecl-gen genereras PHP_ARG_ENABLE som standard till config.m4-filen, vilket var det macro som anger att inget externt C-bibliotek ska användas. För att istället använda externa C-bibliotek används taggen <with>.

Förutom att generera extensions kan pecl-gen även generera dokumentation. I exemplet ovan finns till exempel taggarna <summary> och <description> som båda används i dokumentationsgenerering.

SWIG

I avsnitt 2.2.5 nämndes verktyget SWIG, som är ett verktyg för att generera språkstatiska språkwrappers. Samma input-fil kan ofta användas till att generera språkwrappers till olika språk. Detta verktyg kan användas till att generera PHP-extensions. SWIG anropas då på följande sätt:

```
swig -php min_extension.i
```

Flaggan *-php* anger här att det är en PHP-extension som ska genereras, och *min_extension* är namnet på interface-filen som SWIG tar som input. Här följer ett exempel på en sådan interface-fil:

```
%module min_extension
%{
int min_funk1(int a);
```

```
int min_funk2(int n, int m);
%}
int min_funk1(int a);
int min_funk2(int n, int m);
```

Första raden anger att modulen som ska skapas heter *min_ extension*. Koden inom %{ och %} anger kod som ska kopieras direkt in i den genererade wrapper-koden. Det som kommer efter %} är det som ska wrappas.

Efter att SWIG exekverats med en interface-fil måste källfiler kompileras ihop med den genererade källkods-filen som i exemplet ovan får namnet min_extension_wrap.c. Detta kan se ut på följande sätt:

```
gcc -I/usr/include/php5 -I/usr/include/php5/Zend -I/usr/include/php5/main \
    -I/usr/include/php5/TSRM min_extension.c min_extension_wrap.c -c

gcc -shared min_extension.o min_extension_wrap.o -o min_extension.so
```

Sökvägarna som efterföljer flaggan -I anger sökvägar till header-filer i PHP som krävs av den genererade wrapper-koden.

Sedan måste det delade biblioteket min_extension.so kopieras till den katalog som PHP använder för externa PHP-extensions (vilket i de tidigare avsnittet gjordes automatiskt med make install).

```
[sudo] cp min_extension.so /usr/lib/php5/20051025/min_extension.so
```

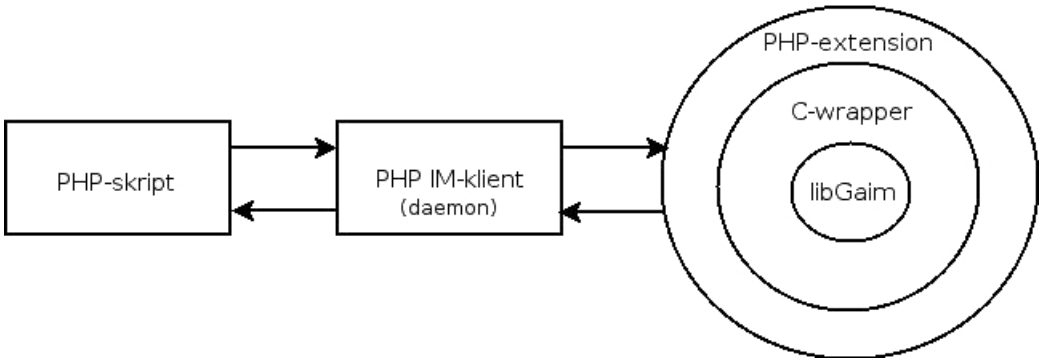
Efter detta kan min_extension laddas in i PHP med

```
dl("min_extension.so");
```

Varför denna design inte valdes

Den stora anledningen till varför denna lösning inte var den bästa för den wrapper som skulle utvecklas var att ett av systemkraven inte uppfylls, nämligen att systemet ska

arbeta som en daemon som är igång hela tiden. Om systemet ska vara igång hela tiden, tillkommer ännu en komponent i slutsystemet, vilket visas i figur 4.3.



Figur 4.3: Version 3 av vår PHP-extension

Sammankopplingen mellan PHP-skriptet och PHP-daemonen måste då ske med hjälp av någon form av interprocesskommunikation. Men om vi ändå måste ha flera processer som ska kommunicera med varandra, vad var det då för mening med att göra en PHP-extension? Då skulle det vara betydligt enklare att bara skapa en C-wrapper till libGaim som arbetar som en daemon, och låta denna kommunicera med PHP genom något medium, till exempel IPC shared memory eller genom en databas. Detta skulle även göra att klienter skrivna i andra språk än PHP kan kopplas in mot libGaim-wrappern.

4.3.2 Jabber-klient

Jabber[21] är en IM-tjänst likt ICQ och MSN. Men till skillnad från dessa är Jabber öppen källkod och använder standardiserade, öppna protokoll (begreppet Jabber används även ofta att beteckna själva protokollet XMPP och även Jabber-serverapplikationer). Jabber bygger på protokoll där meddelandena skickas i form av XML[64] (eXtensible Markup Language). XML passar bra till att strukturera dataströmmar genom att olika delar skärmas av från varandra med *taggar*.

För att visa på hur meddelanden mellan klient och server fungerar i Jabber exemplifieras detta med en registreringssession. För att registrera ett nytt konto mellan en klient och en Jabber-server skickar klienten ett IQ-meddelande (info/query)[40]. Detta kan se ut på följande sätt:

```
K: <iq type='get' id='reg1'>
      <query xmlns='jabber:iq:register'/>
   </iq>
S: <iq type='result' id='reg1'>
      <query xmlns='jabber:iq:register'>
         <username/>
         <password/>
      </query>
   </iq>
K: <iq type='set' id='reg2'>
      <query xmlns='jabber:iq:register'>
         <username>pelle</username>
         <password>mittpasswd</password>
      </query>
   </iq>
```

IM-tjänster som ICQ och MSN bygger på att många användare kopplar upp sig mot en gemensam server. Jabber arbetar istället med en decentraliserad arkitektur, där vem som hellst kan sätta upp sin egen Jabber-server, och därmed även själv bestämma vilken funktionalitet som ska tillhandahållas.

I en Jabber-klient kan *gateways* kopplas in. En gateway (även kallat transport) är en mjukvarumodul som kan kopplas in i en Jabber-server för att ge användare tillgång till andra IM-tjänster än Jabber.

För att kunna använda en gateway måste användaren först skicka ett meddelande till

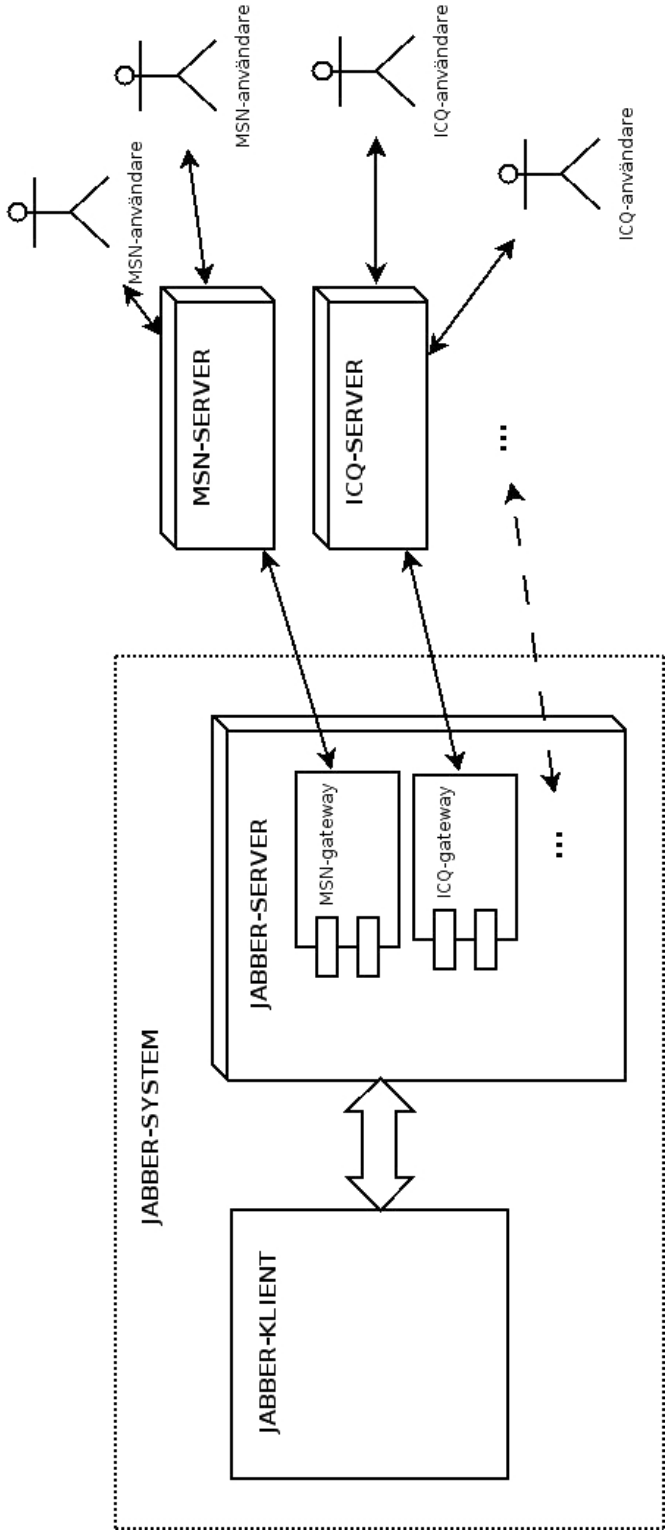
servern att en gateway av en viss typ (som då måste finnas tillgänglig på servern) ska registreras för användaren. Därefter kan användaren koppla upp ett konto mot denna gateway, till exempel kan ett MSN-konto kopplas upp genom en MSN-gateway. På detta sätt har användaren möjlighet att samla alla sina konton i olika IM-tjänster i samma applikation.

Med denna möjlighet till utbyggnad och konfigurering var Jabber en möjlighet för IM-systemet som skulle utvecklas. Jabber-systemet skulle då se ut som i figur 4.4.

En Jabber-server skulle användas som den daemon som exekverar ständigt, och klienten som, om den implementeras i PHP, kan förslagsvis använda sig av Blitzaffe Jabber Client[4].

När en användare kopplar upp sig mot Jabber-servern måste ett nytt Jabber-konto genereras, alternativt används samma konto för alla användare men med olika resources (en resource är ett värde som definierar instansen av ett konto och möjliggör samma konto att vara uppkopplat som flera instanser samtidigt. Om användarnamnet är pelle@jabber.se, läggs resourcen som ett postfix till detta och bildar exempelvis pelle@jabber.se/hemma). Detta för att användaren inte ska få någon kännedom om att det är Jabber som används i bakgrunden. Användaren ska då bara kunna logga in med sina tjänst-specifika inloggningsuppgifter, och slipper skapa ett eget Jabber-konto.

Allt detta leder till tre möjligheter i utvecklingen av systemet; antingen att skriva en ny Jabber-server helt från grunden, vilket skulle vara ett projekt som inte skulle få plats inom tidsramen för detta arbete, eller vidareutveckla en existerande server för att passa våra behov, eller lägga ansvaret för att skicka alla meddelanden som behövs på klienten.



Figur 4.4: Systemet implementerat med Jabber

Om en existerande server ändras för att bättre stödja våra behov (genom att det till exempel räcker med ett inloggningsmeddelande som innehåller protokollnamn och kontouppgifter för att servern ska registrera Jabber-användaren, koppla upp Jabber-kontot, koppla in lämplig gateway, och koppla upp användarens IM-konto) skulle detta leda till att servern blir svårare att uppdatera med nya versioner av ursprungsservern, vilket kan behövas om viktiga brister skulle upptäckas.

Om ansvaret läggs på klienten kan en existerande server användas utan att någon ändring i servern krävs. När en användare kopplar upp sig måste då klienten skicka registreringsmeddelande, ta emot svar från servern, skicka uppkopplingsmeddelande, och så vidare. Detta ska då ske automatiskt utan att användaren märker av det. Innan registreringsmeddelandet skickas måste klienten generera kontonamn alternativt resource till ett befintligt konto. Detta hade även lett till att arbetet tagit en annan form, eftersom det ursprungligen inte var tänkt att vi skulle utveckla klienten, utan bara servern.

Varför denna design inte valdes

Varför valdes då inte denna design? Vilka brister har denna design som gjort att vi valde bort den?

En av anledningarna till att Jabber-designen inte valdes var att komplexiteten i den designen läggs på klienten. Detta är oönskat eftersom klienten kan behöva skrivas om i andra språk i framtiden, och för att detta ska gå så enkelt och snabbt som möjligt bör komplexiteten hållas till ett minimum i klienten.

En annan frågeställning som fanns angående Jabber var företaget AOL's inställning till det. AOL (som äger ICQ och AIM) har tidigare blockerat Jabber-servrar med olika metoder[28]. Skulle till exempel AOL blockera servern som används i detta system skulle detta vara förödande för användningen av systemet.

Under den tidiga utvecklingen av systemet var vi kanske lite väl fokuserade vid att libGaim skulle användas. Vårt uppdrag var att utveckla ett system som använde

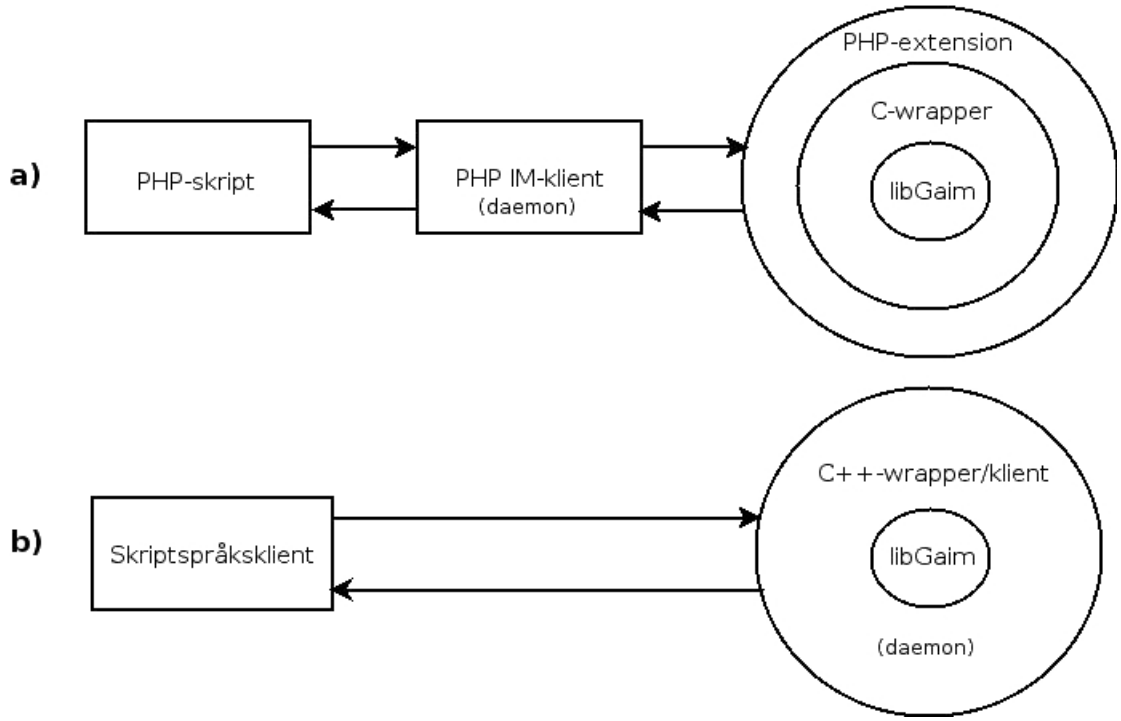
libGaim, vilket ledde till att vi sökte olika lösningar till användningen av detta bibliotek och inte andra lösningar på problemet som helhet. När sedan Jabber-lösningen visade sig hade vi redan påbörjat och kommit en bit in i utvecklingen av libGaim-wrappern. Beslutet togs att fortsätta utveckla detta system, dels för att vår studie är av wrappers (vilket inte varit möjligt på samma sätt om Jabber-lösningen använts), dels för att systemet vi påbörjat upplevdes av oss som att det skulle komma att fungera väl, och dels på grund av anledningarna som nämnts ovan.

Ett starkt argument för att använda libGaim var även att vi visste att det skulle fungera, då det redan finns applikationer av liknande typ som använder just libGaim. Ett sådant exempel är den webbaserade IM-klienten meebo, som använts stundvis som referenspunkt under utvecklingen.

4.3.3 libGaim-wrapper

Som angivits i avsnitt 4.3.1 försvann hela nyttan med en PHP-wrapper i och med att applikationen som använder libGaim måste exekvera kontinuerligt och kräver någon form av kommunikation mellan processer. Genom att förbigå PHP-extensionen blir designen enklare och använder bara språken C/C++, istället för C, möjligen C++, och PHP. Skillnaden mellan de två designerna visas i figur 4.5.

C++-wrappern agerar både som IM-klient, som daemon, och tillhandahåller gränssnittet mot den slutgiltiga skriptspråksklienten. Notera att C++-wrappern är en språkgenerisk wrapper, i och med att den tillåter klienter skrivna i alla språk som kan hantera interprocessmediet att använda den. PHP-extensionen i **a** är istället en språkstatisk wrapper, eftersom den endast tillåter klienter (det vill säga daemonen i **a**) skrivna i PHP att använda den.



Figur 4.5: a) PHP-extension-design och b) C++-wrapper-design

Som påpekat tidigare måste kommunikationen mellan slutklienten och daemonen ske genom någon form av interprocesskommunikation (vilket leder till att det även i design **a** är möjligt att koppla in klienter skrivna i andra språk än PHP). Här finns flera alternativ att välja på:

1. IPC i sin vanliga benämning vilket innefattar sockets, delat minne, och meddelandeköer (och även semaforer, men då detta är en form av synkronisering platsar detta inte i denna lista).

2. En eller flera filer.

3. En databas.

Sockets är ändpunkter för tvåvägskommunikation mellan processer. Domänen för en socket anger i vilken miljö socketen ska användas. Internet är en domän, men den som

skulle användas i detta sammanhang är unix domänen. Servern i systemet skapar först en ny socket och lyssnar sedan efter uppkopplingar mot denna socket. När klienten kopplat upp sig kan meddelandeutbyte ske.

Delat minne är ett sätt att skicka data mellan processer. En process skapar ett minnesutrymme som sedan kan nås av andra processer som har rättigheter.

Meddelandeköer är ett verktyg för kommunikation mellan processer som möjliggör att meddelanden placeras i en kö varur andra processer sedan kan läsa dessa meddelanden.

Användning av en eller flera filer är ett enkelt sätt att skicka information mellan processer. En process skriver ned information till en fil och en annan process läser från filen. För att få en bättre struktur på alla meddelanden kan de delas upp på flera filer och på så sätt simulera en databas. En anledning till att göra på detta sätt istället för att använda en riktig databas kan vara för att öka på prestandan, då det är möjligt att ett databassystem likt MySQL ger en overhead för funktionalitet som inte behövs. Vi har dock inte utfört några tester för att kontrollera detta.

Det som valdes som interprocesskommunikationsmedium var en databas. Att använda en databas har flera fördelar mot de andra alternativen:

- Struktur – Alla meddelanden kan struktureras på ett sätt som blir lättförståligt.
- Färdiga algoritmer – När en design av databasen tagits fram finns färdiga algoritmer för att överföra detta till en modell som går att direkt föra in i databassystemet.
- Testbart – Det är relativt enkelt att snabbt lägga in nya poster i databasen för att testa hur systemet hanterar dem, och sedan se vad som läggs in i databasen av systemet. Manuell testning blir med andra ord enklare.
- Bestående – Om datorn som databasen är installerad på stängs ner av någon anledning försvinner inte innehållet i databasen som det gör vid användning av

sockets, delat minne, och meddelandeköer. Detta ökar möjligheten att stödja kravet om ett kraschsäkert system.

4.4 Sammanfattning

I detta kapitel har kravspecifikationen för systemet presenterats i form av en punktlista sammanställd från muntliga diskutioner med vår handledare på Evolve – Andreas Hassellöf. Denna specifikation har sedan använts i utvärderingen av tre olika förslag på lösningar till systemet.

Jämförelsen av de tre förslagen har givit slutsatsen att en C++-wrapper till libGaim-biblioteket var den mest passande lösningen för att uppfylla kraven som ställts på systemet.

Kapitel 5

Prototyp av libGaim-wrapper

5.1 Introduktion

Målet med arbetet som utförts var att få fram ett system som möjliggjorde användning av IM-tjänster i skriptspråk; huvudsakligen PHP. Beslutet togs att använda IM-biblioteket libGaim som kärnan i systemet, och utveckla en wrapper till denna. Systemet som till slut utvecklades kan ses i figur 1.1 i kapitel 1. De grå delarna i figuren är de som utvecklats i detta arbete.

I detta kapitel redogörs mer ingående än tidigare vad libGaim är och hur det fungerar, och dessutom hur detta är kopplat till det system vi har utvecklat. Här presenteras även varför libGaim är ett välanvänt bibliotek, men även vilka nackdelar det för med sig. Här diskuteras också i korthet hur själva öppen källkods-projektet Gaim är uppbyggt och hur utvecklingen av det fungerar.

I syfte att ge en överblick av prestandaskillnaderna hos de två olika sorterna språkwrappers (språkstatiska och språkgeneriska) presenteras i detta kapitel resultatet av fyra tester som utförts. Dessa tester visar att valet av språkwrapper behöver en stark motivation när prestanda är viktig. Meddelanden skickade via MySQL i en språkgenerisk språkwrapper visade sig ta ungefär 500 gånger längre tid än vanliga funktionsanrop,

medan testet med en PHP-extension (en språkstatisk språkwrapper) endast tog cirka 10 gånger längre tid.

Vidare i rapporten presenteras själva systemet som utvecklats; hur det är uppbyggt och motivationen bakom det.

5.2 libGaim

LibGaim är ett C-bibliotek som tillhandahåller tillgång till IM-tjänster som ICQ och MSN genom användning av insticksmoduler som implementerar protokollen för dessa tjänster.

5.2.1 Struktur

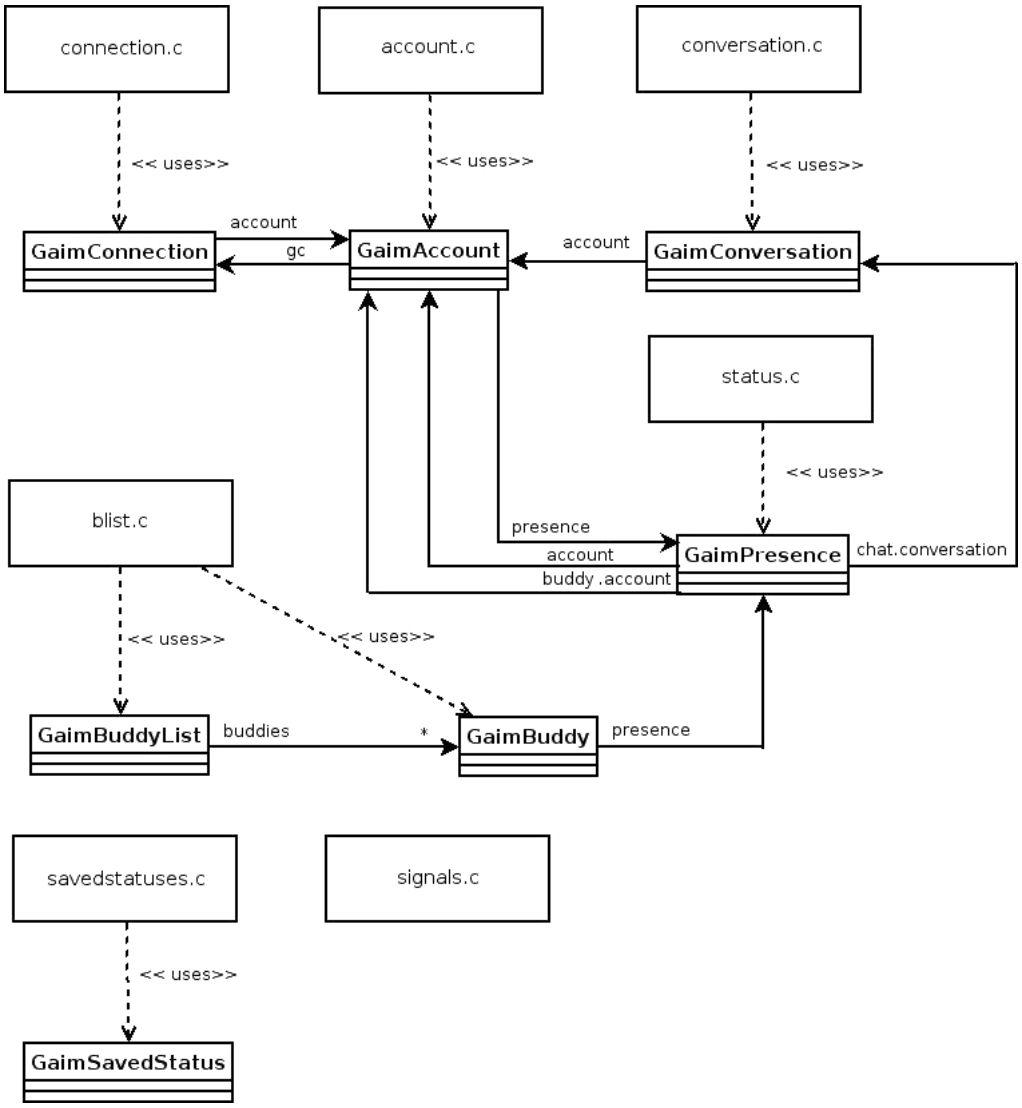
Överblick

LibGaim omfattar cirka 60000 rader C-kod (de protokoll-insticksmoduler som följer med är på sammanlagt cirka 132000 rader), som saknar komplett dokumentation, vilket gör det svårt att skapa en fullständig överblick av biblioteket. Istället är detta ett försök att visa på de viktigaste delarna av libGaim och hur de hänger ihop.

I detta avsnitt kommer begreppet klass användas i en designmässig betydelse, i det avseende att C-structar i libGaim ses som klasser, och funktionerna som ligger i samma fil och arbetar på structen ses som klassens funktioner. Anledningen till att det här talas om klasser trots att språket är C är att libGaim är utvecklat med ett objektorienterat synsätt (se avsnitt 3.4.1).

De viktigaste klasserna i libGaim är GaimAccount, GaimConnection, GaimConversation, GaimBuddyList, GaimBuddy, GaimPresence, och GaimSavedStatus. Till de flesta klasser finns en stor mängd funktioner. Funktionerna för klasserna ovan finns i filerna account.c, connection.c, conversation.c, blist.c (blist.c innefattar funktionerna för både GaimBuddyList och GaimBuddy), status.c, och savedstatuses.c,

som visas i figur 5.1. I figuren visas även signals.c som innehåller signalhanteringsfunktioner, vilket är en viktig del av libGaim.



Figur 5.1: libGaim's viktigaste delar

- GaimConnection är en uppkoppling mellan ett konto och en IM-server. Funktionerna i connection.c hanterar en eller flera uppkopplingar.

- GaimAccount representerar ett IM-konto. Funktionerna i account.c hanterar ett eller flera konton.

- GaimConversation representerar en konversation mellan ett konto och en IM-kompis eller en chat-kompis. Notera att det i figur 5.1 inte finns någon Buddy eller liknande kopplad till GaimConversation, utan denna representeras endast med en sträng. Funktionerna i conversation.c hanterar en eller flera konversationer.

- GaimBuddy representerar en kompis i kompislistan. En kompis är en representation av en annan användares konto som kompisens ägare kan utbyta meddelanden med. GaimBuddy har en referens till det konto kompisen tillhör och även till den GaimPresence som innehåller all statusinformation om kompisen. Funktionerna för att hantera en eller flera kompisar finns i blist.c.

- GaimBuddyList är kompislistan i libGaim. Detta är en lista över alla grupper och kompisar hos alla konton som är uppkopplade. Varje konto har alltså inte sin egen kompislista utan allt ligger i en stor lista. GaimBuddyList refererar till den första noden i kompislistan, men innehåller även en hashtabell över alla kompisar. Funktionerna för att hantera kompislistan finns i blist.c.

- GaimPresence är en samling statusinformation för ett konto, en kompis, eller en chat-kontakt. Denna innehåller *antingen* en referens till ett konto, eller en konversation. Funktionerna för att hantera en eller flera presence's finns i status.c.

- GaimSavedStatus är statusen för alla konton, och representeras på disk av en XML-fil. LibGaim innehåller en lista av GaimSavedStatus:ar. I varje GaimSavedStatus finns en lista av understatusar, som representerar statusen för ett specifikt konto. GaimSavedStatus används för att styra alla kontons statusar, det vill säga att genom att anropa till exempel gaim_savedstatus_activate_for_account(status,account) leder detta till att det

nedkopplade kontot *account* koppas upp (förutsatt att *status* är en GaimSavedStatus med status *available*). Funktioner för att hantera en eller flera GaimSavedStatus:ar finns i savedstatuses.c.

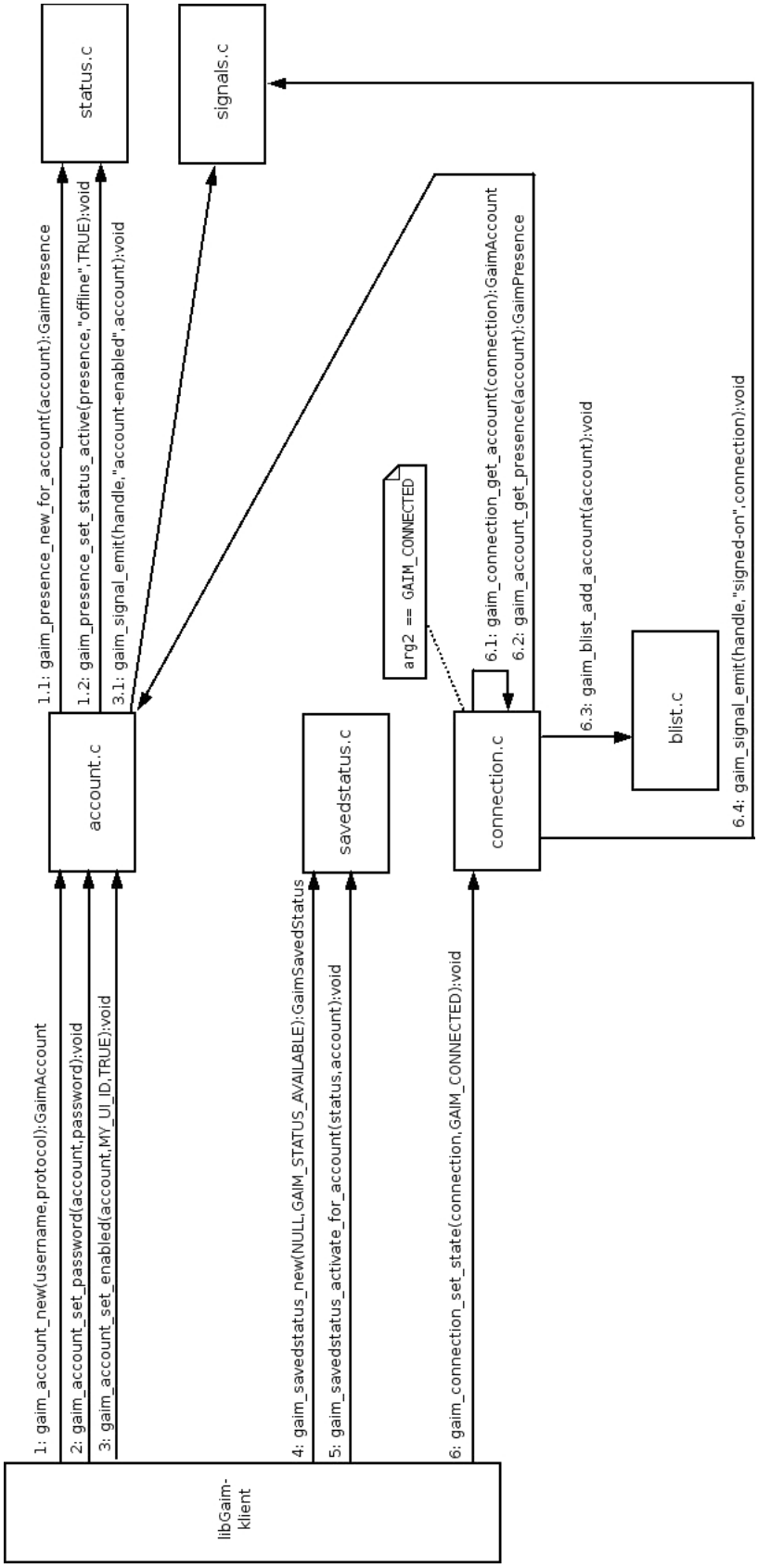
- signals.c är den fil som innehåller funktioner för signalhantering. En signal är bara en sträng som är kopplad till en funktionspekare, vilket gör att när funktionen gaim_signal_emit anropas sökes i en hashtabell efter rätt funktionspekare, och därefter anropas denna. Tack vare detta är det enkelt att ange vilka funktioner som ska ta hand om vilka anrop.

Exempel

För att ge ytterligare inblick i hur libGaim fungerar och hur man använder det följer här ett exempel på själva uppkopplingssekvensen. Kommunikationsdiagrammet över sekvensen visas i figur 5.2.

- 1) gaim_account_new(username, protocol) skapar ett lokalt konto.
- 1.1) gaim_presence_new_for_account(account) skapar en ny samling statusinformation för det nyligen skapade kontot.
- 1.2) gaim_presence_set_status_active(presence,“offline”,TRUE) aktiverar kontot.
- 2) gaim_account_set_password(account, password) sätter lösenordet för kontot.
- 3) gaim_account_set_enabled(account, MY_UI_ID, TRUE) sätter att kontot är klart för användning.
- 3.1) gaim_signal_emit(handle,“account-enabled”,account) skickar signalen att kontot är klart för användning.
- 4) gaim_savedstatus_new(NULL, GAIM_STATUS_AVAILABLE) skapar en ny status.

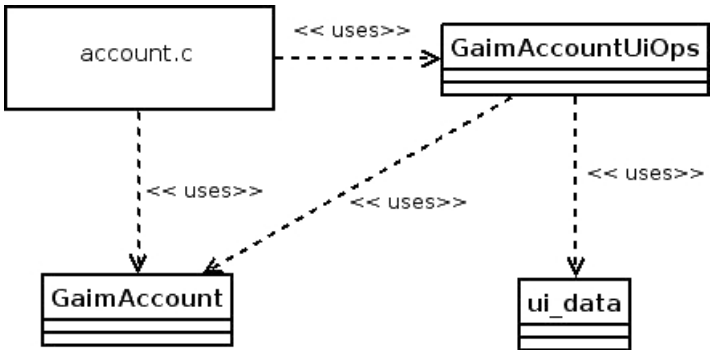
- 5) gaim_savedstatus_activate_for_account(status,account) sätter kontot att ha den tidigare skapade statusen.
- 6) gaim_connection_set_state(my_connection, GAIM_CONNECTED) kopplar upp kontot, eftersom den andra parametern är GAIM_CONNECTED.
- 6.1) gaim_connection_get_account(connection) hämtar det konto som har uppkopplingen *connection*.
- 6.2) gaim_account_get_presence(account) hämtar den presence som är kopplad till det nyligen hämtade kontot.
- 6.3) gaim_blist_add_account(account) lägger till kontot i kompislistan.
- 6.4) gaim_signal_emit(handle,"signed-on",connection) skickar ut signalen att kontot är uppkopplat.



Figur 5.2: Kommunikationsdiagram över uppkoppling med libGaim

Gränssnitt

För att separera logiken från användargränssnittet i libGaim har flera klasser en tillhörande gränssnittsklass som innehåller funktionspekare där det är meningen att gränssnittet definierar hur anrop till gränssnittet ska hanteras. Denna uppdelning visas i figur 5.3 som exemplifierar med GaimAccount.



Figur 5.3: libGaim's uppdelning av logik och gränssnitt

Funktionerna för att hantera logiken for konton är samlade i account.c, medan all data för ett konto samlas i en GaimAccount. Funktionerna för att hantera användargränssnittet ligger i GaimAccountUiOps, medan data för gränssnittet kan samlas i ui_data.

Det finns ett flertal problem med denna uppdelning av logik och gränssnitt. Logiken och gränssnittet är tämligen hårt kopplade trots uppdelningen, eftersom logiken har kännedom om gränssnittet, och för att det fortfarande krävs att ett gränssnitt existerar för att kunna utföra vissa operationer. Till exempel går det inte att hämta information om ett IM-konto utan att först koppla in ett gränssnitt i form av funktionspekare i gränssnittsklassen. Som exempel kan tas att den funktion som skickar ut signalen att kontoinformation tagits emot från nätverket först kontrollerar att gränssnittsfunktioner finns, och om de inte finns avbryts funktionen. Antagandet är här att om ett gränssnitt inte finns, finns ingen anledning att skicka ut informationssignalen eftersom användaren

ändå inte kan se den. LibGaim tycks vara uppbyggt med regeln att utan ett gränssnitt finns ingen användning för det. Med andra ord beror libGaim på gränssnittet, vilket gör det hårt kopplat.

Utvecklingen av libGaim går dock åt en vidare särdelning av logik och gränssnitt. Innan version 1.5 av Gaim användes till exempel klassen GaimConvWindow i libGaim för att lagra alla konversationer i. Denna GaimConvWindow representerade ett fönster med tabbar för olika konversationer, och därmed ett gränssnitt. Nu används istället bara en lista av konversationer som definieras i filen med logikfunktioner.

Kompislistan

Kompislistan är en lista över alla kompisar (representationer av andra IM-användares konton) till alla konton som skapats i libGaim. I figur 5.4 visas hur kompislistan är uppbyggd.

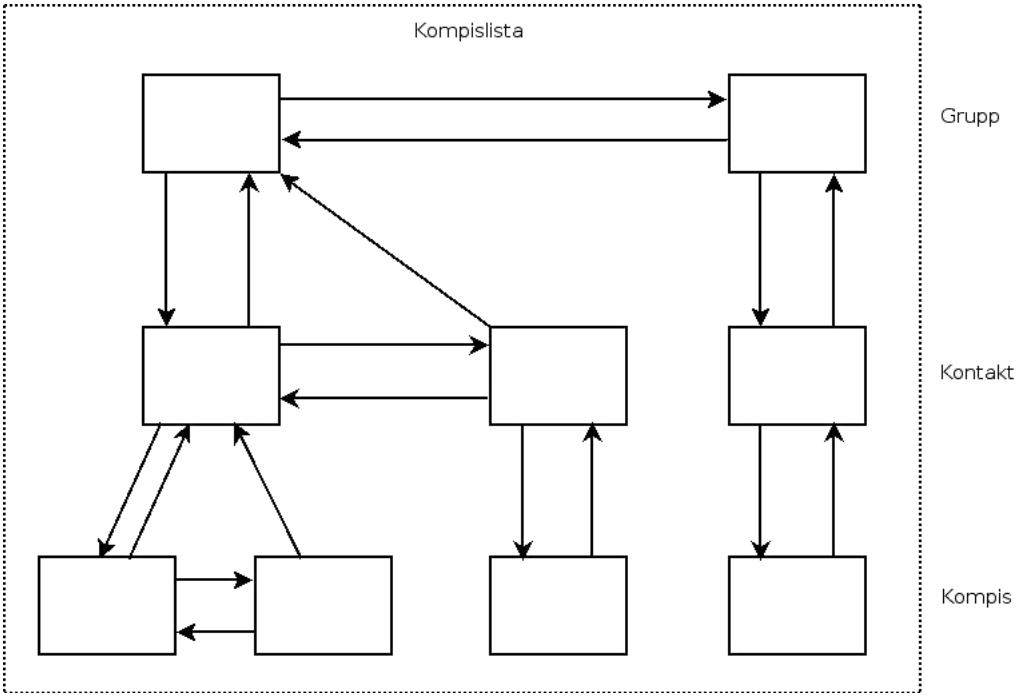
Den lägsta nivån är kompisar. En kompis är en IM-användare som kontot har möjlighet att skicka meddelanden till (om kontot fått rättighet att göra detta av kompisen).

Den andra nivån består av kontakter. En kontakt är en sorts "meta-kompis" som ger ett alias till en samling kompisar. På detta sätt kan en användare som har samma kompis i flera konton samla dessa i en enda kontakt.

Den översta nivån består av grupper. Grupper används för att låta användaren dela in kontakter i sektioner och ge dessa sektioner namn som till exempel "kompisar" eller "arbete".

5.2.2 Kompilering och länkning av libGaim

Som diskuterats i avsnitt 3.5.2 används Autotools till byggning av Gaim och libGaim. Gaim har färdigt stöd för att kompilera insticksmoduler till Gaim utan att behöva skriva en egen Makefile-fil. Detta utförs genom att placera sin C-fil, till exempel my_plugin.c, i



Figur 5.4: Strukturen av kompislistan

plugins-katalogen och exekvera kommandot

```
make my_plugin.so
```

När vi började detta arbete fanns inget färdigt stöd i Gaim för att kompilera en applikation som använder någon del av Gaim, men i andra hälften av arbetet med libGaim-wrappern skrev en av Gaim-utvecklarna en ny Makefile-fil för att kunna bygga applikationer som använder någon del av Gaim utanför Gaims filträd.

För att kompilera ett enkelt program som använder libGaim räcker det att använda en Makefile-fil som ser ut på följande sätt:

```
CC=gcc
CFLAGS=‘pkg-config --cflags gaim‘
LDFLAGS=‘pkg-config --libs gaim‘
```

```
all: my_client
```

Denna information, om hur man kompilerar en applikation som använder libGaim, fanns inte tillgänglig i någon dokumentation vi lyckades finna. Makefile-koden ovanför kommer från en av Gaim-utvecklarna som vi frågade på IRC. Inte långt efter våra utfrågningar skrevs den tidigare nämnda Makefile-filen för att kompilera libGaim-applikationer utanför Gaims filträd.

5.2.3 Kritik mot libGaim

Under utvecklingen har vi funnit att API-dokumentation många gånger snarare gjort utvecklingen svårare än lättare. Detta på grund av att dokumentationen varit felaktig och resultaten vi erhållit inte varit de väntade med avseende på dokumentationen. Dokumentationen i koden är dessutom sparsam och ofta irrelevant. De två källorna som varit den huvudsakliga hjälpen vid utvecklingen var källkoden för andra klienter som använder libGaim, och diskussioner med libGaim-utvecklarna själva. Dessa diskussioner har utförts över IRC[34].

libGaim-koden innehåller en hel del redundans. Samma exakta kod som undersöker samma sak utförs i flera funktioner. Om Gaim-utvecklarna använt kontraktsprogrammering hade detta inte varit ett problem, eftersom du i kontraktsprogrammering anger vad som måste gälla innan funktionen får anropas. Här används istället implicit svaga kontrakt som låter klienten anropa funktionen utan att några förvillkor är uppfyllda.

5.3 Öppen källkodsutveckling och Gaim

LibGaim som varit kärnan i systemet som utvecklats är en del av öppen källkods-projektet Gaim. Det är tack vare att det är ett öppen källkods-projekt som vi

kunnat ta del av källkoden, vilket varit en nödvändighet för att kunna använda libGaim då dokumentationen varit varken uppdaterad eller komplett.

Rollerna inom Gaim är enligt Sean Egan (huvudutvecklare i Gaim) uppdelat som en pyramid. Längst upp är huvudutvecklaren som tar de slutgiltiga besluten i stora frågor och till viss del styr projektet. På andra nivån är utvecklarna; de som har tillgång till att skriva in koden direkt i versionshanteringssystemet utan att någon annan granskar den först. På tredje nivån är de som relativt ofta skriver nya patchar till Gaim, och kallas inom Gaim för "Crazy patch writers". Dessa har inte rättigheter att skriva direkt till versionshanteringssystemet. Till sist på den lägsta nivån, som omfattar störst mängd personer, är de som skriver någon enstaka patch och sedan inte hörs av igen.

Dokumentation tycks inte vara en viktig del i Gaims utveckling. Till exempel nämns aldrig dokumentation i avsnittet om öppen källkod i *Open Source Messaging Application Development: Building and Extending Gaim*[10]. Hur systemet är strukturerat och vilka funktioner som använts till vad tycks till största del hållas vid liv inom utvecklingsgruppen i Gaim. Detta har lett till att motsvarigheten till att läsa dokumentationen för att få reda på hur systemet fungerar blir att fråga utvecklarna själva över IRC. Tillgängligheten av utvecklarna är dock inte alltid den bästa, och det är heller inte konstigt eftersom de arbetar med Gaim på fritiden och inte som ett arbete. Om utvecklarna hade arbetat med Gaim som ett arbete med lön och då på dagstid hade detta vidare försämrat tillgängligheten eftersom de flesta Gaim-utvecklare bor i USA.

Uppfattningen vi fått av Gaim-projektet är att det hålls fungerande till viss del tack vare att det är många som hjälper till att utveckla det. Även om koden inte är den mest välstrukturerade eller logiska, hålls den vid liv för att det finns många utvecklare som hjälper till att hitta felen. Trots att det finns designmässiga fel i koden, och att dokumentationen lämnar en hel del att önska, utvecklas libGaim hela tiden framåt tack vare den uppsjö av medverkande som finns. Vi har själva varit med och förbättrat dokumentationen för libGaim genom att uppdatera deras utvecklings-FAQ (Frequently

Asked Questions) med en beskrivning om hur konversationer fungerar och ett förtydligande av den API-dokumentation som finns tillgänglig.

5.4 Prestandatest av språkwrappers

5.4.1 Utförande

I detta avsnitt jämförs prestandan av att använda en språkstatisk språkwrapper mot att använda en språkgenerisk språkwrapper. Rimligtvis borde en språkstatisk språkwrapper utföra anrop till kärnan snabbare än en språkgenerisk språkwrapper, på grund av kostnaden av interprocesskommunikationen.

Målet med denna undersökning är att visa på skillnaden i tid för funktionsanrop med språkstatisk språkwrapper och språkgenerisk språkwrapper. En implementation av språkstatisk språkwrapper, och två implementationer av språkgeneriska språkwrappers testas. Dessutom utförs ett kontrolltest som inte använder någon wrapper utan utför direkta funktionsanrop.

Alla tester utförs mellan språken C och PHP, då C är ett systemspråk som ofta används till att skriva mjukvara som kräver hög prestanda, och PHP är ett högnivåspråk som ofta använder språkwrappers till C-bibliotek för att tillföra funktionalitet till PHP.

Alla testprogram anropar samma C-funktion genom olika medier. Innan funktionen anropas skrivs tidpunkten ut. Inuti C-funktionen skrivs tidpunkten ut, en uträkning utförs 10 miljoner gånger, och sedan skrivs tidpunkten ut precis innan funktionen returnerar till anroparen. När anroparen tar emot svaret – returvärdet – skrivs tidpunkten ut. Alla tester utfördes 1000 gånger.

Formeln som användes för att få fram ett tvåsidigt konfidensintervall med konfidensgrad 1-α är: $I_\mu = (\overline{x} \mp t_{\alpha/2}(n-1)\frac{s}{\sqrt{n}})$ där $\overline{x}$ är medelvärdet av stickprovet, $t_{\alpha/2}(n-1)$ är värdet som hämtas ur en t-fördelningstabell med n-1 frihetsgrader, n är antalet observationer i stickprovet, och s är en skattning av standardavvikelsen.

De fyra testerna är:

- Jämförelsetest – Ett test som används för jämförelse med vilka tider som erhållits om ingen språkwrapper använts. Detta består av ett C-program som gör ett direkt funktionsanrop till kärnan.

- PHP-extensiontest (språkstatisk språkwrapper) – Detta test bestod av en PHP-extension till kärnan som genererats med SWIG, och som användes av ett PHP-skript.

- IPC-wrappertest (språkgenerisk språkwrapper) – Detta test bestod av en server- och en klientdel. Serverdelen var ett C-program som skapar en meddelandekö och väntar sedan på meddelanden från kön. Klientdelen var ett PHP-skript som använde meddelandekön för att skicka in ett meddelande som sedan togs emot av servern. Servern kontrollerade att meddelandet var korrekt, anropade kärnan, och skickade tillbaks ett svar till klienten via meddelandekön.

- MySQL-wrappertest (språkgenerisk språkwrapper) – Detta test bestod av en server- och en klientdel. Serverdelen var ett C-program som kopplades upp mot en databas och gick sedan in i en loop där en SQL-fråga skickas till databasen i varje iteration. När en post påträffas i ToServer-tabellen kontrolleras att meddelandet är korrekt och i så fall anropas kärnan. Därefter läggs en svar-post in i FromServer-tabellen. Klientdelen bestod av ett PHP-skript som kopplade upp sig mot databasen och lade in en post i ToServer-tabellen. Därefter gick skriptet in i en loop där en SQL-fråga skickas till databasen i varje iteration, tills dess att en post påträffas i FromServer-tabellen.

Resultatet av testerna skrevs till fil med hjälp av > som är en shell output omdirigeringsoperator i linux. Detta sammanställdes sedan i xls-filer med hjälp av ett PHP-skript som skrevs för detta ändamål. Skriptet genererade – förutom de fyra

tidskolumnerna – två kolumner som anger skillnaden i tid från anrop till inne i kärnan, och från kärnan tillbaks till anroparen. Till dessa två kolumner togs medelvärde och 95% konfidensintervall fram. Tabellen över detta redovisas i nästa avsnitt.

5.4.2 Resultat

I tabellen nedanför visas 95% konfidensintervall för medelvärdena som erhållits ur testresultaten. Medelvärdet ligger med andra ord med 95% sannolikhet inom dessa intervall.

	95% Konfidensintervall för medelvärdet av tiden (sekunder)	
	Anrop till kärnan	Returanrop från kärnan
Jämförelsetest	[0.00002422, 0.00002482]	[0.00000786, 0.00000856]
PHP-extension	[0.00028542, 0.00028849]	[0.00001817, 0.00001841]
IPC	[0.00031953, 0.00038357]	[0.00013276, 0.00015802]
MySQL	[0.01452518, 0.01533468]	[0.02591840, 0.02879714]

5.4.3 Utvärdering

Från tabellen ovan kan vi uttyda att även om PHP-extension-testet (den språkstatiska språkwrappern) är cirka 10 gånger långsammare än jämförelsetestet vid anrop till kärnan, har det likt jämförelsetestet en betydligt kortare tid på returanropet från kärnan. Detta kan jämföras med de två testen med språkgeneriska språkwrappers, där tiden är relativt likadan i både anrop till kärnan och svaret tillbaks.

Utöver detta kan vi även se i tabellen att MySQL-testet är relativt mycket långsammare än alla övriga tester. IPC-testet är ungefär 100 gånger snabbare, och vad är då fördelarna med att använda en databas istället för en meddelandekö för interprocesskommunikation till språkgeneriska språkwrappers? En databas har en tydlig struktur, är lättestad, och informationen försvinner inte om datorn startas om. I den

språkwrapper som utvecklats som en del av detta arbete används en databas just av dessa anledningar.

Att välja en språkgenerisk språkwrapper över en språkstatisk behöver inte innebära någon större förlust i prestanda. Om svaret tillbaks efter inropet in till kärnan är oviktigt är PHP-extension-testet och IPC-testet relativt likvärdiga prestandamässigt.

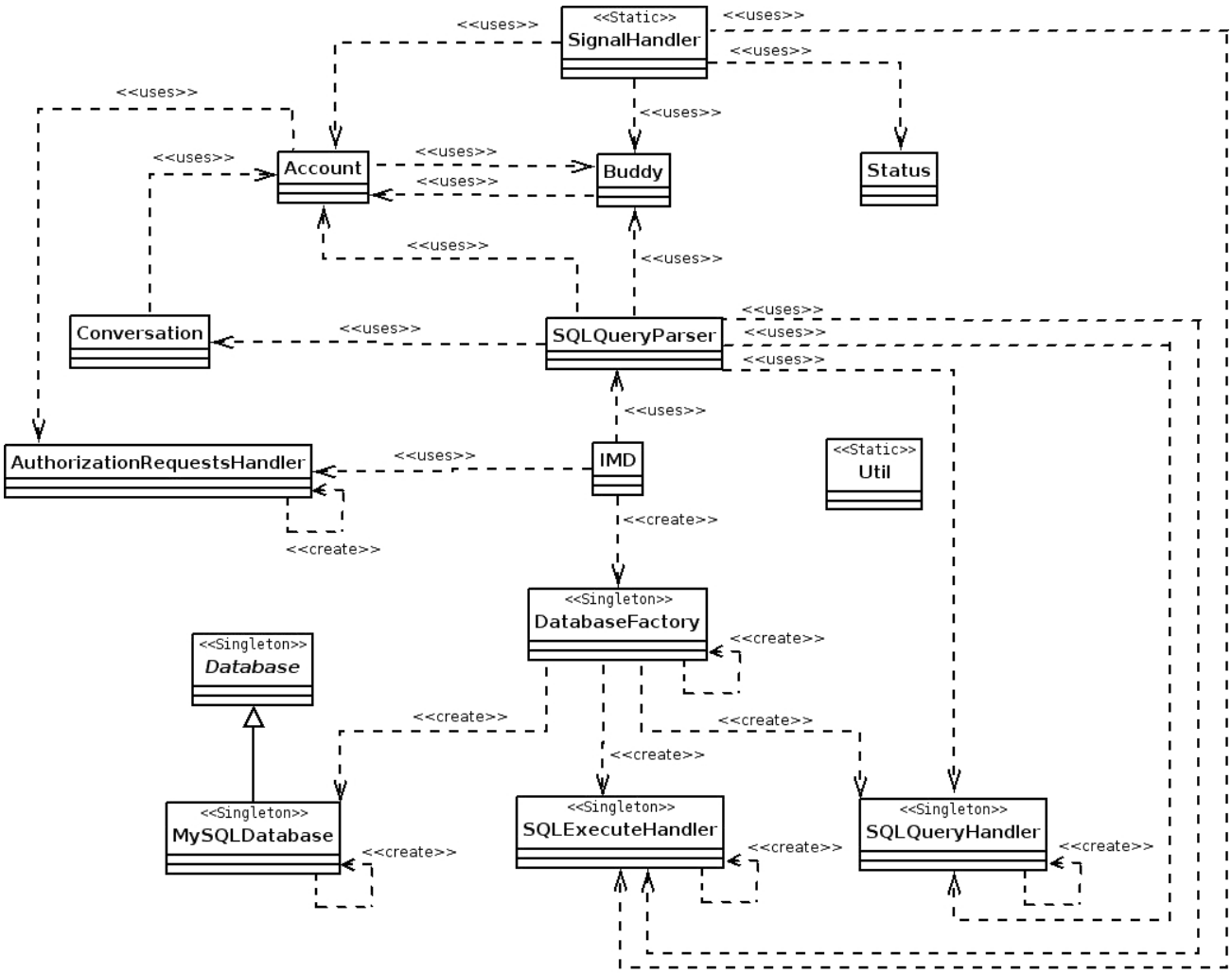
5.5 Design

5.5.1 Systemöversikt

Systemet som till slut utvecklades är uppdelat som i figur 5.5. Figuren visar klasserna i systemet, men utöver dem finns även två filer (LibGaim.h och LibGaim.cpp) som har en procedurell implementation för att enklare koppla samman libGaim, som är skrivet i C, med vårt system som är skrivet i C++.

- IMD är huvudklassen varifrån exekveringen startar. Här initieras libGaim och alla Factory-klasser. Här återfinns även programmets huvudloop som vid varje tidsintervall använder SQLQueryParser till att läsa från databasen.
- SQLQueryParser hanterar SQL-meddelanden och anropar korrekt funktion beroende på meddelandet.
- SQLQueryHandler hanterar hämtning av meddelanden från databasen.
- SQLExecuteHandler hanterar insättningar av meddelanden i databasen.
- DatabaseFactory hanterar skapandet av alla databashanteringsobjekt.
- Database är ett abstrakt databasgränssnitt. Alla databasgränssnitt måste ärva från denna klass.
- MySQLDatabase är ett gränssnitt mot en MySQL-databas.

- SignalHandler hanterar alla signaler från libGaim. Den kopplar in vilka funktioner som ska hantera vilka signaler, och tillhandahåller dessa funktioner.
- AuthorizationRequestsHandler hanterar inkommande tilläggningsförfrågningar, det vill säga när en annan användare skickar en förfrågning om denna får lägga till mottagaranvändaren i sin kompislista.
- Buddy är en kompis i en kompislista.
- Account är ett användarkonto.
- Status är en status för ett konto eller en kompis.
- Conversation är en koversation mellan ett konto och en kompis.



Figur 5.5: Klassdiagram över systemet

5.5.2 Meddelandehantering

Meddelanden mellan klienten och servern sker genom en MySQL-databas. Två huvudgrupper av meddelanden kunde identifieras: IM-meddelanden och Informationsmeddelanden. Ett IM-meddelande är meddelande som skickas mellan slutanvändare, och är antingen ett textmeddelande eller ett notifieringsmeddelande om att den andra användaren skriver eller har slutat skriva.

För att varje tabell skulle ansvara för en väldefinierad uppgift och för att få en tydligare struktur som dessutom är lättare att testa, delades varje meddelandetabell upp i en som hanterade meddelanden till servern, och en som hanterade meddelanden från servern.

Alla meddelanden är kopplade till ett konto, och varje konto har någon gång under sin livstid en kompislista.

Meddelandeutbytet mellan servern och klienten bygger på en mängd värden som anger vad som efterfrågas. För att exemplifiera hur detta utbyte av meddelanden kan se ut presenteras här en sekvens av meddelanden när klienten vill koppla upp ett konto (observera att punkt 5 delas in i två möjligheter: antingen lyckas uppkopplingen eller så lyckas den inte):

1. Klient) Lägger upp nytt konto
   Account: (<räknare>, “<screenname>”, “<protokoll>”, “<lösenord>”, “offline”)
   ex. resultat: (12, “my.email@snail.mail.com”, “prpl-msn”, “ettLösen”, “offline”)

2. Klient) Kopplar upp
   InfoToServer: (<räknare>, <konto id>, 0, “”, “”)
   ex. resultat: (74, 4, 0, “”, “”)

3. Server) Läser in meddelandet från InfoToAccount sammankopplat med Account.

4. Server) Kopplar upp kontot mot en IM-server.

5. Server) Uppkoppling lyckas!

   (a) Server) Tar bort meddelandet i InfoToAccount.

   (b) Server) Uppdaterar konto-status.
   Account: (<räknare>, "<screenname>", "<protokoll>", "<lösenord>", "available")
   ex. resultat: (ändras ej, ändras ej, ändras ej, ändras ej, "available").

   (c) Klient) Kontrollerar konto-status.

   (d) Server) Lägger upp buddylist.
   BuddyList: (<räknare>, <konto id>).
   ex. resultat: (47, 12).

   (e) Server) Lägger upp alla buddies.
   Buddy: (<räknare>, <buddy list id>, "<användarnamn>", "<alias>").
   ex. resultat: (49, 6, "dudes.email@snail.mail.com", "Pelle").

   (f) Server) Skicka buddylist-meddelande.
   InfoFromServer: (<räknare>, <konto id>, 3, "", "").
   ex. resultat: (284, 20, 3, "", "").

   (g) Klient) Tar emot meddelande om buddylist.

   (h) Klient) Hämtar ut alla buddies som hör till buddylist'en.

   (i) Klient) Tar bort alla buddies och buddylist'en, som hör till kontot, från databasen.

   Server) Uppkoppling misslyckas!

   (a) Server) Tar bort meddelandet i InfoToAccount.

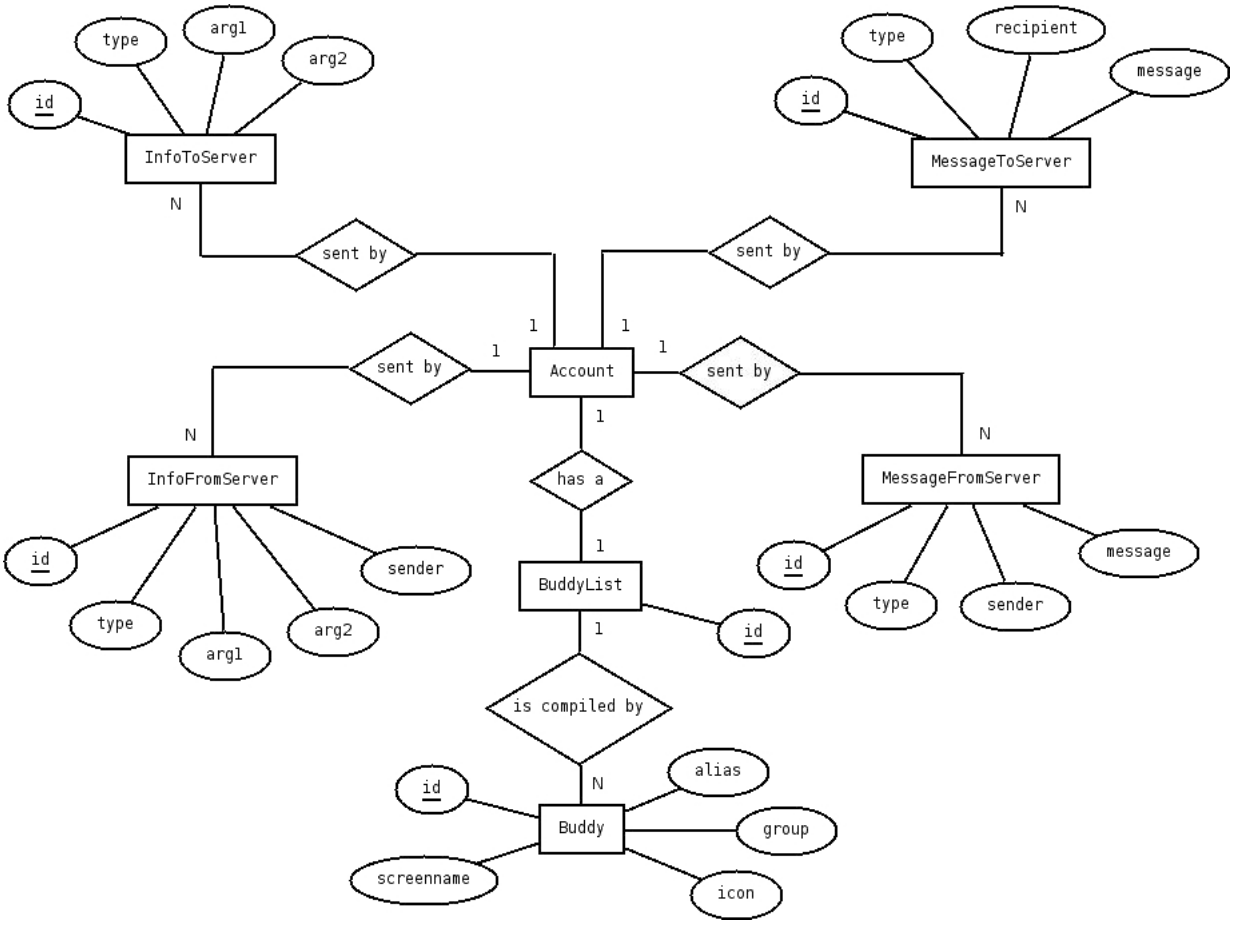
   (b) Server) Meddelar klienten om felet.
   InfoFromServer: (<räknare>, <konto id>, 4, "", "<felmeddelande>").
   ex. resultat: (283, 20, 4, "", "Här står ett felmeddelande").

(c) Klient) Hämtar ut meddelandet.

(d) Klient) Tar bort kontot från Account.

5.5.3 Databas

Databasen som använts till att skicka meddelanden mellan serverdelen och klientdelen i systemet har efter sin ursprungliga design inte genomgått några större förändringar, förutom nya tabeller och attribut som kommit till. I figur 5.6 visas det slutgiltiga E/R-diagrammet för databasen.



Figur 5.6: E/R-diagram

- InfoToServer är meddelanden från klientkonton (det vill säga konton som är uppkopplade mot systemet) till servern. Dessa meddelanden behandlar ärenden om kontot, dess status och dess kompisar. Ett exempel på ett sådan meddelande är ett meddelande som begär att koppla upp ett konto.

- InfoFromServer är meddelanden från servern till klientkonton. Dessa meddelanden är vanligtvis som svar på meddelanden som tidigare skickats in i InfoToServer, men kan även vara till exempel felmeddelande om att IM-servern, till exempel ICQ-servern, kopplat ned kontot.

- MessageToServer är meddelande som ska skickas från ett konto till en av kontots kompisar. Det finns tre typer av meddelanden: meddelande som anger att användaren skriver, meddelande som anger att användaren har slutat skriva, och ett vanligt textmeddelande.

- MessageFromServer är likadant som MessageToServer, men meddelandena går från externa användare till klientkonton istället för tvärtom.

- Account är konton.

- BuddyList är kompislistor som skapas då ett konto kopplats upp. Klienten har ansvaret att ta bort dessa när den hämtat informationen. Kompislistan är inte tänkt att ligga i databasen längre än nödvändigt.

- Buddy är kompisar i kompislistor som skapas samtidigt som kompislistorna.

5.6 Gaim blir Pidgin

Den 6 april skrev Sean Egan, Gaims huvudutvecklare, ett meddelande på Gaims hemsida. I meddelandet berättade Egan att på grund av fejden med AOL om namnet på Gaim, som lett till att AOL hotat med stämning, tvingas nu Gaim byta namn till Pidgin och

libGaim får namnet libPurple. Det system vi utvecklat migrerades inte till libPurple föränn i de sista två veckorna av utvecklingen, och rapporten hänvisar helt igenom till Gaim och libGaim istället för Pidgin och libPurple, då den utveckling som utförts har använt Gaim och libGaim.

5.7 Sammanfattning

I detta kapitel har libGaims struktur presenterats översiktligt med hjälp av klassdiagram, kommunikationsdiagram, figurer, förklarande text, och exempel. Viss kritik mot Gaim har lagts fram, bland annat angående dokumentationen som funnits vara ofta av låg kvalitet och i vissa fall missvisande.

Gaim-projektet är ett öppen källkods-projekt och en kort diskussion om detta och hur Gaim-projektet fungerar har presenterats.

Resultaten av de prestandatester av språkwrappers som utförts har presenterats. Sammanfattningsvis kan sägas att språkgeneriska språkwrappers kan vara relativt likvärdiga prestandamässigt med språkstatiska språkwrappers, då returanropet är oviktigt. Valet av medium i en språkgenerisk språkwrapper kan ha stor inverkan på prestandan, varför valet bör vara väl genomtänkt och motiverat.

Programmet som har utvecklats som en del av detta arbete har presenterats i form av klassdiagram, beskrivande text, och exempel. Gränssnittet till programmet har även presenterats i form av E/R-diagram med tillhörande förklaringar.

Kapitel 6

Utvärdering av libGaim-wrapper

6.1 Introduktion

I detta kapitel utvärderas utvecklingen av IM-systemet; motiveringar bakom val av lösning till problemet, val av design till lösningen, och vad resultatet blev och hur väl det uppfyllde förväntningar ställda på det.

6.2 Val av lösning

För att utveckla IM-klienten som var målet med arbetet, användes en lösning som gick ut på att skapa en språkgenerisk språkwrapper till IM-biblioteket libGaim. Motiveringen bakom valet av denna lösning står beskriven i kapitel 4. De två lösningarna som det till slut stod mellan var libGaim-lösningen och att skapa en Jabber-klient. Det som mest stärkte argumenten för att välja libGaim-lösningen var:

- Det går att skapa ett väl fungerande IM-system med libGaim på ett liknande sätt som vi försökte åstadkomma. Detta visste vi tack vare webbapplikationen Meebo som använder libGaim.

- Med Jabber lösningen läggs komplexiteten på klienten, vilket var oönskat då klienten kan komma att skrivas om i andra språk i framtiden.
- Arbetet med libGaim-lösningen hade redan påbörjats och varit igång några veckor.

Att välja en databas som interprocessmedium var ett lätt val med tanke på att det tillåter att systemet kraschar eller startas om utan att informationen går förlorad. Utöver denna, kanske viktigaste anledning, fanns flera andra:

- Struktur – Databaser är i sin natur välstrukturerade i det avseende att information delas in i tabeller som innehåller en eller flera kolumner, där innehållet kan referera till en kolumn i en annan tabell.
- Färdiga algoritmer – När en databas designats finns färdiga algoritmer för att få fram en bra struktur av designen och på en form som gör det lätt att föra in direkt i databassystemet.
- Lättestat – Genom något av mångfalden av databasgränssnitt går det enkelt att kontrollera innehållet i databasen och att lägga in och ta bort poster, vilket gör systemet lätt att testa.

6.3 Val av design

Systemet är uppbyggt med en objektorienterad design och tar i åtanke flera begrepp inom objektorienterad design, som cohesion, coupling, information hiding, och designmönster.

- **Cohesion** – Cohesion är ett begrepp som anger hur väl ett kodblock mappar mot en välspecificerad uppgift. Hög cohesion är önskvärt då detta ofta kan associeras med robusthet, tillförlitlighet, återanvändbarhet, och lättförståelighet.

  På grund av att designen är begränsad av libGaims design blir cohesion inte hög överallt i systemet. Exempelvis ligger en mängd funktioner som inte har något

annat gemensamt än att de används av libGaims signalsystem i klassen SignalHandler. Här finns exempelvis i denna klass en funktion som fångar upp signalen att kontoinformation för en kompis ska visas. Denna signal skickas ut av libGaim efter att man gjort en förfrågning om kontoinformation för en kompis. Samtidigt finns i denna klass även funktioner som hanterar inkomna IM-meddelanden.

- **Coupling** – Coupling är ett begrepp som anger hur hårt kopplade olika objekt är, det vill säga hur mycket de beror på varandra. Ju mindre kännedom en klass A har om en klass B, desto mindre blir graden coupling mellan dem. Låg coupling är önskvärt då låg coupling innebär att en ändring i en modul inte kräver ändring i en annan.

  Graden coupling i systemet har hållits på en låg nivå. Instansvariabler av klasser existerar inte någonstans i systemet, och klasser har endast kännedom om de klasser som det är rimligt att de bör ha kännedom om. Exempelvis har klassen SQLQueryParser (som hanterar alla inkomna meddelanden från databasen) kännedom om Account eftersom alla meddelanden arbetar med ett konto, men Account har ingen kännedom om SQLQueryParser.

  Inga praktiska undersökningar eller tester har utförts för att få fram ett värde på graden coupling i systemet. Att graden coupling är låg i detta system är den uppskattning vi gjort utifrån hur vi utvecklat systemet.

- **Information hiding** – Information hiding innefattar att gömma designval som kan komma att ändras från klienten. Detta för att klienten inte ska bero på implementationsdetaljer som kan ändras i framtiden. För att stödja detta krävs ett stabilt gränssnitt som gömmer implementationsdetaljer.

  Information hiding stöds i systemet genom att ett stabilt gränssnitt definierats och alla klassvariabler är privata. Eftersom libGaim är skrivet i C finns inga privata

variabler, men detta har löst genom att ange vilka variabler som ska betraktas som privata. Tyvärr saknas dock ett antal åtkomstfunktioner till privata variabler i libGaim, varför åtkomst av dessa måste ske direkt.

- **Designmönster** – Designmönster är mönster för lösningar på kända designproblem. Användning av designmönster kan förhindra framtida problem med systemet genom att designmönstret är vältestat. Det kan även öka läsbarheten av koden för de som känner till designmönstret.

  I systemet har designmönstrena Abstract Factory och Singleton använts.

  – Abstract Factory – Detta designmönster går ut på att enkapsulera en grupp Factory Methods som alla har ett gemensamt tema. Varje Factory Method är en funktion som skapar och returnerar ett visst objekt.

  – Singleton – Detta designmönster går ut på att begränsa mängden instanser av en klass. Alla singleton-klasser i vårt system existerar det bara en instans av.

  Vi har inte haft behov av några andra designmönster än dessa.

6.4 Resultat

Systemet vi utvecklat har fungerat väl i de tester vi utfört, men vid skrivandet av detta har klienten till systemet ej ännu utvecklats. Allt som finns specificerat i gränssnittsspecifikationen för systemet är dock testat, vilket gör att systemet kommer att fungera väl om denna specifikation följs av klienten. I de tester vi utfört har systemet visat sig vara både pålitligt, i den avseende att det utför de uppgifter som förväntas, och relativt snabbt. Inga prestanda-tester av systemet har utförts, men vid manuell testning (till viss del automatiserad på så vis att ett program använts för att lägga in en mängd olika sorters meddelanden i databasen) har systemet uppfört sig likvärdigt med andra IM-system. Mer specifikt:

- Uppkoppling av ett konto sker på några sekunder.
- Ett skickat meddelande är vanligtvis framme inom en sekund.
- Meddelanden från externa användare till användare av systemet tar likvärdigt lång tid som åt andra hållet.
- Om en kompis har en kompisikon kan denna hämtas.
- Kompisar är indelade i den struktur (grupper) som finns specificerat i kontot.

Kraven som ställdes på systemet i avsnitt 4.2 redovisas här, och huruvida systemet lever upp till dem:

1. **Krav:** Systemet som ska utvecklas är en Instant Messaging-klient som gör det möjligt att använda Instant Messaging-tjänster med ett skriptspråk, huvudsakligen PHP, men det är fördelaktigt om det även går att använda andra skriptspråk.

   Lösning: Systemet som har utvecklats är en IM-klient som låter klienter i alla språk med stöd för MySQL använda IM-tjänster. Krav uppfyllt.

2. **Krav:** IM-klienten ska arbeta i bakgrunden, det vill säga vara en *daemon*, och klara av att hantera ett stort antal användare på samma gång.

   Lösning: Systemet arbetar i bakgrunden genom användandet av ett init.d-skript. Vi har inte utfört några tester med ett större antal användare, men Meebo använder libGaim och klarar av att hantera ett stort antal användare (80 miljoner meddelanden per dag enligt dem själva). Vi ser ingen direkt anledning till varför vårt system inte skulle klara av en stor mängd användare, men det är inget vi kan garantera i nuläget. Vi skulle dock rekommendera att vidare testning utförs (vilket kommer utföras av Evolve) innan systemet sätts i drift. Krav delvis uppfyllda.

3. **Krav:** IM-protokollet MSNP[59] ska stödjas av systemet. Andra protokoll har inte lika hög prioritet men det vore önskvärt med stöd för Jabber[21] och OSCAR[60], och även andra IM-protokoll.

   Lösning: MSNP stöds av systemet. Övriga protokoll är nästan inte alls testade men kan vidareutvecklas i framtiden. Krav delvis uppfyllt.

4. **Krav:** Skulle systemet krascha av någon anledning ska det gå att starta om det utan att detta märks i klienten.

   Lösning: Om systemet kraschar påverkar inte detta databasen, och när systemet startas igen hanterar det de konton och meddelanden som finns i databasen utan att klienten blir medveten om detta. Alla konton som var uppkopplade innan systemet stängdes ner kopplas upp igen. Detta är testat ett flertal gånger och fungerar. Krav uppfyllt.

5. **Krav:** Det ska inte gå att krascha systemet eller utnyttja det för att få ut känslig information genom att en användare skickar in ett IM-meddelande innehållandes speciellt utformad text.

   Lösning: Systemet har skydd mot SQL-injections. Krav uppfyllt.

6. **Krav:** Interfacet till IM-klienten ska vara enkelt; det vill säga att komplexiteten inte ska ligga hos klienten, utifall att klienten kan behövas skrivas om i framtiden.

   Lösning: Systemet kräver väldigt lite av klienten. Klienten behöver endast skicka in högnivå-anrop, som till exempel att koppla upp konto x, för att utföra operationer. Krav uppfyllt.

7. **Krav:** Systemet ska fungera i operativsystemet Linux.

   Lösning: Systemet fungerar i Linux. Testat i Ubuntu Linux och CentOS Linux. Krav uppfyllt.

6.5 Sammanfattning

I detta kapitel har valet av lösning diskuterats. Dessutom har designen av systemet utvärderats.

Kraven som ställdes på systemet har jämförts med det slutgiltiga resultatet och det kan konstateras att alla krav är helt eller delvis uppfyllda. Det finns i nuläget ingen garanti för att systemet fungerar väl med ett stort antal användare, och det enda prokoll som testats utförligt är MSNP. MSNP-protokollet är det som vi fokuserat på och var även det som var högst prioriterat. ICQ och Jabber är sparsamt testade men har fungerat i de tester vi utfört.

Slutresultatet blev ett fungerande system som uppfyller kraven ställda.

Kapitel 7

Sammanfattning och slutsats

7.1 Introduktion

I detta kapitel sammanfattas projektet och de slutsatser som dragits från det. Dessutom utvärderar vi vår insats och vad som varit tidskrävande under projektet.

7.2 Utvärdering

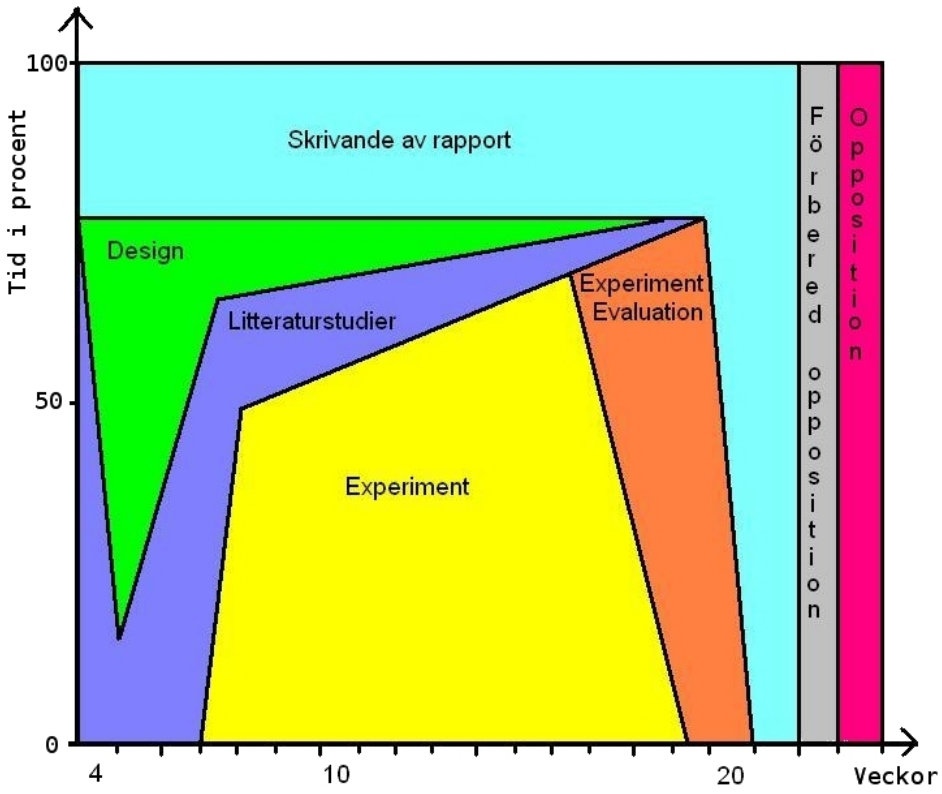
I detta projekt har vi utvecklat en språkwrapper tillika IM-klient till biblioteket libGaim. Kopplat till detta har vi utfört en undersökning av språkwrappers. Denna undesökning av språkwrappers har innefattat ett stickprov av hur begreppet wrapper används inom olika områden av datavetenskapen, en klassificering av begreppet språkwrapper i de två typerna språkstatisk och språkgenerisk, och ett antal tester för att visa på relativa skillnader mellan dessa två typer av språkwrappers och vad det får för konsekvenser.

Att definiera begreppet wrapper var ett åtagande som tog oss längre tid än vad vi ursprungligen väntat oss. Avsaknaden av en standardiserad definition inom datavetenskapen, och den stora mängd egna definitioner som kunde finnas i vetenskapliga rapporter, webbuppslagsverk, böcker och företagssidor gjorde det svårt att koppla

samman en fungerande generell definition. Att finna ett ursprung i begreppet visade sig vara ännu svårare, och vi fann ingen riktigt klar bild av var begreppet tog sin början inom datavetenskapen.

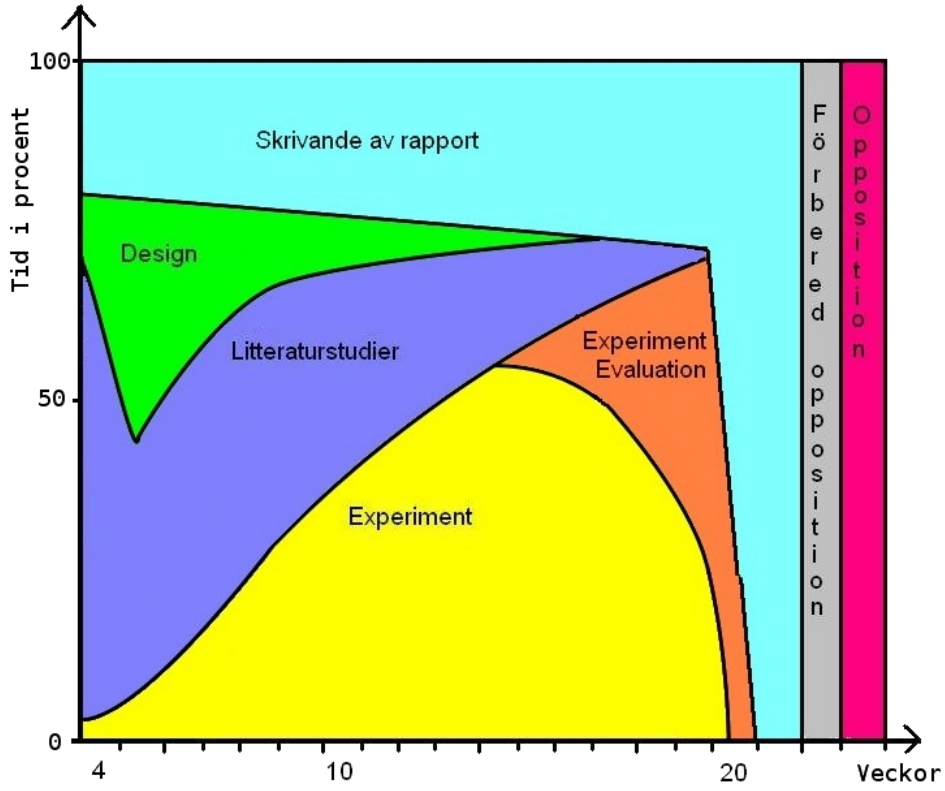
På det hela har detta projekt fungerat mycket väl. De problem vi stött på, både i form av implementation och att finna och förstå korrekt information har lösts i slutändan. En stor del tid har tagits upp av inlärning av nya verktyg (bland annat Autotools och SCons) och språk (SCons använder python).

I figur 7.1 visas ett diagram över den tidsplan som gjordes i början av projektet. Den faktiska tiden som arbetet tog upp visas i figur 7.2.



Figur 7.1: Tidsplanen i början av projektet

I ett jämförande av dessa två figurer kan man se att litteraturstudierna tog upp



Figur 7.2: Den faktiska tidsplanen som blev resultatet

mycket mer tid än vad som ursprungligen planerats. Detta har sin största anledning i att libGaim inte var så väldokumenterat som man hade önskat. Uppskattningsvis skulle utvecklingstiden kunna kortats ner med cirka 20% om libGaim haft en välarbetad dokumentation och kanske till och med övriga hjälpfiler för att sätta sig in i hur libGaim fungerar. De delar som libGaim skulle behövt förbättra i sin dokumentation är:

- Överblick – Överblickande diagram och text för att förklara hur biblioteket hänger ihop.

- Komplett moduldokumentation – Korrekta beskrivningar av alla funktioner och parametrar. Dessutom vore för- och eftervillkor önskvärt.

- Exempel/handledning – Förklarande exempel som visar hur man utvecklar till exempel en väldigt enkel klient till libGaim vore väldigt bra att ha i början av utveckling.

Även om det kan ha sin charm att diskutera hur saker fungerar över IRC med de som verkligen implementerat det, var detta ofta inte möjligt då Gaim-utvecklarna av förståliga anledningar inte var online dygnet runt. Utan denna möjlighet till konversation med Gaim-utvecklarna hade arbetet inte blivit så pass färdigt som det nu blev, då det uppfyller alla krav ställda (två av kraven endast delvis).

Kanske den viktigaste lärdomen från detta projekt är angående dokumentation. Den som tänker sig att utveckla något liknande system som vi gjort måste vara förberedd på att en stor del av tiden, kanske uppåt hälften till och med, kan gå åt till att studera koden i det fall dokumentation saknas eller håller låg kvalitet. Detta bör därför vägas in i bedömningen om den lösning man valt är den rätta. Det är vår uppfattning att knapphändig och bristande dokumentation ofta är ett problem inom öppen källkods-projekt, i varje fall då projekten inte är så stora. Men ett större öppen källkods-projekt är ingen garanti för bra dokumentation, då Gaim-projektet är ett av de större.

I detta projekt har vi även skapat en sammanställning av wrapper-definitioner och försökt skapa en definition som innefattar dessa på bästa sätt. Denna sorts sammanställning av begreppet har vi inte funnit i någon annan litteratur och den kan därför komma att bli användbar för den som söker klassificera mjukvarokomponenter.

Dessutom har vi gett en överblick av språkwrappers och visat på skillnader i prestanda. Våra tester visade att en språkgenerisk wrapper inte behöver betyda någon större förlust av prestanda gentemot en språkstatisk. Samtidigt kan prestandan sjunka relativt mycket vid användandet av språkgeneriska språkwrappers om man till exempel använder en databas som medium.

Ur detta projekt tar vi med oss ett flertal erfarenheter och lärdomar. Utöver de nya

verktyg och språk som vi lärt oss använda, och den kunskap vi samlat på oss som en del av att skriva denna rapport, har vi även fått en inblick i hur det är att arbeta på ett företag ute i verkligheten.

7.3 Framtida arbete

7.3.1 libGaim-wrappern

- Stöd för fler protokoll än MSNP. Även om till exempel ICQ sannolikt fungerar i de flesta avseenden har det inte testats sedan början av arbetet.

- Paketering. Vår slutprodukt är koden, dokumentation, och byggfiler för att kompilera det. För att göra det enklare att installera systemet skulle någon form av paketering behövas som till exempel en deb- eller rpm-fil.

- Stöd för chat. För tillfället stöds endast IM-meddelanden, det vill säga meddelanden som skickas från konto A till konto B. Med chat skulle meddelanden istället kunna skickas till ett chat-rum där flera användare kan läsa det.

  MSNP-protokollet har stöd för chat. För att lägga till detta stöd i systemet krävs en ny typ av meddelande i meddelandetabellen i databasen, och ett antal nya informationsmeddelandetyper som *starta chat med kompis* och *lägg till kompis till chat* för att skicka chat-förfrågningar till fler kompisar. Uppskattningsvis bör det kunna gå att implementera detta i systemet på fem veckor heltid.

- Vi har inte utfört några tester där en stor mängd användare använder systemet, vilket bör göras om det ska komma att användas av ett stort antal användare.

- Från prestandatesterna har vi kunnat konstatera att MySQL som gränssnitt för språkwrappers är betydligt långsammare än bland annat meddelandenköer. För att öka prestandan av systemet vore det då önskvärt att utföra tester för att se vad hos

MySQL som gör att prestandan sjunker så kraftigt. Kanske kunde man använda enbart filer på disk, vilket då inte för med sig den eventuella overhead som MySQL-systemet för med sig, och jämföra prestandan mellan MySQL och detta lösningsförslag.

7.4 Sammanfattning

I detta kapitel har projektet utvärderats med avseende på den planerade tidsplanen jämfört med den faktiska tiden. Den största skillnaden mellan den planerade tiden och hur mycket tid varje moment tog upp i slutändan var litteraturstudier, där informationsinsamling och inlärning av hur libGaim fungerar var en stor del.

Förslag på framtida arbete med libGaim-wrappern har tagits upp, och däribland en utbyggning för att ge möjlighet till chatter.

Referenser

[1] Barnes, David J. & Kölling, Michael. (2006). *Objects First with Java.* Prentice Hall. Harlow, England.

[2] Beasley, David M. et al.. *Perl Extension Building with SWIG.* At the O'Reilly Perl Conference 2.0, August 1998.

[3] Beck, Kent, & Andres, Cynthia. (2004, November). *Extreme Programming Explained – Embrace Change.* Stoughton, Massachusetts. Addison-Wesley.

[4] Blinch, Steve. (2005, December 13). *Blitzaffe Code - PHP Classes - Jabber Client.* (Elektronisk).
Tillgänglig:<http://code.blitzaffe.com/pages/phpclasses/files/jabber_client_52-11>.
(2007-03-27).

[5] Brant, John et al.. *Wrappers to the rescue.* In Proceedings of the 12th European Conference on Object-Oriented Programming, July 1998.

[6] Chidlovskii, Boris et al.. *Wrapper Generation via Grammar Induction.* In ECML 2000, 2000.

[7] Constantine, Larry. (2001, Juni 29). *Management Forum: Hiring the Best.* (Elektronisk). Tillgänglig:<http://www.ddj.com/dept/architect/184415690>. (2007-03-02).

[8] Converse, Tim et al.. (2004). *PHP and MySQL Bible.* Indianapolis, Indiana. Wiley Publishing, inc.

[9] Cunningham & Cunningham Inc. (2007). *History of Patterns.* (Elektronisk). Tillgänglig:<http://c2.com/cgi-bin/wiki?HistoryOfPatterns>. (2007-03-01).

[10] Egan, Sean. (2005). *Open Source Messaging Application Development: Building and Extending Gaim.* Berkeley, CA. Apress.

[11] Epicware och utvecklarna av Fire. (2007). *Fire: A Multi-Protocol IM Client For OS X*. (Elektronisk). Tillgänglig:<http://fire.sourceforge.net>. (2007-03-01).

[12] FOLDOC. (1998-12-15). *Wrapper*. (Elektronisk) Tillgänglig:<http://foldoc.org/index.cgi?query=wrapper&action=Search>. (2007-02-27).

[13] Free Software Foundation, Inc. (2006, November 20). *GNU M4 1.4.9 macro processor*. (Elektronisk). Tillgänglig:<http://www.gnu.org/software/m4/manual/m4.html>. (2007-03-13).

[14] Gamma, Erich et al.. (2005). *Design Patterns – Elements of Reusable Object Oriented Software*. Indianapolis. Addison-Wesley.

[15] Greene, Derek, & O'Mahoney, Donal. *Instant Messaging & Precence Management in Mobile Ad-Hoc Networks*. In Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004.

[16] Grinter, Rebecca E. & Palen, Leysia. *Instant Messaging in Teen Life*. In Proceedings of CSCW '02, New Orleans, LA, 2002.

[17] van Heesch, Dimitri. (2007). *Doxygen Manual*. (Elektronisk). Tillgänglig:<http://www.stack.nl/ dimitri/doxygen/manual.html>. (2007-03-13).

[18] Henstridge, James et al. *Linux man pages for pkg-config*. (Elektronisk). Tillgänglig:<http://www.die.net/doc/linux/man/man1/pkg-config.1.html>. (2007-04-02).

[19] Holt, Gary. (2007). *matrap – Wrap C++ function/classes for various matrix languages*. (Elektronisk). Tillgänglig:<http://lnc.usc.edu/ holt/matwrap/matwrap.html>. (2007-03-01).

[20] Indiatimes News Network. (2007, February 06). *Meet China's Google trouncer*. (Elektronisk) Tillgänglig:<http://infotech.indiatimes.com/Meet_Chinas_Google_trouncer/articleshow/1568771.cms>. (2007-02-20).

[21] Jabber, Inc. (2007). *Jabber: Open Instant Messaging and a Whole Lot More, Powered by XMPP*. (Elektronisk). Tillgänglig:<http://www.jabber.org>. (2007-03-01).

[22] John, George H. et al.. *Irrelevant Features and the Subset Selection Problem*. In Proceedings of the Eleventh International Conference on Machine Learning, San Francisco, CA.

[23] Jupitermedia Corporation (2007). *What is a wrapper?*. (Elektronisk) Tillgänglig:<http://www.webopedia.com/TERM/W/wrapper.html>. (2007-02-09).

[24] Kerner, Sean Michael. (2005, Maj 13). *Mark Spencer, President, Digium.* (Elektronisk). Tillgänglig:<http://www.internetnews.com/dev-news/article.php/3504931>. (2007-03-03).

[25] Knight, Steven, & Roach, Anthony. (2005, December). *Linux man pages for scons.* (Elektronisk). Tillgänglig:<http://www.die.net/doc/linux/man/man1/scons.1.html>. (2007-04-02).

[26] Kushmerick, Nicholas. *Wrapper Induction for Information Extraction.* A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, University of Washington, 1997.

[27] LinuxQuestions.org. (2004, September 24). *Interview with gaim Maintainer Rob Flynn.* (Elektronisk). Tillgänglig:<http://www.linuxquestions.org/questions/showthread.php?t=234803>. (2007-03-03).

[28] McMillan, Robert. (2001, November 15). *The Messenger: An Interview with Jabber's Creator, Jeremie Miller.* (Elektronisk). Tillgänglig:<http://66.102.9.104/search?q=cache:8e5ZA3eSavEJ:www.linux-mag.com/id/902/+aol+blocks+jabber&hl=en&ct=clnk&cd=20>. (2007-04-02).

[29] Meyer, Bertrand. (1997, April). *Object-Oriented Software Construction.* Prentice Hall Ptr. New Jersey.

[30] Morse, Dennis et al. (1999). *Linux man pages for make.* (Elektronisk). Tillgänglig:<http://www.die.net/doc/linux/man/man1/make.1.html>. (2007-04-02).

[31] MySQL AB. (2007). *MySQL AB :: Download Connector/PHP.* (Elektronisk). Tillgänglig: <http://dev.mysql.com/downloads/connector/php/>. (2007-04-16).

[32] MySQL AB. (2007). *MySQL 5.1 Reference Manual.* (Elektronisk). Tillgänglig:<http://dev.mysql.com/doc/refman/5.1/en/index.html>. (2007-03-13).

[33] Nykvist, Conny & Andersson, Mikael. (2005). *Utökning av PHP.* Karlstad Universitet, 2005.

[34] Oikarinen, Jarkko. (2007). *IRC History by Jarkko Oikarinen.* (Elektronisk). Tillgänglig:<http://www.irc.org/history_docs/jarkko.html>. (2007-03-01).

[35] Papakonstantinou, Yannis et al.. *A query translation scheme for rapid implementation of wrappers*. In 4th DOOD, Singapore, December 1995.

[36] Pfaf, Ben. *Linux man pages for aclocal*. (Elektronisk). Tillgänglig:<http://www.annodex.net/cgi-bin/man/man2html?1+aclocal>. (2007-04-02).

[37] PostgreSQL Global Development Group. (2007). *PostgreSQL Documentation*. (Elektronisk). Tillgänglig:<http://www.postgresql.org/docs/>. (2007-03-13).

[38] Jabber Software Foundation. (2004, October). *RFC 3920 – Extensible Messaging and Presence Protocol (XMPP): Core*. (Elektronisk) Tillgänglig:<http://www.ietf.org/rfc/rfc3920.txt>. (2007-02-20).

[39] Sewell, Peter, & Vitek, Jan. *Secure Composition of Untrusted Code: Wrappers and Causality Types*. In Proc. IEEE Computer Security Foundations Workshop, July 2000.

[40] Snellman, Ronnie. (2004, Oktober). *XML-RPC över Jabber*. Department of Computer Science, Åbo Akademi University. Åbo, Finland.

[41] Spencer, Mark. (2001, Juni 25). *More Trouble With AOL And GAIM*. (Elektronisk). Tillgänglig:<http://yro.slashdot.org/article.pl?sid=01/06/25/2115200>. (2007-03-03).

[42] Stogov, Dmitry. (2004). *Using Perl Code from PHP*. (Elektronisk) Tillgänglig:<http://www.zend.com/php5/articles/php5-perl.php>. (2007-02-14).

[43] SWIG community. (2007). *SWIG documentation*. (Elektronisk). Tillgänglig:<http://www.swig.org/doc.html>. (2007-03-01).

[44] The PHP Group. (2007, Mars 26). *Package Information: CodeGen_PECL*. (Elektronisk). Tillgänglig:<http://pear.php.net/package/codegen_pecl/>. (2007-03-26).

[45] The PHP Group. (2007, April 16). *PHP: Image Functions - Manual*. (Elektronisk). Tillgänglig:<http://se.php.net/gd>. (2007-04-16).

[46] The PHP Group. (2007). *PEAR - PHP Extension and Application Repository*. (Elektronisk). Tillgänglig:<http://pear.php.net/>. (2007-03-20).

[47] Voas, Jeffrey, & Payne, Jeffrey. *COTS Software Failures: Can Anything be done?*. In IEEE Workshop on Application specific Software Engineering and Technology. 1998.

[48] Walleij, Linus. (2004). *Att använda GNU/Linux*. Lund, Studentlitteratur.

[49] Ware, Mike. (2003, Augusti). *Background of Computer Programming.* (Elektronisk). Tillgänglig:<http://www.developerfusion.co.uk/show/3653/>. (2007-03-02).

[50] Wengo. (2007). *WengoPhone – call, talk, chat and share for free with anyone.* (Elektronisk). Tillgänglig:<http://openwengo.com>. (2007-03-01).

[51] whatis.com (2007). *What is a wrapper?.* (Elektronisk). Tillgänglig:<http://searchwebservices.techtarget.com/sDefinition/0,290660,sid26_gci213388,00.html>. (2007-02-27).

[52] Wikimedia Foundation, Inc. (2007, Mars 31). *AOL Instant Messenger.* (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/AOL_Instant_Messenger>. (2007-04-02).

[53] Wikimedia Foundation, Inc. (2007, Mars 30). *AOL.* (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/Aol>. (2007-04-02).

[54] Wikimedia Foundation, Inc. (2007, Mars 5). *Concurrent Versions System.* (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/Concurrent_Versions_System>. (2007-03-13).

[55] Wikimedia Foundation, Inc. (2007, Mars 30). *HTML.* (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/Html>. (2007-04-02).

[56] Wikimedia Foundation, Inc. (2007, Februari 28). *ICQ.* (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/ICQ>. (2007-03-01).

[57] Wikimedia Foundation, Inc. (2007, Februari 5). *Internet Engineering Task Force.* (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/IETF>. (2007-03-01).

[58] Wikimedia Foundation, Inc. (2007, Mars 28). *LaTeX.* (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/LaTeX>. (2007-04-02).

[59] Wikimedia Foundation, Inc. (2007, Februari 13). *Microsoft Notification Protocol.* (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/MSNP>. (2007-03-01).

[60] Wikimedia Foundation, Inc. (2007, Februari 10). *OSCAR protocol.* (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/OSCAR_protocol>. (2007-03-01).

[61] Wikimedia Foundation, Inc. (2007, Mars 26). *Portable Document Format.* (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/Pdf>. (2007-04-02).

[62] Wikimedia Foundatin, Inc. (2007, March 5). *Subversion (software).* (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/Subversion_%28software%29>. (2007-03-06).

[63] Wikimedia Foundation, Inc. (2007, Februari 21). *Wrapper*. (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/Wrapper>. (2007-02-27).

[64] Wikimedia Foundation, Inc. (2007, Mars 25). *XML*. (Elektronisk). Tillgänglig:<http://en.wikipedia.org/wiki/XML>. (2007-03-27).

[65] Zend Technologies Ltd. (2007). *Zend Engine - In Depth*. (Elektronisk). Tillgänglig:<http://www.zend.com/products/zend_engine/in_depth>. (2007-03-26).

Bilaga A

Definitioner och förkortningar

A.1 Defintioner

Cohesion – Hur väl en kodmodul mappar mot en välspecificerad uppgift.

Coupling – Hur hårt sammankopplade två mjukvarumoduler är.

Design by Contract – En metodik för mjukvaruutveckling som innebär att alla mjukvarogränssnitt ska ha specifikationer i form av kontrakt och invarianter.

Designmönster – Mönster för lösningar på kända designproblem.

Filter – Kod som hanterar en ström av data, exempelvis en funktion som gör om en ström text från ett språk till ett annat.

Gaim – En IM-klient som använder biblioteket libGaim (dessa har nu bytt namn till Pidgin och libPurple).

Information hiding – En utvecklingsstrategi som innefattar att gömma designval som kan komma att ändras i framtiden. Detta för att klienten inte ska bero på implementationsdetaljer.

Instant Messaging – En typ av kommunikationssystem som innebär synkron kommunikation i textform mellan en eller flera användare.

Kompis – En kompis är en representation av en IM-användare och används i kompislistor i de flesta IM-klienter.

Meebo – En webbaserad IM-klient som använder libGaim och som enligt företaget själva hanterar 80 miljoner meddelanden om dagen.

PHP – Ett skriptspråk som vanligen används till att utveckla webbapplikationer.

SCons – Ett byggverktyg som kan jämföras med Autotools.

Språkwrapper – En wrapper vars huvudsakliga syfte är att möjliggöra användning av wrapperns kärna i ett språk som skiljer sig från kärnans.

Subversion – Versionshanteringssystem likt CVS.

Tcl – Ett skriptspråk från 1988 som vanligen används till prototyper och testning.

Wrapper – En wrapper är en mjukvarumodul, det vill säga kod, som använder en annan mjukvarumodul i syfe att antingen utvidga denna, öka kompatibiliteten utåt, förstärka säkerheten, öka enkelheten, eller addera en vy, utan att den nya mjukvarumodulen (wrappern + det som wrappas) avviker för mycket från syftet hos det som wrappas. Wrappern begränsar hur kärnan (det som wrappas) hanteras, och hur omgivningen hanterar den.

Öppen källkod – eng. Open Source. Innebär att källkoden är tillgänglig för att läsas, modifieras, och vidaredistribueras. Begreppet förknippas även med en utvecklingsprocess där ett flertal individer arbetar på samma projekt där koden är öppen och de medverkande ofta håller kontakt över internet.

A.2 Förkortningar

COTS – Commercial off-the-shelf.

IETF –Internet Engineering Task Force.

IM – Instant Messaging.

PEAR – PHP Extension and Application Repository.

PERL – Practical Extraction and Report Language.

PHP – PHP: Hypertext Preprocessor

SVN – Subversion.

SWIG – Simplified Wrapper and Interface Generator.

XP – eXtreme Programming.