Datavetenskap

Peter Labraaten

Stefan Larsson

Portering av ett handdatorprogram för registrering av teleutrustning

---

Examensarbete, C-nivå

2003:12

Portering av ett handdatorprogram för registrering av teleutrustning

Peter Labraaten

Stefan Larsson

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

Peter Labraaten

Stefan Larsson

Godkänd, 2003-06-03

Handledare: Robin Staxhammar

Examinator: Stefan Lindskog

Sammanfattning

Detta examensarbete är en konstruktionsuppgift som görs på uppdrag av Prevas AB. Arbetet innebär att utveckla ett program kallat TULP utifrån ett befintligt program. Det nya programmet skall utvecklas för plattformen .NET och kunna köras på handdatorer med operativsystemet Pocket PC 2002.

Programmet används för att registrera teleutrustning, bland annat i basstationer, med hjälp av streckkoder. Vid registreringen används handdatorer försedda med streckkodsläsare.

I uppsatsen beskrivs och analyseras det befintliga systemet och utifrån detta utvecklades det nya programmet. Hur utvecklingen gick till beskrivs i stegen analys, konstruktion, implementation och testning.

Resultatet av arbetet blev ett mer effektivt program med samma funktionalitet som det befintliga men med ett förbättrat användargränssnitt. Det nya programmet är också förberett för en framtida multispråksanpassning.

Porting a palmtop program to record telecommunication equipment

Abstract

This bachelor's project is a construction assignment given by Prevas AB. The assignment is to develop a program called TULP on the basis of an existing program. The new program should be developed for the .NET platform and should run on palmtop computers using the operating system Pocket PC 2002.

The program uses barcodes to record telecommunication equipment, such as components located in radio base stations. Recording is done using palmtop computers equipped with barcode scanners.

In this report the existing system is described and analyzed and the new program is developed on that basis. The development is described stepwise with analysis, construction, implementation and testing.

The result of the work is a more efficient program with the same functionality as the existing one, but with an improved graphical user interface. The new program is also prepared for a future multi language adaptation.

Innehållsförteckning

Figurförteckning

1 Inledning

1.1 Bakgrund

Företaget Prevas AB har idag en produkt som heter Tracy Update and Label Printing (TULP). Produkten används vid installation och uppdatering av bland annat Ericssons teleutrustning världen över.

Ericsson har en databas som heter Tracy som lagrar information om alla komponenter för respektive basstationer som har levererats. Tracy innehåller bland annat information om vilka komponenter som sitter i vilken basstation. En basstation är en station som hanterar sändning och mottagning av mobil telekommunikation. En anledning till att Ericsson sparar information om detta är att de kan behöva byta ut komponenter och då måste de veta var komponenterna finns.

Produkten TULP består idag av två delprogram. Ett som körs på handdatorer utrustade med streckkodsläsare och ett som körs på stationära datorer. Handdatorprogrammet har till uppgift att med hjälp av streckkoder registrera komponentstrukturer i basstationer. Detta kan göras både vid en förstagångsregistrering efter en installation och vid uppdatering efter service. Programmet för stationära datorer används för att behandla data från handdatorn samt att uppdatera databasen Tracy.

Det befintliga handdatorprogrammet som är skrivet i Microsoft eMbedded Visual Basic 3.0 har efterhand byggts på med ny funktionalitet och detta har medfört att programmet blivit långsamt. Detta är ett problem och därför vill Prevas i syfte att göra programmet effektivare byta till den modernare plattformen .NET. Att programspråket Visual Basic .NET skall användas beror på att Visual Basic är det språk som Prevas använder för denna typ av tillämpning.

1.2 Mål

Att för operativsystemet Pocket PC, med hjälp av Microsoft Visual Basic .NET, skriva om det delprogram som ingår i TULP och körs på handdatorer. I konstruktionen av det nya programmet ska det utöver den befintliga funktionaliteten även tas hänsyn till att möjliggöra

en framtida multispråksanpassning. Till det nya programmet skall även en teknisk dokumentation skrivas.

1.3 Uppsatsens upplägg

I det nästkommande kapitlet börjar vi med att beskriva hur det befintliga systemet ser ut. Detta för att ge läsaren en förståelse och bakgrund till arbetet. Efter beskrivningen av det befintliga systemet gör vi en analys där vi presenterar vad systemet har för för- och nackdelar. Utifrån analysen av det gamla systemet konstruerar vi det nya.

Beskrivningen av det nya programmet börjar med en genomgång av konstruktionslösningen där användargränssnittets utseende och programmets uppbyggnad tagits fram. Med grund i denna konstruktionslösning utfördes sedan implementationen som beskrivs i det efterföljande kapitlet som i sin tur följs av ett kapitel om testningen av programmet.

Uppsatsen avslutas med att vi presenterar de resultat vi nått och de rekommendationer vi vill ge. Till sist summeras projektet.

För att ge en förståelse för den plattform som användes vid utvecklingen av det nya programmet finns en bilaga där .NET beskrivs övergripande. Den är emellertid inte nödvändig att läsa för att förstå innehållet i uppsatsen.

2 Det befintliga systemet

För att få förståelse för programutvecklingen krävs att man vet hur systemet fungerar och används. Handdatorprogrammet är en del av ett större system och i detta kapitel kommer vi att förklara hur systemet fungerar i sin helhet.

Till att börja med kommer systemets historia och bakgrund att behandlas. Systemet kommer sedan beskrivas översiktligt och till sist kommer systemets olika delar att beskrivas.

Efter denna genomgång av systemet kommer i nästa kapitel den del av systemet som skall nyutvecklas att analyseras.

2.1 Bakgrund

Namnet TULP står för Tracy Update and Label Printing. Detta syftar på att TULP används för att uppdatera databasen Tracy samt att skriva ut streckkodsetiketter. TULP härstammar från ett program som användes för att skriva ut streckkodsetiketter men har under åren byggts på med spårbarhetsfunktionalitet vilket idag är programmets huvudfunktion. Spårbarhetsfunktionalitet innebär funktionalitet för att söka efter och spåra utrustning med hjälp av identifieringsnummer och databaser. Den typ av teleutrustning som TULP är anpassad för är främst tillverkad av Ericsson men klarar även av andra leverantörers utrustning.

För att hantera en operatörs mobiltelefontrafik inom ett visst område används basstationer. Basstationerna är utrustade med en mängd kretskort av olika slag. En basstation är hierarkiskt uppbyggd med ett förälder/barn förhållande där varje basstation har ett eget identifieringsnummer vilket står högst i hierarkin. I en basstation finns ett antal rack och strömförsörjningsenheter och dessa utgör barn till basstationen. I varje rack sitter det en mängd kretskort som då blir barn till respektive rack.

För att effektivisera underhåll och support av sin utrustning lagrar Ericsson information om alla levererade komponenter i en stor databas som heter Tracy. Vid underhåll och uppdatering av en basstation behöver Tracy uppdateras och för detta används TULP.

Anledningen att programmet utvecklades var att det inte fanns något system för att uppdatera Tracy. All information i Tracy lades in vid leverans och förblev sedan ouppdaterad. Ville köparen sedan hålla sin egen databas uppdaterad var alla förändringar tvungna att

skrivas ned för hand för att sedan läggas in manuellt. Detta tog lång tid och risken för felaktiga uppdateringar var stor.

2.2 Systemöversikt

TULP är ett spårbarhetssystem som används för att skapa och underhålla spårbarhetsinformation. Spårbarhetsinformation är i detta fallet information om teleutrustning lagrad i streckkodsformat. Streckkoderna kan till exempel innehålla information om produktens serienummer, mjukvaruversion, tillverkare, produktnummer och tillverkningsdatum.

Ett typiskt TULP-system består av en bärbar dator med dockningsstation och en eller flera handdatorer.

Figur 2.1; Typiskt TULP-system.

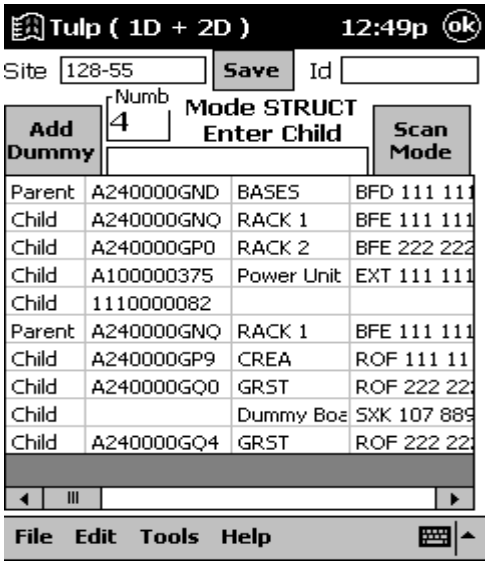
Handdatorerna används för att samla in information vilken sedan överförs till den bärbara datorn. Den bärbara datorn behandlar informationen för att sedan skicka den vidare till en databas som innehåller information om företagets samtliga basstationer. Denna databas kan i sin tur skicka informationen vidare till Ericssons centrala databas Tracy.

2.3 Handdatorns funktionalitet i TULP

Vid insamling av information används oftast en handdator utrustad med streckkodsläsare. I de fall då en handdator inte används kan en PC användas vilket beskrivs i avsnitt 2.4.

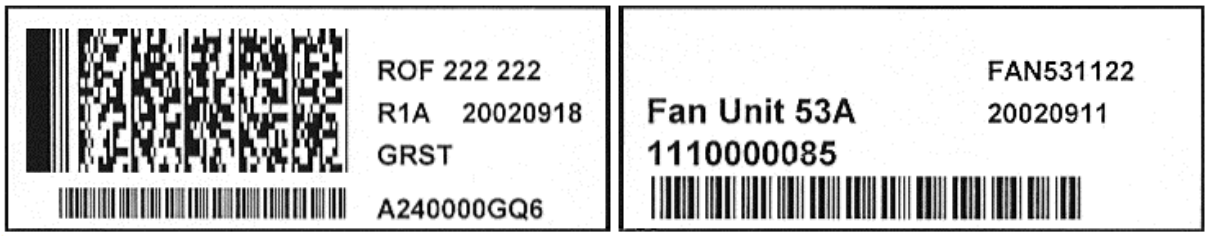
Som underlag till detta avsnitt har [4] använts.

Med hjälp av streckkodsläsaren kan användaren enkelt läsa av streckkoderna på de olika komponenterna. Programmet behandlar sedan informationen som sedan presenteras i en tabell på skärmen enligt figur 2.2.



Figur 2.2; Skärmbild med tabell.

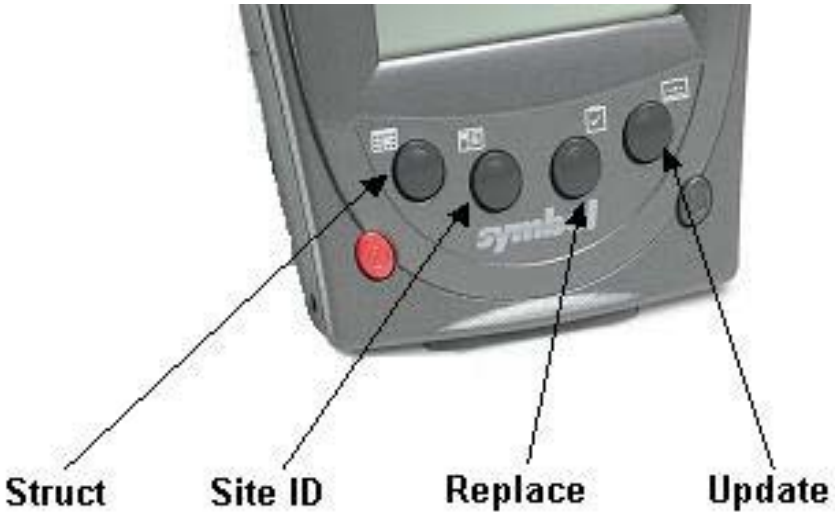
Streckkodsläsaren kan läsa en mängd olika streckkodstyper men i detta program är den begränsad till att läsa två typer av endimensionella koder samt en typ av tvådimensionell kod. Att endast dessa koder tillåts beror på att det är de enda som förekommer på den här typen av komponenter. Exempel på hur streckkodsetiketter kan se ut visas i figur 2.3.



Figur 2.3; Exempel på streckkodsetiketter. 2D till vänster och 1D till höger.

Det vanligaste är att en etikett innehåller både en endimensionell och en tvådimensionell streckkod. Den tvådimensionella streckkoden innehåller mer information och för att undvika att läsa in den endimensionella streckkoden av misstag kan inläsning av den endimensionella streckkoden stängas av och på medan den tvådimensionella alltid är påslagen. Vilket av de

olika streckkodsalternativen som skall användas vid ett visst tillfälle går att välja under körning. Detta är bra eftersom den tvådimensionella koden inte finns på alla komponenter.
Handdatorn används vid installation, underhåll och uppdatering av komponenter i basstationer. För att komma åt de olika funktionaliteterna använder man handdatorns knappar. Vilka knappar som finns visas i figur 2.4.



Figur 2.4; Handdatorns hårdvaruknappar.

Vid installation på en basstation registreras komponenternas struktur genom att läsa in förälder/barn hierarkin. För att göra detta trycker man först på knappen ”Struct”. Programmet ber då användaren att läsa in föräldern. När detta gjorts ombeds användaren att läsa in ett barn till den tidigare inlästa föräldern. Då ett barn lästs in har användaren möjligheten att läsa in ett nytt barn. Detta upprepas tills användaren antingen väljer en ny funktion eller att spara. Vill användaren börja om med en ny förälder trycks ”Struct”-knappen på nytt.

Ett annat användningsfall är då ett kort går sönder. Kortet måste bytas ut och för att göra detta trycker användaren på knappen ”Replace”. Användaren läser först in det gamla kortet och sedan det nya. Proceduren upprepas för varje kortbyte och när reparationen är slutförd väljer användaren antingen att spara eller en ny funktion.

Underhåll kan bestå av att en komponents mjukvara uppdateras. Även detta måste registreras och uppdateras i Tracy. För att registrera en uppdatering av mjukvaran trycker användaren på knappen ”Update” och läser sedan in streckkoden på det kort som uppdaterats. Användaren läser in samtliga kort som har uppdaterats i en följd.

Genom att trycka på ”Site ID”-knappen finns möjlighet att läsa in utrustning utan att strukturera i ett förälder/barn förhållande. Inläsningen av utrustning upprepas tills användaren väljer en ny funktion eller sparar. Detta är bra då man endast vill registrera positionen på en viss utrustning och att den inte ingår i en struktur.

Flera av dessa åtgärder kan utföras vid samma tillfälle. Skulle användaren läsa in en felaktig rad finns möjlighet att ta bort den och läsa in på nytt. Det finns även möjlighet att läsa in ID för den aktuella basstationen samt personligt ID för användaren. Då en basstation inte tillåts ha några tomma kortplatser sitter på dessa platser så kallade ”dummy kort”. Dessa har ingen funktion utan är bara utfyllnad men måste ändå läsas in. För att registrera ett sådant kort kan det oftast läsas in som vanligt men ibland kan streckkod saknas. För detta fall finns mjukvaruknappen ”Add Dummy” som visas i figur 2.2. Då knappen används läggs ett "dummy kort” till i tabellen.

2.4 PCns funktionalitet i TULP

Denna del av TULP körs på en PC utrustad med en dockningsstation för handdator. När handdatorn sätts i dockningsstationen överförs informationen som handdatorprogrammet genererat. Det finns även möjlighet att koppla en fristående streckkodsläsare till datorn och läsa in streckkodsinformation med samma funktionalitet som handdatorn. Denna lösning används sällan eftersom den inte ger samma flexibilitet som handdatorn.

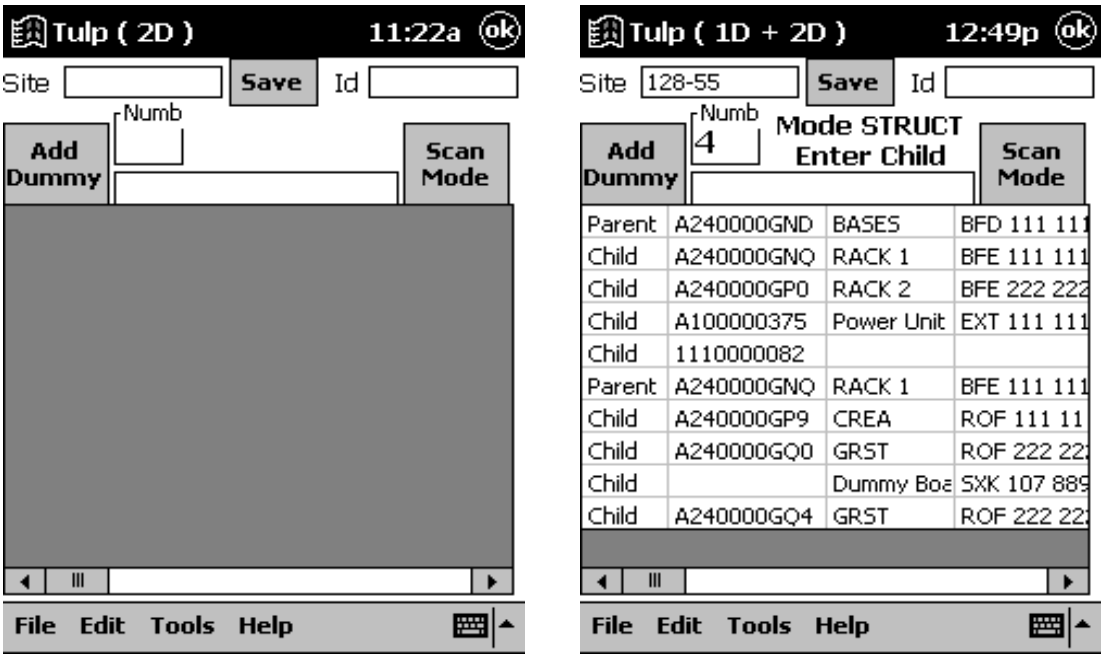
Efter att ha överfört informationen till TULP-PC finns möjlighet att göra ändringar och slutligen godkänna innehållet. Programmet kan sedan uppdatera företagets egna databas och kan även uppdatera Tracy via e-post.

Den ursprungliga funktionaliteten för utskrift av streckkodsetiketter finns kvar i denna del av TULP. Nya streckkodsetiketter skrivs ut då kretskort uppdateras men även på grund av att de gamla slits och blir svåra att läsa. För att skriva ut streckkodsetiketter kopplas en speciell etikettskrivare till datorn.

3 Analys av det befintliga programmet

I detta kapitel kommer en analys av det befintliga programmet att ges. Analysen har delats upp mellan användargränssnitt och funktionalitet. Fördelar och nackdelar med det befintliga programmet kommer att identifieras och möjliga förändringar kommer att beskrivas. Denna analys ligger till grund för konstruktionslösningen som beskrivs i nästa kapitel.

3.1 Användargränssnitt



Figur 3.1; Det grafiska användargränssnittet.

3.1.1 Fördelar med gränssnittet

- Tabellen som visar den inlästa informationen är stor vilket är viktigt för att ge en bra överblick.
- Den viktigaste informationen är placerad till vänster i tabellen vilket medför att rullningslisten inte behöver användas så ofta.
- Det finns möjlighet att ändra storlek på tabelltext och kolumnbredd mellan två förbestämda lägen.
- Återkoppling med ljud vid lyckad inläsning och avslutad inläsningssekvens.

3.1.2 Nackdelar med gränssnittet

- Ingen möjlighet till manuell finjustering av kolumnbredd.
- Inga kolumnnamn.
- Bristande symmetri i designen. Komponenternas placering följer inte ett enhetligt mönster så att linjer uppstår längs dess kanter.
- Aktuell streckkodstyp för streckkodsläsaren står i namnlisten längst upp i gränssnittet vilket kan misstolkas som en del av programnamnet.
- Liten text i dialogrutor.
- Ingen förklaringstext för den inmatningsruta som är placerad direkt ovanför tabellen. Inmatningsrutan används för att manuellt mata in ID-nummer då streckkod saknas.

3.1.3 Möjliga förändringar

För att förtydliga visningen av data i tabellen vore det bra att visa kolumnnamn. Då de inlästa värdena kan variera i storlek vore det även bra att manuellt kunna justera bredden på kolumnerna.

Designen av gränssnittet är något otydlig och ostrukturerad. Ett exempel på detta är att den aktuella inställningen för streckkodsläsaren står i namnlisten längst upp i gränssnittet vilket medför att det kan tolkas som att det hör till programnamnet. Är till exempel streckkodsläsaren inställd för endast tvådimensionell inläsning visas texten ”TULP (2D)” i namnlisten. Detta skulle istället kunna skrivas ut i anslutning till knappen som ändrar denna inställning. De grafiska komponenterna är dåligt grupperade och bör därför omplaceras så att linjer uppstår längs dess kanter och ett mönster uppstår. Förklaringstext bör läggas till den manuella inmatningsrutan.

När dialogrutor visas kan med fördel en större text användas då det är viktigt att användaren inte misstolkar meddelanden.

De fördelar med användargränssnittet som identifierats bör behållas.

3.2 Funktionalitet

Programmet är utvecklat från ett mindre program och allt eftersom nya behov uppstått har ny funktionalitet lagts till. Den befintliga funktionaliteten är därför nödvändig och bör inte ändras. Eftersom programmet körs på en handdator måste funktionaliteten begränsas till att

omfatta endast det nödvändiga för att programmet skall vara så effektivt och enkelt att använda som möjligt.

3.2.1 Fördelar i funktionaliteten

- Automatisk anpassning av streckkodsläsarens inställning i lägen där endast en streckkodstyp är möjlig.
- Ingen onödig funktionalitet.
- Felkontroller ger få felinmatningar. Till exempel kan en förälder inte kan läsas in som barn till sig själv. Vid felaktig inmatning visas ett felmeddelande.

3.2.2 Nackdelar i funktionaliteten

- Hårdvaruknapparna som används för att nå huvudfunktionaliteterna ”Struct”, ”Update”, ”Replace” och ”Site ID” är egentligen avsedda för annan funktionalitet som till exempel att öppna en kalender eller en adressbok.
- Vid kretskortsbyte skapas en extra rad i tabellen. Denna rad används då ett icke giltligt Ericsson ID har lästs in och innehåller då dess förälder. Skall ett kort med Ericsson ID bytas blir raden tom och är därför överflödig.

3.2.3 Möjliga förändringar

Hårdvaruknapparna är som standard tänkta att användas för att starta program som till exempel kalender och adressbok. För att inte låsa upp hårdvaruknapparna med programfunktionalitet kan istället det grafiska användargränssnittet användas.

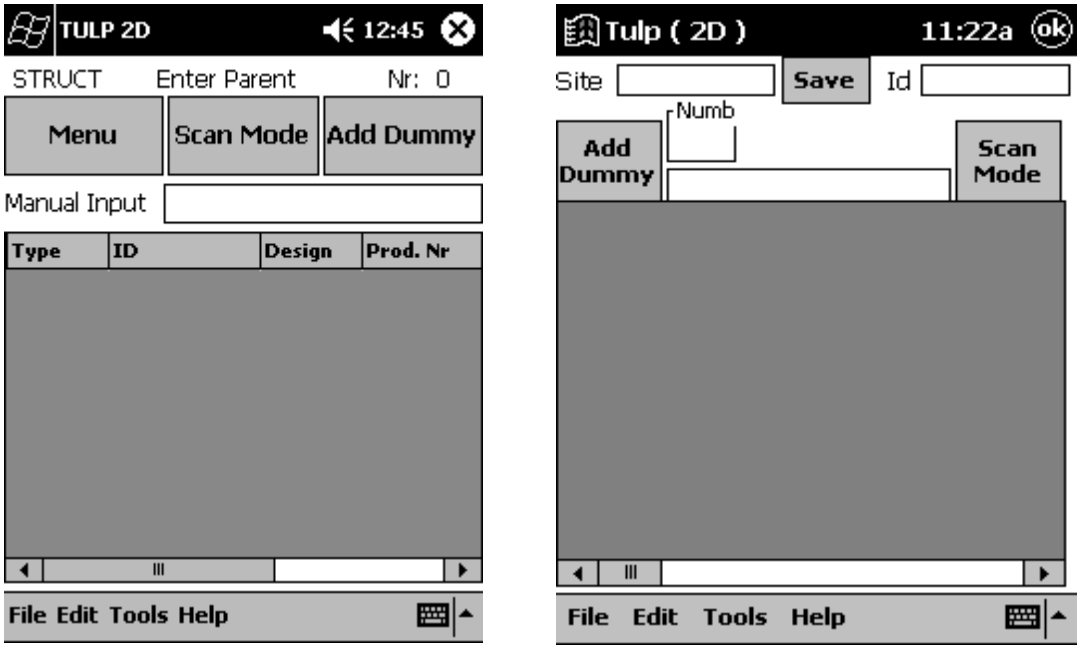
En ändring i grundfunktionaliteten kan göras för registrering av kretskortsbyte. Programmet lägger då till en extra tabellrad som endast används i vissa undantagsfall men ändå alltid visas. En ändring skulle kunna vara att endast lägga till denna rad då den verkligen behövs.

4 Beskrivning av konstruktionslösningen

I detta kapitel beskrivs den konstruktionslösning som tagits fram. Till att börja med beskrivs användargränssnittets utseende och motiveringar ges för tagna beslut. Därnäst kommer programmets uppbyggnad att beskrivas. Även här kommer de olika besluten att motiveras. Konstruktionslösningen ligger till grund för implementationen som beskrivs i kapitel 5.

4.1 Användargränssnitt

För konstruktion av det nya gränssnittet ligger det som framkom vid analysen i avsnitt 3.1 till grund. Utifrån analysen har en ny gränssnittsprototyp tagits fram och visas tillsammans med det gamla gränssnittet i figur 4.1.

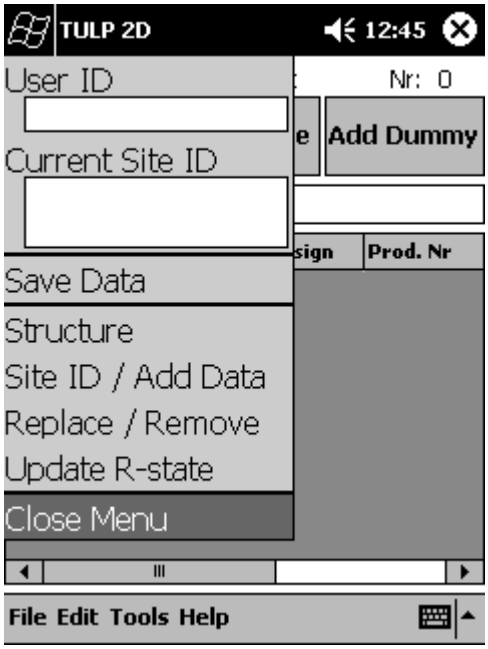


Figur 4.1; Nya gränssnittet till vänster och det gamla till höger.

- Tabellens storlek och kolumnordning har behållits men kolumnnamn har lagts till för att förtydliga.
- Det skall finnas möjlighet att manuellt ändra kolumnbredd och textstorlek i cellerna.
- Likartade komponenter har grupperats radvis för att få lättöverskådlig struktur. Anledningen till komponenternas placering är att det skall vara lätt att använda

knapparna utan att komma åt fel komponent. Denna gruppering gör också att knapparna tar minimal plats i höjdled vilket gör att tabellen ges större utrymme. En rad med utmatningsfält har placerats längst upp på skärmen så att knapparna blir placerade en bit ner på skärmen. Detta gör att operativsystemets menyknapp som är placerad längst upp till vänster på skärmen inte aktiveras av misstag då en knapp används.

- Fältet för manuell inmatning har nu en förklarande text.
- Gränssnittet har fått en ny knapp som används för att visa en meny där de viktigaste funktionerna lätt kan nås. Denna meny visas i figur 4.2. I menyn återfinns bland annat funktionalitet för att mata in användar-ID, plats-ID samt att spara den inlästa informationen. Dessa funktionaliteter återfanns tidigare i huvudfönstret men har lagts i menyn för att spara plats. I menyn återfinns även de fyra grundfunktionerna ”Structure”, ”Site ID”, ”Replace” och ”Update”. Dessa funktioner nåddes förut via hårdvaruknapparna som nu kan användas till sin ursprungliga funktionalitet. Det finns även ett menyval för att stänga menyn. Menyval kan göras på två sätt. Det ena är att trycka på önskat menyval via skärmen och det andra är att använda handdatorns piltangenter och valknapp.



Figur 4.2; Menyn.

I analysen konstaterades att streckkodsläsarens inställning inte borde visas i namnlisten. Vid design av prototypen framkom dock att den vinst av utrymme som denna placering

medför är tillräckligt stor för att överväga nackdelen med att det kan misstolkas som en del av programnamnet. Därför kommer placeringen i namnlisten att behållas.

4.2 Programmets uppbyggnad

Programmet skall till skillnad från den tidigare versionen vara objektorienterat vilket är ett krav från uppdragsgivaren. Ett objektorienterat program innebär att det blir en naturlig uppdelning av programmet i form av klasser. I det tidigare programmet finns all logik i samma programdel som de grafiska komponenterna men i det nya programmet kommer logiken och användargränssnittet att skiljas åt. Detta medför att programmet blir mer modulärt och därmed lättare att underhålla. Det innebär även att logikkoden i framtiden kan återanvändas om ett nytt användargränssnitt tas fram. Det är även möjligt att använda samma logikkod om ett nytt program för stationära datorer utvecklas. Programmet kommer alltså att delas upp i två huvuddelar, en för det grafiska användargränssnittet och en för att hantera logiken.

Ett önskemål från Prevas är att det nya programmet skall vara multispråksanpassat. Denna anpassning kommer att ske i användargränssnittet.

5 Implementation

I detta kapitel ges en beskrivning av hur implementationen utifrån den tidigare beskrivna konstruktionslösningen gått till. Vid implementationen användes programspråket Visual Basic .NET som använder plattformen .NET. För detta kapitel behövs ingen kunskap om målplattformen men för den som är intresserad ges en översiktlig beskrivning av .NET i bilaga A.

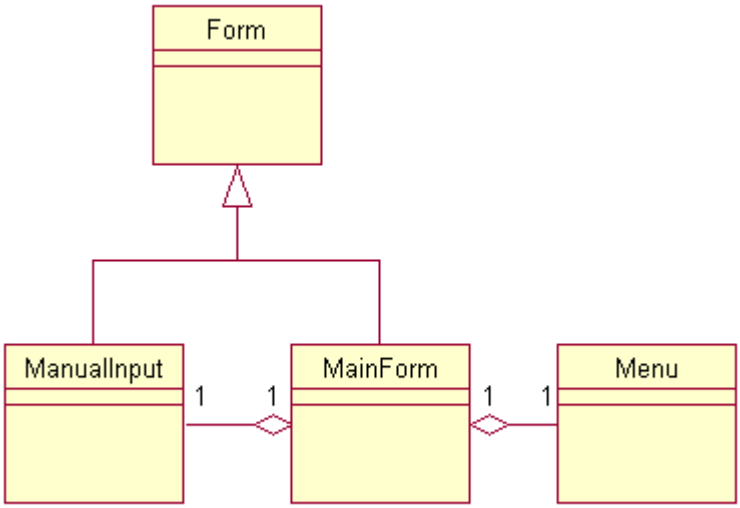
I detta kapitel ges först en övergripande beskrivning av designen. Därefter beskrivs den funktionalitet som har implementerats och det är denna del som ligger till grund för testningen som beskrivs i kapitel 6. Slutligen kommer de problem vi stött på under implementationen att tas upp.

5.1 Design

Programmet är uppdelat i två huvuddelar, logik och användargränssnitt. Beskrivningen av de två delarna har delats upp klassvis. Programmet består av totalt fem egendefinierade klasser med varsitt ansvarsområde. Detta för att varje del skall vara utbytbar vid eventuell vidareutveckling.

Först beskrivs användargränssnittet, sedan logiken och därefter beskrivs hur de är sammankopplade.

5.1.1 Grafiska användargränssnittet



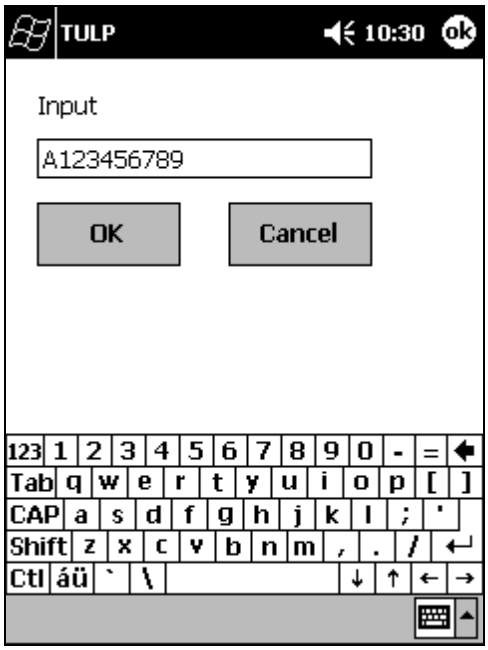
Figur 5.1; Klassdiagram för användargränssnittet.

- **MainForm**

Klassen MainForm är programmets grafiska huvudfönster och laddas när programmet startar. Klassen har tre huvuduppgifter. Det ena är att låta användaren styra programmet genom knapptryckningar och menyval och förmedla dessa vidare till den logiska delen. En annan uppgift är att visa den inlästa informationen för användaren i en tabell. Den tredje huvuduppgiften är att tala om för användaren vad programmet förväntar sig att användaren skall utföra i nästa steg i en inläsningssekvens.

MainForm håller inte själv reda på vilket tillstånd programmet befinner sig i utan kommunicerar med logikdelen för att uppdateras då en förändring har skett.

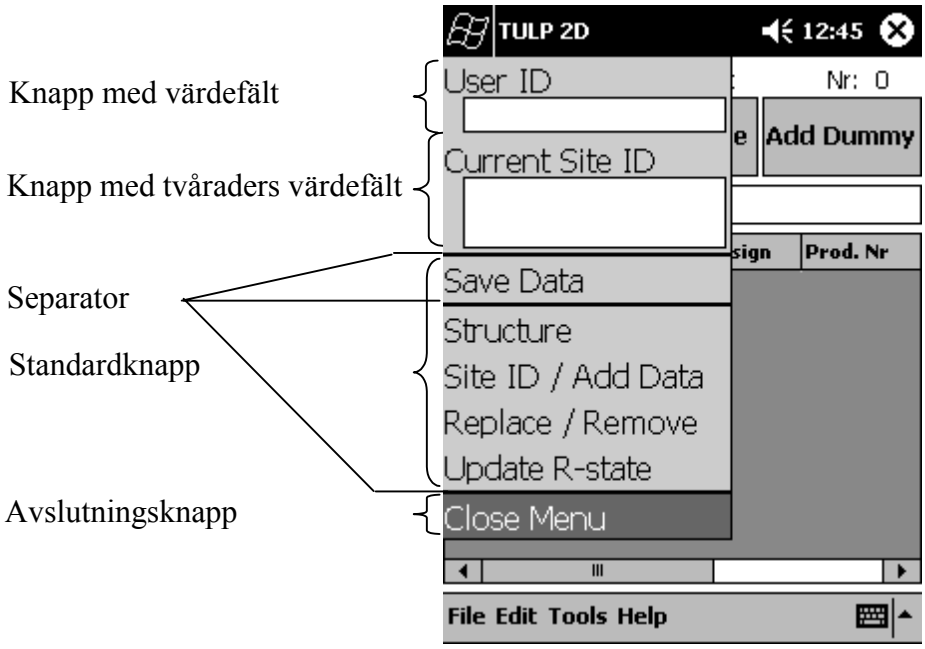
- **ManualInputForm**



Figur 5.2; Manuell inmatning.

Denna klass används för att manuellt mata in värden och används i klassen MainForm. Genom att ärva från standardklassen Form kan ManualInputForm klassen visa ett grafiskt fönster med ett inmatningsfält där handdatorns inbyggda tangentbord kan användas för att mata in ett ID-nummer istället för att använda streckkodsläsaren. Då ett värde matats in skickas det vidare för behandling i den logiska enheten via klassen MainForm.

- **Menu**

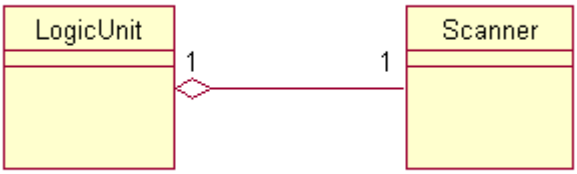


Figur 5.3; Menyn.

Menu-klassen används i klassen MainForm och skapar den meny som används för att komma åt programmets olika funktioner. Menyn består av fyra olika knapptyper och separatorer och när den skapas kan dess utseende ställas in. De olika knapparna är standardknapp, knapp med värdefält, knapp med tvåraders värdefält och avslutningsknapp. Menyn reglerar själv sin storlek och anpassar sig efter de knappar som finns. För att stänga menyn utan att välja någon funktionalitet kan användaren trycka på ”Close Menu”, men även då de andra knapparna används så stängs menyn. Då menyn stängs behåller den sitt tillstånd vilket medför att den senast använda knappen är markerad då menyn öppnas igen.

Menyn byggs upp med hjälp av .NETs standardklass Panel som är en tom yta som kan fånga upp olika händelser och innehålla andra element.

5.1.2 Logiken



Figur 5.4; Klassdiagram logikdel.

- **Scanner**

Klassen Scanner innehåller all funktionalitet för att hantera handdatorns streckkodsläsare. Med hjälp av denna klass kan man starta och stoppa streckkodsläsaren, ändra tillåtna streckkodstyper och hämta inläst information. När en streckkod har lästs in delas informationen upp i olika beståndsdelar och lagras i klassen. Efter uppdelningen meddelas LogicUnit-klassen att en streckkod har lästs in och informationen kan hämtas från Scanner-klassen.

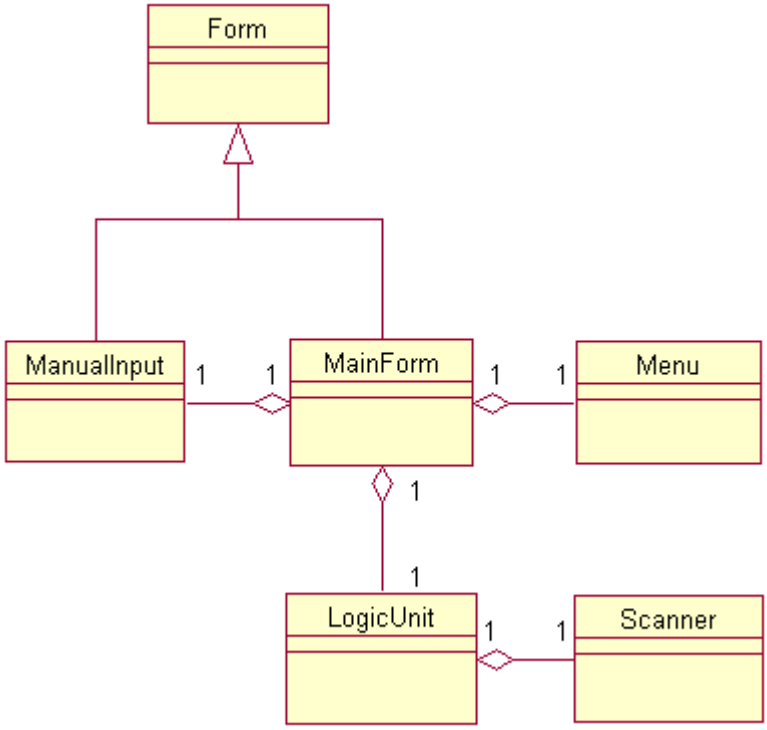
- **LogicUnit**

Klassen LogicUnit fungerar som programmets motor. Den hämtar information från Scanner-klassen, bearbetar informationen och kommunicerar med användargränssnittet.

Det är LogicUnit-klassen som håller programmet i ett korrekt tillstånd genom att kontrollera att inte otillåtna värden läggs till i tabellen. Ett exempel på en kontroll som görs för att upptäcka en felaktig inläsning är om en förälder läses in som barn till sig själv i en struktur.

5.1.3 Programmets klassdiagram

Den grafiska och logiska delen kopplas samman enligt klassdiagrammet i figur 5.5.



Figur 5.5; Klassdiagram hela systemet.

5.2 Implementationsbeskrivning

I detta avsnitt beskrivs den funktionalitet som implementerats i klassen LogicUnit. Beskrivningen har delats upp i två delar. Den ena med grundfunktionalitet för inläsning och den andra delen med funktionalitet för att redigera redan inläst information.

5.2.1 Grundfunktionalitet

- **Replace / Remove**

Funktionaliteten ”Replace / Remove” används då ett kort skall bytas ut eller tas ur bruk. För att utföra ett byte (Replace) läses det gamla och det nya kortet in. Beroende på kortens ID-nummer skall eventuellt även det gamla kortets förälder läsas in. Då ett kort skall tas ur bruk (Remove) läses det aktuella kortet in två gånger vilket programmet tolkar som en borttagning. För att visa vilken kortstruktur kortet tas bort ifrån skall även föräldern till kortet läsas in.

- **Update**

Funktionen ”Update” används när ett korts R-läge, det vill säga kortets mjukvara, har uppdaterats. För att registrera en uppdatering läses det uppdaterade kortet in och läggs till i tabellen.

- **Struct**

”Struct” används då en hel struktur av kort skall läsas in. För att läsa in en struktur läses först det kort som är förälder i strukturen in och läggs till i tabellen. Sedan läses alla barnen i strukturen in. När en struktur läses in sker ett flertal kontroller av ID-numret. Ett exempel på en kontroll är att föräldern inte kan läsas in som barn till sig själv. Då ett barn går igenom kontrollen läggs det till i tabellen.

- **Site ID**

”Site ID” används för att läsa in kort utan hänsyn till struktur. För att göra detta läses ett kort in och läggs till i tabellen utan beroende till något annat, tidigare eller senare inläst kort.

- **Current Site ID**

För att läsa in ID-numret för den fysiska plats som användaren befinner sig på används ”Current Site ID”. ID-numret för platsen läses in men läggs inte till i den tabell som används för att visa inlästa kort. Istället visas ID-numret i ett fält på menyn. Då en plats inte alltid har en streckkod med ID-nummer går det att manuellt mata in detta ID-nummer med hjälp av operativsystemets grafiska tangentbord.

- **User ID / PPC**

För att läsa in användarens ID-nummer används ”User ID / PPC”. Personens ID-nummer läses in och visas i ett fält på menyn. Saknas streckkod med användarens ID-nummer går det att manuellt mata in detta nummer med hjälp av operativsystemets grafiska tangentbord.

- **Save**

När den inlästa informationen sparas skrivs teckensträngar som representerar de inlästa raderna i tabellen till en fil. Dessa textsträngar byggs upp med hjälp av förbestämda prefix och inläst data. Filens namn är unikt och skapas med hjälp av tid, datum och delar av den inlästa informationen.

5.2.2 Redigering

- **Insert Row**

”Insert Row” används för att infoga en rad i tabellen med den inlästa informationen. Detta är användbart då användaren har missat ett kort i en struktur och istället för att göra om hela strukturen kan användaren med denna funktionalitet lägga till det missade kortet på rätt plats i strukturen. Tillvägagångssättet är att markera den rad i tabellen där kortet skall sättas in och ”Insert Row” väljs. Därefter läses kortet in och behandlas som om det hade lästs in samtidigt som de övriga korten i strukturen vilket innebär att samma kontroller görs som vid en vanlig inläsning av barn i en struktur.

- **Remove Row**

Om användaren upptäcker att en felaktig inläsning har gjorts så kan ”Remove Row” användas för att korrigera den felaktiga inläsningen. Detta kan göras överallt i tabellen och för alla inlästa rader. Tillvägagångssättet för att ta bort en rad är att markera den rad som skall tas bort och därefter välja ”Remove Row”. Då en rad tas bort sker många kontroller för att upprätthålla ett korrekt tillstånd i tabellen. Ett exempel på en sådan kontroll är när en förälder i en struktur tas bort så måste även alla barn i strukturen tas bort. Detta för att förhindra att barn utan förälder existerar. Detta innebär att funktionen även kan användas för att ta bort hela strukturer.

- **Remove All Rows**

”Remove All Rows” används för att tömma tabellen på allt innehåll. När ”Remove All Rows” väljs och tabellen inte redan är tom får användaren en fråga om alla rader verkligen skall tas bort. Vill användaren ta bort alla rader görs detta, annars görs ingenting.

5.2.3 Multispråksanpassning

Prevas önskemål om multispråksanpassning har förberetts genom att strängkonstanter använts vid alla utskrifter i det grafiska användargränssnittet. Dessa kan vid ett senare tillfälle bytas ut mot anrop till metoder i en klass som hanterar de olika språken.

Det nuvarande klassdiagrammet för det grafiska användargränssnittet kan ses i figur 5.1. En språkklass bör kopplas samman med klassen MainForm som är den klass som visar det grafiska användargränssnittet.

5.3 Problem

- **Ingen dialog från logikdelen**

Eftersom avsikten är att skilja användargränssnittet från logiken måste all dialog med användaren ligga i klassen MainForm. Detta skapade problem i de fall då programmet behöver ett val från användaren för att utföra rätt logik. Eftersom logikdelen inte tillåts fråga användaren direkt tvingas många funktioner att delas upp i flera steg vilket gör koden mer svårläslig.

- **Rullningslist**

Då ett nytt värde läggs till i tabellen skall denna rad alltid synas. Detta betyder att tabellens rullningslist måste regleras av programmet. Rullningslisten är i .NET Framework åtkomlig via arv men i Compact Framework som vi använder finns inte denna möjlighet. Vi löste problemet genom att leta igenom den samling av kontroller som finns i klassen som visar tabellen på skärmen och där hittade vi rullningslisten. Efter att rullningslisten hittats så typkonverterades den från basklassen Control till VScrollBar som är en vertikal rullningslist. Denna teknik för att komma åt rullningslisten är inte att rekommendera eftersom den inte finns dokumenterad i .NET. Hur rullningslisten används visas nedan.

```
CType(grdMain.Controls.Item(1), VScrollBar).Value = 100
```

I tabellen grdMain hämtas kontrollen med index 1 ut ur kontrollsamlingen Controls. Typkonvertering sker till VScrollBar med hjälp av Ctype. VscrollBars medlem value sätts till 100.

- **Ljuduppspelning**

I programmet används ofta ljud som återkoppling då till exempel en inläsning har utförts. .NET compact Framework har inget inbyggt stöd för att spela upp ljud. Detta innebär att systemoperationer som finns i Dynamic Link Library (DLL) filer måste användas. Problemet med detta var att ta reda på vilken DLL-fil som skall användas och vilka funktioner som finns i den. Efter mycket sökande hittades lösningen i dokumentationen för .NET Compact Framework.

Funktionen för ljuduppspelning som finns i DLL-filen ”CoreDll.dll” används på följande sätt:

```
Public Declare Function TULP_PlaySound Lib "CoreDll.dll" Alias "PlaySound" (ByVal szSound As String, ByVal hMod As IntPtr, ByVal flags As Integer) As Integer
```

Funktionen TULP_PlaySound deklareras med hjälp av nyckelordet Declare. Genom nyckelordet Lib anges DLL-filen ”CoreDll.dll” och med Alias anges funktionen PlaySound som finns i DLL-filen. Ett anrop till funktionen sker enligt följande:

```
TULP_PlaySound(“Windows\Alarm1.wav”, 0, SND_FILENAME)
```

Vid anropet spelas ljudfilen som anges i första argumentet upp, i detta fall ”Alarm1.wav”. Andra argumentet används inte och sätts därför till 0. Det tredje argumentet anger att första argumentet är en ljudfil.
Att detta beskrivs beror på att det kan vara bra information för uppdragsgivaren då det är annorlunda gentemot övrig kod.

- **Egna kontroller**

När menyn skulle konstrueras var det naturliga valet egentligen att konstruera en egen kontroll för att representera menyn. Problemet är att detta inte stöds av .NET Compact Framework. För att konstruera menyn användes därför en annan lösning där kontroller av typen Panel placerades ut för att symbolisera de olika knapparna vilket var ganska omständigt. Paneler är specialiserade kontroller eftersom de har ett bestämt beteende och därför skulle det vara effektivare och enklare att skapa menyn med hjälp av egendefinierade kontroller. Eftersom stöd saknades var detta tyvärr inte möjligt.

- **Användning av streckkodsläsare**

Handdatorns tillverkare tillhandahåller ett utvecklingspaket för .NET, som bland annat innehåller de klasser som möjliggör åtkomst av streckkodsläsaren. Detta utvecklingspaket fanns under utvecklingen endast som en beta-version vilket innebar att dokumentationen för utvecklingspaketet var bristfällig. Det var därför svårt att komma igång med användandet av

streckkodsläsaren i .NET men med hjälp av medföljande exempel kunde vi komma fram till hur streckkodsläsaren skulle ställas in och användas.

6 Testning

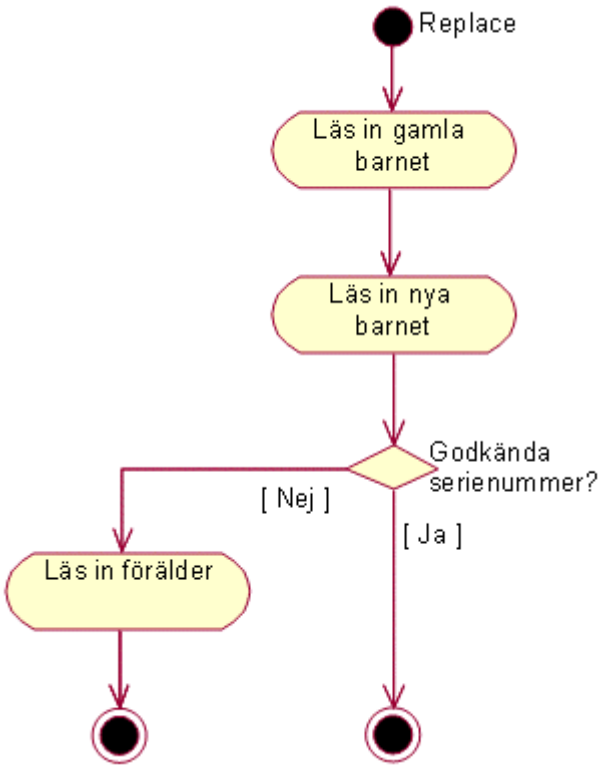
I kapitlet beskrivs testfasen där den implementerade funktionaliteten som beskrivs i avsnitt 5.2 testas. Detta görs med hjälp av olika testfall som beskrivs i första avsnittet. I andra avsnittet ges en kort summering av testernas utfall.

6.1 Testfall

För att testa programmet har så kallad ”black box”-testning använts. Detta innebär att det inte är koden som ligger till grund för utformningen av testfallen utan endast programmets avsedda beteende. Testfall för de viktigaste funktionaliteterna tagits fram med hjälp av flödesdiagram.

I några av diagrammen används ”Godkända serienummer?”. Detta innebär att det med hjälp av serienumren kontrolleras om utrustningen är kompatibel.

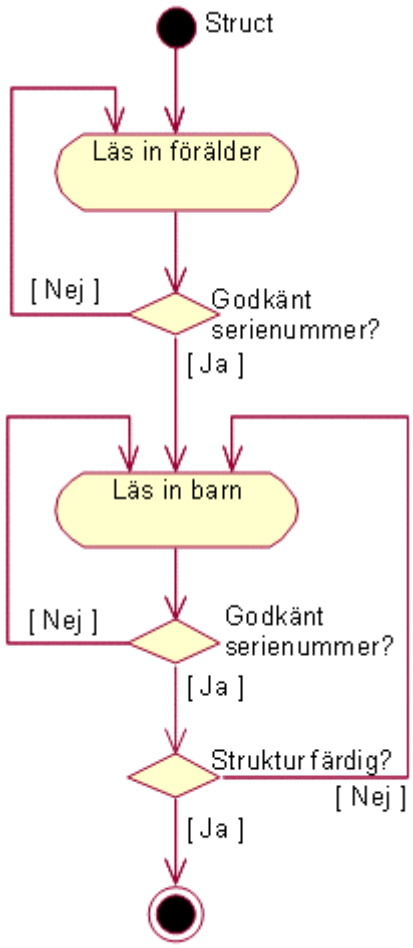
- **Replace**



Figur 6.1; Flödesdiagram Replace.

Testet av ”Replace” börjar med att det gamla barnet läses in. Efter detta läses det nya barnet in och efter detta kontrolleras så att de båda barnen har godkända ID-nummer. Är de båda ID-numren godkända är bytet klart. Annars skall föräldern läsas in och då detta är gjort är bytet klart.

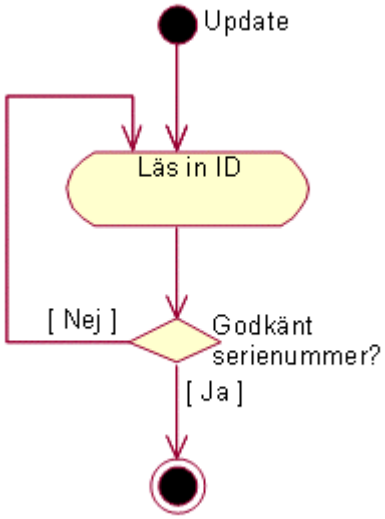
- **Struct**



Figur 6.2; Flödesdiagram Struct.

Testet av ”Struct” börjar med att föräldern läses in. Denna förälder måste ha ett godkänt ID-nummer. Efter föräldern läses barnen in. Även barnen kontrolleras så att de har godkända ID-nummer. Är strukturen färdig avslutas funktionen, annars läses nästa barn in.

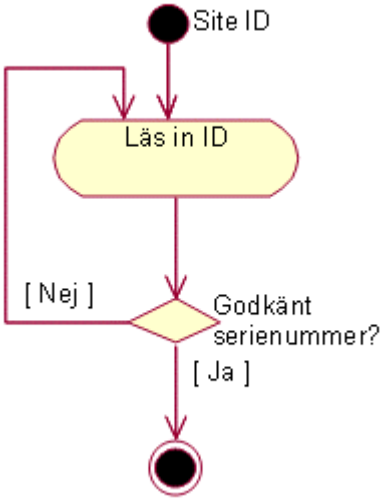
- **Update**



Figur 6.3; Flödesdiagram Update.

Testet börjar med att ett ID läses in. Är det ett godkänt ID-nummer är ”Update” klar.

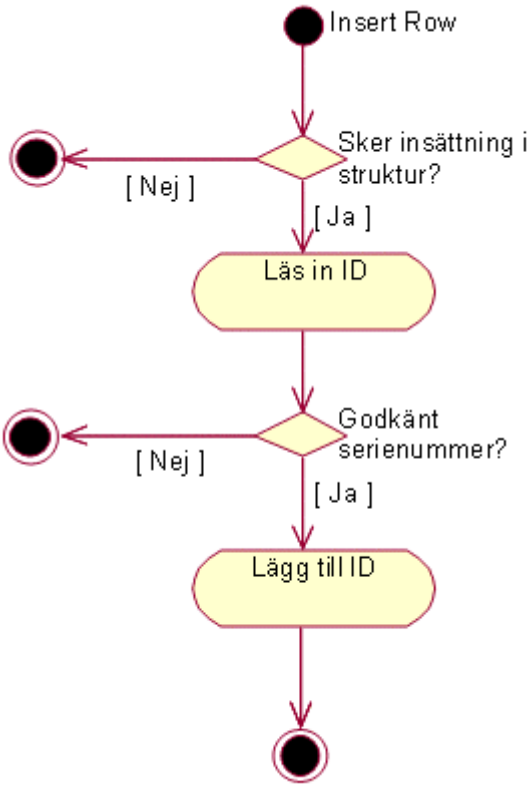
- **Site ID**



Figur 6.4; Flödesdiagram Site ID.

Testet börjar med att ett ID läses in. Är det ett godkänt ID är ”Site ID” klar.

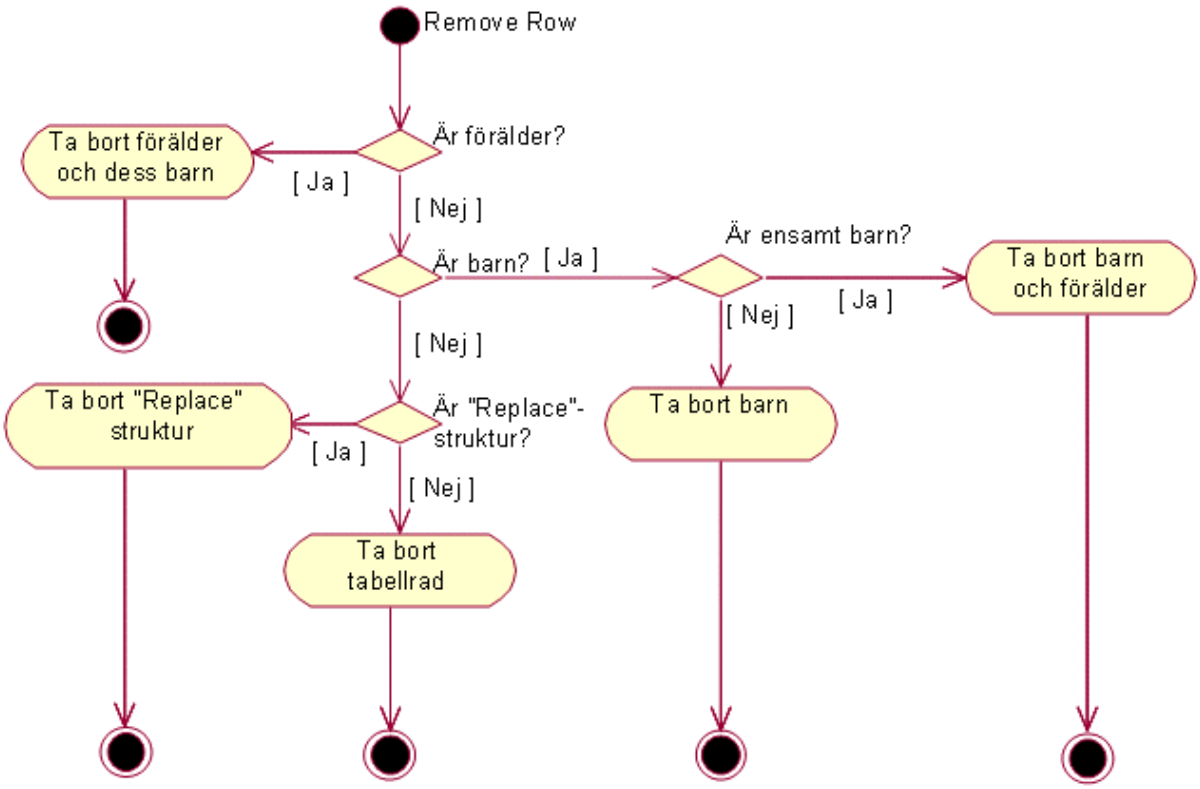
- **Insert Row**



Figur 6.5; Flödesdiagram Insert Row.

”Insert Row” är endast tillåtet att göras i en struktur så därför kontrolleras detta först. Är så fallet läses det ID som skall sättas in i strukturen in. Detta ID kontrolleras så att det är godkänt och i så fall är insättningen utförd.

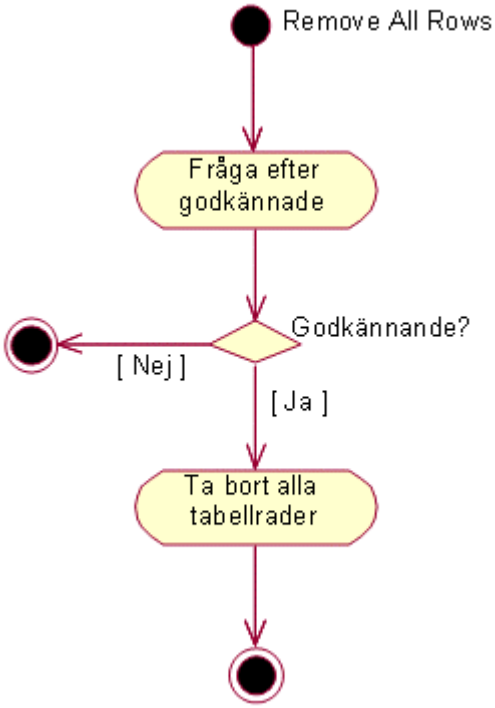
- **Remove row**



Figur 6.6; Flödesdiagram Remove Row.

Först kontrolleras om den rad i tabellen som skall tas bort är en förälder. Är så fallet tas föräldern och alla dess barn bort. Är det ett barn som står ensamt som skall tas bort tas även föräldern bort, annars tas bara barnet bort. Skulle den rad som skall tas bort tillhöra en ”Replace” struktur så tas hela strukturen bort. I alla övriga fall tas den önskade raden bort utan att det påverkar någon annan rad.

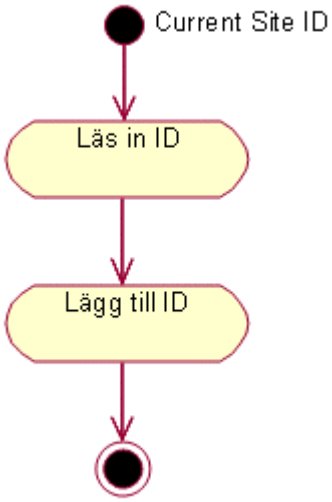
- **Remove All Rows**



Figur 6.7; Flödesdiagram Remove All Rows.

Vid ”Remove All Rows” måste ett godkännande ges av användaren. Görs detta tas alla rader bort.

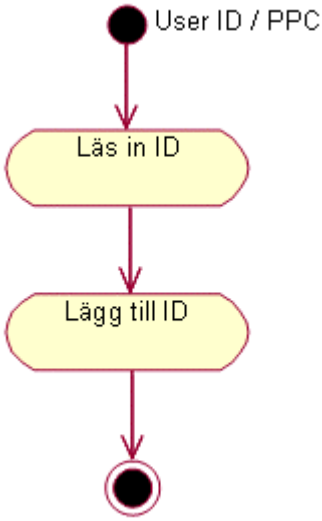
- **Current Site ID**



Figur 6.8; Flödesdiagram Current Site ID.

Här är det tillåtet med alla ID-nummer.

- **User ID/ PPC**



Figur 6.9; Flödesdiagram User ID / PPC.

Här är det tillåtet med alla ID-nummer.

6.2 Resultat av testerna

Testningen av programmet var lyckat då alla testfall gick igenom utan att några fel uppstod.

7 Resultat och rekommendationer

I detta kapitel summeras resultatet av det utförda arbetet. De mål som angavs i avsnitt 1.2 kommer att mätas och rekommendationer för vidareutveckling av programmet kommer att ges.

De mål som sattes upp var att utveckla ett program för operativsystemet Pocket PC med hjälp av Visual Basic .NET. Programmet skulle ha samma funktionalitet som det delprogram som ingår i TULP och körs på handdatorer. Det nya programmet skulle även möjliggöra en framtida multispråksanpassning. Till det nya programmet skall också en teknisk dokumentation skrivas.

Det program som utvecklats kan köras på operativsystemet Pocket PC 2002 och är skrivet i Visual Basic .NET. All funktionalitet som fanns i det gamla programmet finns även i det nya programmet.

Målet att möjliggöra en framtida multispråksanpassning har uppnåtts. All text i användargränssnittet för det nyutvecklade programmet representeras med hjälp av strängkonstanter. Vid en eventuell vidareutveckling av mulitspråksanpassning i TULP rekommenderar vi att dessa strängkonstanter byts ut mot anrop till metoder i en egenutvecklad språkklass. Denna metod skulle då returnera den aktuella textsträngen för det valda språket.

Målet att skriva en teknisk dokumentation för det nya programmet har inte nåtts eftersom tiden inte har räckt till. Detta är det enda mål vi inte nått.

Utöver de mål vi nått har vi förändrat användargränssnittet till det bättre och programmets funktioner är numer åtkomliga på nya och mer intuitiva sätt.

Det nyutvecklade programmet upplevs effektivare än det gamla. Inga tekniska mätningar har kunnat genomföras men det är ändå en märkbar skillnad vid normal användning.

En förändring som bör göras i programmet avser implementationen av metoden för att komma åt rullningslisten som beskrevs i avsnitt 5.3. Om det finns möjlighet att komma åt och använda rullningslisten på annat sätt bör detta övervägas eftersom den nuvarande implementationen inte är dokumenterad i .NET och därför kan inga garantier ges för funktionsdugligheten i framtida .NET versioner.

I det program vi har utvecklat har det grafiska användargränssnittet skilts ifrån logiken för att dessa delar skall vara oberoende av varandra och därför underlätta vidareutveckling. Skulle

vi göra om programmet skulle vi överväga om detta är en bra teknik i det här fallet. Om delarna vore integrerade skulle logiken vara enklare att implementera samtidigt skulle modulariseringen försämras. Vad som avgör om delarna bör vara integrerade eller skilda är hur stor sannolikheten för en framtida vidareutveckling är. Vi är dock nöjda med resultatet av vårt val att separera logiken från användargränssnittet. Därmed har vi möjliggjort att enklare kunna vidareutveckla programmet.

8 Summering av projektet

I detta kapitel kommer vi att sammanfatta arbetet, ta upp vilka slutsatser som dragits och vad vi har lärt oss.

Av de mål som sattes upp i avsnitt 1.2 och mättes i kapitel 7 hade vi förväntat oss att klara samtliga. Anledningen till att vi inte uppfyllde målet att skriva en teknisk dokumentetion var att arbetet tog längre tid än vad vi från början räknade med. En stor anledning till att det tog längre tid än beräknat var att det tog lång tid att komma igång. Det som tog mest tid i början var att sätta upp en utvecklingsmiljö för den avsedda plattformen tillsammans med handdatorns utvecklingspaket. Då handdatorns utvecklingspaket för .NET plattformen inte var färdig användes en testversion som saknade delar av dokumentationen vilket ledde till att det var svårt att komma igång med användningen.

En slutsats om själva arbetet är att moment ofta tar längre tid än vad man först tror och det är därför svårt att konstruera en bra tidsplan vid arbetets början. Vi har under arbetets gång inte stött på något problem där vi helt kört fast. Däremot har vi stött på många mindre problem som tillsammans har tagit lång tid att lösa.

Andra slutsatser som dragits under arbetets gång rör främst .NET plattformen och det programspråk som använts. Vi tycker att .NET är en bra plattform för den här typen av projekt. Program skrivna för .NET på handdatorer kan även köras på stationära datorer som har .NET Framework installerat. Utvecklingsmiljön innehåller en stor mängd standardklasser som är enkla att använda och underlättar utveckling för plattformen. Programspråket Visual Basic .NET som användes i projektet passade mycket bra eftersom språket är utvecklat speciellt för .NET. Det är även ett enkelt språk att använda vid utveckling av grafiska användargränssnitt.

Under arbetets gång har vi dragit många nya lärdomar. Bland annat har vi lärt oss det nya programspråket Visual Basic .NET som vi arbetat med under hela arbetet. Vi har även lärt oss hur man utvecklar program speciellt för handdatorer vilket vi inte hade någon erfarenhet alls av innan vi påbörjade arbetet. Det var också lärorikt att hantera den speciella hårdvaran som streckkodsläsaren innebär. I övrigt har arbetet gett oss erfarenhet av mjukvaruutveckling i allmänhet.

Referenser

[1] Craig Utley. A Programmer's Introduction to Visual Basic .NET. Sams, 1:a upplagan, 2001.

[2] Dave Grundgeiger. Programming Visual Basic .NET. O'Reilly & Associates, 1:a upplagan, 2001.

[3] Steven Roman, Paul Lomax, Ron Petrusha. VB.NET Language in a Nutshell. O'Reilly & Associates, 1:a upplagan, 2001.

[4] Lars-Ove Larsson. User Manual PPC, Prevas AB, KS010B03.

A Beskrivning av .NET

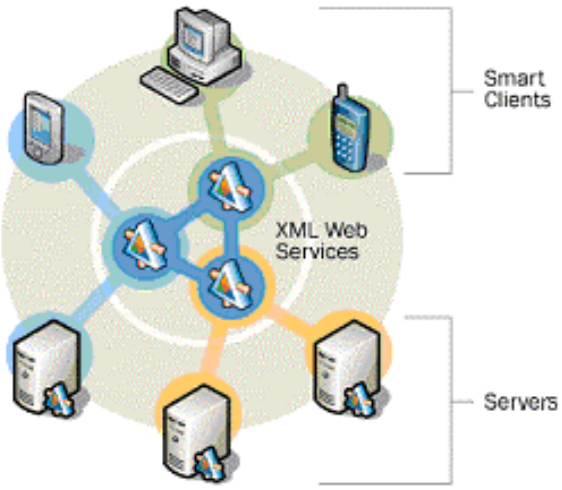
Vid utvecklingen av TULP används plattformen .NET. För att få förståelse för den utvecklingsmiljö som .NET utgör kommer plattformen att beskrivas i denna bilaga. .NET är ett stort område och vikten har lagts på att beskriva de delar som är viktiga för vår uppgift. Andra viktiga komponenter i .NET har beskrivits överskådligt.

Först ges en kort översikt av .NET som följs av en noggrannare beskrivning av .NET Framework och dess mest grundläggande delar. Därefter ges en beskrivning av .NET Compact Framework som är den version av ramverket som används vid utvecklingen av TULP. Slutligen beskrivs Visual Basic .NET som är det programspråk som används.

För att skriva detta kapitel har vi använt oss av [1],[2] och [3].

A.1 Översikt

Microsoft .NET är en plattform som ger förutsättningar för att koppla samman olika enheter på ett effektivt sätt. Dessa enheter kan till exempel vara handdatorer, servrar, PC-datorer och mobiltelefoner.



Figur A.1; .NET kommunikationsöversikt.

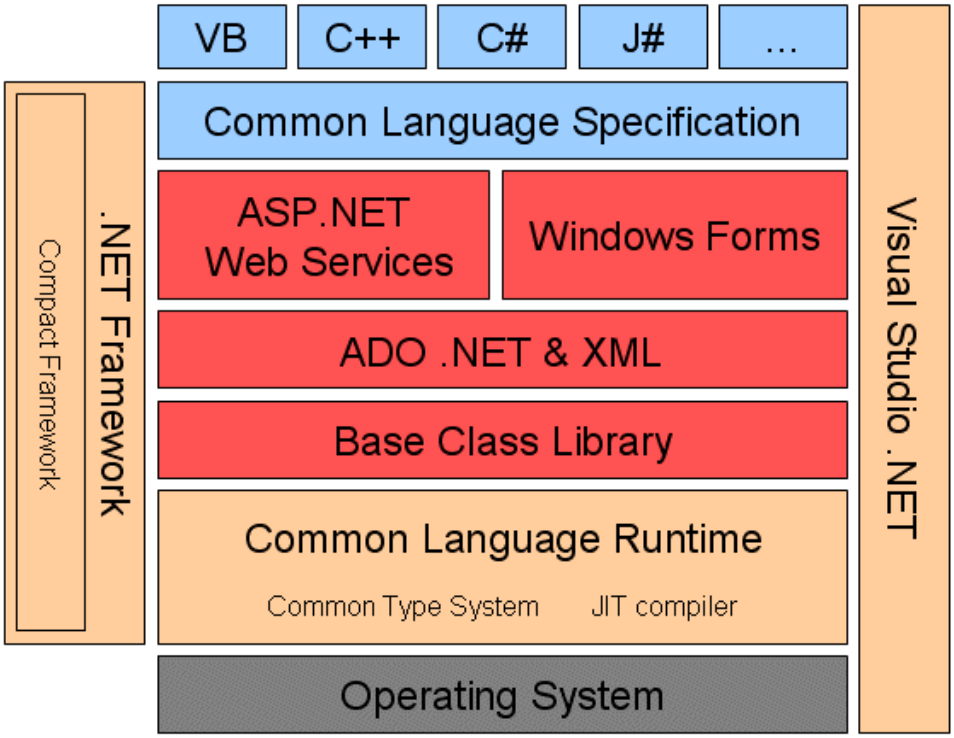
Gemensam funktionalitet och regler för kommunikation mellan de olika enheterna finns i ramverket .NET Framework som beskrivs i avsnitt A.2.

.NET introducerar en ny teknik för att skapa distribuerade system som kommunicerar över nätverk som till exempel Internet. Denna teknik kallas för XML Web Services och är en viktig del av .NET-plattformen. Tekniken använder Extensible Markup Language (XML) för att skicka data mellan program på enheter ansluta till ett nätverk.

A.2 .NET Framework

.NET Framework är en samling av tjänster och klasser som fungerar som ett lager mellan program och operativsystemet. .NET Framework är inte enbart bundet till operativsystemet Windows utan kan användas ihop med samtliga operativsystem, men i dagsläget finns enbart en version för Windows implementerad. Program skrivna för .NET kompileras inte direkt till maskinkod vilket innebär att programmen är plattformsoberoende och kan köras på alla operativsystem som har .NET Framework installerat. .NET är konstruerat för att vara programspråksoberoende och alla programspråk kan använda denna modell. Modellen innebär bland annat att datatyper har samma representation för alla programspråk byggda för .NET. Exempel på detta är att datatypen integer alltid representeras med 32 bitar.

Delarna i .NET Framework visas i figur A.2.



Figur A.2; Delarna i .NET Framework.

A.2.1 Common Language Runtime

Common Language Runtime (CLR) är en mekanism för att ladda och köra program. CLR definierar ett standardiserat typsystem vilket möjliggör att data kan skickas mellan olika komponenter oberoende av vilket programspråk de är skrivna i. CLR tillhandahåller även funktionalitet för undantagshantering, debugging, säkerhet och versionshantering. Under körning hanterar CLR objekts livstid genom att ta hand om referenser till objekt samt att ta bort dem från minnet då de inte längre används. CLR möjliggör även att ett program kan köras på olika operativsystem utan att behöva kompileras om.

- **Common Type System**

Common Type System (CTS) är det typsystem som finns i CLR. CTS definierar hur olika typer skall representeras och användas under körning. Då samma grundtyper används underlättas kommunikation mellan komponenter skrivna i olika programspråk. Både primitiva och egendefinierade typer som till exempel klasser och strukturer innehåller typinformation som gör att det är lätt att skicka och ta emot objekt mellan programmoduler skrivna i olika programspråk.

- **Just-In-Time kompilering**

När källkod kompileras för .NET plattformen skapas kod i ett språk som kallas Intermediate Language (IL). IL är ett maskinoberoende språk som genereras vid kompilering oavsett vilket programspråk källkoden är skriven i. IL-koden sparas vanligast som en exe-fil. Vid körning av exe-filen startar CLR den så kallade Just-In-Time (JIT) kompilatorn. JIT-kompilatorn kompilerar IL koden och skapar maskinberoende exekverbar kod. Denna mekanism kan jämföras med Javas byte-kod och Java Virtual Machine (JVM). JIT kompilering kan ske på två sätt. Det ena sättet är att kompilera all kod direkt vid körning och det andra sättet, som är det vanligaste, är att kompilera koden allt eftersom den används.

Ett exempel på hur IL kod kan se ut ses nedan.

Visual Basic .NET kod:

```
Module HelloWorld
```

```
    Sub Main()
        System.Console.WriteLine("Hello World!")
    End Sub

End Module
```

Genererad IL kod:

```
.method public static void  Main() cil managed
{
  .entrypoint
  .custom instance void
[mscorlib]System.STAThreadAttribute::.ctor() = ( 01 00 00 00 )
  // Code size       14 (0xe)
  .maxstack  8
  IL_0000:  nop
  IL_0001:  ldstr      "Hello World!"
  IL_0006:  call       void
[mscorlib]System.Console::WriteLine(string)
  IL_000b:  nop
  IL_000c:  nop
  IL_000d:  ret
} // end of method HelloWorld::Main
```

A.2.2 Base Class Library

.NET Base Class Library (BCL) är en stor samling av användbara klasser och datatyper. Klassbiblioteket är hierarkiskt indelat i så kallade namnutrymmen (namespace) vilket är en logisk uppdelning av relaterade klasser. Denna uppdelning gör att klasserna är lätta att särskilja och använda. Under körning tillhandahåller biblioteket systemservice till det exekverande programmet i form av till exempel undantagshantering. BCL definierar även de primitiva datatyperna.

A.2.3 ADO.NET & XML

ActiveX Data Objects (ADO) .NET är en modell för dataåtkomst. ADO.NET utgörs av en mängd klasser som använder XML för att kommunicera med databaser.

A.2.4 ASP.NET & Web Services

Active Server Pages (ASP) .NET är en metod för att skapa dynamiska websidor. ASP.NET komponenter kan skrivas i alla .NET anpassade programspråk. När en webserver får en förfrågan om en ASP.NET websida exekveras en ASP.NET komponent och HTML-kod genereras och skickas till webläsaren på klientsidan.

Syftet med Web Services är att kunna placera program på olika servrar och sedan anropa dessa genom HTTP-protokollet. Fördelen med Web Services är att man kan distribuera program över hela Internet istället för att vara bunden till ett internt nätverk. ASP.NET kan användas för att skapa websidor som fungerar som grafiska användargränssnitt till Web Service.

A.2.5 Windows Forms

Windows Forms är samling klasser som används för att skapa grafiska användargränssnitt för alla program som använder sig av .NET Framework. Detta gör att användargränssnitt ser likadana ut oberoende av vilket programspråk som använts.

A.2.6 Common Language Specification

Common Language Specification (CLS) specificerar hur olika komponenters gränssnitt skall se ut. Språk anpassade för .NET Framework måste följa CLS specifikationen vilket innebär att till exempel ett objekt representeras på ett givet sätt. Ett språk måste inte innehålla alla delar av CLS för att kunna använda ramverket men de delar som finns måste följa CLS. Genom att ett programspråk följer CLS garanterar ramverket ett språkoberoende gentemot övriga språk som följer CLS.

A.3 .NET Compact Framework

.NET Compact Framework är en plattform för så kallade smarta enheter som till exempel handdatorer och telefoner. Vid utveckling av TULP kommer därför detta ramverk att användas. Compact Framework en är nerskalad version av .NET Framework som gör det möjligt för utvecklare att skriva vanliga .NET-program för smarta enheter. Detta leder till att en utvecklare kan använda valfritt programspråk och arbeta i samma utvecklingsmiljö som används för utveckling av vanliga PC-program. Compact Framework tillhandahåller en säker miljö som skyddar enheter mot programkrascher och skadlig kod. Compact Framework och det vanliga ramverket fungerar på samma sätt men skiljer sig åt innehållsmässigt. Compact Framework innehåller mycket färre klasser för att få plats på enheter med begränsat

minnesutrymme. .NET Framework har ett utbrett stöd för återanvändning av äldre komponenter men detta har skalats bort i Compact Framework.

A.4 Visual Basic .NET

Visual Basic .NET är ett programspråk som är utvecklat för .NET Framework vilket medför att alla fördelar som .NET-plattformen erbjuder kan utnyttjas.

Visual Basic .NET har i stor utsträckning samma syntax som äldre versioner men skiljer sig i övrigt mycket åt. Ett exempel på detta är att när en parameter skickas till en funktion utan att det anges om den skall skickas som värde (by-value) eller referens (by-reference). Själva funktionsanropet ser likadant ut som i föregående versioner men i .NET versionen skickas parametern som värde medan tidigare versioner skulle ha skickat en referens. Visual Basic .NET är första versionen av Visual Basic som stödjer objektorienterad programmering fullt ut. Tidigare versioner har varit objektbaserade men nu kan man till exempel använda sig av arv. Visual Basic .NET använder tillsammans med alla programspråk byggda för .NET ett gemensamt ramverk vilket medför att ett program skrivet i Visual Basic vid kompilering genererar samma slags maskinoberoende kod som övriga programmeringsspråk. Detta gör att ett program skrivet i Visual Basic .NET blir lika effektivt som ett program skrivet i C++ för .NET.

B Förkortningar

1D	Endimensionell
2D	Tvådimensionell
ADO	ActiveX Data Objects
ASP	Active Server Pages
BCL	Base Class Library
CLR	Common Language Runtime
CLS	Common Language Specification
CTS	Common Type System
DLL	Dynamic Link Library
IL	Intermediate Language
JIT	Just-In-Time
JVM	Java Virtual Machine
HTML	Hyper Text Markup Langauge
HTTP	Hyper Text Transfer Protocol
TULP	Tracy Update and Label Printing
XML	Extensibel Markup Language