Datavetenskap

Ingela Pettersson Lindqvist

Utredning av Server för hantering av Felmeddelanden och Åtgärdsförslag

---

Examensarbete, C-nivå

2001:28

Investigation concerning a Server to handle Error messages and Recovery Solutions

Ingela Pettersson Lindqvist

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

Ingela Pettersson Lindqvist

Godkänd 2001-10-03

Handledare: Hannes Persson

Examinator: Stefan Lindskog

Sammanfattning

Detta arbete har bestått i att utveckla ett koncept, som är avsett att utmynna i en ny standard. I konceptet behandlas hur felmeddelanden och förslag på åtgärder till fel, kan presenteras för en datoranvändare. I detta arbete ingick även att arbeta fram en prototyp till en modul som skall ta emot felkoder, samt returnera åtgärdsförslag. Allt enligt klient-servermodellen.

Att realisera modulprototypen har inneburit beredningsarbete genom intervjuer, och att genom dessa arbeta fram en kravspecifikation. Utifrån kravspecifikationen valdes sedan vilka områden som var lämpliga att börja arbeta med. Vidare har det inneburit att hitta ett lämpligt sätt att implementera och testa textpaket, för att kunna sända paket över ett nätverk som använder sig av TCP/IP.

Detta arbete har gått relativt problemfritt. Att sända paket över nätverket var inga större problem. Däremot åtgick mera tid än beräknat, till att implementera textpaketen. Detta på grund av utprovning av olika metoder för att tilldela, samt packa upp ett paket.

Arbetet har vidare bestått av att i modulen integrera en databas, och att koppla ett generellt gränssnitt till denna. Det innebar inga större problem med att konfigurera ett ODBC-gränssnitt. Avsikten var dessutom att lägga till ett ADO-interface, för att konceptet skulle kunna användas till alla tänkbara databaser. Detta visade sig dock innebära vissa problem, och ADO-gränssnittet fick läggas åt sidan tills vidare.

Slutligen kan sägas att jag lärt mig mycket, och att arbetet varit mycket varierande. På grund av att jag följt arbetsgången i ett helt projekt, har jag fått inblick i alla dess moment. Resultatet har blivit sådant, att jag är nöjd med min arbetsinsats, och jag hoppas och tror att detta koncept kan vidareutvecklas.

Investigation concerning a Server to handle Error messages and Recovery Solutions

Abstract

This work has consisted of developing a concept, which is supposed to result in a new code standard. This concept is dealing with how errormessages and recovery solutions, can be presented to a computer user. A part of this work is to produce a prototype for a module which will receive error codes, and return suggestions what to do about errors. This according to the client-servermodel.

To realize the module prototype has meant interviews as preparation work, and from these develope a requirement specification. From this specification was then selected which parts that was suitable to begin work with. Further work has been to find a suitable way to implement and test text packages, and to be able to send packages over a network that uses TCP/IP.

This work has been relatively free from difficulties. To send packages over the network was not a problem. The implementation of the text packages on the other hand, took some more time than I had expected. The course of this was trying different methods concerning allocation, and how to unpack a package.

Further, this work has included that, in the module integrate a database, and to connect a general interface to it. The configuration of an ODBC-interface was not a problem. The intention was also to add an ADO-interface, to make the concept useful for all conceivable databases. Unfortunately this turned out to be troublesome, and the ADO-interface was put aside until further notice.

Finally I want to say, that I have learned a lot, and that the work has been very variable. Since I have followed a whole project, I have been able to catch a glimpse of all its elements.

The result has become as such, that I am satisfyed with my work, and I hope and believe that this draft can be developed further.

Tack

Jag vill tacka personalen på Industrisystem i Karlskoga AB, för all hjälp och stöd jag fått till detta arbete.

Dessutom vill jag särskilt tacka mina handledare: Hannes Persson på Karlstad Universitet, samt Roine Karlsson på ISAB, för konstruktiv handledning och support.

Innehållsförteckning

Figurförteckning

1 Inledning

Datorn skapades för att kunna rationalisera arbete, genom att hantera och bearbeta stora mängder information. Samtidigt leder datorerna till många missförstånd och konflikter för dem som är beroende av dem.

Det är lätt att glömma bort samarbetet mellan människa och maskin. Det vill säga, antingen handlar det bara om teknik, eller så gäller det bara människor. När det gäller tekniken, blir både hårdvara och mjukvara snabbare och effektivare. Beträffande människorna vill man att de skall kunna prestera mera, och kanske kunna ägna sig åt andra arbetsuppgifter utöver terminalarbete. Det ställer alltså högre krav på människan och hennes kunskaper. Ofta har det som skall produceras också en bestämd tidsgräns. Tidgränsen kan vara ganska snäv, och detta gör att människan måste klara en viss mängd stress. Om då tekniken krånglar, kan stressen öka och användaren blir förutom stressen också frustrerad. Detta gör att användaren ställer högre krav på tydlig information och ordentlig utbildning.

Exempelvis så är det troligt att den programmerare som arbetat fram mjukvaran, tycker att informationen är tydlig. Men har han eller hon frågat någon som inte arbetar med programmering hur den uppfattas? Nu görs datorerna alltmer användarvänliga, och vid utvecklingen av informationssystem tänker man mer och mer på användarens krav.

Dessutom kan det finnas intresse av att samla information på ett ställe. Informationen skall vara åtkomlig för flera användare, till exempel över ett nätverk.

Detta examensarbete är inriktat på dessa intresseområden.

1.1 Presentation av Industrisystem i Karlskoga AB

ISAB (Industrisystem i Karlskoga AB) är ett konsultföretag som hanterar gränssnittet/gränssnitten mellan produktionen på fabriksgolvet, och rapporteringen uppåt till överordnade affärssystem. Ett affärssystem kan beskrivas som ett program som kan övervaka och styra flödet av information, varor och transaktioner inom ett företag. Företaget ISAB arbetar med hela flödet: Från godsmottagning av artiklar eller komponenter, fram till utleverans av färdiga produkter till kund. ISAB:s affärside är ”att effektivisera kundernas material och informationsflöde, från produktionsnivån till överordnade, administrativa system” [1].

1.2 Presentation av examensarbetet

Examensarbetet hos ISAB består i följande saker:

- Att utreda hur en användare vill att ett felmeddelande skall visas på datorskärmen
- Att utreda hur en användare vill att instruktioner om hur man åtgärdar ett fel skall visas
- Arbeta fram en kravspecifikation som skall ligga till grund för en ny standard beträffande felsupport
- Utveckla en modul som kan ta emot felkoder, samt ge åtgärdsförslag till dessa
- Utforma datapaket innehållande felkod/åtgärdsförslag som skall kunna skickas över TCP/IP, till och från modulen
- Skapa en testdatabas för att till den kunna ställa frågor i SQL, beträffande felkoden som mottagits av modulen, samt få respons på frågorna från modulens databas
- Testa genom att skicka ett paket innehållande felkod från klienten, och att få svar från databasen

1.3 Presentation av innehållet i denna uppsats

Kapitel 2 behandlar bakgrund, problem och syfte med examensarbetet. Detta inkluderar var idén till modulen kom ifrån, vilka problemställningar som utgicks ifrån, samt varför ISAB vill skapa nämnda modul.

I kapitel 3 beskrivs förutsättningar, och de krav som modulen skall uppfylla. Texten beskriver också vilken funktionalitet som önskas av modulen, och hur det är tänkt att kontakten mellan klienter och modulen skall fungera.

Det förberedande arbetet behandlas i kapitel 4, och beskriver tillvägagångssätten innan utvecklingen av modulen kunde påbörjas. Kapitlet beskriver de undersökande intervjuer som gjordes, samt den kravspecifikation som togs fram utifrån dessa.

Följande kapitel (5) tar upp arbetet med att ta fram en prototyp för modulen. Det flödesschema som nämns, visar olika tänkta scenarion mellan klient och server. Ett av dessa scenarion beskrivs med att ett klientmeddelande skickas till servern. Hur meddelandet är uppbyggt, hur det hanteras på serversidan, samt hur serversidan ger respons till klienten. Dessutom nämns olika tekniker för att arbeta fram ett gränssnitt mot modulens databas.

Beskrivning av implementering, och hur testning av prototypen har gått till, återfinns i kapitel 6. Bland annat förklaras den paketeringsklass som utvecklats, för att möjliggöra att

paketerad information kan skickas med protokollet TCP/IP. Dessutom beskrivs två olika gränssnitt som kan användas vid kontakt med en databas.

De återstående tre kapitlen 7, 8 och 9 visar erfarenheter och rekommendationer, som tar upp några av de egna upplevelserna av projektutvecklingen. Där nämns kort några av de problem som förekommit vid examensarbetet, och slutligen avrundas med slutsatserna.

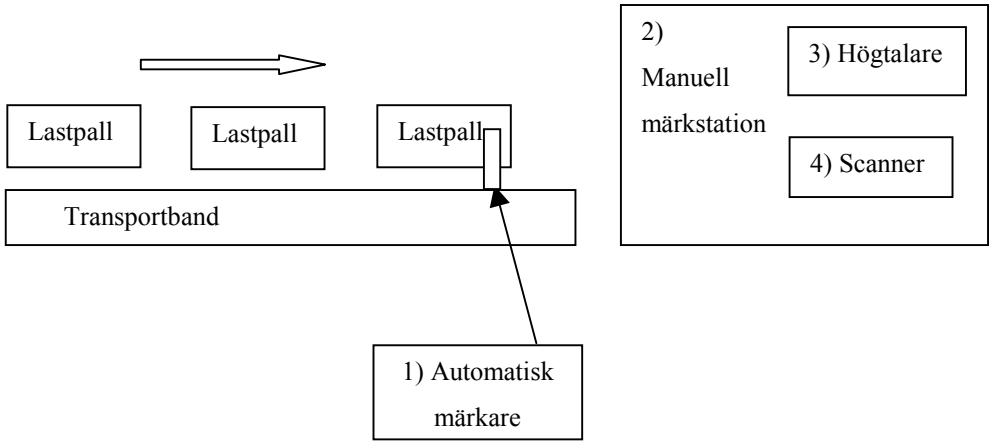
2 Bakgrund, problem, och syfte

Företaget Industrisystem i Karlskoga AB vill ha en modul som skall hantera varnings- och felmeddelanden. Med en modul menas en avgränsad enhet, som är avsedd att utföra en eller flera väldefinierade uppgifter. Modulen skall kunna lagra filer i en databas, där filerna innehåller förklaringar till felkoder. Det vill säga instruktioner om hur ett fel kan åtgärdas. Vidare skall modulen fungera inom ett nätverk som använder TCP/IP. Detta kapitel tar upp var idén till modulen kommit ifrån, samt några frågeställningar angående användares önskemål. Dessutom behandlas syftet med modulen.

2.1 Bakgrund

En idé om en modul med felsupport framkom genom att en kund hade önskemål om kontroll av märkning av pallar. Om den automatiska märkningen av en pall misslyckades av någon anledning, ville kunden att ett meddelande på något sätt skulle upplysa om detta. Sedan skulle pallen märkas manuellt. Ett system togs fram för detta ändamål, som fungerade på följande sätt (se illustrationen 2-1 nedan):

1) Lastpallarna förflyttas på ett transportband, där en automatisk märkare finns monterad.
2) När en pall passerat märkaren, kommer den till en manuell märkstation. Där kontrolleras pallen.
3) Om den inte är automatmärkt, säger en högtalarröst: ”Läs streckkod”.
4) I den bemannade manuella märkstationen används då en scanner, för att märka den missade pallen.



Figur 2-1: Kontroll av pallmärkning

Kunden tyckte att lösningen utföll till belåtenhet, och ISAB började då fundera på om de kanske kunde utveckla idén till att användas även inom andra områden.

2.2 Problem

Problemet med att skapa ovan nämnda modul måste delvis lösas genom att göra en grundlig undersökning av användares önskemål och krav. Ofta visas felmeddelanden, samt hur ett fel skall avhjälpas, på ett – för en användare - tvetydigt, eller till och med obegripligt sätt. Särskilt om användaren är ovan med att arbeta med någon form av datorterminal är detta ett problem.

Examensarbetet utgick ifrån följande frågeställningar:

- Hur skall man kunna presentera ett felmeddelande på ett tydligt sätt?
- Skall hjälpen presenteras på flera sätt än ett? Till exempel, skall man förutom beskrivande text även ha bilder? Vill en användare se en instruerande film?
- Hur skall dessa instruktioner lagras? Skall de ligga centralt på en server som alla kunder kan koppla sina felmeddelanden till? Skall de hanteras med hjälp av en - för ett företag – lokal modul, som innehar en databas med instruktioner?
- Eller, skall varje användare ha instruktionerna lagrade i sin egen terminal?

Vilket är bäst? Och bäst för vem?

2.3 Syfte

Syftet är att underlätta felhantering för alla inblandade (kunder, användare och support på ISAB), och att ta hand om den ”grundläggande” felhanteringen. Detta för att ISAB sedan skall kunna bygga på med mera avancerade tillämpningar till modulen. ISAB vill alltså ha hjälp med att utarbeta en ny standard för felsupport.

Syftet för kunder och användare är att de skall kunna förstå de felmeddelanden som kommer upp på datorskärmen. Genom att också kunna göra viss felsökning på egen hand när det gäller enklare fel, behöver en användare inte kontakta support med en gång. Detta kan vara bra om supporten kanske inte går att komma i kontakt med.

Syftet för de som står för supporten, är att kunna koncentrera sig på att avhjälpa allvarligare fel. Det vill säga sådana fel som en användare inte klarar av att åtgärda. Om användaren har klarat en enklare felsökning på egen hand, kan den som har hand om supporten koncentrera sig på de besvärligare felen.

Nästa kapitel behandlar vilka egenskaper som är önskvärda hos modulen.

3 Egenskaper hos modulen

Modulen skall fungera som en serverenhet som svarar på användarnas (klienternas) förfrågningar. Servern skall kunna ta emot felkoder från användarnas olika applikationer. När ett fel uppstår i mjukvara/hårdvara, skall information om felet visas i text för användaren. Det fel som inträffat skall sedan ges ett felnummer på heltalsform (en felkod). Denna kod är sedan avsedd att skickas med TCP/IP över nätverket till modulen. Detta kapitel beskriver hur en klient skall kunna utnyttja modulen, samt modulens grundläggande egenskaper.

3.1 Ett möjligt felscenario

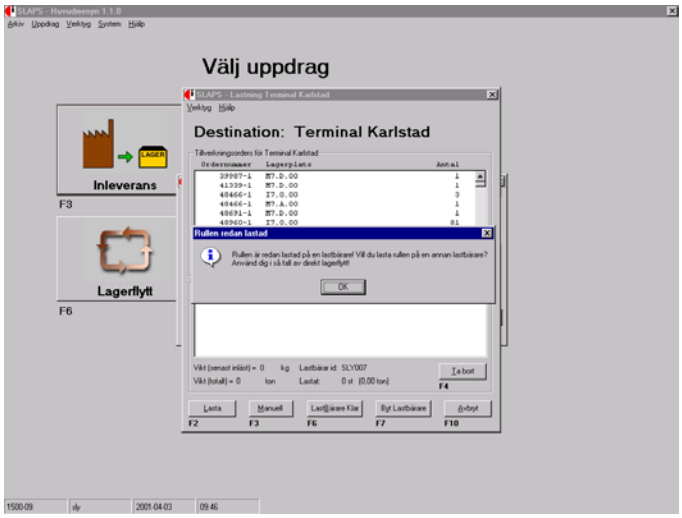
Exempel på en möjlig felaktig händelse är vid lagerhantering [2] (se figur 3-1):

1) Lastning av pappersrullar skall ske från ett lager. En dator finns i trucken.
2) Pappersrullarna är märkta med streckkoder
3) Trucken lastar, men tar fel pappersrulle/pappersrullar
4) Ett felmeddelande visas på truckdatorns skärm

1) Truck som är på väg att hämta rullar från lagret

2) Pappersrulle med streckkoder

3) Lastning av pappersrullar



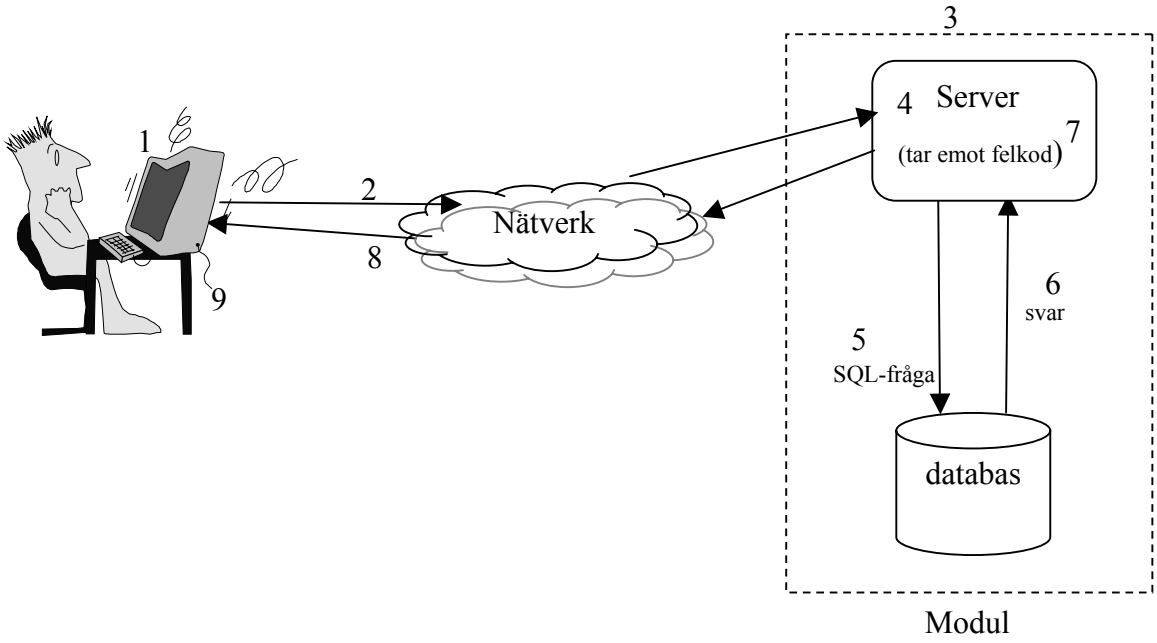
4) Meddelandet i informationsrutan i truckens dator lyder:
Rullen är redan lastad på en annan lastbärare! Vill du lasta på en annan lastbärare?
Använd dig i så fall av direkt lagerflytt!

Figur 3-1: Informationsruta i truckdator

3.2 Modulens egenskaper

Vid diskussioner med handledaren på ISAB framkom vissa tänkta egenskaper hos modulen (se figur 3-2). De användare som nämns är de anställda som finns hos ISABs kunder.

1) När en användare (en klient) sitter vid en dator och ett program (applikation) på något sätt genererat ett felmeddelande, ska detta felmeddelande visas i text för användaren. En användare skall kunna använda sig av modulens tjänster, även om användaren är ovan med terminalarbete. Därför måste detta meddelande vara otvetydigt, så att det inte kan tolkas på olika sätt.
2) Felmeddelandet skall sedan ges en felkod, som skall packas så att koden kan skickas över ett nätverk som använder TCP/IP. Detta innebär att användaren måste ha tillgång till ett nätverk för att kunna använda modulen. Flera klienter (användare) skall också kunna använda modulen samtidigt. Gränsen för hur många som kan använda den sätts av nätverkets kapacitet, samt av serverns prestanda.
3) På mottagarsidan skall en modul utvecklas, bestående av en server samt en databas. Servern skall kunna ta emot kodmeddelandet från nätverket. Modulen skall ha filer och texter som lagras i databasen. Avsikten med dessa, är att de skall innehålla förklaringar till felkoder. Dessa förklaringar och instruktioner kan sedan visas i text, bild, ljud eller multimedia. När modulen har tagit emot en felkod, skall den skicka respons från databasen tillbaka till användaren.
4) Meddelandet skall packas upp, och de delar som behövs för att ställa en fråga till en databas skall plockas ut. Delarna är ämnade att motsvara söknycklar i databasen.
5) En SQL-fråga görs med hjälp av ovanstående nämnda delar, för att söka i databasen.
6) Ett svar returneras av databasen.
7) I servern packas svaret om till ett TCP/IP-anpassat paket.
8) Svaret skickas sedan tillbaka över nätverket till klienten
9) Svaret skall packas upp på klientsidan, där en applikation skall visa svaret i text för klienten.



Figur 3-2: Modulens egenskaper

Modulen skall på detta sätt lämna instruktioner om hur ett fel avhjälps. ISAB vill också att en kund själv lokalt, skall kunna bygga på med egna instruktioner om vad som gäller just på den arbetsplatsen.

Ett krav är att svarstiden från modulen skall begränsas. Användaren får inte uppleva systemet som långsamt.

Vad beträffar databasen, skall den endast vara åtkomlig för ändringar av systemansvarig på ISAB. Den ”interna” funktionalitet som önskas av en kund, får dock hanteras som kunden själv önskar.

Nästkommande kapitel beskriver beredningsarbetet som inkluderar intervjuer, och den kravspecifikation som arbetats fram utifrån dessa intervjuer.

4 Beredning

Innan arbetet med själva modulen kunde påbörjas, insamlades material genom intervjuer med ISABs anställda. För att ha något att utgå ifrån under intervjuerna, hjälpte handledaren på ISAB till med att ta fram en kort lista med frågor. Dessutom framkom vid intervjuerna nya frågor, och nya problemställningar som i sin tur behövde besvaras. Utifrån intervjuerna utarbetades sedan med handledarens hjälp, en kravspecifikation. Kravspecifikationen definierar vad modulen skall ha för uppgifter.

Kapitlet beskriver arbetet med intervjuerna och kravspecifikationen.

4.1 Intervjuer

Beträffande intervjuerna skulle det bästa naturligtvis ha varit att intervjua de som använder ISABs utrustningar. Detta var dock inte möjligt, med tanke på den begränsade tid som ett examensarbete innebär. Den näst bästa lösningen var att intervjua de som har kontakt med dessa användare.

Därför undersöktes ämnet genom att utfråga en del av ISABs personal. Eftersom det är bra att få olika infallsvinklar på ett problem, utvaldes anställda med olika arbetsområden; såsom elektronik, hårdvara, försäljning och programmering.

Mycket intressant kom fram vid dessa intervjuer. Intervjuerna är sammanställda i bilaga A, med avsikt att kunna välja ut vilket/vilka önskemål som verkade lämpligt att börja arbeta med. Bilagan innehåller inte bara vad användare vill ha, utan också andra saker (till exempel personliga önskemål, tekniska begränsningar) som diskuterades vid intervjuerna.

Beredningsdokumentet ligger till grund för kravspecifikationen, och därigenom också för denna examensuppsats. Förhoppningsvis skall dokumentet även kunna användas i framtiden, för att bygga vidare på detta examensarbete.

Nedan följer en sammanställning av intervjuerna och kravspecifikationen.

4.1.1 Felmeddelanden

Det fortsatta beredningsarbetet innebar att undersöka hur användare vill att felmeddelanden skall presenteras på datorskärmen. Innan intervjuerna påbörjades existerade egna reflektioner över vad som kan vara viktigt. Något som kan upplevas som mycket irriterande är när ett kryptiskt felmeddelande, typ: ”Fel 107B i enhet A” skrivs på skärmen. Det vore mycket bättre

om det stod: ”Papperstrassel i skrivaren”, så att användaren slipper söka i dokumentation efter vad felet betyder. Det visade sig också senare vid intervjuerna, att just detta var ett önskemål som de flesta nämnde. Intervjuernas innehåll beträffande felmeddelanden följer nedan.

Ett textmeddelande skall vara kortfattat och på svenska. Engelska uttryck får endast användas när det gäller till exempel dataenheter som inte har någon tillfredsställande översättning till svenska språket.

När ett fel uppstår skall felmeddelandet visas på ett sådant sätt på datorskärmen att användaren förstår dels i vilken fysisk enhet felet uppstått, dels vad det är för typ av fel. Om flera liknande enheter finns (exempelvis skrivare) skall det i meddelandet finnas angivet vilken av enheterna meddelandet avser. Avsikten med ett tydligt felmeddelande är att datoranvändaren genast skall förstå innebörden och inte behöva leta i manualer efter felkod och vad koden betyder.

I de fall ett felmeddelande visas, där felet är av allvarligare slag kan det vara motiverat att lägga till en kod. I dessa fall räcker det inte med modulens hjälp, utan användaren måste få hjälp per telefon av ISAB. Om koden underlättar för att ta reda på vad det är för sorts fel, kan användaren uppge denna i telefon.

Avsikten med den tänkta modulen är alltså att vara behjälplig med råd och instruktioner, om hur användare åtgärdar fel, vars orsak ej är av allvarligare slag.

4.1.2 Instruktioner från modulen i form av text

Egna funderingar fanns också, beträffande önskemål om hur instruktioner avseende felkorrigering skulle kunna presenteras. En beskrivande text, med eventuellt en teckning eller bild, skulle kunna vara fullt tillräcklig. Om en instruerande filmsnutt (multimedia) måste följas, kan det upplevas som både frustrerande och stressande. Det är bättre att kunna arbeta i sin egen takt, och då kan en text upplevas som det mest effektiva.

När användaren fått ett felmeddelande och ISABs modul har kontaktats, får användaren tillbaka en text som instruerar hur felet ska åtgärdas. Om felet är av enklare slag räcker det med en kort instruktion. Om ett fel kan ha flera olika orsaker, kan användaren få förslag om vad som kan ha hänt.

Eventuellt kan en testlista användas, där användaren får göra enklare tester av systemet. Denna testlista kan innehålla vissa förslag på åtgärder, som användaren kan prova ett i taget för att felsöka. Testlistan kan genereras som text av modulen. Om en åtgärd visar något som inte fungerar som det skall, kan ett nytt meddelande skickas till modulen. Ett nytt svar kan då returneras, som förhoppningsvis kan visa hur felet korrigeras. Om användaren har gått igenom

hela testlistan, och ändå inte funnit en lösning på problemet får man söka hjälp på vanligt sätt genom telefonsupport.

Om ett visst fel har benägenhet att uppkomma ofta, och användaren har sett testinstruktionerna förut, måste det finnas möjlighet att kunna välja bort dem så man slipper gå igenom sådant man redan provat.

4.1.3 Instruktioner från modulen i form av bilder

Det är bra att komplettera en text med bilder. Särskilt om flera instruktioner ingår i samma meddelande kan man använda sig av en bildsekvens. Bilderna kan numreras där det visas steg för steg hur man skall göra. Möjlighet att backa tillbaka till en tidigare bild måste också finnas.

4.1.4 Instruktioner från modulen i form av ljud

Många arbetsmiljöer är bullriga. Ett ljudmeddelande med en instruerande röst kan vara svårt att avlyssna. Om ett system behöver anpassas till användare med nedsatt syn, är det dock en bra möjlighet att kunna komplettera modulen med röstmeddelanden. Ljud är också motiverat när en larmsignal behövs.

4.1.5 Instruktioner från modulen i form av multimedia

Möjlighet skall finnas att bygga vidare på modulen med multimedia. En film kan visas som instruerar användaren hur felet skall åtgärdas. Om användaren önskar få instruktioner från en film eller dylikt, skickas sökvägen med i responsen. Instruktioner på film skall alltså i första hand finnas lokalt hos användaren.

Nackdelen med multimedia kan dock vara att användaren blir frustrerad och stressad om han/hon är tvingad att se en film. När felet inträffar innebär det stress och irritation, och om man dessutom måste sitta och följa en filmsekvens kan personen bli ännu mera irriterad. En film kan dock användas som tillval. Om en instruerande text visas skulle användaren kunna välja om han vill se en film.

Den största fördelen med en film kan möjligen vara vid nyinstallation av mjuk-/hårdvara. Detta för att användaren skall få en ordentlig genomgång av hur produkten ska användas. Man har då möjlighet att gå tillbaka i filmen om något missats, eller flytta framåt om det är något som kan hoppas över.

4.1.6 Hur gränssnittet bör kunna hanteras

När ett felmeddelande visas, eller när ett svar returnerats från modulen skall dessa texter visas på användarens skärm i ett GUI. Gränssnittet skall hela tiden ligga synligt, så att även om användaren klickar i en underliggande applikation skall ej GUIt försvinna bakom. Detta för att om texterna avser den nämnda underliggande applikationen, måste den vara åtkomlig att arbeta med, samtidigt som användaren kan läsa texten. Särskilt om texten avser ett svar från modulen som är av instruerande typ. Det måste också gå att flytta och ändra storlek på hjälprutan, så att den inte skymmer det som är viktigt i applikationen. Användaren ska också kunna stänga hjälprutan om så önskas. Det kan vara så att en användare flera gånger sett samma meddelande, och då kan det upplevas som irriterande om rutan ej går att ta bort.

4.1.7 Kundens egna mjuka komplettering av modulen

Det skall finnas möjlighet för en kund att komplettera modulens hjälptexter med egna instruktioner. Instruktionerna gäller då lokalt på den egna arbetsplatsen. Det kan också vara så att en kund har ett eget LAN med flera arbetsplatser inom till exempel en koncern.

Ett meddelande skulle till exempel kunna lyda: ”Kontakta Nisse på IT-avdelningen med telnr: 12345, mailadress: Nisse@Business, och så vidare. Det skulle kunna innebära att en användare först utnyttjar ISABs modul, och sedan de interna hjälpmöjligheterna.

Fördelarna med detta är att ISABs direktsupport blir avlastad. Kunden behöver inte heller ägna tid åt att söka support per telefon. Dessutom kan kunden skräddarsy sin egen självhjälp.

4.1.8 När en arbetsstation är obemannad

I vissa situationer när en användare inte finns vid sin terminal, måste någon göras uppmärksam på felhändelser. Det skulle kunna åstadkommas genom att ett mail skickas automatiskt till någon systemansvarig, eller i vilket fall till en terminal som man vet är bemannad hela tiden.

Om felet är av allvarligare art som kan orsaka diverse fel och skador av produktionen eller människorna, kan till felmeddelandet kopplas ett automatlarm med ljud och eventuellt en ”saftblandare”.

4.2 Kravspecifikation

Beredningsarbetet fortskred med att utveckla en kravspecifikation över modulen (bilaga B). Till kravspecifikationen har ISABs egna mall använts. Definitionen av en kravspecifikation lyder: En kravspecifikation är ett dokument som ger en samlad beskrivning

av de krav användarna ställer på det framtida informationssystemet [3]. Den allmänna avsikten med en kravspecifikation framgår också av bilagans punkt 7.

Kravspecifikationen har också sitt största värde för ISAB i att den visar riktlinjerna för den nya standarden.

En diskussion med handledaren på ISAB, utifrån beredningslistan, gav vad som skulle ingå i kravspecifikationen.

Följande stycken beskriver vad kravspecifikationen (se bilaga B) innehåller. Rubrikerna motsvarar kravspecifikationens egna rubriker.

4.2.1 Allmänt

Introduktionen (se bilaga B, rubrik Allmänt) visar vad ISAB vill ta fram för sorts produkt (modulen). Varför man menar att produkten behövs, vad den skall användas till och hur man har tänkt sig att den skall fungera. Det är också formulerat till vilka användare produkten är avsedd för.

Den översiktliga beskrivningen förklarar att modulen skall ta emot, samt skicka respons på en felkod som skickas över ett nätverk.

4.2.2 Referenser

Rubriken referenser innehåller de dokument som kravspecifikationen hänvisar till. I detta fall är det beredningslistan som arbetats fram genom intervjuer.

4.2.3 Definitioner

Stycket förklarar och definierar de uttryck som har använts i kravspecifikationen.

- Med en användare avses den som använder applikationen
  Betydelse: Med användare menas en människa som använder en applikation. Skall ej blandas ihop med en applikation som använder en annan applikation eller tjänst.
- Med en användarapplikation avses den applikation som en användare arbetar med
  Betydelse: En applikation (programvara) som en datoranvändare arbetar med.
- Med ett felmeddelande avses ett textmeddelande som visas på en datorskärm
  Betydelse: Olika sorters fel kan manifesteras på olika sätt. Det vill säga, att alla fel som kan inträffa i ett system kanske inte har ett felmeddelande som kan visas på datorskärmen.
- Med ”tyst” mode avses att när servern svarat med respons, så förväntar den sig inte någon reaktion från användaren

Betydelse: När en felkod skickats från en klient och servern returnerar ett svar så kopplar servern ner förbindelsen.

- Med ”icke tyst” mode (interaktivt) menas att när en felkod skickats från en klient och servern returnerar ett svar så kopplar servern ej ner förbindelsen
  Betydelse: Servern kan då ta emot ytterligare förfrågningar från samma klient, och sända ytterligare svar. Om servern tar emot förfrågning nummer två, och denna visar sig vara att klienten meddelar att inget mera önskas, kopplar servern ner förbindelsen.

4.2.4 Tekniska krav

De funktionella kraven är:

- Modulen skall ta emot felmeddelanden i form av kod som genereras av en applikation.
  Betydelse: Modulen måste implementeras så att den kan buffra ett felmeddelande för att sedan uttolka betydelsen.
- Felmeddelanden skall tas emot via TCP/IP sockets.
  Betydelse: Genom att använda sockets kan servern ta emot meddelanden från flera olika klienter. Förutsättningen är naturligtvis att en klient har tillgång till nätverket över TCP/IP.
- Programvaran skall visa meddelanden i text för användaren.
  Betydelse: Ett textmeddelande presenteras i ett grafiskt gränssnitt.
- Modulen skall kunna konfigureras i ”tyst” samt ”icke tyst” mode.
  Betydelse: Klienten skall kunna bestämma själv vilken funktionalitet som önskas av modulen.
- Andra länder skall kunna få svar från modulen på sitt eget lands språk.
  Betydelse: Om en kund från ett annat land än Sverige vill utnyttja modulens tjänster, skall textmeddelanden och textrespons kunna konfigureras för det specifika landet.

De ickefunktionella kraven är:

- Svarstiden är av betydelse. Klienten vill ha snabb respons.
  Betydelse: Klienten vill ha snabb respons från modulen. Om svarstiden är lång kanske det ej är möjligt att vänta, felet kanske skadar produktionen eller dylikt.
- Modulens svar på ett felmeddelande skall vara kortfattat.
  Betydelse: Användaren har inte möjlighet att läsa en stor textmängd. Innehållet skall vara det som är relevant för ett visst fel, och ingenting mer.
- Modulens svar skall vara otvetydigt.
  Betydelse: Svaret får inte kunna uppfattas på olika sätt.

- Databasen skall endast vara åtkomlig för ändringar för systemansvarig hos kunden.
  Betydelse: Någon bör ha huvudansvaret, om något behöver ändras i de lokala önskemålen. Tillräckligt med kunskap måste finnas för att få ändra i databasen. Det är heller inte bra om hur många som helst kan gå in och ändra, eftersom olika personer tycker det skall se ut och fungera på olika sätt.

4.2.4.1 Yttre gränsytor

De gränsytor som finns är mot användaren, och mot användarapplikationen.

Kravspecifikationen tar upp följande gränssnitt mot användaren:

- Felmeddelandet skall ligga överst, tills användaren valt att ta bort det
  Betydelse: Med överst menas framför en eventuell applikation som finns på dataskärmen.
- Felmeddelandet skall vara flyttbart över skärmytan
  Betydelse: Det är viktigt att användaren skall kunna se vilken del av skärmens yta han/hon vill.

I underrubriken Gränsytan mot användarapplikationen finns angivet:

- Modulen skall använda sig av TCP/IP, det vill säga ett socketbaserat protokoll
  Betydelse: Orsaken till att det måste vara socketbaserat, är att flera klienter skall kunna kontakta modulen samtidigt.

4.2.4.2 Funktionskrav

De viktigaste funktionskraven för att kunna arbeta fram en prototyp för modulen är:

- Modulen skall fungera som en server som tar emot meddelanden
  Betydelse: Med meddelande menas de felkoder som tidigare nämnts.
- Modulen skall lagra information i en databas
  Betydelse: Vilken databas som helst som är kompatibel med ODBC skall kunna användas.
- Modulen skall skicka ”tyst”/”icke tyst” respons tillbaka till användaren eller applikationsanvändaren enligt protokollet
  Betydelse: En flagga i meddelandepaket visar vilket mode (läge) som klienten önskar använda.
- Modulen skall ha möjlighet att lägga till ljud- och bildstöd i etapp 2
  Betydelse: Först skall prototypen utvecklas till att bli en färdig produkt. Efter det kan etapp 2 börja, och andra media än text kan tillföras.

4.2.4.3 Kapacitetskrav

Stycket visar vad modulen skall klara av att hantera:

- Antal användare som samtidigt skall kunna komma åt systemet skall endast begränsas av nätverkets kapacitet

  Betydelse: Modulens tekniska lösning skall ej vara begränsad vad beträffar antalet användare.

- Stöd för mera än en applikation samtidigt

  Betydelse: Som tidigare nämnts under punkt 4.2.4, skall TCP/IP-sockets användas för att flera klienter skall kunna kontakta modulen samtidigt.

4.2.4.4 Säkerhetskrav

- Den ende som skall kunna ändra i databasen är den systemansvarige hos kunden, via ett systemadministrationsverktyg

  Betydelse: I regel är det inte önskvärt att användaren direkt skall kunna ändra, eller i värsta fall sabotera ett system. Då finns möjligheten att välja att skapa ett eget program (ett systemadministrationsverktyg) som medger tillägg/ändringar i system.

4.2.4.5 Konstruktionskrav

- Felmeddelanden skall skickas som paket över TCP/IP

  Betydelse: Kunden måste ha tillgång till ett nätverk med protokollet TCP/IP

- Modulen skall vara kompatibel med Windows NT 4.0 / 98 / 2000

  Betydelse: Äldre system måste möjligen uppdateras för att kunna utnyttja modulens tjänster

4.2.4.6 Användarens kapacitet och kompetens

- En helt oinitierad användare skall kunna använda sig av modulens tjänster

  Betydelse: Användaren skall ej behöva någon form av utbildning för att använda sig av modulen.

4.2.4.7 Övrig information

- Denna kravspecifikation omfattar ej handdatorer

  Betydelse: Handdatorer används ofta i ISABs system. Dessa kräver dock en specifik utredning, då egenskaperna skiljer sig från persondatorer och datorer som är installerade i truckfordon.

5 Inledande arbete med prototypen

För att kunna ta fram en prototyp över modulen, behövdes ett flödesschema för att se vilka olika kommunikationsscenarion som kan inträffa mellan klient och server. För att göra det möjligt för dessa att utbyta information över ett nätverk, behövdes någon form av paketering av informationen. Dessutom skulle möjligheten finnas att testskicka dessa paket över ett fiktivt nätverk, för att se att kommunikationen mellan klient och server fungerade tillfredsställande.

Detta kapitel beskriver flödesschemat, hur ett meddelande skickas över ett nätverk samt uppbyggnaden av detta paket. Några olika gränssnitt beskrivs också med en förklaring till varför dessa gränssnitt valdes.

5.1 Flödesschema

Flödesschemat som återfinns i bilaga C, beskriver ”trafiken” enligt en klient-server-modell. En klient skickar ett paket, som innehåller bland annat en felkod, och servern som hämtar information från databasen, returnerar ett paket med lämpligt svar. Intentionen är att en applikation hos klienten på något sätt ska generera en felkod, när någon typ av problem inträffat i nämnda applikation. Förutom felkoden som skickas, ska ett felmeddelande också visas för användaren i ett GUI.

Servern skall kunna exekveras i två lägen:

I ”Tyst” läge sker ingen interaktion med klienten. Endast ett svar returneras.

I ”Icke-tyst” (interaktivt) responsläge returneras ytterligare svar, efter att datoranvändaren gjort en tangenttryckning.

5.2 Kommunikation mellan klient och server

Utifrån ett befintligt program med upp-, och nedkoppling mellan klient och server, samt trådhantering, fanns möjlighet att skicka textmeddelanden. Programmet var utbyggbart, samt ett testprogram med ett GUI (se 6.2) fanns tillgängligt för att skriva in en textsträng som klient, och GUIt visade då att serversidan tog emot texten.

Prototyparbetet fortsatte med att utforma paketering av den text, som skulle skickas över nätverket mellan klienten och servern.

5.2.1 Utformning av paket

När ett fel inträffat och en applikation genererat ett felmeddelande (som beskrivs under 5.1) till användarens datorskärm, skall också ett paket (en textsträng) innehållande kodad information om felet skapas. Ett paket kan se ut på olika sätt, till exempel så här (se figur nr 5-1):

G	E	T		T	E	X	T		1	0	1		7	2				osv

(Hela exemplet är mycket förenklat, och motsvarar inte exakt hur ett paket ser ut).

Figur 5-1: Exempel på ett paket

Paketet skall bland annat innehålla information som avser ovan nämnda felmeddelande. Arbetet med prototypen har dock inte inkluderat att applikationen ska generera meddelanden, mappa om meddelande till kod, samt att skicka koden till servern. Därför har ett paket implementerats direkt i koden. Paketet är i form av en vektor som har fasta storlekar på sina fält. Det enda fält som kan variera i omfång är datafältet.

En paketeringsklass arbetades fram för att kunna skapa ett format för att skicka paket. Ett objekt av denna paketeringsklass skapas på klientsidan, (här benämns objektet för Klientpaket). När ovan nämnda paket har genererats, plockas ett fält i taget ut ur strängen, och innehållet lagras i Klientpaketets variabler:

Klientpaket Förfrågan=Get Text
Felnummer=101
Programnummer=72

Variablerna används på detta sätt till att lagra de olika byggstenar som ett paket är uppbyggt av. Klientpaketet är tänkt att även kunna användas för andra framtida behov. Exempelvis kan det vara bra att kunna välja ut de variabler som kan vara av intresse för loggfiler och dylikt.

Paketet (textsträngen) paketeras och skickas sedan till serversidan, med hjälp av TCP/IP över ett nätverk. På serversidan skapas ett liknande paketobjekt av paketeringsklassen som på klientens sida. (Objektet benämns här för Serverpaket). Det inkomna paketet packas upp av Serverpaketet, och varje del ur paketet läggs för sig, i varsin variabel i detta objekt.

Dessa variabler skall senare användas för att ställa frågor i SQL. Det innebär att en variabels innehåll kan motsvara en söknyckel i databasen. Variablerna kan också precis som på klientsidan, användas till loggfiler och dylikt i framtida utveckling av denna modul.

Svaret som fås från databasen, packas på samma sätt till ett paket av paketeringsklassen, och skickas över nätverket tillbaka till klienten.

Nästa steg i att ta fram en prototyp bestod i att göra ett gränssnitt mot databasen.

5.3 Gränssnitt mot databasen

För att servern skulle kunna kontakta databasen, behövdes ett gränssnitt av någon form (se figur 5-2). Dessutom skulle det vara bra om gränssnittet inte endast var användbart till en enda typ av databas, utan av en mer generell form. Den huvudsakliga orsaken till detta är att det skall kunna vara möjligt att använda olika databaser, men ändå endast ha ett enda gränssnitt för att kunna kontakta dessa.

Därför valdes nedanstående tekniker.

5.3.1 ODBC

Att arbeta fram ett gränssnitt mot databasen innebar att konfigurera ett ODBC-gränssnitt. ODBC är implementerat som ett SQL-baserat gränssnitt. Det är standardiserat för att kunna samarbeta med databaser av olika slag. Bakom gränssnittet finns program för varje databas som tar emot funktionsanrop, och konverterar dem till anrop för databasens särskilda gränssnitt.

En liten testdatabas skapades i Access (se bilaga G) för att servern skulle kunna hämta data. Sedan konfigurerades en ODBC-datakälla till databasen. Denna datakälla är till för att peka till databasen. Vidare utformades ett ”programskal”, i detta fall ett grafiskt gränssnitt. Därigenom visades innehållet från databasens (enda) tabell.

När konfigurationen fungerade som avsedd, kunde det grafiska gränssnittet tas bort. Det behövdes inte, eftersom syftet var att databashanteringen skulle byggas på med ADO. (Se nästkommande delkapitel).

5.3.2 ADO

ADO är en teknik för databasåtkomst. Tekniken är avsedd att kunna användas med alla programmeringsspråk och program. Om en SQL-, eller Oracle-server används behöver ADO ej gå via ODBC eftersom där finns OLE databas-gränssnitt [4].

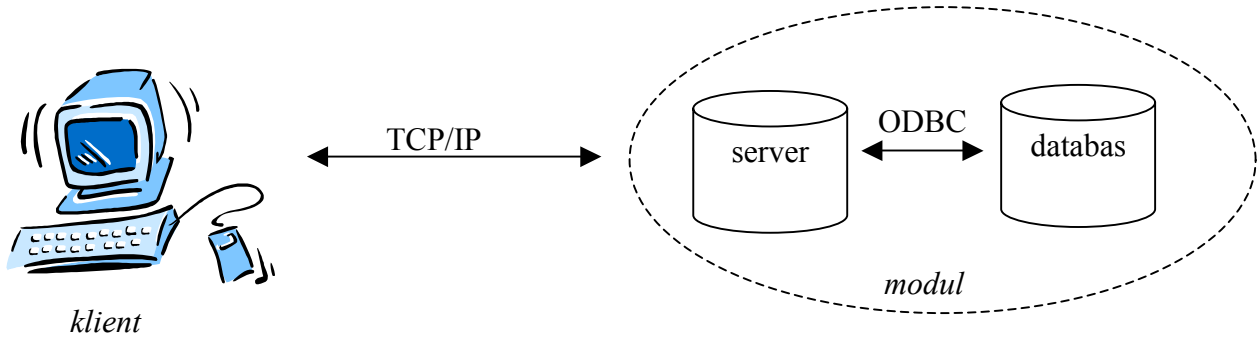
OLE är Microsofts standard för att länka ihop applikationer, det vill säga för att få olika program att samverka. Det innebär att objekt som behövs i ett eller flera program lagras centralt på ett enda ställe i datorns minne. (OLE blev som handelsnamn nedlagt i början av 1998 då Microsoft bestämde att OLE, ActiveX, COM och DCOM alla ska heta COM) [4].

Syftet med att använda ADO ”ovanpå” ODBC är att komma åt de databaser som inte har ett OLE DB-gränssnitt. Bilaga D visar en översiktlig bild över hur detta är avsett att fungera.

ISAB vill ha databashanteringen så att den skall vara kompatibel med olika programmeringsspråk som därför kan använda ADO.

Tyvärr fungerade inte detta med ADO som det var tänkt, och det fick räcka med att använda ODBC, (se under rubriken Implementering och test).

Orsaken till att just ADO valdes, är att ISAB arbetar i Microsoft-miljö, och därför har de valt att ADO-nivån skall vara den som gäller.



Figur 5-2: Databasens gränssnitt

Nästkommande kapitel behandlar implementeringsdelen och test av denna.

6 Implementering och test

Microsoft Visual C++ 6.0 har använts som utvecklingsmiljö, med operativsystemet Windows NT 4. Innan någon kod implementerades, testkördes ett program (se 6.1) i en ”socket-testningsapplikation” med ett tillhörande GUI (se 6.2). Möjligheten att koppla upp sig till ett nätverk fanns, och en textsträng kunde skrivas in från ”klientsidan”. I programkoden fanns då en liten funktion, som konverterade den inskrivna texten från gemena bokstäver till versaler, och tvärtom.

Arbetet beträffande implementationsdelen som beskrivs i detta kapitel, har huvudsakligen bestått av att arbeta med vad som händer när klienttråden befinner sig i Run()-metoden. Alltså när en förbindelse upprättats mellan en klient och servern, och ett datapaket skall skickas över ett nätverk.

Implementationsdelen föregicks som nämnts av testkörning av programmet, som tas upp i nästa stycke.

6.1 Program

Ett redan befintligt program fanns tillgängligt att bygga vidare på. Det innehåller förutom mainfunktionen, klassen thread som är abstraktion av trådar. I thread sköts till exempel start, paus och stopp av tråd.

Trådklassen ärvs av två klasser (bilaga E): serverthread, samt clientthread. Servertråden har bland annat en socket, och en port att lyssna till. Servertrådklassen är alltså till för att kunna ta emot klientanrop, och eftersom flera sockets kan skapas, kan den hantera flera klienter (det vill säga, flera klienttrådar).

Klassen clientthread används till exempel för att initiera eller stänga ner en klienttråd. Här kontrolleras också om en förfrågan kommit från en klient. Om så är fallet hanteras denna i klienttrådens Run()-metod.

6.2 GUI för nätverkstest

För att kunna testköra ovan nämnda program, användes ett klient-server-program (Castalia Socket Tester Application) med ett GUI (bilaga F). Genom detta var det möjligt att koppla upp sig till ett nätverk, genom att ange en IP-adress och en port. I fältet ”Data to send:” kunde

en textsträng skrivas in från ”klientsidan”. I programkoden fanns då (som tidigare nämnts) en liten funktion, som konverterade den inskrivna texten från gemena bokstäver till versaler, och tvärtom. Den konverterade texten skickades så tillbaka över nätverket till klientsidan, och visades i GUIts ruta ”Transmission log”. När instuderingen av klient/server-modellen var klar, lades egen kod till i programmet.

GUIt användes inte endast för att testa kommunikationen över ett nätverk. Det användes också för att kontrollera att innehållet i de variabler som hämtades, verkligen placerades på rätt plats i textsträngen (se kapitel 6.4.1).

6.3 Paketeringsklassen

En paketeringsklass skapades först, som skall hantera uppackning och paketering av de paket (textsträngar) som sedan ska skickas över ett nätverk. Bilaga H visar några paketformat. Paketeringsklassen har också medlemsvariabler, som används för att lagra de olika fälten som finns i ett paket. Medlemsvariablerna är av typen char array (en vektor) med olika längd, beroende på vilket fält som skall lagras. Variablerna skall bland annat användas för att ställa SQL-frågor till databasen. Tanken är också att man skall kunna använda variablerna till exempelvis loggfiler och dylikt, i framtida utveckling av modulen. Bilaga I innehåller implementeringen av paketeringsklassen.

6.3.1 Utvecklingsarbetet med paketeringsklassen

Utformningen av paketeringsklassen började med att skapa två metoder: set_mForFragan och get_mForFragan. Dessa var avsedda att hantera den variabel (mForFragan) som skulle lagra vilken typ av förfrågan, som senare var avsedd att skickas till servern.

För att testa tilldelning och hämtning av denna variabel, skapades ett objekt av paketeringsklassen. Sedan gjordes en tilldelning till variabeln med metoden set_mForFragan, där variabelinnehållet bestämdes till ”GET_VERSION”.

För att se till att innehållet i variabeln endast var dessa tecken, lades ett strängslutstecken: ’\0’ till. Sedan hämtades variabelinnehållet med metoden get_mForFragan, och lades i ett paket. Paketet skickades sedan från klienten till servern på det sätt som tidigare beskrivits i kapitel 6.2. Paketinnehållet – i detta fall alltså ”GET_VERSION” – visades i rutan ”Transmission log” som också beskrivits i kapitel 6.2.

Paketeringsklassens utformning fortsatte sedan med att lägga till de övriga variabler som skulle behövas. Dessa hade framkommit vid diskussioner med handledaren på ISAB. Varje variabel fick sedan en set-, och en get-metod för tilldelning och hämtning.

6.4 Klienten skickar ett paket

På klientens sida skapas ett objekt av paketeringsklassen (namngivet Clientpaket). Objektets medlemsvariabler tilldelas olika sorts information. Till exempel vilken typ av förfrågan (flera olika kan förekomma), vilket sekvensnummer, och allvarlighetsgrad på felet som inträffat. Olika typer av förfrågningar kan till exempel vara SHOW_ERROR, SHOW_WARNING och GET_VERSION.

För att testa paketeringsklassen har variablerna tilldelats följande värden: ”SHOW_ERROR” (typ av förfrågan), ”123” (sekvensnummer), ”4567” (programnummer), ”abcdefg” (felnummer), samt ”1” (felnivå). Dessa värden saknar betydelse i nuläget. Det vill säga att inget av fälten ”SHOW_ERROR”, ”123”,”4567”, ”abcdefg” och ”1” har någon speciell innebörd. Tilldelningen av värden har kunnat vara precis vilka tecken som helst, bara för att se att exakt dessa tecken kommer fram till serversidan. Just på detta sätt har testningen gått till: Värdena har bytts mot vilka tecken som helst. Dessutom har olika längder på variabelinnehållen provats. Orsaken till detta var förutom att testa paketeringsklassen, också att indexeringen skulle fungera utan fel.

En vektor skapas som skall användas som paket (namngiven Skickas, se bilaga I), och med hjälp av en paketeringsmetod (beskriven i kapitel 6.4.1) i objektet Clientpaket, tilldelas ”Skickas” innehållet från Clientpaketets medlemsvariabler. I detta fall hämtas alltså de testvärden som nämnts. Detta paket (vektorn) skickas sedan över ett nätverk med TCP/IP.

6.4.1 Paketeringsmetoden

För att kunna placera de olika variabelinnehållen i ett paket som skall skickas från klienten, skapades först en textvektor med den fördefinierade storleken 100 (antal möjliga tecken).

En ny metod benämnd mPacka, lades till i paketeringsklassen. Metoden implementerades så att den kan ta emot ovan nämnda textvektor som inparameter. Innehållet i den variabel som tidigare tilldelats typ av förfrågan, hämtades och placerades med sitt första tecken på textvektorns första plats.

Nästa variabelinnehåll (”sekvensnummer”) hämtades sedan. Detta innehåll placerades också på en bestämd plats med index nummer 15 i vektorn.

För att kontrollera att detta ”sekvensnummer” verkligen placerades på rätt plats i textsträngen, användes det tidigare beskrivna GUIt. De indexerade platserna som blivit tomma mellan ”förfrågan” och ”sekvensnummer” tilldelades samtliga med siffran noll. Sedan skickades strängen med användning av GUIt, och visades där i dess svarsruta.

Genom att räkna antal tecken: GET_VERSION (11 stycken) och lägga till antal nollor som visades (4 stycken), blev alltså teckensumman 15. Eftersom teckensträngens indexering börjar på 0, var alltså index 0 till och med 14 fyllda med ovanstående innehåll. Slutsatsen kunde alltså dras, att nästa fält i strängen började på index nummer 15. Alltså fungerade detta som det var tänkt.

På samma sätt hämtades innehållet från de återstående variablerna, placerades i textsträngen, samt testades med klient-server-GUIt. (Bilaga H visar index och fältstorlekar för teckensträngen).

På första lediga positionen efter sista fältet, placerades slutligen ett strängslutstecken för att visa att inga fler tecken ingår i textsträngen.

6.5 Servern tar emot ett paket

När paketet tagits emot på serversidan, skapas ett paketeringsobjekt även där (namngivet Serverpaket). Med hjälp av en uppackningsmetod (beskriven i kapitel 6.5.1) i objektet, plockas de olika fälten ut ur paketet. Fältens innehåll läggs i objektets medlemsvariabler (precis som på klientsidan i ”Clientpaket”). Från dessa variabler skall det sedan väljas ut vilken/vilka som ska användas för att ställa SQL-frågor till databasen. I testfallet har dock använts en fråga som implementerats direkt i koden: ”Select ID, Data from help where ID=1”, det vill säga att inga värden hämtas ur Serverklassens variabler. Orsaken till detta är att tid saknades att utveckla denna del färdigt.

6.5.1 Uppackningsmetoden

För att kunna plocka ur innehållet ur en textsträng på serversidan, skapades en uppackningsmetod. Denna metod tar precis som packningsmetoden emot en textsträng som inparameter. En temporär vektor skapas först i metoden, som används för att mellanlagra inparameterns textstränginnehåll. Eftersom flera olika förfrågningstyper kan finnas, jämförs innehållet i textsträngens början med de förfrågningar som finns fördefinierade. Orsaken till att detta görs, är att kunna avgöra antalet tecken i förfrågningen, samt att lägga till ett strängslutstecken. Sedan tilldelades innehållet till variabeln mForFragan. För att se att variabeln fått rätt innehåll, hämtades detta med paketeringsklassens get-metod, och återigen användes GUIt för att titta på innehållet.

Tilldelning och kontroll av de övriga variablerna gjordes på samma sätt. Dock skedde i dessa fall ingen jämförelse mellan strängar som förekommer i förfrågningsfallet.

6.6 Kontakt med databasen

Ovan nämnda SQL-fråga ställs till databasen. Syftet med SQL-frågan är att välja ut de tabeller och fält som är av intresse från databasen. Svaret från databasen, i detta fall ett heltal och en text, läggs i två av Serverpaketets variabler. Databassvaret packas sedan av Serverpaketets metod mPackaSvar, till ett paket (namngivet aUtbuf) och skickas tillbaka över nätverket till klienten.

6.6.1 Packa svar-metoden

Den sista klassmetoden som lades till var mPackaSvar. Den är tänkt att användas av serversidan, för att paketera svaret som erhållits från databasen (se kap 6.6.2.2). De variabler som finns att tillgå är:

- mFlagga som skall lagra ”OK” eller ”NOK” (se bilaga C), för att visa om databasen är åtkomlig
- mFelNiva, som skall lagra ett fels allvarlighetsgrad
- Text1 som har skapats för att kunna lagra en instruerande text

Metoden tar som inparameter en textsträng, vilken är avsedd att användas för att innehålla det svar som skall packas, och som sedan skall skickas till klientsidan. Paketeringen går till på liknande sätt som i metoden mPacka (se kapitel 6.4.1).

För att klienten skall kunna packa upp innehållet ur serverns svarspaket, behövs en metod liknande den som beskrivs i kapitel 6.5.1. Tiden räckte dock ej till att utveckla denna.

6.6.2 ODBC-gränssnitt

Innan SQL-frågor kunde ställas till databasen, konfigurerades en ODBC-datakälla som skulle peka till denna. Med hjälp av Windows fördefinierade ODBC-konfiguration angavs namnet på datakällan till ”SupportMan”, samt vart källan finns lagrad. Ett namn, samt konfiguration måste finnas för att veta vilken databas som avses, om man exempelvis vill kunna använda flera databaser under gränssnittet.

6.6.2.1 Uppkoppling till databasen

För att koppla upp sig till databasen anropades några ODBC-funktioner. Först användes funktionen SQLAllocEnv(&sHenv) som allokerar minne till ett ”miljöhandtag”. Funktionen initierar ODBC-gränssnittet så att applikationen skall kunna använda miljön. Miljöhandtaget lagras i variabeln sHenv. Sedan anropades SQLAllocConnect(sHenv,&sHdbc) som allokerar

minne till ett förbindelsehandtag inom miljön, som i den första nämnda funktionen identifieras av sHenv. Sedan lagras förbindelsehandtaget i variabeln sHdbc.

Genom att använda funktionen SQLConnect etablerades en uppkoppling till datakällan ”SupportMan” (se 6.6.2). Parametrarna är de tidigare nämnda variablerna sHenv och sHdbc. De avslutande SQL_NTS är standarduttryck som används om till exempel användarnamn och password inte behövs. Detta var inte särskilt angeläget vid utvecklingen av modulen.

En kontroll sker sedan med funktionen SQLError, om returvärdet från SQLConnect är negativt. Då har något inträffat, som gör att uppkoppling ej kan ske. Om däremot returvärdet visar att uppkoppling har skett, kan information hämtas från databasen.

6.6.2.2 Kommunicera med databasen

I nästa steg används funktionen SQLAllocStmt som allokerar ett ”handtag” till ett statement, och associerar handtaget med kopplingen som tidigare specifierats av sHdbc. Statementhandtaget lagras i variabeln sStmt.

En vektor skapas sedan, namngiven till sSQLKommando. I den lagras den SQL-fråga som skall ställas till databasen. I detta fall är det frågan ”SELECT ID, Data FROM Help where ID=1”. För att ställa frågan till databasen, används funktionen SQLExecDirect med strängen sSQLKommando som parameter. För att sedan identifiera kolumnerna ID och Data som skall hämtas, anropas funktionen SQLBindCol. I nästa steg hämtar SQLFetch en rad med data. Fältinnehållet från kolumnen Data, där radens ID=1, läggs i en textsträng, benämnd Text1, och lagras i en medlemsvariabel i objektet Serverpaket. Även fältinnehållet från kolumnen ID hämtas, för att kontrollera att både Data och ID har de förväntade värdena. Däremot lagras inte ID-fältet i någon variabel. Endast Datafältet lagras, för att sedan skicka detta i paket till klientsidan.

6.6.2.3 Nedkoppling av databasen

För att frigöra minne, samt för att koppla ned från databasen har följande funktioner använts:

- SQLFreeStmt lämnar tillbaka miljöhandtagets minne
- SQLTransact gör en rollback operation för alla update-, insert-, and delete-transaktioner som associeras med en databaskoppling. Om inga transaktioner är aktiva har funktionen ingen effekt
- SQLDisconnect stänger ner kopplingen som är associerad med sHdbc and

- SQLFreeConnect kopplar loss miljöhandtaget sHdbc, och frigör allt minne som associeras med detta
- SQLFreeEnv frigör förbindelsehandtaget sHenv, samt allt minne som associeras med detta

6.6.3 ADO

Meningen var att ADO skulle användas ”ovanpå” ODBC-datakällan. ADO behövs ej vid exempelvis en Oracle-databas, eftersom ett OLE DB-gränssnitt redan finns tillgängligt. Däremot används ADO för att kunna nå databaser som ej har detta gränssnitt. Alltså var avsikten att konceptet skulle kunna användas till de flesta tänkbara databaser.

Tyvärr visade det sig inte vara genomförbart. Det gick inte att få en anslutning till Access-databasen med ADO. Boken ”Visual C++ på tre veckor” [5] hade instruktioner, men de var inte tillräckliga, och ingen vid ISAB fanns tillgänglig som hade erforderliga kunskaper om ämnet. Detta problem kanske kan lösas vid vidare arbete och utveckling av konceptet.

6.7 Databasen

Microsoft Access användes till att skapa databasen, och för att kunna ställa SQL-frågor. Databasen innehåller för närvarande endast en tabell, benämnd ”help”, som använts för att se att det svar som returnerades vid en SQL-fråga var det som förväntats.

För att testa databasen görs först en koppling till denna. Om kopplingen fungerar som den skall, ställs en SQL-fråga. Frågan är (som nämnts under 6.4) skriven som textsträng direkt i koden, och lagras i vektorn ”sSQLKommando”. Svaret som returneras är i detta testfall ett ID-nummer, samt en text, båda från ovan nämnda tabell. Endast texten lagras i en vektor: Text1. Innehållet i vektorn kan sedan plockas ut och skickas i ett paket till klienten. ID-numret var också avsett att lagras i en vektor, för att även detta skulle kunna skickas till klientsidan. Det har dock ej gjorts på grund av att examensarbetet började nå sitt slut, och tiden räckte ej till.

Kapitel 7 tar upp erfarenheter och rekommendationer beträffande detta arbete.

7 Erfarenheter och rekommendationer

Stycke 7.1 Intervjuer, tar kort upp några av de egna upplevelserna under beredningsfasen. Under 7.2 följer en avstämning mot kravspecifikationen för att tydliggöra vilka punkter som åtgärdats, och vad som återstår att arbeta med, samt vissa rekommendationer hur det kan vara lämpligt att gå till väga.

Dessutom följer ytterligare rekommendationer samt förslag, beträffande vissa avsnitt som ingår i 7.2, och som behöver förklaras utförligare.

7.1 Intervjuer

Intervjuerna gav förutom material till beredningen, också ökad självkännedom. Vissa förslag upplevdes som mindre bra, och skulle kunna ignorerats, eller åtminstone göra någon förbättring/förändring. Det hade inte varit bra, eftersom syftet med intervjuerna då hade frångåtts. I detta avseende är det felplacerat att komma med egna åsikter som färgar andras. Koncentrationen bör ligga på att vara objektiv och inte få fram ”önskade” svar.

7.2 Kravspecifikation

Att ta fram en kravspecifikation kan vara ganska komplicerat. Det som är svårt är att definiera kraven tydligt. Alla krav måste vara entydiga och exakta, så att inga oklarheter finns. Därför var det viktigt att kontrollera om handledaren avsett innebörden i de punkter som nedskrivits i kravspecifikationen. Kravspecifikationen måste också vara så detaljerad att den är till nytta. Den ska vara en bra utgångspunkt för det fortsatta arbetet.

För att visa vad som hittills åtgärdats, och rekommendationer för vad som fortsättningsvis kan utvecklas vid arbetet av modulen, följer kravspecifikationens grundläggande delar:

7.2.1 Tekniska krav

- *Modulen skall ta emot felmeddelanden i form av kod som genereras av en applikation*
  De felmeddelanden som förekommer vid testningen, är i dagsläget inskriven i programmets implementation. Modulen är implementerad så att den kan buffra ett felmeddelande i objekt av paketeringsklassen FPaket.
  Att åtgärda: Att en felkod istället genereras automatiskt av den applikation, i vilken ett fel har uppstått.

- *Felmeddelanden skall tas emot via TCP/IP-sockets*

  Kravet var redan uppfyllt i och med det redan befintliga programmet (se kap 5.2).
- *Programvaran skall visa meddelanden i text för användaren*

  De textmeddelanden som är avsedda att visas behöver skapas. Dessutom behöver textmeddelandena på något sätt kopplas till ett grafiskt gränssnitt. Det nuvarande gränssnittet har endast använts för att kunna följa kommunikationen mellan klient och server.
- *Modulen skall kunna konfigureras i ”tyst” samt ”icke-tyst” mode*

  Arbetet med prototypen har hela tiden koncentrerats på ”tyst” mode. Lämpligast skulle vara att färdigutveckla prototypens ”tysta” mode. Först när det fungerar tillfredsställande kan arbetet med ”icke-tyst” mode påbörjas.
- *Andra länder skall kunna få svar från modulen på sitt eget lands språk*

  Konceptet bör vara färdigutvecklat, innan detta krav kan uppfyllas.
- *Svarstiden är av betydelse, klienten vill ha snabb respons*

  Detta krav blir aktuellt först när systemet skall installeras hos en kund.
- *Modulens svar skall vara kortfattat och otvetydigt*

  Testdatabasen skall bytas ut mot den databas som avses att användas. Databasens svar på en felkod skall lämpligen formuleras så att texten blir kort och endast kan tolkas på ett enda sätt.
- *Databasen skall endast vara åtkomlig för ändringar för systemansvarig hos kunden*

  De ändringar som avses i kravspecifikationen och skall göras i databasen, blir inte aktuella förrän konceptet kan anses färdigutvecklat.

7.2.1.1 Yttre gränsytor:

- *Felmeddelandet skall vara flyttbart över skärmytan, samt ligga överst tills användaren valt att ta bort det*

  När det grafiska gränssnittet mot användaren utvecklats, skall den önskade funktionaliteten läggas till.
- *Modulen skall använda sig av TCP/IP, det vill säga ett socketbaserat protokoll*

  Gränssnittet mot användarapplikationen är färdigt, och TCP/IP kan användas.

7.2.1.2 Funktionskrav

- *Modulen skall fungera som en server som tar emot meddelanden*

Modulen är utvecklad enligt servermodell, för att kunna ta emot meddelanden i form av felkod

- *Modulen skall lagra information i en databas*

  En testdatabas i Microsoft Access är utvecklad. Vad beträffar databasen är det lämpligt att fortsätta arbetet med att lägga till den information som önskas. Som några kolumnexempel till en tabell kan ges: Text1 som skall innehålla förklaring till en felkod. Text2 som kan innehålla ytterligare, och eventuellt mera utförlig förklaring till samma felkod.

- *Modulen skall skicka ”tyst”/”icke tyst” respons tillbaka till användaren eller applikationsanvändaren enligt protokollet*

  Som tidigare nämnts har arbetet hittills endast varit angående ”tyst” respons. I framtida utveckling av modulen, kan en flagga läggas till som visar vilket mode klienten önskar använda.

- *Modulen skall ha möjlighet att lägga till ljud- och bildstöd i etapp 2*

  Detta har i nuläget ej varit aktuellt, utan först är det lämpligt att utveckla prototypen till en färdig produkt. Sedan kan andra media än text tillföras.

7.2.1.3 Kapacitetskrav

- *Antal användare som samtidigt skall kunna komma åt systemet skall endast begränsas av nätverkets kapacitet*

  Denna punkt har ej varit aktuell att arbeta med. Ämnet kan utforskas när produkten är klar.

- *Stöd för mera än en applikation samtidigt*

  Eftersom TCP/IP-sockets hela tiden har använts av modulen, borde denna redan ha denna funktionalitet.

7.2.1.4 Säkerhetskrav

- *Den ende som skall kunna ändra i databasen är den systemansvarige hos kunden, via ett systemadministrationsverktyg*

  Denna punkt har ej varit aktuell att arbeta med. Ämnet kan utforskas när produkten är klar.

7.2.1.5 Konstruktionskrav

- *Felmeddelanden skall skickas som paket över TCP/IP*

Att en kund skall ha tillgång till ett nätverk med nämnda protokoll, blir aktuellt när produkten är klar att säljas.

- *Modulen skall vara kompatibel med Windows NT 4.0/98/2000*
  Denna punkt har ej varit aktuell att arbeta med. Ämnet kan utforskas när produkten är klar.

7.2.1.6 Användarens kapacitet och kompetens

- *En helt oinitierad användare skall kunna använda sig av modulens tjänster*
  Denna punkt har anknytning till hur ett textmeddelande visas, och blir aktuell vid det vidare arbetet av hur texten skall formuleras.

7.2.2 Övrig information

- *Kravspecifikationen omfattar ej handdatorer*
  Om ISAB anser att den färdiga produkten har den funktionalitet som önskats, kan modulen vidareutvecklas till att även kunna kontaktas av handdatorer.

7.3 Rekommendationer samt förslag beträffande vissa avsnitt

Under rubrik 7.2.1 finns angivet: ”Modulen är implementerad så att den kan buffra ett felmeddelande i objekt av paketeringsklassen FPaket.”

Om så önskas går det att lägga till/ta bort variabler, vad beträffar paketeringsklassen FPaket. Variablerna är definierade i FPaket.h, i klassdefinitionens privata del. Om tillägg sker i ett paket, exempelvis ett nytt fält: ”nnnn”, kan en ny variabel läggas till i paketeringsklassen. Då kan det också läggas till en metod set_Nytt fält för att lagra ”nnnn”, samt en metod get_Nytt fält för att kunna hämta innehållet ”nnnn” från variabeln.

Om annan funktionalitet förutom packnings-/uppackningsmetoder önskas från paketeringsklassen, är det möjligt att lägga till denna till nuvarande implementation. Dessutom behövs troligen läggas till en metod på klientsidan, för att kunna packa upp svaret från servern (se kapitel 6.3.1.3).

Alla namn på variabler, metoder och dylikt, är troligen inte enligt ISABs standard. Därför kan vid fortsatt arbete med modulen rekommenderas att byta dessa namn.

Under rubrik 7.2.1.1 står: ”Gränssnittet mot användarapplikationen är färdigt, och TCP/IP kan användas”.

Gränssnittet är färdigt så till vida att en klient har möjlighet att få kontakt med servermodulen. För att få den önskade funktionaliteten av denna kontakt, behövdes ett

textformat utvecklas för att kunna skicka ett paket över protokollet TCP/IP. Detta textformat har fått uppbyggnaden av en vektor, vars innehåll består av ett antal fält. Fälten har bestämda storlekar, och varje fält kan innehålla text och heltal. För att vidareutveckla vektorn är det lämpligt att fortsätta bygga på med ytterligare fält, vars innehåll bestäms av ytterligare information som modulen kan behöva.

7.4 Gränssnitt mot databasen

För att göra det möjligt att använda en databas, eller flera stycken olika databaser kan det vara bra att använda ett generellt gränssnitt. Det är ganska opraktiskt att varje databas kräver ett särskilt gränssnitt, vid användning av flera olika databaser. Därför finns det ett ODBC-gränssnitt som är avsett att åtgärda det problemet. Dessutom var tanken att kunna använda ADO, som i sin tur använder ODBC, för att nå databaser som inte har ett OLE DB-gränssnitt (se bilaga D).

7.4.1 ODBC

ODBC Administrator användes för att konfigurera en ODBC-datakälla som pekar till databasen. För att göra databasen åtkomlig gavs den ett namn: ”SupportMan”, samt var databasen fanns lokaliserad. Ett GUI utformades som användes vid testkörning. Sedan plockades det bort, eftersom avsikten var att byta ut det mot ett GUI i ADO. Att konfigurera datakällan, samt att tillverka GUIt var imponerande enkelt med hjälp av boken ”Visual C++ på tre veckor” [5].

7.4.2 ADO

För att ansluta ADO till ODBC-datakällan skulle förutom denna också anges namn på databasen. Detta var ganska enkelt med hjälp av bokens [5] anvisningar. Däremot var det ganska krångligt att förstå sig på de kontroller som används, för att på olika sätt kunna kommunicera med databasen. Dessutom gick inte heller att få en anslutning till databasen ”SupportMan”, och orsaken till detta är fortfarande okänd.

Om ISAB bestämmer sig för att göra ett nytt försök att lägga till ADO, rekommenderar jag litteratur som behandlar ADO mycket mera utförligt.

Om företaget å andra sidan bestämmer sig för att nöja sig med ODBC, är kopplingen till databasen redan klar. Det som återstår är då att utveckla ett nytt GUI, med önskad funktionalitet för ODBC.

8 Problem

De största problem som har förekommit, har dels varit beträffande ADO-gränssnittet, och dels paketeringen av textsträngen.

Vad beträffar ADO gick det tyvärr inte att få fram varför det inte fungerade. Det som hände var, att vid försök att kontakta databasen returnerades ett negativt svar. (Rent programmeringsmässigt så returnerades värdet –1). Det återstår att ta reda på vid vidareutveckling av modulen, om ISAB bestämmer sig för att utreda den saken, om man skall hålla fast vid att använda ADO. Vad beträffar problemet med ADO-gränssnittet, är det möjligt att det går att lösa genom att ta reda på ordentligt hur ADO egentligen fungerar. Det finns troligen bättre och utförligare litteratur än den som använts i detta arbete

Angående paketeringen har problemet mest legat i att ett pakets beståndsdelar har lagts på de rätta indexerade platserna i vektorn.

Dessutom var valet av namn på paketeringsklassen lite olyckligt: FPaket. Det är lätt att förväxla med paketet som skall sändas över nätverket.

9 Slutsatser

De intervjuer som förekom i inledningsskedet, gav mängder med information och idéer. De frågor som fanns att utgå ifrån, ledde till intressanta diskussioner och förslag på lösningar. I många fall gav också diskussionerna nya problem och frågor.

På detta sätt framkom alltså en stor mängd av information. Den här fasen i beredningen var lätt att arbeta med. Det som var lättarbetat var att informationen inte behövde bedömas. Varken hur bra, eller dålig den var, utan bara samla ihop den.

Det som var svårare var att plocka ut de delar av intervjuerna som skulle lämpa sig för att ta med i konceptet. En ordentlig begränsning skulle göras, och det som verkade vara ”lagom” omfattande, visade sig vid närmare granskning vara större. Vid diskussionen med ISABs handledare beträffande vilka delar från intervjuerna som borde ingå i examensarbetet, gjordes en snävare begränsning som verkade vara mera realistisk.

Utifrån detta skulle sedan en kravspecifikation tas fram. I den skall varje detalj för sig tas fram, och vilka krav som gäller för projektet skall anges.

Sedan kunde implementationsdelen påbörjas. Microsoft Visual C++ hade förekommit som hjälpverktyg vid laborationer vid Karlstad Universitet, och mina kunskaper om applikationen var ganska grundläggande. Jag hade implementerat kod, kompilerat och kört mindre program. Nu fick jag lära mig hur ett större projekt skapades, och hur felsökning med hjälp av debugger sker. Det var intressant, och just debuggern visade sig vara ett oumbärligt verktyg.

När det gäller kopplingen ADO-ODBC-databas var det synd att det inte gick att arbeta med ADO. Det verkade så enkelt att få till med hjälp av boken [5], men i praktiken fungerade det inte alls. Trots all hjälp och assistans av ISABs programmerare, visade det sig att tillräckliga kunskaper saknades om ADO. Kontentan av detta: Att det fungerar i teorin, säger inte att det kommer att fungera i praktiken. Däremot var det inga större problem med att arbeta fram kopplingen mellan ODBC och databasen. Jag hade gjort en tabell i Access som jag kunde ställa en SQL-fråga till och få ett svar därifrån.

Jag tycker att jag lärt mig mycket av exjobbet. Särskilt för att jag arbetat med ett helt projekt från beredningsfas, till implementerings- och testfas. Varje del av projektet har gett nya erfarenheter: Beredningsfasen med att det är viktigt att kunna avgränsa arbetet, och hur svårt det är att göra en kravspecifikation. Implementationsfasen med att det aldrig är så enkelt som det verkar när instruktionerna i en bok följs. Det innebar också att tidplanen inte riktigt

höll för denna del av projektet. Och slutligen testfasen som löpt parallellt med implementationsfasen, och som gav vissa insikter: Sådant som ”borde” fungera, kanske inte alls gjorde det, eller i alla fall inte som det var tänkt. Men till sist kändes det mycket tillfredsställande att kunna ställa en fråga till databasen, och att få det svar jag förväntade mig.

Därför tror och hoppas jag att mitt arbete kan ligga till grund för fortsatta planer avseende modulen.

Förkortningar

GUI = Graphical User Interface

SQL = Structured Query Language

LAN = Local Area Network

TCP/IP = Transport Control Protocol / Internet Protocol

ADO = ActiveX Data Objects

ODBC = Open DataBase Connector

OLE = Object Linking and Embedding

Referenser

[1] ”Nyheter och allmän information” , informationsblad från ISAB.

[2] Bilderna är tagna på Stora Enso. Fotograf: Sylvester Rowinski, ISAB.

[3] Systemutveckling – principer, metoder och tekniker (Erling S Andersen). ISBN 91-44-31042-0, Studentlitteratur, Lund 1994

[4] Texten är hämtad ur kapitlet om ADO i boken Lär dig Visual C++ på 3 veckor (Davis Chapman). ISBN 91-636-0520-1, Pagina Förlags AB, 2000.

[5] Lär dig Visual C++ på 3 veckor (Davis Chapman). ISBN 91-636-0520-1, Pagina Förlags AB, 2000.

Ordlista

Saftblandare: Roterande varningsljus.

Pinga: Skicka ett meddelande från en dator till en annan för att kolla att den finns, är ansluten och svarar på tilltal. Ordet är troligen hämtat från ekolodning, där ljudet "ping" tyder på en träff.

Mode: Läge.

Felsupport: Samma betydelse som åtgärdsförslag.

Definitioner från kravspecifikationen:

Användare: Med användare menas en människa som använder en applikation.

Användarapplikation: En applikation (programvara) som en datoranvändare arbetar med.

Felmeddelande: Ett textmeddelande som visas på en datorskärm

”Tyst” mode: När en felkod skickats från en klient och servern returnerat ett svar så kopplar servern ner förbindelsen.

”Icke tyst” mode: Servern kan ta emot ytterligare förfrågningar från samma klient, och sända ytterligare svar. Om servern tar emot förfrågning nummer två, och denna visar sig vara att klienten meddelar att inget mera önskas, kopplar servern ner förbindelsen.

Intervjuer

BILAGA A

Vad vill användaren ha? Vad skall visas?

1) Tydlighet!!! Korta, koncisa meddelanden. Det får inte finnas tvetydigheter, utan det ska framgå klart vad som menas.

2) Klartext, kombinerat med kod. Koden är inte intressant för användaren, men kan användas när kontakt med supporten måste tas.

3) Hjälprutan skall ligga överst hela tiden, så att den inte försvinner när man klickar i den applikation man använder för tillfället.

4) Hjälprutan skall dock vara flyttbar, så att den inte lägger sig över den text som är viktig.

5) Meddelanden skall skrivas på svenska. Engelska ord får användas när det inte finns något motsvarande uttryck på svenska.

6) Om man sett ett meddelande ett antal gånger vill man ej se det mera. Antal gånger användaren får försöka. Begränsning?

7) Instruktionstext skall komma upp när användaren gjort sitt val om han/hon vill försöka olika åtgärder.

8) Översikt av möjliga lösningar. Möjlighet att välja. Detaljerad felbeskrivning visas sedan.

9) Möjlighet för en kund att lägga till interna meddelanden. Kunden (systemansvarige) skall kunna komplettera felhanteringstexten från den globala databasen. Exempel på text: ”Kontakta Nisse på IT-avdelningen, telnr, o s v”.

10) Användaren kanske inte sitter vid en arbetsstation hela tiden. Hur skall då ett fel visas? Kan ett larm kopplas till någons telefon? Kan ett meddelande skickas till systemansvarig? Det måste också bero på hur allvarligt felet är.

11) Om en fil saknas för att kunna utföra ett arbete, skall detta meddelas. Arbetet får inte bara försvinna för att något saknas. Det kan hända att den saknade filen bara har felaktig path el dyl.

12) Hellre enkel bild som säger det mesta, än en omständlig film.

13) Möjligen kan en film finnas som valmöjlighet om användaren skulle vilja se en.

14) En instruerande film kan upplevas som stressande för en användare som har bråttom. Svårt att hänga med instruktionerna och samtidigt utföra dem. En film kan också bli överpedagogisk?

15) En viktig synpunkt på film, ljud o s v är att det kanske inte är möjligt att använda i de äldre systemen som har mera begränsade resurser.

16) Enda gången det kan vara motiverat med film är vid nyinstallation, för att få en ”grunduppfattning” om hur man skall göra.

17) Felhantering med ljud är ej så bra p g a bullriga lokaler.

18) Felhantering med ljud kan motiveras i de fall ett fel kan innebära allvarliga problem. Då kan ”saftblandare” och/eller larmljud användas.

19) Vid exempelvis inloggningsproblem skall användaren kunna kontrollera om han/hon matat in korrekta uppgifter. Om så är fallet och användaren kanske inte har rättigheter ska meddelande komma upp. ”Här får du inte vara”. D v s möjligheter att visa upp inställningar och ge förslag.

20) Testfunktioner om en enhet är trasig ex: ”ScanDisc”. Användaren har möjlighet att göra vissa kontroller själv. Detta gör att han/hon inte behöver ringa efter support direkt → Kan spara tid för både användare och support.

21) Meddelande som kommer upp i stegform: 1) Kontrollera detta. 2) Kontrollera detta. Bilder, teckningar med möjlighet att backa tillbaka om man gjort fel, eller inte uppfattat allt.

22) Åtgärdslista vid fel: 1) Visa i text vilket fel som inträffat. 2) Visa knappar: Vill du gå igenom de möjliga åtgärder som finns? Där kan man också visa andra valmöjligheter för användaren. T ex Vill du se en film om hur du ska göra? Vill du se bilder och text? Vill du se endast text? Om du redan gått igenom åtgärdslistan, vad skall visas. Om användaren vill gå igenom åtgärdslistan och prövat alla de olika alternativen, skall längst ner i ”alternativslistan” stå exv: Ring systemansvarige.

23) Användaren skall också ha möjlighet att titta i en loggfil. Det kanske inte är nödvändigt att skicka över den till supporten.

Övrigt

Vid allvarligare fel som kräver experthjälp

Vid en krasch skall meddelande komma upp, även om det är så att användaren inte kan åtgärda felet. Om ett system däremot ”totalkraschar” går det naturligtvis inte att visa något felmeddelande.
(Om ett meddelande är tydligt och det inte finns någon möjlighet att användaren kan åtgärda ett fel räcker det med en OK-knapp).

Sådant som användaren inte bryr sig om

Hjälphänvisningen skall ligga i samma bibliotek som applikationen, fast i en särskild hjälpmapp. D v s ingen hjälp i en gemensam databas(server) som kan gå ned, och ingen användare kan få hjälp.

Vid överföringsfel av olika slag: Kolla om kontakt med nätverk är OK. Pinga defaultgateway. Kontrollera kontakt med skrivare, scanner o s v.

En allmän åtgärdslista skall finnas. De åtgärder som inte är aktuella för ett visst system gråmarkeras och kan inte användas. Detta för att förenkla framtagningen av åtgärdslista.

Kan alla program använda felhanteringen? Är den bakåtkompatibel?

Man lägger felhanteringen lokalt (databas(server)??, varje dator har egen felhantering??). Varje gång som användaren loggar in i systemet sker en uppkoppling mot en perifer databas för att kontrollera om ändringar gjorts. Om så är fallet uppdateras den lokala informationen.
Kommunikation till enhet. Exempel: Om en enhet ej svarar. Om en enhet svarar fel.
Returmeddelande vid fel: Information om vem användaren är, namn, nummer.
”Behörighetsproblem”.

Hur ska man kvittera meddelanden?
VIKTIGT: Logga. Felen kollas i logg och åtgärdas sedan. Loggfilerna skall ha ”bäst-före-datum” el liknande så att de gamla tas bort automatiskt. Hur detaljerat skall det loggas? Kan man ha en låg loggnivå om systemet fungerat felfritt. Går det att automatiskt öka till högre loggnivå om systemet ”märker” att något ej är OK?

Felhanteringen får inte vara så avancerad med exv ljud och bild att inte äldre system kan använda den.

Om det inte går att överföra data från en enhet till en annan ska man kunna lagra den tillfälligt. Det måste vara möjligt att kunna skicka den senare om det är kommunikationsproblem. Användaren skall kunna veta att inget jobb försvinner → Trygghet.

Betyder ett felnummer samma sak hos alla kunder? Behövs en kundunik tabell läggas i databasen? Det kanske inte räcker att endast skicka ett felnummerpaket som inparameter till servern. Man kanske måste lägga till flera parametrar? Exempelvis ID-nummer till en arbetsstation.
En arbetsplats kan ha flera skrivare. På en plats används en bläckstråleskrivare, på en annan laserskrivare. Skall ett felnummer kopplas till generell ”Skrivare”, eller kan man koppla ett arbetsplatsnummer till felmeddelandet. Tabellen kan innehålla ID-nummer, koppling till vilken skrivare som finns just vid den arbetsstationen. Hur ska detta lösas?

Krav:

1) Skicka felmeddelande 111. Modulen drar igång presentation. Klar. Returnera OK. Returnera Ej OK om presentationen ej gick att visa.
2) Bra alternativ ur teknikers synpunkt. En ”global” modul att underhålla, slippa åka till företag om databasen strular.

Skicka felmeddelande 111. Returnera: Så här ska du göra; exv Message Box, eller applikationen skall dra igång något annat. ”Arbetet utförs hos användarapplikationen”.

Tekniska begränsningar
Ytstorlek på skärmen. Vid ”normal skärmstorlek” ett grafiskt gränssnitt.
Truckdatorskärm. Storlek ca 14’’.
Handdator. Skärm? ”Fel på Movex” i text.

Äldre system med dåliga resurser.

Krav på databasservern. Kapacitet.

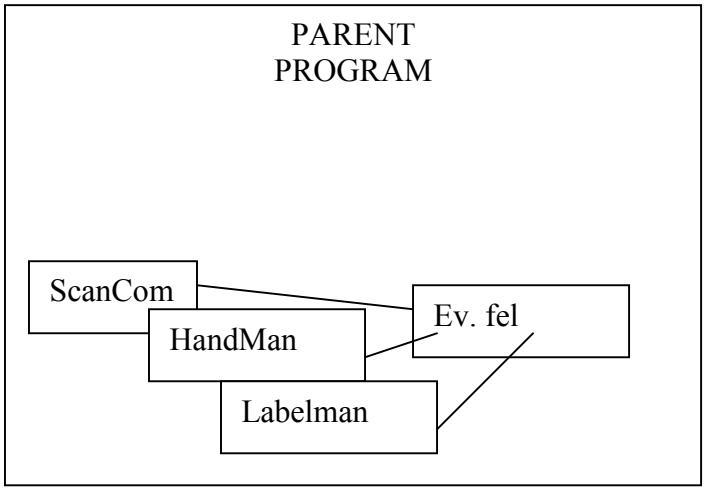
Framtiden
Möjligheter att visa upp inställningar och ge förslag. Exempelvis om fel uppgifter är inmatade någonstans. Skall instruktioner visas för hur det skall se ut för att fungera?
Sinnen? Defekt färgseende? Nedsatt hörsel o s v.
Möjligheter till specialanpassning? Andra fysiska handikapp?

Om en användare försökt alla åtgärder och skall kontakta support: En särskild knapp för att skicka loggfilerna: ”Skicka loggfiler”, automatiskt till supporten med exv email. Filerna kan komprimeras automatiskt innan sändning. Om användaren inte har internetbehörighet kan loggfilerna skickas till systemansvarige el dyl. → Skicka vidare. Då behöver inte supporten ägna tid åt att tala om för användaren hur man skall skicka filerna. Supporten kan titta på loggfilerna i lugn och ro, slipper sitta i telefon med användaren. Användaren har kanske någon annan arbetsuppgift att utföra under tiden, supporten kan ringa upp så snart som möjligt istället.
Det kanske finns möjlighet att det automatiskt väljs ut vilka loggfiler som skall skickas, dvs inte alla, utan endast de viktigaste som berör felet som inträffat???

Kunna välja mellan olika varianter. 1) Välj handdator. 2) PC med grafiskt gränssnitt.

Felmeddelande till handdator: Exv ”Får ej tag i överordnat system”, om det är problem vid överföring av information. Då vet användaren att gå till stationär dator, eller systemansvarige?

Avgränsa en viss operation från programmoduler. Om man stänger ner
Parentfönstret följer Childfönstren med. Felhanteringen kopplas direkt till Childrenprogram.



1. Allmänt

Introduktion

Många användare upplever felmeddelanden som kryptiska och obegripliga. Därför vill ISAB ta fram en modul som visar dessa felmeddelanden på ett tydligare sätt. Det är också meningen att instruktioner för att hantera fel skall samlas på samma ställe, dvs i en modul. Modulen skall fungera som en server där en databas skall lagra instruktionerna. Flera applikationer och användare skall kunna använda databasen samtidigt.
De användare som avses är ISABs kunder och de anställda hos en kund.

Översiktlig beskrivning

När en applikation vill visa felmeddelande för användaren, skickas ett paket innehållande felkoden över TCP/IP. Modulen tar emot felkoden och visar en lämplig åtgärd mot felet. Åtgärden kan presenteras som text, ljud eller bild.

2. Referenser

Lista på dokument: I) Förslag på vad en användare vill ha.

3. Definitioner

- Med en användare avses den som använder applikationen
- Med en användarapplikation avses den applikation som en användare arbetar med
- Med ett felmeddelande avses ett textmeddelande som visas på en datorskärm
- Med ”tyst” mode avses att när servern svarat med respons, så förväntar den sig inte någon reaktion från användaren
- Med ”icke tyst” mode (interaktivt) menas att servern svarar med att visa ett meddelande, och väntar på reaktion från användaren. När användaren gjort någon form av tangenttryckning skickar servern OK tillbaka

4. Tekniska krav

Funktionella krav:

- Modulen skall ta emot felmeddelanden i form av kod från användare/applikationsanvändare
- Felmeddelanden skall tas emot via TCP/IP sockets
- Programvaran skall visa felmeddelanden i text för användaren, eller hantera felkod enligt två modeller:
  - Modulen skall kunna konfigureras i ”tyst” mode
  - Modulen skall kunna konfigureras i ”icke tyst” mode
- Andra länder skall kunna få svar från modulen på sitt eget lands språk

Ickefunktionella krav:

- Svarstiden är av betydelse - klienten vill ha snabb respons
- Modulens svar på ett felmeddelande skall vara kortfattat
- Modulens svar på ett felmeddelande skall vara otvetydigt
- Databasen skall endast vara åtkomlig för ändringar för systemansvarig hos kunden

4.1 Yttre gränsytor

Mot användaren:

- Felmeddelandet skall ligga överst, tills användaren valt att ta bort det
- Felmeddelandet skall vara flyttbart över skärmytan

Mot användarapplikationen:

- Modulen skall använda sig av TCP/IP, dvs ett socketbaserat protokoll

4.2 Funktionskrav

1) Modulen skall fungera som en server som tar emot meddelanden
2) Modulen skall lagra information i en databas
3) Modulen skall skicka ”tyst”/”icke tyst” respons tillbaka till användaren/applikationsanvändaren enligt protokollet
4) Modulen skall ha möjlighet att lägga till ljud- och bildstöd i etapp 2

4.2.1

Modulen skall kunna serva flera användare samtidigt.

4.2.2

Databasen skall kunna lagra text och bilder, för att kunna ge stöd till användaren.

4.2.3

Modulen skall kvittera användares/applikationsanvändares meddelanden.

4.3 Samband mellan funktionerna

Se 4.2

4.4 Driftskrav

Se dok: ISABdrift.pdf

4.5 Kapacitetskrav

- Antal användare som samtidigt skall kunna komma åt systemet skall endast begränsas av nätverkets kapacitet
- Stöd för mera än en applikation samtidigt

4.6 Säkerhetskrav

- Den ende som skall kunna ändra i databasen är den systemansvarige hos kunden, via ett systemadministrationsverktyg

4.7 Konstruktionskrav

- Felmeddelanden skall skickas som paket över TCP/IP.
- Modulen skall vara kompatibel med Windows NT 4.0 / 98 / 2000

4.8 Användarens kapacitet och kompetens

- En helt oinitierad användare skall kunna använda sig av modulens tjänster

4.9 Övriga krav

5 Övrig information

Denna kravspecifikation omfattar ej handdatorer.

6. Checklista för granskning av kravspecifikation

Allmänt

- Bristande information i ett dokument är ett fel
- Överflödig information i ett dokument är ett fel
- Alla rubriker enligt mallen skall finnas med. Är en huvudrubrik inte tillämplig skall den vara kvar med texten ”Ej tillämplig” och ev underrubriker slopas

1. Identifieras programvaran entydigt?
2. Saknas något krav?
3. Behövs alla krav?
4. Finns motstridiga krav?
5. Finns kravet bara på ett ställe i dokumentet?
6. Är alla formuleringar klara och entydiga?
7. Finns alla nödvändiga definitioner?
8. Är något krav för detaljerat beskrivet?
9. Är något krav för allmänt beskrivet?
10. Begriper beställaren kravspecifikationen?
11. Är alla krav testbara och mätbara?
12. Används ordet **skall** för att beskriva absoluta krav.
13. Förekommer onödig text eller onödiga figurer?
14. Anges kraven på alla yttre gränssnitt?
15. Anges krav på all funktionalitet?
16. Följer kravspecifikationen mallen?

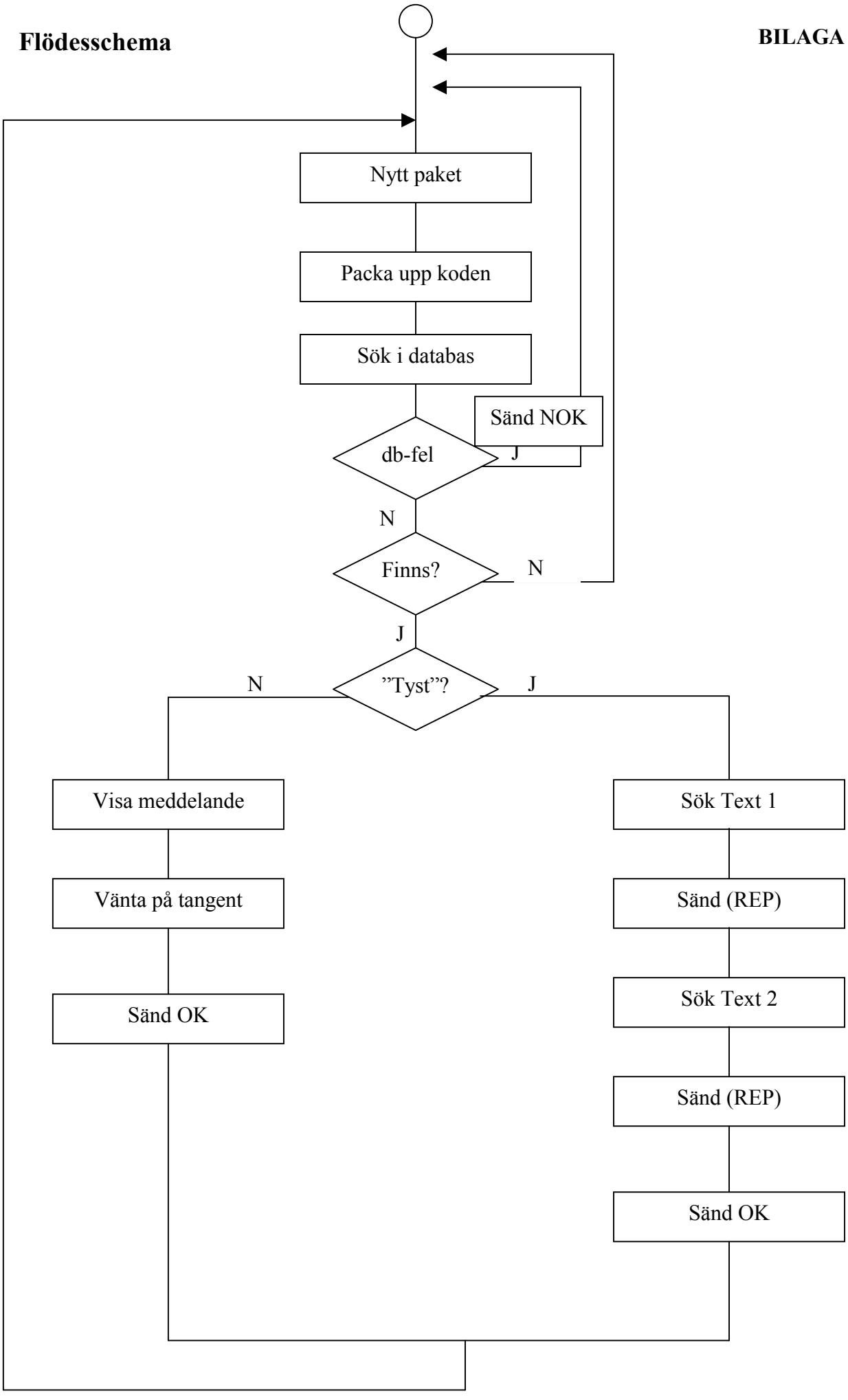
7. Råd vid upprättandet

Målet skall vara att:

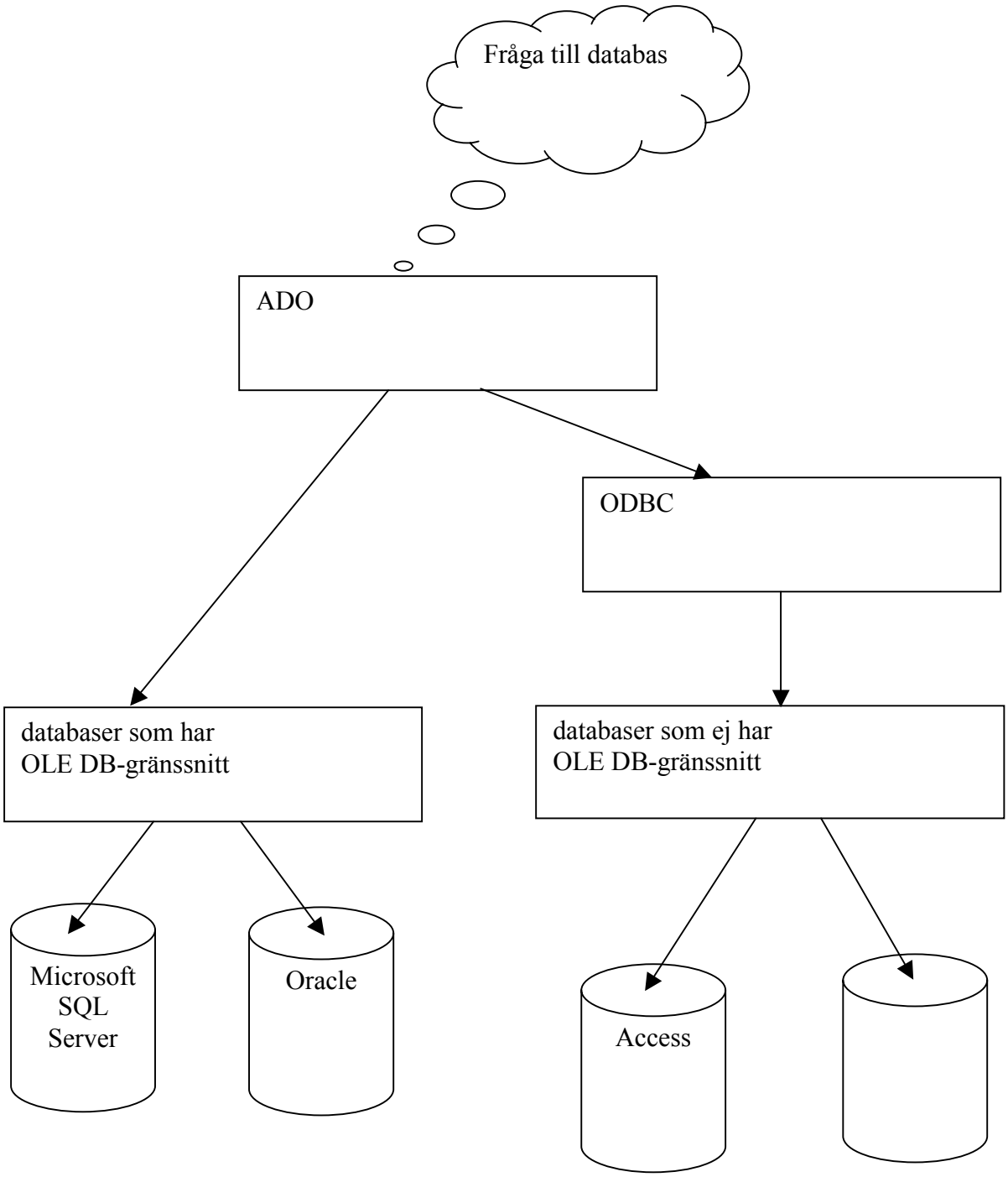
- beskriva systemets yttre egenskaper.
- den endast innehåller mätbara krav.
- kraven bör numreras för att underlätta spårning och testning.
- den kan ange prioriteringar för olika krav.

Att tänka på:

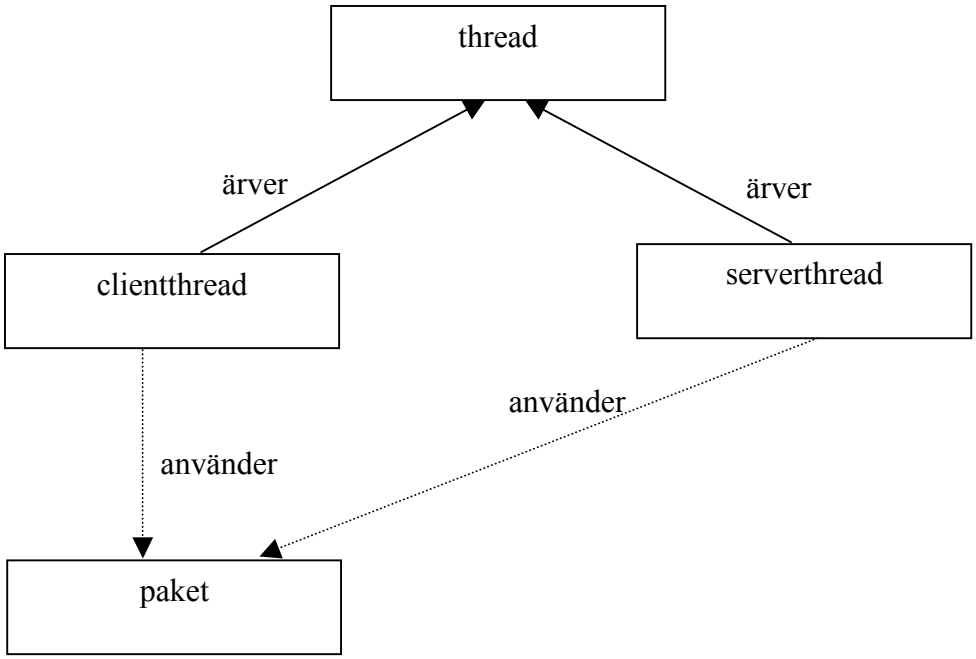
- Identifiera kunden. Ta reda på vem som egentligen bestämmer.
- Se till att prata med rätt personer redan från början.
- Analysera det existerande systemet. Gärna grafiskt. Överensstämmer inte denna bild med kundens syn på det existerande systemet har du problem.
- Intervjua användare.
- Hur ser omgivningen ut? Ställer den speciella krav på utrustning mm.
- Gör en hierarkisk problembeskrivning. Beskriv problemet på alltmer detaljerade nivåer. Tala om vad som skall och inte skall lösas på respektive nivå. Du skall dock **inte beskriva** hur.
- Dokumentera. Använd ett icke tekniskt språk som kunden förstår.
- Låt kunden känna ansvar och delaktighet i kravspecifikationen. Låt kunden fortlöpande godkänna och komma med synpunkter under arbetets gång.
- Kravspecifikationen är inget designdokument, det säger inget om hur problemen löses.
- Allt som systemet skall kunna göra är specificerat i kravspecifikationen.
- Kraven som specificeras är skrivna på ett sådant sätt att man kan kontrollera den färdiga programvaran mot den.

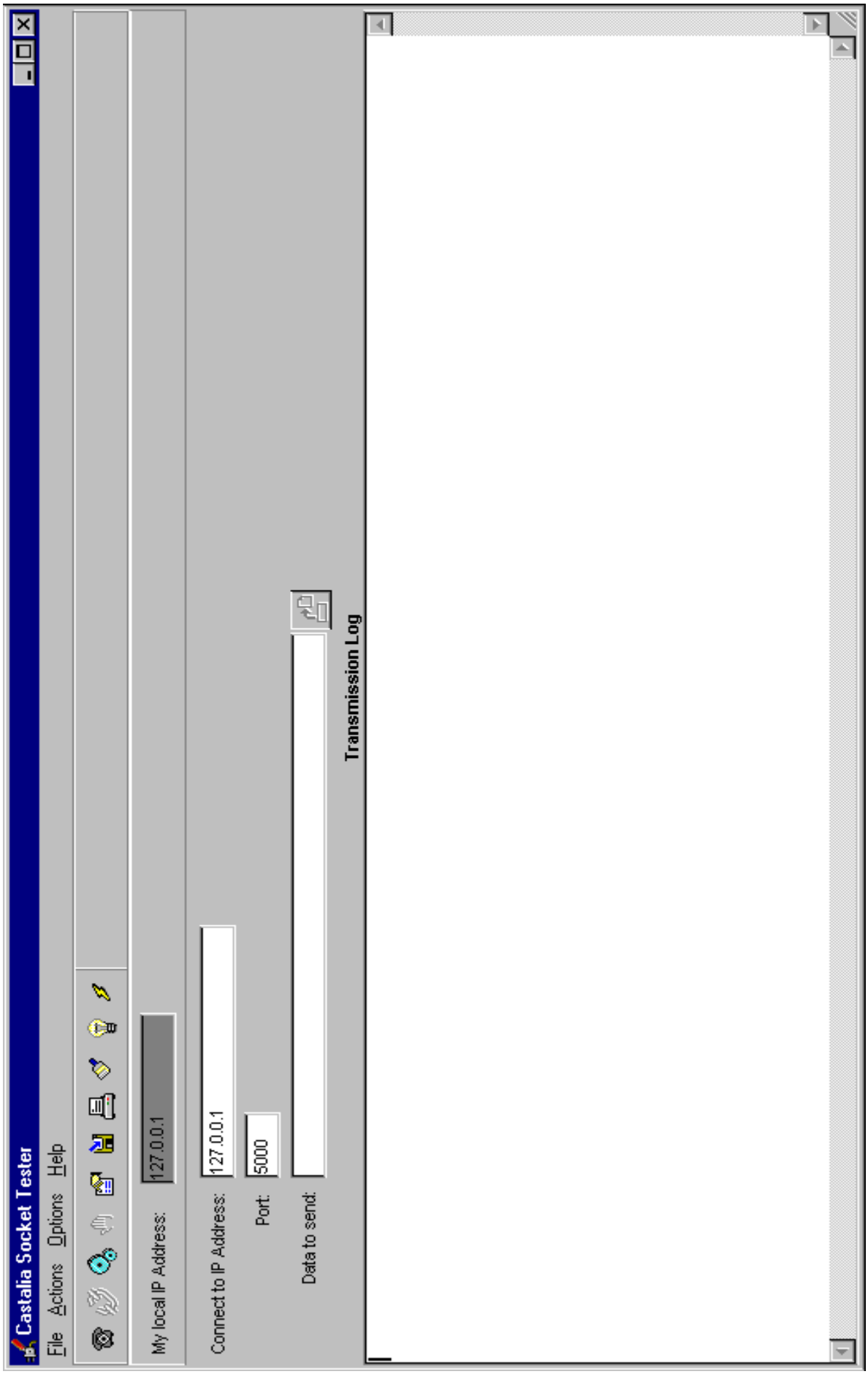

Nytt paket
Packa upp koden
Sök i databas
Sänd NOK
db-fel
J
N
Finns?
N
J
N
”Tyst”?
J
Visa meddelande
Vänta på tangent
Sänd OK
Sök Text 1
Sänd (REP)
Sök Text 2
Sänd (REP)
Sänd OK

Databasgränssnitt



Klassdiagram




Castalia Socket Tester
File Actions Options Help
My local IP Address:
127.0.0.1
Connect to IP Address:
127.0.0.1
Port:
5000
Data to send:
Transmission Log

Tabellen från testdatabasen

Bilaga G

HELP

ID	Data
1	Kasta ut datorn
2	Dansa runt en sväng
3	Ta ledigt
4	Drick kaffe

Paketformat

Textsträng från klientsidan

Beskrivning	Obligatoriskt	Från	Till	Längd	Kommentar
Meddelande	Ja	0	14	15	Typ av förfrågan
Sekvensnummer	Ja	15	17	3	Vilket paket i ordningen
Programnummer	Ja	18	21	4	Vilken applikation som genererat en felkod
Feltyp	Ja	22	28	7	Vilken felkod som avses
Felnivå	Ja	29	29	1	Allvarlighetsgrad

Textsträng som svar från serversidan

Beskrivning	Obligatoriskt	Från	Till	Längd	Kommentar
Flagga	Ja	0	14	15	OK/NOK
Sekvensnummer	Ja	15	17	3	Samma som sekvensnummer från klientsidan
Text1	Nej	18	179	100	Instruerande text från serversidan

// File : main.cpp

#include <windows.h>

```
#include <shellapi.h>
#include "resource.h"
#include "serverthread.h"

#define WM_SYSTRAY_MSG           (WM_USER + 1)

static NOTIFYICONDATA gNid;
static GServerThread* gServerThread;

/*************************************************************************
***/
/** Hanterar meddelanden från dialog */
INT_PTR CALLBACK MainDialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
           if(uMsg == WM_COMMAND) EndDialog(hwndDlg, FALSE);
           return DefWindowProc(hwndDlg, uMsg, wParam, lParam);
}

/*************************************************************************/
/** Anropas när användaren dubbleklickar på systray ikonen */
void OnSysTrayDblClick(HWND aHwnd)
{
           INT_PTR sResult = DialogBox(
                      (HINSTANCE)GetWindowLong(aHwnd, GWL_HINSTANCE),
                      MAKEINTRESOURCE(IDD_MAIN_DIALOG),
                      aHwnd,
                      (DLGPROC)MainDialogProc
                      );
           if(sResult == FALSE)
           {
                      DestroyWindow(aHwnd);
           }
}

/*************************************************************************/
/** Skapar server objekt och systray ikon */
BOOL OnCreate(HWND aHwnd)
{
           // skapa server objekt
           gServerThread = new GServerThread(6543);
           if(!gServerThread) return FALSE;

           // starta server objekt
           gServerThread->Start();

           // lägg till systray ikon
           ZeroMemory(&gNid, sizeof(gNid));
           gNid.cbSize = sizeof(gNid);
           gNid.hWnd = aHwnd;
           gNid.uID = 0;
           gNid.uFlags = NIF_ICON | NIF_MESSAGE | NIF_TIP;
           gNid.uCallbackMessage = WM_SYSTRAY_MSG;
           gNid.hIcon = LoadIcon((HINSTANCE)GetWindowLong(aHwnd,
GWL_HINSTANCE), MAKEINTRESOURCE(IDI_SYSTRAY_ICON));
           strcpy(gNid.szTip, "ISAB Felhantering");
```

```
            Shell_NotifyIcon(NIM_ADD, &gNid);

            return TRUE;
}

/*************************************************************************
***/
/** Stänger server objekt och tar bort systray ikon */
void OnRelease(HWND aHwnd)
{
            Shell_NotifyIcon(NIM_DELETE, &gNid);
}

/*************************************************************************
***/
/** Anropas för varje meddelande som går till huvuddialogen */
LONG CALLBACK MessageWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
            LONG sResult = TRUE;
            switch(uMsg)
            {
            case WM_CREATE:
                        sResult = OnCreate(hWnd);
                        break;

            case WM_SYSTRAY_MSG:
                        if(lParam == WM_LBUTTONDBLCLK)
OnSysTrayDblClick(hWnd);
                        break;

            case WM_CLOSE:
                        DestroyWindow(hWnd);
                        break;

            case WM_DESTROY:
                        OnRelease(hWnd);
                        PostQuitMessage(0);
                        break;

            default:
                        sResult = DefWindowProc(hWnd, uMsg, wParam, lParam);
                        break;
            }
            return sResult;

}

/*************************************************************************
***/
/** Huvudfunktion som anropas vid start av programmet */
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE pInst, LPSTR szCmdLine, INT
nCmdShow)
{
            WNDCLASS sWc;
            HWND sHwnd;
            MSG sMsg;

            // skapa applikationsklass
            memset(&sWc, 0, sizeof(sWc));
            sWc.style = 0;
```

```
        sWc.lpfnWndProc = MessageWndProc;
        sWc.hbrBackground = (HBRUSH)(COLOR_BTNFACE + 1);
        sWc.hInstance = hInst;
        sWc.lpszClassName = "ISAB_MsgPump";
        RegisterClass(&sWc);

        // skapa dolt fönster för att hantera windows meddelanden
        sHwnd = CreateWindow(
                    "ISAB_MsgPump",
                    "ISAB Felhantering",
                    WS_POPUP,
                    0,
                    0,
                    0,
                    0,
                    NULL,
                    NULL,
                    hInst,
                    NULL);

        // meddelande loop
        while(GetMessage(&sMsg, sHwnd, 0, 0) > 0)
        {
                    TranslateMessage(&sMsg);
                    DispatchMessage(&sMsg);
        }

        return 0;
}
```

```
// File         : thread.h
// Description  : Abstraktion av trådar
// Rev          : 0.0
// Created      : 14 feb 2001
// Author       : Anders Pistol
// Mail         : anders.pistol@isab.se
//
//
// Copyright Industrisystem i Karlskoga AB
// Badstugatan 24A
// SE-691 32 Karlskoga
// Sweden
// tel int +46 (586) 596 00
// fax int +46 (586) 596 13
//
// The copyright to the computer program(s) herin
// is the property of Industrisystem i Karlskoga AB, Sweden.
// The program(s) may be used and/or copied only with
// the written permission of Industrisystem i Karlskoga AB
// or in accordance with terms and conditions stipulated
// in the agreement/contract under which the program(s)
// have been supplied.
//
#ifndef _THREAD
#define _THREAD

class GThread
{
protected:
  void* mThread;                //m:medlemsvariabel

public:
          GThread();                        //constructor, prefix G

public:
  int Start();                              //metoder
  int Stop();
  int Pause();
  int Resume();
          bool Running();                   //flagga

public:
  static void Sleep(int aTimeMilliSec);             //a:argument

public:
  virtual int Init();
  virtual void Close();
  virtual int Run();
};

#endif  // _THREAD
```

```
// File         : thread.cpp
// Description  : Standard tråd hantering under Win32
// Rev          : 0.0
// Created      : 14 feb 2001
// Author       : Anders Pistol
// Mail         : anders.pistol@isab.se
//
//
// Copyright Industrisystem i Karlskoga AB
// Badstugatan 24A
// SE-691 32 Karlskoga
// Sweden
// tel int +46 (586) 596 00
// fax int +46 (586) 596 13
//
// The copyright to the computer program(s) herin
// is the property of Industrisystem i Karlskoga AB, Sweden.
// The program(s) may be used and/or copied only with
// the written permission of Industrisystem i Karlskoga AB
// or in accordance with terms and conditions stipulated
// in the agreement/contract under which the program(s)
// have been supplied.
//
#include <windows.h>
#include "thread.h"
#include <string.h>

/*************************************************************************
***/
/** Avfyrningsramp för tråd */
static DWORD WINAPI ThreadDispatcher(LPVOID aThreadObject)
           //ulong 32b
{
           GThread* sThreadObject = (GThread *)aThreadObject;
                      //pekare till trådobj
           DWORD sReturnValue = 0;
           if(sThreadObject != 0)

           //om pekaren har obj att peka till
           {
                      if((sReturnValue = sThreadObject->Init()) == 0)
                                 //ptr t returv virt int
                      {
                                 sReturnValue = (DWORD)sThreadObject-
>Run();                          //   ", cast till ulong
                                 sThreadObject->Close();

           //stoppa tråd
                      }
           }
           return sReturnValue;

}

/*************************************************************************
***/
/** CTOR. Rensar variabler */
GThread::GThread()              //konstruktor
{
           mThread = 0;                    //init m:medlemsvar ptr till
voidfunkt
```

```
}

/*************************************************************************
***/
/** Startar tråd objekt */
int GThread::Start()
{
  DWORD sThreadId;                          //ulong
  mThread = (void *)new HANDLE;
  *(HANDLE *)mThread = CreateThread(
    NULL,
    0,
    (LPTHREAD_START_ROUTINE)ThreadDispatcher,
    (LPVOID)this,
    0,
    &sThreadId
  );
  return 0;
}

/*************************************************************************
***/
/** Stoppar tråd */
int GThread::Stop()
{
           TerminateThread(*(HANDLE *)mThread, 0);
  return 0;
}

/*************************************************************************
***/
/** Pausar tråd, kan startas åter mha Resume() */
int GThread::Pause()
{
           SuspendThread(*(HANDLE *)mThread);
  return 0;
}

/*************************************************************************
***/
/** Startar tråd som pausats mha Pause(). */
int GThread::Resume()
{
           ResumeThread(*(HANDLE *)mThread);
  return 0;
}

/*************************************************************************
***/
/** Returnerar sant om tråden körs */
bool GThread::Running()
{
           DWORD sResult;
           GetExitCodeThread(*(HANDLE *)mThread, &sResult);
           return (sResult == STILL_ACTIVE);
}

/*************************************************************************
***/
/** Låter tråden "vila" i X antal millisekunder */
void GThread::Sleep(int aTimeMilliSec)
```

```
{
            ::Sleep(aTimeMilliSec);
}

/*************************************************************************
***/
/** Tom funktion för virtuella Init() */
int GThread::Init()
{
            return 0;
}

/*************************************************************************
***/
/** Tom funktion för virtuella Close() */
void GThread::Close()
{
}

/*************************************************************************
***/
/** Tom funktion för virtuella Run() */
int GThread::Run()
{
            return 0;
}
```

```
// File : serverthread.h

#ifndef _SERVERTHREAD_H
#define _SERVERTHREAD_H

#include <winsock.h>
#include <list>
#include "thread.h"
#include "clientthread.h"

class GServerThread: public GThread
{
private:
           SOCKET     mServerSocket;
           WORD                  mListenPort;
           std::list<GClientThread*> mClientThreads;
           bool                  mQuitLoop;

public:
           GServerThread(WORD aListenPort):           //constructor med 16b
portnr
                      mListenPort(aListenPort),
                      mQuitLoop(false)
                      {}

public:
           int Stop();

public:
  virtual int                    Init();
  virtual void        Close();
  virtual int                    Run();
};

#endif     // _SERVERTHREAD_H
```

```
// File   : serverthread.cpp

#include "serverthread.h"
using namespace std;

/*************************************************************************
***/
/** Anropas när programmet vill stoppa denna tråd */
int GServerThread::Stop()
{
          mQuitLoop = true;
          if(WaitForSingleObject(*(HANDLE *)mThread, 5000) !=
WAIT_OBJECT_0)
          {
                    // Tråd låst, använder systemanrop för att avsluta...
                    return GThread::Stop();
          }
          return 0;
}

/*************************************************************************
***/
/** Initierar server tråd */
int GServerThread::Init()
{
          struct sockaddr_in sLocalAddr;              //serverstruct
          WSADATA sWsaData;

          // starta winsock
          if(WSAStartup(MAKEWORD(1, 1), &sWsaData) != 0)
          {
                    // !!! FEL !!!
                    return 1;
          }

          // skapa socket handtag för inkommande anslutningar
          mServerSocket = socket(AF_INET, SOCK_STREAM, 0);
          if(mServerSocket < 0)
          {
                    // !!! FEL !!!
                    return 1;
          }

          // ändra så att socket återanvänder addresser effektivare
          setsockopt(mServerSocket, SOL_SOCKET, SO_REUSEADDR, "TRUE",
sizeof(int));

          // bygg adress struktur
          memset(&sLocalAddr, 0, sizeof(sLocalAddr));
          sLocalAddr.sin_family = AF_INET;                        //sätt
structvariabler
          sLocalAddr.sin_port = htons(mListenPort);
          sLocalAddr.sin_addr.s_addr = htonl(INADDR_ANY);

          // bind adress till anslutning
          if(bind(mServerSocket, (LPSOCKADDR)&sLocalAddr,
sizeof(sLocalAddr)) < 0)
          {
                    // !!! FEL !!!
                    return 1;
          }
```

```
		// aktivera anslutning
		if(listen(mServerSocket, SOMAXCONN) < 0)
		{
			// !!! FEL !!!
			return 1;
		}

		return 0;
}

/**************************************************************************
***/
/** Stänger server tråd och alla dess anslutningar */
void GServerThread::Close()
{
		list<GClientThread*>::iterator sIter;

		// gå igenom lista med klienter och avsluta dessa
		for(sIter = mClientThreads.begin(); sIter !=
mClientThreads.end(); sIter++)
		{
			GClientThread* sClientThread = (*sIter);
			if(sClientThread->Running() == true)
			{
				// tvingar klient att avsluta
				sClientThread->Stop();
			}
			delete sClientThread;
		}

		// stäng server anslutning
		closesocket(mServerSocket);
		WSACleanup();
}

/**************************************************************************
***/
/** Hanterar anslutningar */
int GServerThread::Run()
{
		list<GClientThread*>::iterator sIter;
		GClientThread* sClientThread;
		TIMEVAL sTo = { 0, 100 };
		SOCKET sClientSocket;
		fd_set sRdFds;

		while(mQuitLoop == false)
		{
			// kontrollera om några användare har avslutats
			if(mClientThreads.size() > 0)
			{
				sIter = mClientThreads.begin();
				do
				{
					sClientThread = (*sIter);
					if(sClientThread->Running() ==
false)
					{
						sIter =
mClientThreads.erase(sIter);
```

```
                                                delete sClientThread;
                                        }
                                        else
                                        {
                                                sIter++;
                                        }
                                }
                                while(sIter != mClientThreads.end());
                        }

                        // kontrollera om några användare väntar på att bli
accepterade
                        FD_ZERO(&sRdFds);
                        FD_SET(mServerSocket, &sRdFds);
                        if(select(0, &sRdFds, 0, 0, &sTo) > 0)
                        {
                                // acceptera användare
                                sClientSocket = accept(mServerSocket, 0,
0);
                                if(sClientSocket != INVALID_SOCKET)
                                {
                                        sClientThread = new
GClientThread(sClientSocket);
                                        sClientThread->Start();

            mClientThreads.push_back(sClientThread);
                                }
                        }
            }

            return 0;
}
```

// File : clientthread.h

```
#ifndef _CLIENTTHREAD_H
#define _CLIENTTHREAD_H

#include <winsock.h>
#include "thread.h"



class GClientThread: public GThread         //ärver GThread
{
private:
           SOCKET     mSocket;
           bool                  mQuitLoop;

public:
           GClientThread(SOCKET aSocket):   //constructor
                      mSocket(aSocket),
                      mQuitLoop(false)
                      {}


public:
           int        Stop();


public:
  virtual int                    Init();
  virtual void        Close();
  virtual int                    Run();
};

#endif     // _CLIENTTHREAD_H
```

// File : clientthread.cpp

```
#include "clientthread.h"
#include "FPaket.h"
#include <stdio.h>
#include <string.h>
#include "sqlext.h"



#define FORFRAGAN 15
#define SEKVENSNR 3
#define PROGNR 4
#define FELNR 7


/************************************************************************
***/
/** Anropas när server vill stoppa denna tråd */
int GClientThread::Stop()
{
           mQuitLoop = true;
           if(WaitForSingleObject(*(HANDLE *)mThread, 5000) !=
WAIT_OBJECT_0)
           {
                      // Tråd låst, använder systemanrop för att avsluta...
                      return GThread::Stop();
           }
           return 0;
}

/************************************************************************
***/
/** Initiera klient */
int GClientThread::Init()
{
           return 0;
}

/************************************************************************
***/
/** Stäng klient */
void GClientThread::Close()
{
}

/************************************************************************
***/
/** Utför data från klienten */
int GClientThread::Run()
{
           HENV sHenv;
           HDBC sHdbc;

           SQLAllocEnv(&sHenv);
           SQLAllocConnect(sHenv,&sHdbc);

           RETCODE sRetcode;
           char sFelText[200];
           //char sDsn[200];

           TIMEVAL sTo = { 1, 0 };
```

```
        fd_set sRdFds;


        while(mQuitLoop == false)
        {
                // kontrollera om någon data har kommit från klienten
                FD_ZERO(&sRdFds);
                FD_SET(mSocket, &sRdFds);

                if(select(0, &sRdFds, 0, 0, &sTo) > 0)
                {

                        FPaket Clientpaket;
                //      Clientpaket.set_mForFragan("GET_VERSION");
                //
        Clientpaket.set_mForFragan("SHOW_WARNING");
                        Clientpaket.set_mForFragan("SHOW_ERROR");
                        Clientpaket.set_mSekvensNr("123");
                        Clientpaket.set_mProgNr("4567");
                        Clientpaket.set_mFelNr("abcdefg");
                        Clientpaket.set_mFelNiva("1");

                        char Skickas[INBUFFERLENGD];
                //Skapa array för avsändning
                        memset(Skickas, '0', INBUFFERLENGD-1);
                        Clientpaket.mPacka(Skickas);
                //Packa array med fasta platser
                        int lengd=(strlen(Skickas));

                        char aInbuf[INBUFFERLENGD];

                //      strcpy(aInbuf,Skickas);
                                //Kopiera till ny array???


        //aInbuf=Skickas

                        char aUtbuf[INBUFFERLENGD];
                        memset(aUtbuf,'0',INBUFFERLENGD-1);
                        FPaket Serverpaket;

                        if(recv(mSocket, aInbuf, INBUFFERLENGD, 0)
<= 0)
                        {
                                break;
                        }

                //      Serverpaket.mPackaUpp(aInbuf);
                //Packa upp från fasta platser
                        Serverpaket.mPackaUpp(Skickas);

                        //Connection till db
                        sRetcode=SQLConnect(sHdbc, (unsigned char
*)"SupportMan",SQL_NTS,
                                        (unsigned char
*)"",SQL_NTS,(unsigned char *)"",SQL_NTS);

                        /*Fel i databasen*/
                        if(sRetcode==SQL_ERROR)
                        {
```

```
          SQLError(sHenv,sHdbc,SQL_NULL_HSTMT,NULL,NULL,(unsigned char
*)sFelText,sizeof(sFelText),NULL);

                                        SQLFreeConnect(sHdbc);
                                        SQLFreeEnv(sHenv);
                                        sHenv=NULL;
                                        sHdbc=NULL;
                              //        return;
                              }
                    /*Databas OK*/
                    else if(sRetcode==SQL_SUCCESS_WITH_INFO)
                    {
                              SQLError(sHenv,sHdbc,SQL_NULL_HSTMT, NULL,
NULL,(unsigned char *)sFelText,sizeof(sFelText),NULL);
                    }

                    HSTMT sStmt;          //"Handtag" till hämtad dbinfo
                    char sSQLKommando[200];
                    SDWORD cbTemp;


                    sRetcode=SQLAllocStmt(sHdbc,&sStmt);
                    //sprintf(sSQLKommando, "SELECT ID, Data FROM help");
                    sprintf(sSQLKommando, "SELECT id, Data FROM help
where ID=1");

                    /*(SQLHSTMTStatementHandle,SQLCHAR
*StatementText,SQLINTEGERTextLength);*/
                    sRetcode=SQLExecDirect(sStmt,(unsigned char
*)sSQLKommando, SQL_NTS);
                    LONG sID;
                    char Text1[500];

                    //Bind kolumn 1 för att ta heltal (ID)

          //(StatementHandle,Kolumnnummer,TargetType,TargetValuePtr,Buffer
längd,Stränglängd)

          sRetcode=SQLBindCol(sStmt,1,SQL_C_LONG,&sID,sizeof(sID),&cbTemp)
;

          sRetcode=SQLBindCol(sStmt,2,SQL_C_CHAR,&Text1,500,&cbTemp);

                    /*(SQLHSTMTStatementHandle,
          SQLUSMALLINTColumnNumber,SQLSMALLINTTargetType,

          SQLPOINTERTargetValuePtr,SQLINTEGERBufferLength,SQLINTEGER
*StrLen_or_IndPtr);*/

          //sRetcode=SQLBindCol(sStmt,2,SQL_C_CHAR,Text1,strlen(Text1),&cb
Temp);


                              sID=0;
                              sRetcode=SQLFetch(sStmt);

                              if((sRetcode==SQL_NO_DATA_FOUND) ||
(sRetcode==SQL_ERROR))
                              {
                                        break;
```

```
                                }
                                else
                                {
                                          char sData[100];
                                          sprintf(sData, "%ld",sID);

                                }

                    SQLFreeStmt(sStmt,SQL_DROP);
          //Avallokera minnesutrymmet för recset


                    SQLTransact(sHenv,sHdbc,SQL_ROLLBACK);
                    SQLDisconnect(sHdbc);
                    SQLFreeConnect(sHdbc);
                    SQLFreeEnv(&sHenv);

                    //Om svar från server OK
                              Serverpaket.set_mText1(Text1);
                              Serverpaket.set_mFlagga(1);
                              Serverpaket.mPackaSvar(aUtbuf);
          //Packar aUtbuf för sändning över nätverk

                              //sprintf(aUtbuf, "%s %s %s %s %s",
Serverpaket.get_mForFragan(), Serverpaket.get_mSekvensNr(),
Serverpaket.get_mProgNr(), Serverpaket.get_mFelNr(),
Serverpaket.get_mFelNiva());
                    send(mSocket, aUtbuf, INBUFFERLENGD, 0);
                    }
          }

          return 0;
}
```

// File : FPaket.h

```
// FPaket.h: interface for the FPaket class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_FPaket_H__2A27ADB1_13A6_11D5_A63D_0004760DCC72__INCLUDED_)
#define AFX_FPaket_H__2A27ADB1_13A6_11D5_A63D_0004760DCC72__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define INBUFFERLENGD 100
#include <String.h>


class FPaket
{
public:

           FPaket();
           virtual ~FPaket();
           void set_mForFragan(char *);                //Ex: SHOW_ERROR,
GET_VERSION
           const char * get_mForFragan() const;
           void set_mSekvensNr(char *);                //Sekvensnummer
           const char * get_mSekvensNr() const;
           void set_mProgNr(char *);
           //Applikationsnummer
           const char * get_mProgNr() const;
           void set_mFelNr(char *);                                  //Felkod
           const char * get_mFelNr() const;
           void set_mFelNiva(char *);                                //Felets
allvarlighetsgrad
           const char * get_mFelNiva() const;
           void set_mFlagga(int);
           //Svar OK/NOK
           const char * get_mFlagga() const;
           void set_mText1(char *)
           //Text från db
           const char * get_mText1() const;
           void mPacka(char *);                                      //Paketera
paket
           void mPackaUpp(char *);
           //Packar upp paket
           void mPackaSvar(char *);                                  //Paket
från serversidan

private:
           char mForFragan[16];
           char mSekvensNr[4];
           char mProgNr[5];
           char mFelNr[8];
           char mFelNiva[2];
           char mFlagga[4];
           char Text1[100];

};

#endif //
!defined(AFX_FPaket_H__2A27ADB1_13A6_11D5_A63D_0004760DCC72__INCLUDED_)
```

// File : FPaket.cpp

```
// FPaket.cpp: implementation of the FPaket class.
//
//////////////////////////////////////////////////////////////////////


#include "FPaket.h"


//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

FPaket::FPaket()
{

}

FPaket::~FPaket()
{

}


void FPaket::set_mForFragan(char * aFraga)
{
           strcpy(mForFragan,aFraga);
           mForFragan[15]='\0';
}

const char * FPaket::get_mForFragan() const
{
           return mForFragan;
}

void FPaket::set_mSekvensNr(char * aSekNr)
{
           strcpy(mSekvensNr, aSekNr);
           mSekvensNr[3]='\0';
}

const char * FPaket::get_mSekvensNr() const
{
           return mSekvensNr;
}

void FPaket::set_mProgNr(char * aProg)
{
           strcpy(mProgNr,aProg);
           mProgNr[4]='\0';
}

const char * FPaket::get_mProgNr() const
{
           return mProgNr;
}

void FPaket::set_mFelNr(char *aFel)
{
           strcpy(mFelNr,aFel);
           mFelNr[7]='\0';
```

```
}

const char * FPaket::get_mFelNr() const
{
          return mFelNr;
}

void FPaket::set_mFlagga(int aFlagga)
{
          if(aFlagga==1)
          {
                    strncpy(mFlagga, "OK", 2);
                    mFlagga[2]='\0';
          }
          else
          {
                    strncpy(mFlagga,"NOK",3);
                    mFlagga[3]='\0';
          }
}

void FPaket::get_mFlagga() const
{
          return mFlagga;
}

void FPaket::set_mFelNiva(char * aNiva)
{
          strcpy(mFelNiva,aNiva);
          mFelNiva[1]='\0';
}

const char * FPaket::get_mFelNiva() const
{
          return mFelNiva;
}

void FPaket::set_mText1(char *aText)
{
          strncpy(Text1,aText,(strlen(aText)));
          Text1[strlen(aText)]='\0';
}

const char * FPaket::get_mText1() const
{
          return Text1;
}

void FPaket::mPacka(char * Skickas)
{
          int len=strlen(get_mForFragan());
                    //Kolla frågelängd
          strncpy(Skickas,get_mForFragan(),len);
          //Innehåll i clientpaket läggs i "formaterad" vektor
          memset((Skickas+len),'0',(15-len));
          strncpy((Skickas+15),get_mSekvensNr(),3);
          strncpy((Skickas+18),get_mProgNr(),4);
          strncpy((Skickas+22),get_mFelNr(),7);
          strncpy((Skickas+29),get_mFelNiva(),1);
          Skickas[30]='\0';
}
```

```
void FPaket::mPackaUpp(char *Skickas)
{
            char sFraga[20];
            char temp[INBUFFERLENGD];
            memset(temp,'0',INBUFFERLENGD-1);
            char sGet[]="GET_VERSION";
            char sShowW[]="SHOW_WARNING";
            char sShowE[]="SHOW_ERROR";
            strncpy(sFraga,Skickas,15);
            //Jämförelsesträng med förfrågan
            sFraga[15]='\0';

            if(strncmp(sFraga,sGet,11)==0)              //Om Get_Version
            {
                        sFraga[11]= '\0';
                        set_mForFragan(sFraga);
            }

            if(strcmp(sFraga,sShowW)==0)     //Om Show_Warning
            {
                        sFraga[12]= '\0';
                        set_mForFragan(sFraga);
                        memset(temp,'0',INBUFFERLENGD-1);
            }

            if(strcmp(sFraga,sShowE)==0)     //Om Show_Error
            {
                        sFraga[10]= '\0';
                        set_mForFragan(sFraga);
                        memset(temp,'0',INBUFFERLENGD-1);
            }


            strncpy(temp,(Skickas+15),3);    //15, 16, 17
            temp[3]='\0';
            set_mSekvensNr(temp);
            get_mSekvensNr();
            memset(temp,'0',INBUFFERLENGD-1);

            strncpy(temp,(Skickas+18),4);    //18,19,20,21
            temp[4]='\0';
            set_mProgNr(temp);
            get_mProgNr();
            memset(temp,'0',INBUFFERLENGD-1);

            strncpy(temp,(Skickas+22),7);    //22-28
            temp[7]='\0';
            set_mFelNr(temp);
            get_mFelNr();
            memset(temp,'0',INBUFFERLENGD-1);

            strncpy(temp,(Skickas+29),1);    //29
            temp[1]='\0';
            set_mFelNiva(temp);
            get_mFelNiva();

}

void FPaket::mPackaSvar(char * aUtbuf)
{
```

```
            int length=strlen(get_mFlagga());
            strncpy(aUtbuf,(get_mFlagga()),length);
            strncpy((aUtbuf+15),get_mSekvensNr(),3);
            strncpy((aUtbuf+18,get_mText1(),(strlen(get_mText1())));
}
```