

# **Winflex PlantMaker**

## **Programvara för konfigurering av styr och reglerutrustning**

---

Ett examensarbete i Datalogi

Utfört av:

Olof Zandrén CTH

Handledare:

Håkan Sundell (DV)  
Gunnar Åberg (ReginTech)

# Sammanfattning

PlantMaker är ett projekt inom Regin Technology AB som syftar till att ta fram en Windows baserad programvara för konfigurering av reglersystem. PlantMaker skall användas vid konstruktion, installation och underhåll av byggnadsautomationssystem. Programmet skall klara av att konfigurera två olika kommunikationsprotokoll, SRDLink och LonWorks. SRDLink är ett protokoll framtaget av ReginTech för att sköta kommunikationen mellan deras egna enheter. LonWorks är en nätverksplattform framtagen av det amerikanska bolaget Echelon. Det har på senare år blivit en standard inom byggnadsautomatisering.

Byggnadsautomation är styr- och regler funktioner som automatiseras hos byggnaders fasta installationer. Det kan vara funktioner som sköter värme, ljus, ventilation etc. Detta görs för att minska kostnaderna, underlätta underhållet, öka säkerheten och framförallt göra det så bekvämt som möjligt för de som bor i byggnaden.

Hela projektet går ut på att ta fram en fungerande programvara för PC datorer. Resultatet av examensarbetet är ett användargränssitt med de basfunktioner som skall finnas i den färdiga produkten. Det som återstår för en komplett programvara är att implementera kommunikation mellan datorn och styrenheterna. Programstrukturen är uppbyggd för att underlätta utbyggnad med de nödvändiga kompletteringarna.

# Abstract

The purpose of the PlantMaker project is to develop software for design/programming of HVAC control systems for Regin Technology AB. PlantMaker will be used in construction, installation and management of building automation control systems. The application shall be able to configure two different communication protocols, SRDLink and LonWorks®. SRDLink is a protocol developed by ReginTech to operate the communication between its own units. LonWorks is a standard protocol used world wide, and is developed by the American company Echelon.

Building automation is when the main control functions of a building are automated. It could be heating, air conditioning, light, security equipment, etc. The purpose of this development is to reduce the costs, make it easier to maintain, increase the security but most of all to increase the comfort of building occupants.

The goal of the project is to develop software for a PC. The result of this essay is the user interface and the basic functions for this application. The remaining implantation for a full-fit application is to implement the communication between the computer and the actual units. The code is structured to make it easy to complement it with the necessary functions.

# Förord

Projekt Winflex PlantMaker är ett examensarbete i Datalogi vid Institutionen för Datavetenskap (DV), Chalmers Tekniska Högskola. Arbetet har utförts på Regin Technology AB där handledarna Gunnar Åberg och Peter Kandemalm har varit till stor hjälp. Vill även nämna företagsledarna Sven Emanuelsson och Kent Samuelson som gav mig möjligheten att göra examensarbetet på företaget. Min handledare vid institutionen, Håkan Sundell skall ha ett stort tack för handledning av hela examensarbetet och främst vid rapportskrivningen. Ett stort tack till ovanstående samt till övrig personal på ReginTechnology.

# Innehållsförteckning

1 Inledning	1
1.1 Problemområde	1
1.1.1 Standardprotokoll	3
1.2 Regin Technology AB	3
1.3 Disposition	4
2 Teori	4
2.1 LonWorks	4
2.1.1 Lon-hårdvara	5
2.1.2 LonTalk	5
2.1.3 Interoperabilitet	6
2.2 SRDLink	7
2.2.1 Transportlager	7
2.2.2 Paketstruktur	8
2.2.3 Applikationslager	8
2.2.4 Nätverk	8
2.3 XIF-filen	9
2.4 Resursfiler	9
2.5 Kommunikationsbibliotek	11
3 Systemkrav	12
4 Analys	13
4.1 Verktyg	13
4.2 Användarna	13
4.3 Användargränssnittet	14
4.3.1 Automate	14
4.3.2 PlantMaker	15
4.4 Programstruktur	17
4.4.1 Klasser	18
4.5 Speciella lösningar	19
4.5.1 Lägg till DUC	20
4.5.2 Lägg till DUC typ	21
4.5.3 DUCens format	22
4.5.4 Editering	23
4.5.5 Standardvärden	23
4.5.6 Variabelinitiering	24
5 Resultat	24
5.1 Implementering	25
5.2 Utveckling	25
6 Slutsats	27
Referenser	28
Ordlista	29
Appendix A	30

# 1 Inledning

Regin Technology AB (ReginTech) utvecklar produkter för fastighetsautomation. Det är framför allt regulatorer som kan anpassas efter kundens behov. Dessa är installerade med ett kommunikationsmedium som i dagsläget består av antingen LonWorks eller SRDLink. För att underlätta installation och konfigurering av dessa enheter så skall en programvara utvecklas som skall göra det möjligt att göra detta från en dator. I programmet som kallas PlantMaker skall man kunna bygga upp ett system av olika enheter och konfigurera dessa. Programmet bygger upp installationspaket som skickas till enheten via exempelvis Internet. Med hjälp av applikationen skall det även vara möjligt att styra enheterna online. Det största problemet är att binda ihop de två olika standarderna LonWorks och SRDLink. Eftersom programmet skall kunna styra båda, krävs det en allmän lösning som klarar detta. För att veta hur enheterna ser ut används det s.k. XIF-filer (External Interface File) och resurs filer. XIF-filen beskriver hur enheten är uppbyggd med olika variabler, ingångar och utgångar. I resurs filerna finns det information om hur variablerna är uppbyggda. Det är t.ex. typ, min/max, format etc. Genom att kombinera inläsning från dessa olika filer byggs enhetens struktur upp i programmet.



*Figur 1.1: Bild på SRD3100 och dess display. En av ReginTechs produkter som använder sig av SRDLink.*

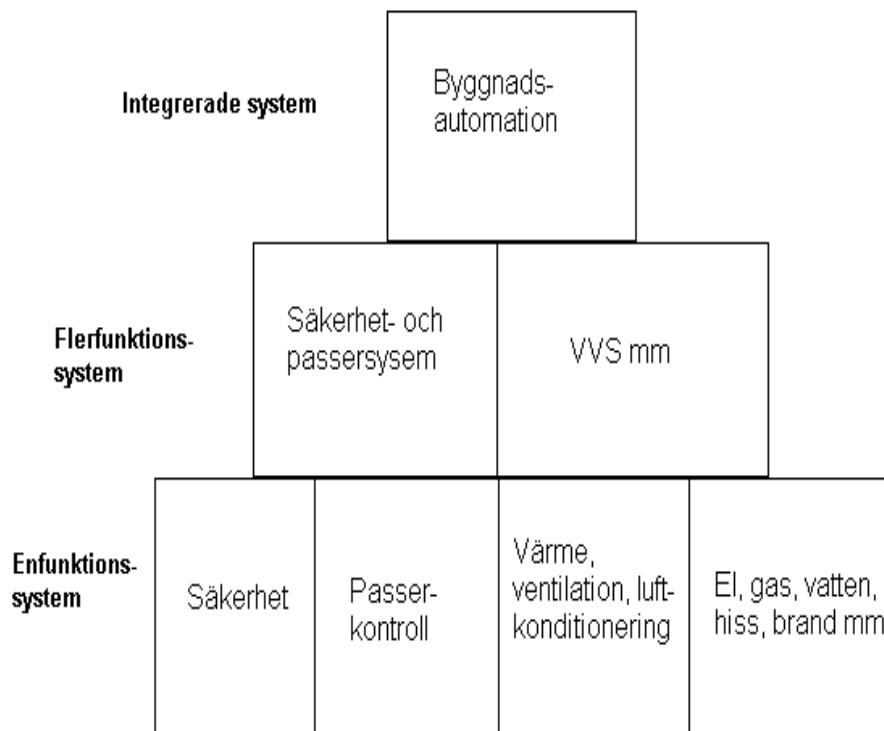
Idag finns det en programvara (AutoMate), utvecklad av ReginTech, som sköter kommunikation och konfigurering av en enhet. Det finns tre nackdelar med detta program som gör det nödvändigt att utveckla en ny applikation. Den första nackdelen är att programmet bara kan sköta en enhet i taget. Den klarar inte av att bygga upp ett helt system av enheter med kommunikation mellan dessa. Den andra nackdelen är att den använder sig av en databas som är komplicerad och gör programmet långsamt. Slutligen går den bara att använda på enheter som använder sig av SRDLink. Den här lösningen har nått sina begränsningar och blir alldeles för dyr och kostnadskrävande att uppgradera till de nya kraven. PlantMaker är tänkt som en efterträdare till denna programvara.

## 1.1 Problemområde

Med byggnadsautomation avser man oftast automatiserade styr- och reglerfunktioner av en byggnads fasta installationer som t ex värme, ventilation, ljus, säkerhetssystem o s v. Målet med automatiseringen är bland annat att minska energiförbrukningen, förenkla underhållet, öka säkerheten samt att göra det mer bekvämt för människorna i byggnaden [9]. Vanliga styrsystem för fastigheter har sedan långt tillbaka arbetat med centraliserade styrlösningar. Det började runt 1950 med relälösningar, omkring år 1960 kom datorer, i början på 1980-talet

kom fastigheter att börja använda DUC-system (*DataUnderCentral*). Dessa typer av centraliserade systemlösningar baserar sig på att en ledning dras från varje givare, terminal, motor, spjäll etc. till centralenheten. I fastigheter är problemet extra besvärligt eftersom varje objekt är unikt och dessutom byggs om då och då. Det gör installation och kabeldragning både dyrbar och besvärlig. I dessa anläggningar är ombyggnad av ledningsnäten 2-3 ggr dyrare än vid nybyggnad. Det är vanligt att det finns helt individuella ledningsnät för varje enskilt system i fastigheter som t ex brandlarm, tjuvlarm, personsökning, belysningsstyrning, larmöverföring, ventilation, tidstyrning, mätarinsamling, elvärme, värme, effekt mm (enfunktionssystem) [10].

För att förenkla styr- och reglerförfarandet samt minska projekterings- och underhållskostnaden i enfunktionssystem, finns ett behov av mer flexibla system. Modern teknologi möjliggör automatisk styrning och reglering av en mängd olika funktioner, men för att automatiseringen verkligen skall löna sig krävs att de olika enfunktionssystemen integreras med varandra. Uppgiften för ett modernt byggnadsautomationssystem skall vara att effektivt, flexibelt och driftsäkert styra och reglera alla olika funktioner i byggnaden. Vinster på minskade kostnader för t ex underhåll och energi inte bara rättfärdigar utan nödvändiggör i många fall investeringen i ett sådant system [9]. Under slutet av 80-talet anammades koncept med en gemensam kommunikationsplattform av fastighetsinstallationsbranschen. Genom sammanslutningar av leverantörer från olika områden som belysning, styr och regler, passage- och inbrottslarmssystem m.m. görs det ett försök att överbrygga de traditionella entreprenadgränserna och skapa integrerade system för byggnader (se figur 1.2).



**Figur 1.2:** Från enfunktionssystem till integrerade system.

### 1.1.1 Standardprotokoll

Det talas mer och mer om olika standardprotokoll för kommunikation inom och mellan styrsystem för VVS-installationer. Anledningen är att en sådan utveckling underlättar livet för de flesta aktörerna på styrmarknaden, inte minst för de som ska knyta ihop alla funktioner i de allt mer integrerade systemen. En kort översikt av de viktigaste protokollen presenteras i tabell 1.1 nedan. Några av dessa protokoll har en liten betydelse på byggnadsautomationsmarknaden och ser inte ut att vinna mark, t.ex. FND, WordFIP och

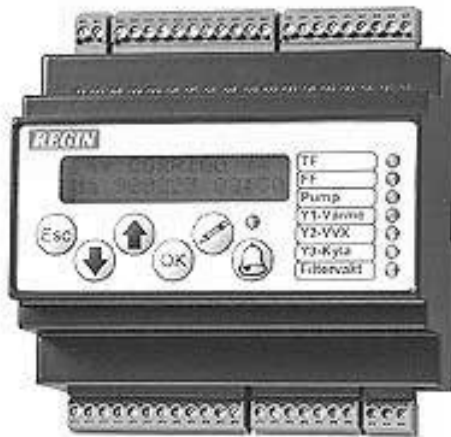
Namn	Företag/Organisation	Övrigt
BAC-net	ASHRAE	Amerikansk standard ANSI 135-1995 ASHRAE är den amerikanska motsvarigheten till VVS-tekniska föreningen. Stark ställning.
FND		Från Tyskland, används främst för anslutning av överordnade system till byggnadsautomation i statliga byggnader
PROFIBUS		Från Tyskland. Europeisk industristandard, EN 50170. Stark ställning.
WordFIP		Fransk-Amerikanskt protokoll. Europeisk industristandard, EN 50170
LonWorks	Echelon/LonMark	Amerikanskt, har rönt stort intresse inom byggnadsautomationsindustrin.
EIB	Siemens	Primärt för elektriska installationer som belysning mm.
BATIBUS		Från Frankrike. Används främst för enklare funktioner i byggnader
EHS		EU-projekt med syfte att ta fram ett protokoll lämpligt för automation av bostäder

**Tabell 1.1:** Översikt av de viktigaste standardprotokollen inom byggnadsautomation.

EHS. Andra däremot, till exempel LonWorks och BACnet har en tydlig inriktning på byggnadsautomation och en ”neutral” organisation i bakgrunden. Det är därför troligt att dessa protokoll kommer att hävda sig väl i framtiden [10].

## 1.2 Regin Technology AB

Regin Technology AB har utvecklat ett flertal produkter för byggnadsautomation och då främst styr- och reglerenheter för värme och ventilation. Produkterna kan delas upp i tre olika grupper: Regulatorer som är helt anpassningsbara till kundens applikation, regulatorer som delvis kan anpassas efter kundens krav samt överordnad programvara för PC-miljö. I dagsläget använder sig ReginTech av två olika kommunikationsstandarder, LonWorks och det egenframtagna SRDLink.



*Figur 1.3: Bild på en Corrigo C20. En produkt från ReginTech som använder sig av LonWorks protokollet.*

## 1.3 Disposition

Rapporten börjar med en teoridel som beskriver grundläggande begrepp och de olika tekniker som projektet kommer i kontakt med. Därefter kommer en analys och metodbeskrivning där problemet beskrivs mer genomgående och tillvägagångssättet för att lösa problemet. Till slut finns det en resultatdel med diskussioner och slutsatser.

## 2 Teori

### 2.1 LonWorks

LonWorks® är en nätverksplattform som det amerikanska företaget Echelon står bakom. Echelon började utveckla konceptet i slutet på 80-talet och fortsatte en bit in på 90-talet. Med tiden har tekniken blivit erkänd och används idag över hela världen för automation av fastigheter, energitillämpningar, industrier och mycket mer [8]. LonWorks betecknar tekniken för det fysiska **styr- och reglernätverk** som Echelon har utvecklat. Tekniken gör det möjligt för utrustning och apparater från olika tillverkare att samexistera på ett gemensamt nätverk, tack vare det gemensamma protokollet **LonTalk** [3].

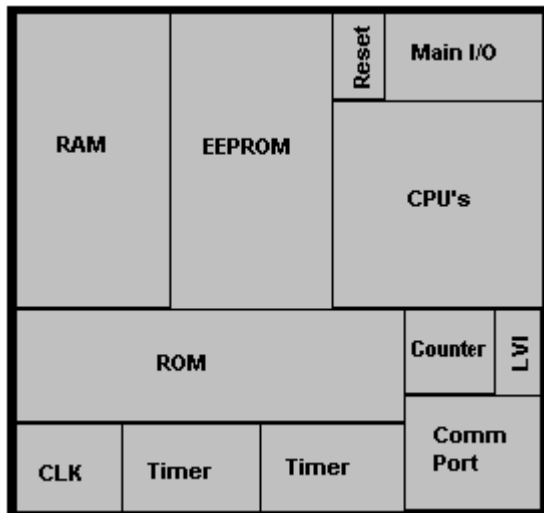
Genom att bygga upp LonWorks-nätverk erhålls funktioner för att effektivisera och förbättra exempelvis uppvärmning, ventilation, passagekontroll, belysning och mycket mer. Flera tusen företag utvecklar produkter baserade på LonWorks-tekniken och genom dessa produkter kan intelligenta nätverk med smarta funktioner byggas upp av installatörer och systemintegratörer. Några fördelar med LonWorks är flexibilitet, energieffektivisering, lättare och bättre drift och förbättrad säkerhet. Ett LonWorks-nät kan bestå av mellan 2 och 32 000 **noder**. Nätet kan enkelt kompletteras och byggas ut i takt med att behoven ökar [8].

LonWorks-tekniken består i huvudsak av följande beståndsdelar:

- LonWorks-hårdvaran: Neuronprocessor och transceiver (sändare/mottagare).
- LonWorks firmware: Innehåller bland annat kommunikationsprotokollet LonTalk och ett distribuerat realtidsoperativsystem.
- LNS – LonWorks Network Services: Tillhandahåller en plattform för att utveckla Windows-baserade applikationer för LonWorks-system.

### 2.1.1 Lon-hårdvara

**Neuronen** är en programmerbar integrerad krets med utgång för kommunikation via en transceiver till nätverket. Ett Neuron-chip innehåller tre mikroprocessorer, tre minnesområden (ROM, RAM samt EEPROM) samt kommunikations- och I/O- portar. Figur 2.1 visar en schematisk bild av neuronen.



Figur 2.1: Bild över Neuronen.

I ROM finns kommunikationsprotokollet LonTalk och ett I/O funktionsbibliotek. Flashminnet (EEPROM) är till för konfigurationsdata och applikationer. De är således säkrade mot förlust vid strömavbrott. Varje Neuron-chip förses vid tillverkningen med en 48 bitars ID-nummer, vilken säkerställer att två neuroner med samma ID-nummer inte existerar. De flesta LonWorks-enheter är försedda med en servicepinne som används när noder adresseras på ett nätverk. När servicepinnen aktiveras sänds neuronens ID-nummer ut på nätverket. Detta möjliggör att ett anslutet installationsverktyg kan fånga upp ID-numret och lägga in noden i databasen. Neuronen är framtagen av Echelon tillsammans med Motorola och Toshiba.

**Transceivern** (sändaren och mottagare) utgör ett kommunikationsgränssnitt mellan neuronchipsen och ett fysiskt media, s k. **kanal**, som transporterar datapaket till andra enheter i nätverket. Varje enhet som är uppkopplad mot en specifik kanal måste ha en kompatibel transceiver som körs på samma bitrate. Echelon kan erbjuda transceivers för flera olika sorters kanaler där de vanligaste är partvinnad kabel, elnätet, koaxialkabel, fiberkabel, IR-sändare och radio [3].

### 2.1.2 LonTalk

LonTalk-protokollet utvecklades för att kunna erbjuda interoperabilitet i styr- och reglernätverk inom industri- och byggnadsautomation. LonTalk är för närvarande en industristandard för branscher som uppvärmning, ventilation, luftkonditionering, halvledartillverkare (SEMIC) och för EIA (the Electronics Industry Alliance). Till en början var LonTalk patentskyddat, vilket gjorde att stora resurser sattes in för att ta fram

konkurrerande protokoll. LonTalk har sedan dess blivit en öppen standard som vem som helst kan ta del av. LonTalk stödjer samtliga av OSI-modellens sju lager (Open System Interconnection) som är standardiserade av ISO (International Standardization Organization). Det betyder för användaren att samtliga lager är transparenta och tillgängliga. Detta medför att användare av produkter med Lon-kommunikation kan bygga upp ett system med komponenter från olika tillverkare, därav uttrycket interoperabilitet. [2]

I tabell 2.1 presenteras OSI-modellens sju lager och hur de används i LonTalk.

	OSI Layer	Purpose	Service Provided
7	Application	Application Compatibility	Standard Network Variable Types
6	Presentation	Data Interpretation	Network Variables; Foreign Frame Transmission
5	Session	Remote Actions	Request-Response; Authentication; Network Management; Network Interface
4	Transport	End-to-End Reliability	Acknowledged & Unacknowledged; Unicast & Multicast Authentication; Common Ordering; Duplicate Detecting
3	Network	Destination Addressing	Addressing Routers
2	Link	Media Access and Framing	Framing; Data Encoding; CRC Error Checking; Persistent CSMA; Collision Avoidance; Optional Priority & Collision Detection
1	Physical	Electrical Interconnection	Media-Specific Interfaces and Modulation Schemes <small>(twisted pair, power line, radio frequency, Coaxial cable, infrared, fiber optic)</small>

*Tabell 2.1: OSI-modellens sju lager och hur de används i LonTalk. [3]*

### 2.1.3 Interoperabilitet

För att produkter från olika tillverkare inom samma segment skall kunna kommunicera utan hinder finns en oberoende organisation, LonMark, som arbetar för att produkter designas på ett föreskrivet sätt. På detta sätt kan en installatör enkelt sätta ihop system med produkter från mängder av tillverkare under förutsättning att dessa arbetat efter LonMarks riktlinjer.

Normalt i ett LonWorks-system skickar noderna inte kommandon till varandra, utan de utbyter datapaket som innehåller information om t ex temperatur, tryck, tillstånd, datum och tid. Noderna använder sedan informationen på olika sätt beroende på den funktion som noden har. Inom LonWorks kan dessa datapaket ses som globala variabler som finns på nätverket och de har därför fått namnet **nätverksvariabler**. Om en nod uppdaterar en nätverksvariabel skickas denna automatiskt ut på nätverket, vilket medför att andra noder blir medvetna om det nya värdet. För att uppnå fullständig **interoperabilitet** i ett nätverk räcker det inte med att finnas på samma nätverk, ha samma typ av transceiver och kunna skicka nätverksvariabler. Noderna måste också kunna förstå innehållet i nätverksvariablerna, vilket erhålls i LonWorks genom att endast använda standardiserade typer av nätverksvariabler. Dessa typer benämns **SNVT** (Standard Network Variable Types) och de innehåller bland annat information om enhet, upplösning och vilka värden typen kan anta. Även om det finns många olika sorters

SNVT: är att välja mellan kan det hända att det inte finns en typ som passar för en viss funktion. I ett sådant fall tvingas man att skapa en egen typ av nätverksvariabel. Den egna typen benämns då som en **UNVT** (User defined Network Variable Type).

Ofta behöver man **konfigurera** en nod innan den kan tas i bruk och utföra de funktioner som den är avsedd för. Hur konfigurationen utförs beror på hur programmet i noden är konstruerat. En vanlig metod är att använda en speciell typ av nätverksvariabler som kallas för **nci** (Network Configuration Variable Input) eller konfigureringsvariabler. Denna metod är enkel att implementera och använda men då antalet nätverksvariabler är en begränsad resurs på noden är den mindre lämplig då det rör sig om större mängder av konfigurationsdata. I dessa fall finns det en mer kompakt metod där **konfigureringsparametrar** lagras i filer på själva noden. För att läsa och skriva till dessa används antingen filöverföring eller speciella nätverkskommandon för direkt åtkomst till nodens minne. Metoden med konfigureringsparametrar är betydligt mer komplicerad att implementera. På samma sätt som för nätverksvariabler kan även konfigurationsegenskaper vara standardiserade eller egendefinierade. Standardtyper för dessa benämns **SCPT** (Standard Configuration Property Types) medan de egendefinierade benämns **UCPT** (User defined Configuration Property Types). [3]

## 2.2 SRDLink

SRDLink (Styr Regler Duc) är ett kommunikationsprotokoll som togs fram av Kent Samuelsson och Gunnar Åberg i början av 80-talet. Dessa två är grundare och delägare i ReginTechnology AB. Det är ett ASCII-baserat nätverksprotokoll som används av en produktgrupp framtagen av ReginTech. Protokollet består av en tre lagars modell enligt tabell 2.2. Underst finns ett fysiskt lager som sköter själva kommunikationen m h a RS 485 eller RS232 uppkoppling. Över det ligger ett transportlager som sköter inkapslingen och adresseringen av meddelandena. Överst finns ett applikationslager som tar hand om innehållet i det skickade meddelandet.

	<b>Layer</b>	<b>Purpose</b>	<b>Service provided</b>
3	Application	Application Compatibility	Sending/Receiving data
2	Transport	End-to-end Reliability	Acknowledged & Unacknowledged Duplicate Detecting Direct or Broadcast adressing
1	Physical	Electrical Interconnection	RS485 or RS232

*Tabell 2.2: SRDLinks tre skikt och hur de används.*

### 2.2.1 Transportlager

Transportlagret är ett meddelande system som skickar paket till den angivna adressen. Lagret har ansvar för att data, definierad av applikationslagret, kommer fram till rätt enhet. Datan som skickas är av ASCII format. Skickad data skall alltid besvaras med ett meddelande innehållande en bekräftelse från mottagaren att meddelandet är mottaget på ett korrekt sätt.

## 2.2.2 Paketstruktur

Paketstrukturen är indelad i två delar som visas i figur 2.2. Den första delen innehåller adressen till mottagaren av meddelandet. Adressen består av skrivbara ASCII tecken, och

STX	Address header	Generic data	ETX
-----	----------------	--------------	-----

*Figur 2.2: Visar meddelandestrukturen i transportlagret.*

följs av en datadel som kan innehålla en valfri ström av tecken som inte innehåller något kontrolltecken. Dessa två delar kapslas i sin tur in av ett start- (STX) och ett stopp- (ETX) tecken. Den maximalt tillåtna storleken, inklusive start och stopp tecken, är 253 tecken.

## 2.2.3 Applikationslager

Lagret har hand om den faktiska data som skickas mellan två enheter. Sändningen är ett resultat av en direkt läsning eller skrivning till minnet där data finns. Detta lager kapslas alltid in av transportlagret. Den första byten är ett tecken som identifierar vilket paket i ordningen det är. Detta tecken används främst till att kasta bort överflödiga meddelanden som har skickats av någon anledning t.ex. för sent ankommande svar på tidigare sändning. Det kan även användas till att meddela mottagaren om att det kommer flera paket som den skall svara på.

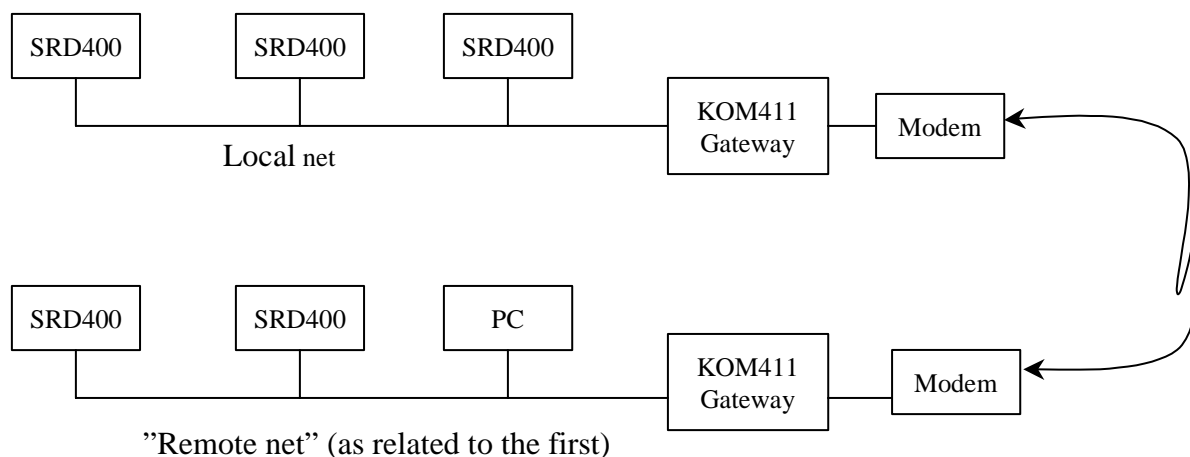
Paketet är indelat i fyra sektioner, som visas i figur 2.3. Den första sektionen är en kommandoinstruktion, som anger vilken operation som skall utföras. Den innehåller även det löpande paketnumret. Den andra delen innehåller en minnesadress. Därefter följer en sektion innehållande längden på den data som skall läsas eller skrivas. Till sist kommer en kontrollsumma på två tecken.

Packet type	Packet contents			
Read	Read command and packet #	20 Bit address	Length	Check sum
Write	Write command and packet #	20 Bit address	Data	Check sum
Read reply	ACK code and packet #	---	Data	Check sum
Acknowledge/Write reply	ACK code and packet #	---	---	Check sum
Negative ACK	NAK code and packet #	---	---	Check sum
Alarm event	Alarm code and packet #	6 char DHC-ID	6 char own ID	Check sum

*Figur 2.3: Schematisk bild över paketstrukturen i applikationslagret.*

## 2.2.4 Nätverk

Ett SRDLink-nätverk är uppbyggt av en eller flera lokala nätverk (se figur 2.4), vilka kan kontrolleras externt m h a en uppkoppling till nätets gateway. Denna gateway består av, i RegnTechs nätverk, en KOM411-enhet och ett tillhörande modem.



*Figur 2.4: Visar kommunikationen mellan två lokala nätverk.*

Varje nod i ett nät har en adress som sträcker sig mellan 0 och 255. Denna adress är bara unik inom det enskilda nätet. Transportlagret skiljer sedan mellan lokala- och fjärrnät vid adresseringen.

## 2.3 XIF-filen

XIF-filen (External Interface File) beskriver en enhets egenskaper. Filen innehåller de SNVT och UNVT variabler som skall bindas och ställas in för att uppnå önskade funktionalitet/samexistens med andra enheter. Det kan finnas olika XIF-filer till samma produkt beroende på hur tillverkaren har tänkt sig att produkten skall fungera. I filen finns det ingen information om de inre algoritmerna i en enhet, utan den beskriver bara vad som är synligt för användarna av enheten. [3]

Filen finns att tillgå i två olika format, binärt eller textformat. PlantMaker läser in XIF-filer av det senare formatet. Textfilen innehåller följande sektioner:

- Header
- Nätverksvariabler och meddelande definitioner.
- Fil definitioner (version 4.0 eller senare)
- Standardvärden på nätverksvariablerna (version 4.0 eller senare).

[7]

## 2.4 Resursfiler

En resursfil innehåller definitioner av resurser som behövs när nätverksverktyg skall tolka mottagen data samt koda data som skickas till en enhet. De här resurserna inkluderar funktionsprofiler, nätverksvariabler, konfigureringsparametrar, strängar och format. Resursfiler används till att definiera både standard och de egendefinierade resurser. I standardfilerna finns all information om de standarddefinierade resurserna. Motsvarande

information för de egendefinierade resurserna finns i filer med namn efter företaget eller enheten. Resursfiler delas in i olika set, där varje set innehåller resurser för en viss enhetstyp. Den variabel som beskriver vilka enheter som berörs av filen kallas ”the scope” (omfattning). Den kan som exempel specificera att resursfilen gäller en speciell enhetstyp, eller till alla enhetstyper. Det finns sju olika nivåer för resursfilens omfattning:

- 0 – Standard** – omfattar alla enheter.
- 1 – Device Class** – omfattar alla enheter med den specificerade enhetstypen
- 2 – Device Class and Subclass** – omfattar alla enheter med den specificerade typen och undertypen.
- 3 – Manufacturer** – omfattar alla enheter från en viss tillverkare.
- 4 – Manufacturer and Device Class** – omfattar alla enhetstyper från en viss tillverkare.
- 5 – Manufacturer, Device Class and Subclass** - omfattar alla enheter från en viss tillverkare med den specificerade typen och undertypen.
- 6 - Manufacturer, Device Class, Subclass and Device Model** – omfattar en viss modell från den specificerade tillverkaren.

I resursfilen finns det ett program-ID som används av nätverksverktyg för att hitta en lämplig fil till en viss enhet. Nivå 0 till 2 är reserverade för standarddefinitioner som är publicerade av Echelon. Eftersom nivå 0 omfattar alla enheter så finns det bara ett set på denna nivå som kallas ”standard resource file set”. Varje set kan innehålla definitioner av följande resurser:

<b>Nätverksvariabler</b>	Information om nätverksvariablerna, vilken inkluderar storlek, enhet, skalfaktor och typ (flyttal, heltal etc.) för varje variabel. Nätverks variabler kan antingen innehålla ett enkelt värde eller en struktur som innehåller multipla fält. (t.ex. SNVT_date_call, som innehåller fält för år, månad och dag). Variabeln kan även innehålla uppräkningsstyper.
<b>Konfigurationsparametrar</b>	Innehåller data rörande konfigureringsparametrar. Datan inkluderar storlek, enhet, skalfaktorer och typ för varje parameter. Liksom nätverksvariabler kan den innehålla strukturer och uppräkningsstyper.
<b>Funktionsprofiler</b>	Funktionsprofiler definierar en mall för funktionsblock, vilket är en samling av nätverks variabler och konfigureringsparametrar som är designade för att utföra en funktion på en enhet. Innehåller både obligatoriska och frivilliga variabler och parametrar. För att implementera ett funktionsblock, måste minst alla obligatoriska data i funktionsprofilen implementeras.
<b>Uppräkningsstyper</b>	En uppräkningsstyp är en lista med heltal, där varje tal är associerad med en textsträng. Uppräkningsstyper definieras antingen i resursfilen eller i en separat header-fil.

## Språksträngar

Alla de tidigare resurserna kan innehålla textreferenser, som används till att beskriva dess namn, enhet eller funktion. Samtliga textsträngar finns sparade i en eller flera språkfiler. Det finns en språkfil för varje språk som resursfilen stödjer. Varje fil har en unik förlängning vilket möjliggör att filen har samma namn som övriga resursfiler i setet.

## Format

Varje nätverksvariabel och konfigureringsparameter måste minst ha ett format definierat. Formatet beskriver hur variabeln eller parametern visas för nätverksanvändaren. Det finns möjlighet att definiera flera format för samma variabel eller parameter (t.ex. kan en temperatur presenteras i Celsius eller Fahrenheit).

[6]

## 2.5 Kommunikationsbibliotek

LonMark tillhandahåller en kommunikationskomponent som är språkoberoende och fungerar som ett interface mellan resursfiler och program. Den versionen som används i PlantMaker ger bara access till att läsa från filerna. Anledningen är att den är gratis och att det aldrig kommer bli aktuellt att skriva till resursfilerna från programmet. Komponenten består av sex DLL (Dynamic Link Library) uppdelade efter olika funktionsområden. Här följer en genomgång av komponenterna:

- **LdrfCatalog** – Det finns en katalog där alla resursfiler samlas för att programmet skall hitta dem. Komponenten innehåller funktioner som används för att sköta denna katalog. En viktig funktion är `OpenCatalog`, som sköter kopplingen mellan programmet och katalogen. Här finns även funktioner som lägger till och tar bort filer från katalogen.
- **LdrfGeneral** – Innehåller funktioner som tar hand om kopplingen till en speciell fil. Den mest använda av dessa är `OpenFile`, som öppnar en fil och returnerar ett nummer som senare används av programmet när filen skall användas. Det finns även funktioner som returnerar version, information om vilket språk som används etc.
- **LdrfLangResource** – Används nästan uteslutande till att hämta strängar som är språkberoende. Det är funktionen `GetStringResource` som används av PlantMaker.
- **LdrfTypes** – Komponenten används till att få ut information ur en speciell resursfil. Via funktionerna ges det övergripande information om vilka variabler som finns i filen. Det går även att få ut typ, struktur och storlek på de variabler som finns i filen. Här ligger även funktioner som öppnar och läser filer innehållandes olika uppräkningsstyper.
- **LdrfTypeTree** – När en nätverksvariabel öppnas med en `LdrfType` så returnerar den en trädstruktur som beskriver hur variabeln är uppbyggd. För att läsa detta träd används de olika funktionerna i `LdrfTypeTree`. Dessa funktioner ger all den information som används vid kontroll av inskrivna värden.

- **LdrfFuncProf** – Används till att få information om de funktionsprofiler som finns lagrade i resursfilerna. Fungerar på liknande sätt som LdrfTypes vad det gäller funktioner och deras uppgifter.

[5]

### 3 Systemkrav

Den främsta orsaken till att ReginTech har valt att utveckla programvaran är att binda ihop de två olika världarna, SRDLink och LonWorks. För att göra detta krävs det ett konfigurationsprogram som klarar av båda typerna. För att klara av det skall PlantMaker använda sig av XIF-filer och resursfiler, samt ha ett användargränssnitt som är applicerbart på båda typerna. Lösningen skall vara så allmän som möjligt och kunna konfigurera alla produkter som har en XIF-fil och tillhörande resursfil.

Det skall i PlantMaker var möjligt att:

- Skapa projekt som innehåller ett antal olika styrenheter.
- Kunna läsa in en XIF-fil för respektive enhet för att tillåta användaren att unikt konfigurera olika produkter med samma verktyg.
- kunna hantera resursfiler så att även komplexa datatyper kan representeras vid editering.
- Kunna läsa/spara konfigureringar till projektfiler och konfigurationsfiler.
- Via speciell drivrutin läsa/skriva konfigurering till/från styrenheter.

För att programmet skall bli användarvänligt skall stor vikt läggas på utformningen av gränssnittet.

Hela projektet omfattar 6-8 arbetsmånader, därför har följande avgränsningar för exjobbet införts:

- Konstruera användargränssnittet.
- Kunna läsa in beskrivning av styrenhet via XIF-filer och anpassa editeringsfunktioner för aktuell styrenhet.
- Att kunna spara och läsa in projekt till hårddisk.

I examensarbetet ingår inte:

- Att skriva drivrutiner för kommunikation mot de olika systemen.
- Att skriva verktyg för att skapa och underhålla resurs- och XIF-filerna.

## 4 Analys

### 4.1 Verktyg

Till utvecklingen av PlantMaker har Delphi valts som programspråk. Delphi är ett utvecklingsverktyg som är framställt av Borland. Det har en grafisk programmeringsmiljö integrerad med vanlig textbaserad programmering. Delphi har ett bibliotek, Visual Component Library (VCL), som innehåller komponenter för att konstruera Windows applikationer. Detta bibliotek gör att det är enkelt och snabbt att utveckla användargränssnitt i Windowsmiljö. Verktøget har sin grund i Borland Turbo Pascal, som introducerades i mitten av 1980-talet. Det underliggande programmeringsspråket är idag Objekt Pascal som är en objektorienterad utveckling av Pascal [1].

Den främsta anledning till att Delphi valdes var att det hade använts tidigare inom ReginTech, och att kompetensen om programmet redan fanns. En annan bidragande orsak var vikten vid ett användarvänligt program. Med Delphi är det lätt att ta fram prototyper som kan testas gentemot användaren. Det är även lätt att ändra på redan implementerade lösningar under utvecklingsfasen. Verktøget möjliggör även implementering av komplicerade algoritmer tack vare det bakomliggande programspråket. Detta var en nödvändighet för att språket skulle vara användbart för det här projektet.

Nackdelarna med Delphi är att det inte är plattformsoberoende. Detta medför att det krävs en PC med Windows installerat. Det slutgiltiga programmet blir inte helt oberoende av vilken version av Windows som används. Det kan skilja sig en del åt vilket gör det svårt att testa den slutgiltiga produkten, tack vare att den måste testas på olika datorer med olika miljöer. Objekt Pascal är inte helt fullkomligt vilket medför att det kan vara svårt att lösa vissa programmeringsuppgifter. Ett exempel är när objekt skall klonas i programmet. Det finns inte något enkelt sätt att göra det i Delphi. Antingen får en speciell kopieringsfunktion skrivas för varje objekt som skall klonas eller så får andra lite mer komplicerade lösningar användas. En annan sak är tvåvägskommunikation mellan klasser, som är svårt att implementera. Det finns bara möjlighet att anropa funktioner och procedurer från ett håll, vilket medför att det krävs speciella lösningar i vissa situationer.

### 4.2 Användarna

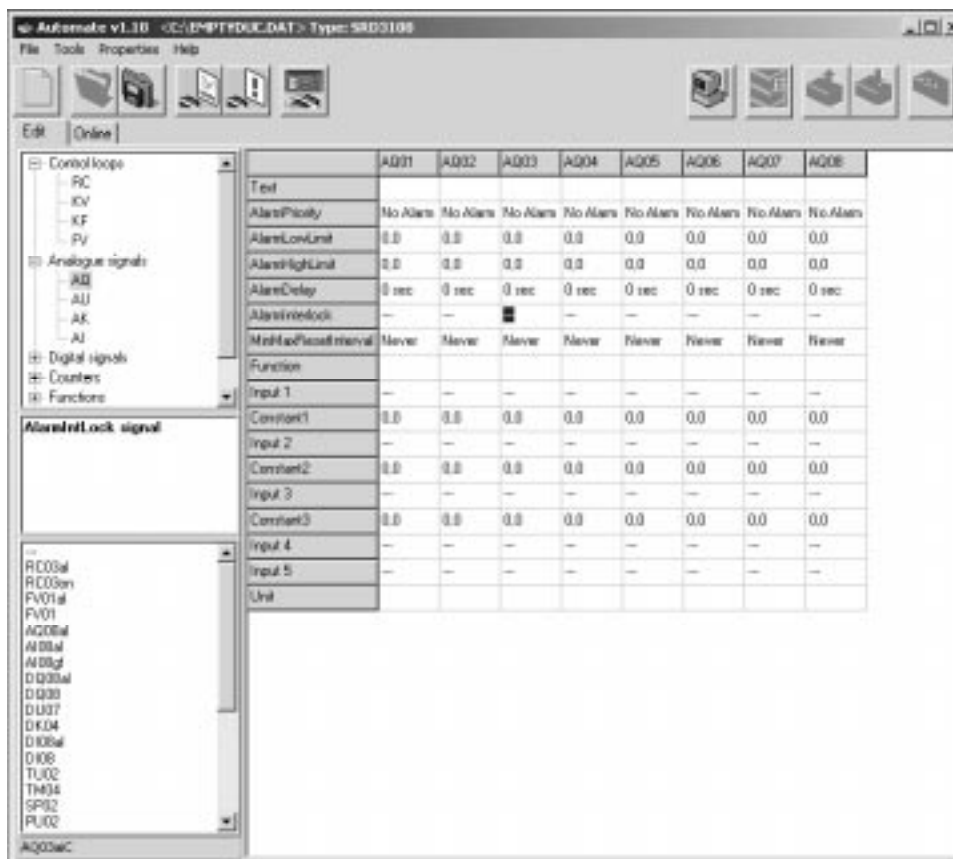
De tänkta användarna av PlantMaker är styr- och reglerinstallatörer. Programmet skall användas vid konstruktion av styr- och regleranläggningar inom byggnadsautomation. Programmet kan även användas för att omprogrammera ett system om dess användningsområde ändras efter installationen. När online-styrningen tillkommer så blir det möjligt att använda programmet som en statistikinsamlare. PlantMaker kan då ringa upp ett system och samla ihop data rörande t.ex. temperatur eller vattenflöde. Främst kommer dock användningsområdet vara konstruktion av nya nät.

Användarna kommer inte att vara dataspecialister med ett stort kunnande inom programmering. Därför är det viktigt att programmet blir tydligt och lättanvänt, samt att det skall vara möjligt att använda med måttliga kunskaper om datorer.

## 4.3 Användargränssnittet

### 4.3.1 Automate

Programmet är uppbyggt så att det skall vara lätt att använda, och lätt att få en översikt över systemet som är uppbyggt. Grunden till layouten ligger i utseendet hos AutoMate som är en föregångare till PlantMaker. Meningen är att användarna skall känna igen sig i den strukturen och på så sätt underlätta övergången.



Figur 4.1: Användargränssnittet hos Automate, föregångaren till PlantMaker.

Upp till vänster i figur 4.1 finns en trädstruktur som beskriver DUCens uppbyggnad. Här visas in- och utgångar, beräkningsvariabler och funktioner. Det här trädet har ungefär samma utseende i PlantMaker. I tabellen sker all inskrivning och ändring av värden till variablerna. Ifall det finns flera av samma sort så visas de i var sin kolumn i tabellen. Detta sker inte i den nya applikationen, utan där ligger de som löv i trädstrukturen för att spara plats på skärmen. I PlantMaker finns det möjlighet att bygga upp ett helt system med enheter och detta tar större plats på skärmen. Det har även kommit en önskan från användarna av Automate att det skulle bli enklare att hitta rätt efter denna åtgärd. De övriga rutorna som ger information om valbara värden etc. ser ungefär likadana ut, dock med en annan placering.



- SRD-trädet delas upp i olika sektioner utefter vilken funktion objektet har. Följande indelningar finns: regleringar, analoga signaler, digitala signaler, pulsräknare, funktioner inställningar och annat. Under dessa rubriker i trädet finns det fler uppdelningar inom det aktuella området som t.ex. in- eller utsignaler. Skillnaden gentemot LonWorks är fokuseringen på objekt istället för variabler. Här ses en grupp variabler som ett objekt som skall konfigureras i enheten.

## Tabellen

Upp till höger finns det en tabell där all inmatning och ändring av värden sker. (se figur 4.3) Här visas en vald nod i det andra trädet och dess löv med punktnotation. Om den översta noden är markerad så visas alla variabler i tabellen.

Name [nvi]	Data
nviSpaceTemp	5
nviSetpoint	4
nviSetptOffset	6
nviSetptShift.occupied_cool	100
nviSetptShift.standby_cool	50
nviSetptShift.unoccupied_cool	20
nviSetptShift.occupied_heat	150
nviSetptShift.standby_heat	120
nviSetptShift.unoccupied_heat	50
nviOccSchedule.current_state	OC_UNOCCUPIED
nviOccSchedule.next_state	OC_STANDBY
nviOccSchedule.time_to_next_state	30
nviOccManCmd	3
nviOccSensor	4
nviApplicMode	0
nviEnergyHoldOff.value	43,5
nviEnergyHoldOff.state	1
nviCoolSrcTemp	43
nviSpaceDewPt	32
nviHeatPriSlave	42
nviHeatSecSlave	5

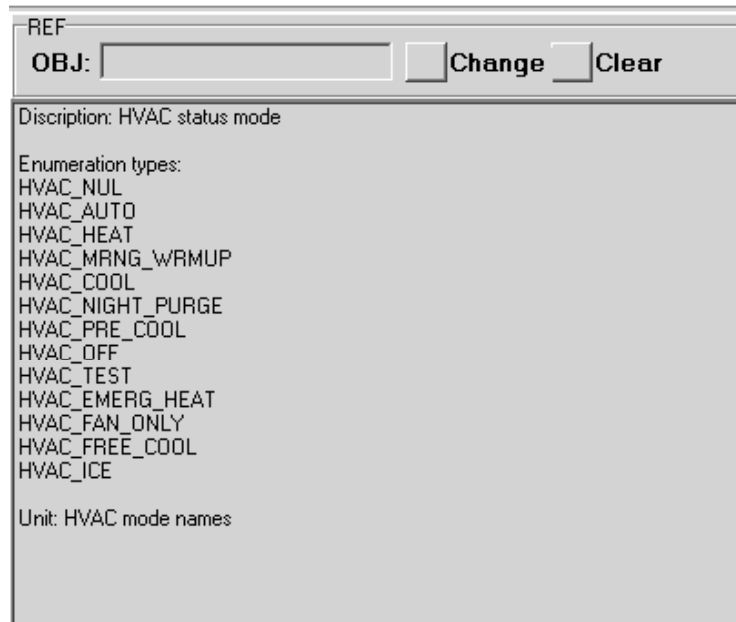
Vid inmatande av ett värde, som sker i kolumnen "Data", så sker en kontroll av värdet. Det skall vara av rätt format och noggrannhet samt vara inom min och max gränsen. Vid inmatande av en uppräkningsstyp så finns det en rullgardinsmeny med valbara alternativ i fältet för inmatning. Om dessa krav inte uppfylls meddelas detta och inmatning underkänns. När denna del av fönstret lämnas så måste alla de inmatade värdena vara godkända. Om något värde skulle vara felaktigt så visas ett informationsfönster där det står beskrivet vad det är för fel på variabeln.

*Figur 4.3: Tabell för editering av variabelvärden.*

## Informationsfönstret

Längst ner till höger finns en informationsruta. I den ges information om den markerade variabeln i tabellen. Figur 4.4 visar information om en uppräkningsvariabel. Där finns en förklaring, vilka typer som är valbara och vilken enheten är. Om det är en variabel som inte är av uppräkningsformat så finns det, istället för vilka typer som är valbara, information om min och maxvärden samt noggrannheten.

I PlantMaker finns det en möjlighet att ta in variabler från andra enheter i samma system. Då måste det finnas en möjlighet att markera vilken variabel och vilken enhet som den skall hämtas från. Detta sker i informationsfönstret genom att trycka på "Change" knappen. Då visas det ett träd som är en kombination mellan de två träden i huvudfönstret. Istället för att visa den inre strukturen hos en enhet i ett nytt träd, så visas den som ett underträd till den enheten i samma träd. När rätt variabel är funnen så markeras den och namnet kommer upp i Obj-rutan.

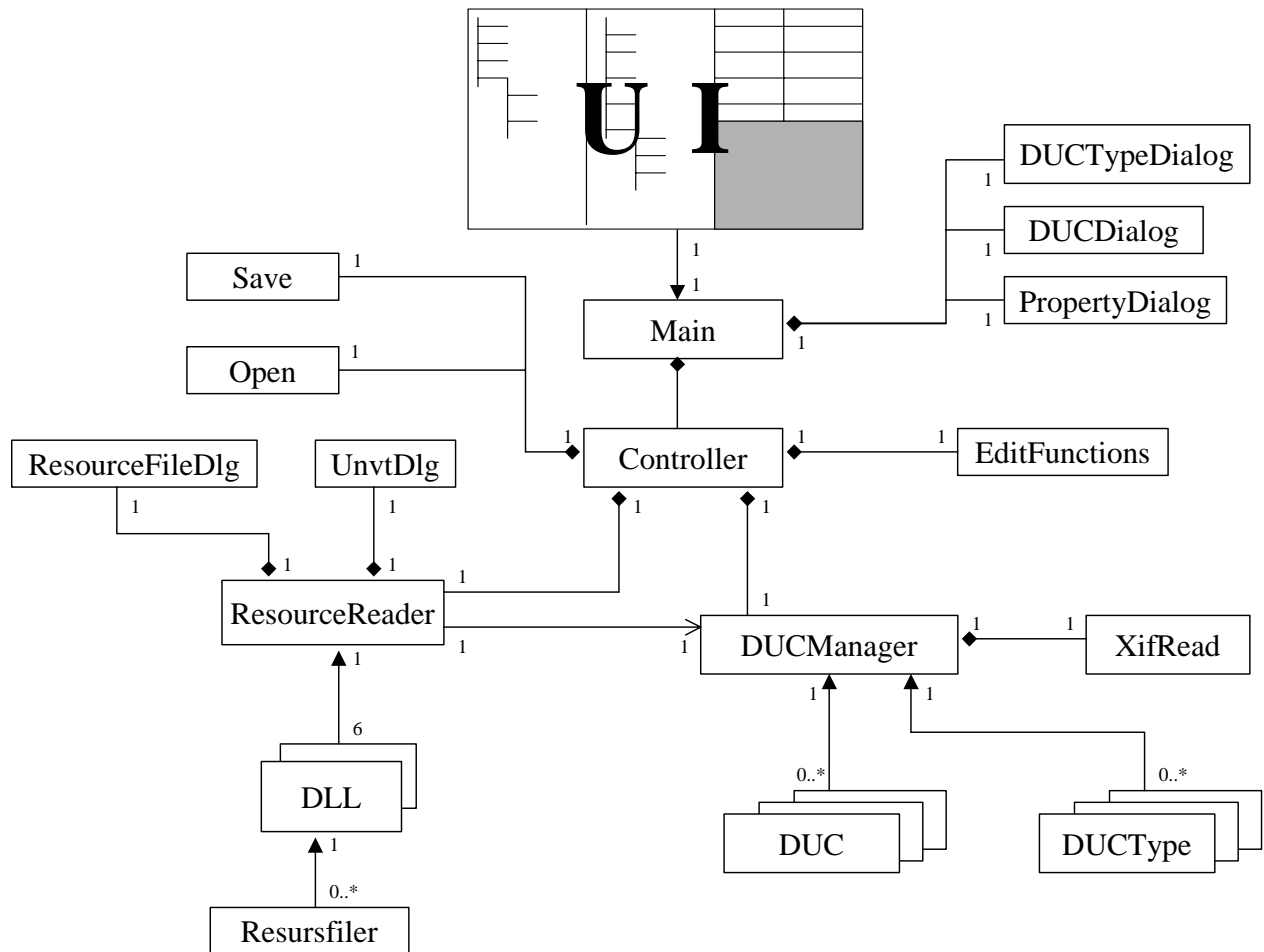


*Figur 4.4: Fönster som visar information om vald variabel.*

## 4.4 Programstruktur

Programstrukturen är en enkel uppställning med uppdelning efter funktion. Procedurer och funktioner med liknande uppgift är samlade i samma klass. Strukturen skall ge en lösning som undviker ett starkt beroende mellan klasserna. Tanken är att så många anrop som möjligt skall gå via "Controller" för en enklare översyn av anropen. Controller skall även fungera som en samordnare mellan klasserna och på så sätt avlasta Main-klassen i vissa uppgifter. Det finns en stark uppdelning mellan de olika delarna i programmet. I huvudsak finns det tre grupper i strukturen. En grupp, med "Main" som huvudklass, sköter funktionerna kopplade till användargränssnittet. En andra grupp som sköter hanteringen av enheterna i systemet, och en tredje som har ansvar för läsningen av data från resurs filerna. Tanken med Controller-klassen är att den skall fungera som en spindel i nätet genom att ta emot anrop och distribuera dessa till ansvarig klass. I det här stadiet av programutvecklingen är den lite överflödigt, men programmet kommer att kompletteras med många nya klasser och funktioner. Med hjälp av den här uppbyggnaden finns det utrymme att utöka programmet men ändå hålla kvar uppdelningen i programstrukturen.

Förutom vid spara och öppna så används designmönstret "Expert". Det innebär att den klass som har den mesta av information som behövs för att utföra en speciell operation, också får uppdraget att göra detta. Undantaget som tidigare nämndes är klasserna "Save" och "Open", där informationen som skall sparas skickas till dessa klasser för att sparas på hårddisken. Denna lösning valdes för att det enkelt skall gå att hitta och analysera all kommunikation mellan programmet och hårddisken. Om varje klass skulle spara eller öppna allt lokalt så skulle det bli komplicerat att hitta de avsnitten som sköter detta. Med en gemensam klass så kan även funktioner återanvändas i flera sammanhang genom att ändra parametrarna. För en mer detaljerad genomgång av klasserna och deras funktioner, se Appendix A.



Figur 4.5: Programstrukturen hos PlantMaker.

#### 4.4.1 Klasser

##### Main

Klassen tar hand om alla händelser kopplade direkt till användargränssnittet. Den innehåller alla objekt som används i huvudfönstret och deras händelsehanterare. Den största delen handlar om konfigureringen av trädstrukturerna. Det är funktioner och procedurer som bl.a. sköter lägga till/ta bort enhet, klippa ut/klistra in, ta bort etc. Main håller även reda på i alla tangent- och mustryckningar som sker i huvudfönstret.

## DUCManager

Huvuduppgiften är att hålla reda på enheterna i systemet. Hit kommer alla händelser som har med konfigurerings av systemets uppbyggnad. Den innehåller främst två listor med information. Den ena håller reda på vilka olika typer av DUCar som är valbara vid tillägg av enhet i systemet och den andra har hand om alla enheter som finns i systemet. Utöver dessa listor och tillhörande funktioner finns det funktioner och procedurer som håller reda på vilken enhet som är vald i det vänstra trädet i huvudfönstret.

## XifRead

Den här klassen har en funktion, och det är att läsa in XIF-filer (External Interface File) till lämpligt format. Klassen läser in all information om enheten och dess variabler från en textfil till ett format som finns deklarerad i klassen DUCType.

## ResourceReader

Klassen fungerar som ett interface till det kommunikationsbibliotek som finns beskrivet tidigare i rapporten. Funktionerna i biblioteket ger mer information än vad som efterfrågas i varje situation. Det är även aktuellt att kombinera olika funktionsanrop för att få ut all önskad information. För att förenkla anropen i programmet finns det i ResourceReader funktioner som är anpassade till de önskemål om information som finns i programmet. Klassen använder sig av dialogfönster för att få information från användaren om saker som inte står i XIF-filen.

## Editfunctions

Här sker all kontroll av inmatade värden i tabellen. Det finns funktioner som översätter strängen, som innehåller det inmatade värdet till aktuellt format och kontrollerar att det är rätt. Det sker även en kontroll gentemot min och max värden som är satta för variabeln. Här ligger även funktioner som visar redan inskrivna värden på ett lämpligt sätt.

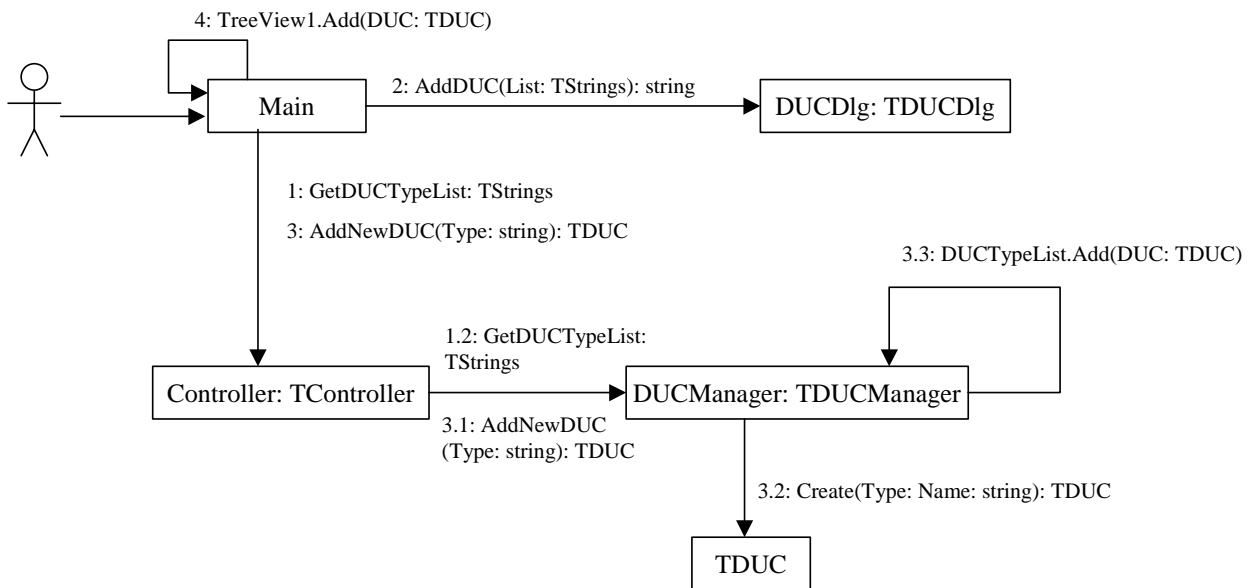
## Save/Open

I dessa klasser ligger alla funktioner som sköter allt som skall sparas eller öppnas från hårddisken. Här sparas listor, initieringsinformation, uppbyggda system etc. Det finns ingen databas som informationen sparas i, utan all information sparas i filer på hårddisken. Detta för att dels slippa databaskopplingen, som kan vara svår att göra så generell så att den fungerar på alla versioner av Windows. Ett annat bidragande argument är att denna databas skulle bli komplicerad och svåräst, vilket har skett i föregångaren (Automate). I det programmet är tiden det tar att läsa in databasen orimligt lång. För att minska tiden har lösningen med att använda resursfiler och spara information i filer använts.

## 4.5 Speciella lösningar

I detta avsnitt presenteras några utvalda lösningar på olika funktioner i programmet. De diagram som presenteras i samband med texten är av formatet UML (Unified Modeling Language). Det är ett "språk" som används för att specificera, visualisera och konstruera mjukvarusystem. UML används främst vid framtagning av objektorienterade programstrukturer. [4]

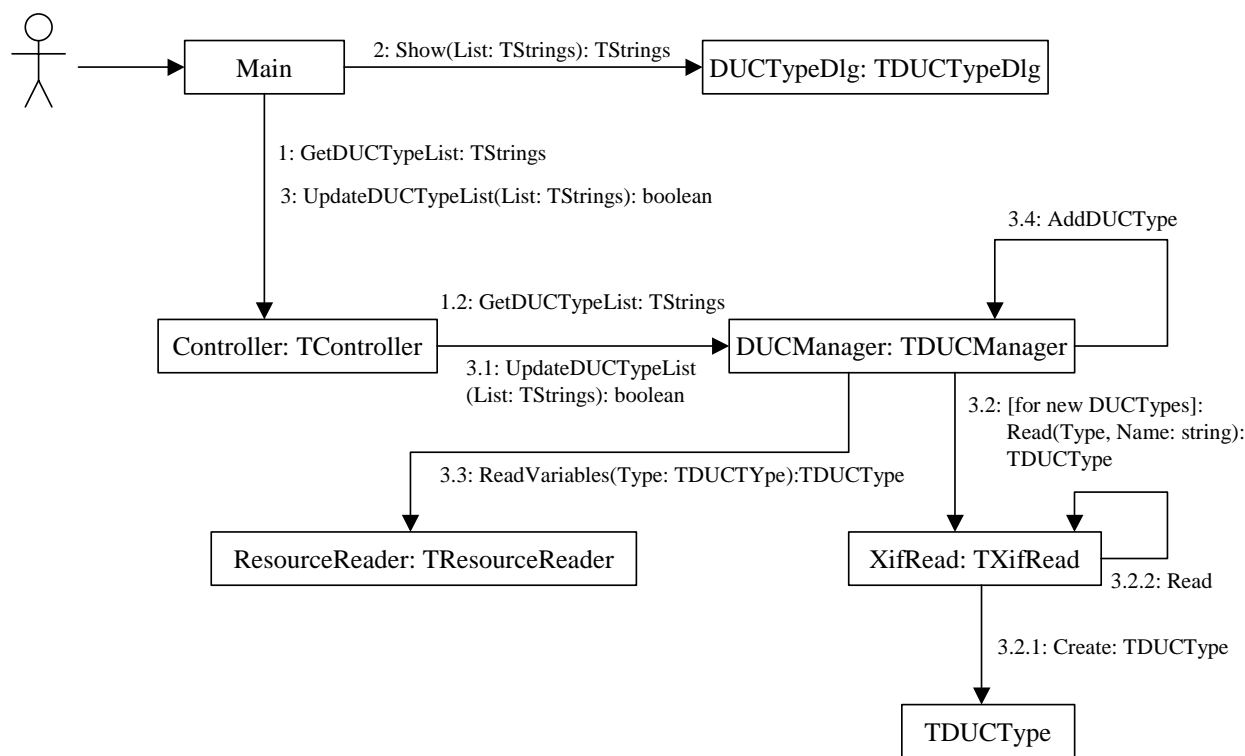
## 4.5.1 Lägg till DUC



**Figur 4.6:** Kollaborationsdiagram för händelsen "AddDUC".

Starten för förhållandet i figur 4.6 är att användaren ger en signal att han eller hon vill lägga till en enhet i systemet. Detta sker genom att aktivera händelsen med en knapptryckning, val i menyn eller genom ett kortkommando. Det första Main gör är att anropa `GetDUCTypeList` i `Controller`. Denna funktion returnerar en lista med strängar som innehåller alla namnen på de typer av enheter som är valbara. Denna lista skickas till `DUCTypeDlg` där den visas i ett speciellt fönster. I detta fönster får användaren välja vilken typ av enhet som skall läggas till i systemet. `DUCTypeDlg` returnerar en sträng innehållandes namn på typen. Därefter skickas namnet på enheten med när `AddNewDUC` anropas i `Controller`. Denna skickar strängen vidare till `DUCManager` där en ny DUC av den aktuella typen skapas. Ett temporärt namn ges till enheten vid skapandet. Om skapandet av den nya DUC'en går bra så adderas den till `DUCList` i `DUCManager`, som är en lista med insatta enheter. Funktionen `AddNewDUC` returnerar en pekare till det skapade objektet, som läggs till i det vänstra trädet i huvudfönstret. I detta träd kan användaren byta namn, kopiera eller flytta enheten efter önskemål.

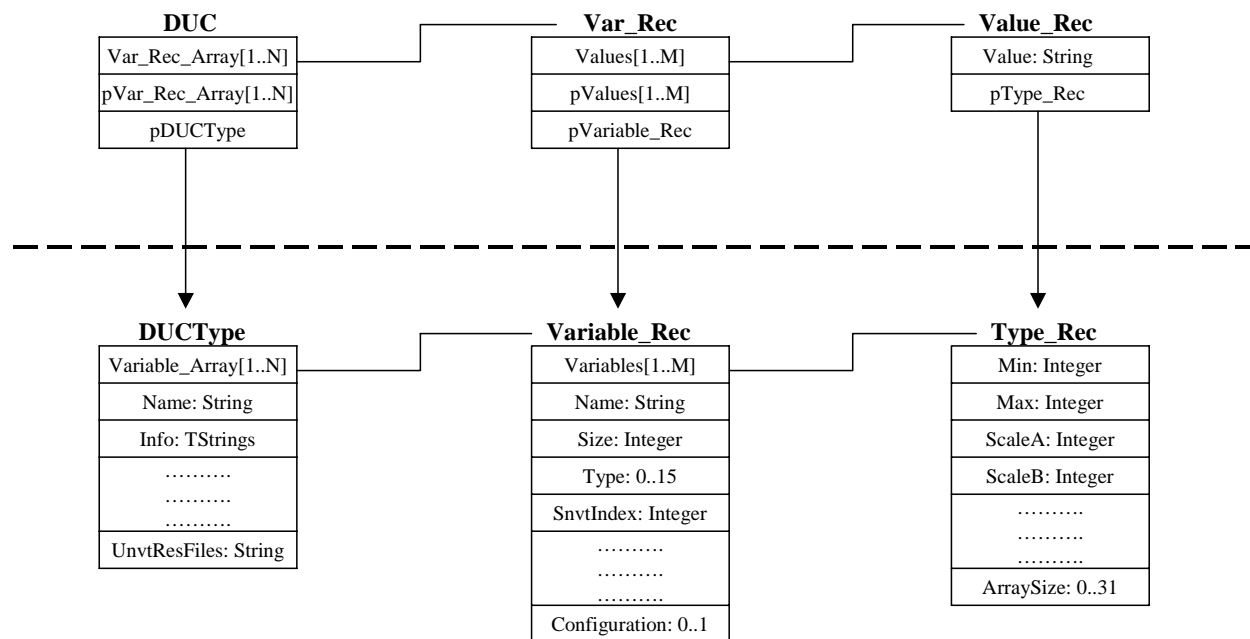
## 4.5.2 Lägg till DUC-typ



**Figur 4.7:** Kollaborationsdiagram för händelsen "AddDUCType".

Här liksom när en enhet läggs till, sker starten med en signal från användaren. Main anropar GetDUCTypeList i Controller som returnerar en lista med redan inlagda typer. Denna lista skickas till AddTypeDlg där den visas i ett nytt fönster. I detta fönster finns det möjlighet att lägga till eller ta bort redan existerande DUC typer. När en ny typ skall läggas till i listan behövs två saker, namn och sökväg till tillhörande XIF-fil. Dessa två strängar skrivs in i dialogfönstret. Här finns även möjlighet att söka efter filen med hjälp av datorns filhanterare. När användaren har lagt till och tagit bort alla önskvärda typer så returneras listan till Main. Listan vidarebefordras till Controller genom att skicka med den som inparameter när UpdateDUCTypeList anropas. Controller skickar vidare listan till DUCManager som har hand om administreringen av DUCTypeList. Här jämförs listan med den gamla som finns lagrad i DUCManager. Alla nya typer som har tillkommit läggs till och XIF-filen läses in av XifRead. Återstående data fås genom att ResourceReader anropas med funktionen ReadVariables. De enheter som saknas i den nya listan tas bort från listan och minnesutrymmet släpps fritt. I samband med att programmet startas läses listan från hårddisken där den sparades när programmet avslutades gången innan.

### 4.5.3 DUCens format



**Figur 4.8:** Schematisk bild över en DUCs representation i minnet hos programmet.

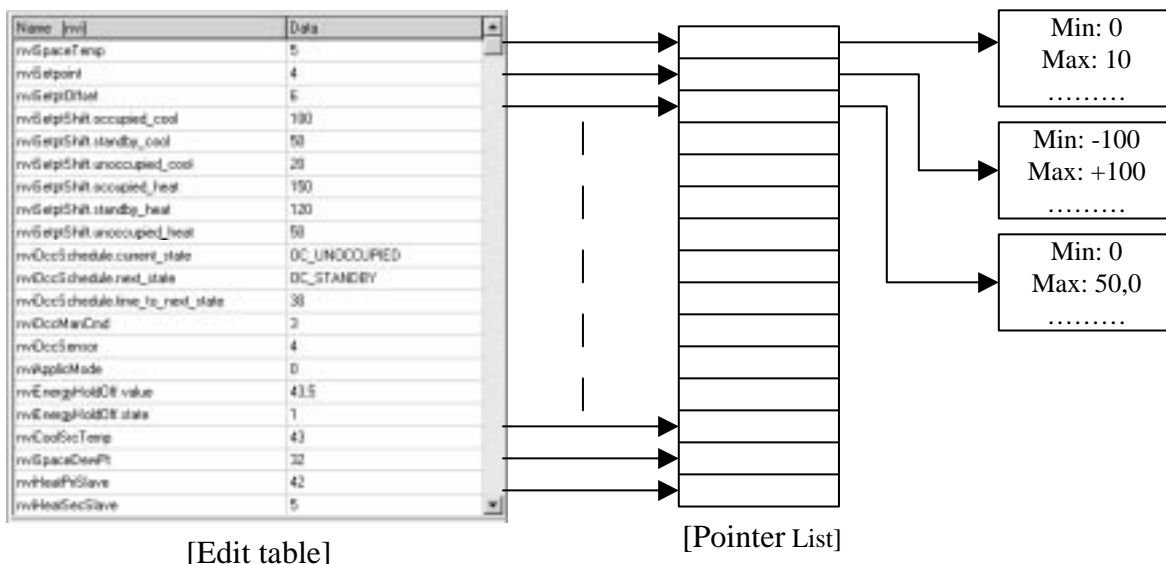
En viktig del i programmet är hur en enhet representeras i minnet på datorn. Under projektets gång har formatet ändrats flera gånger utefter att olika situationer har dykt upp. Från början sparades all information som hörde till en enhet i samma klass. Sedan skapades det en instans av klassen för varje enhet som lades till systemet. Nackdelen med denna lösning var att det blev mycket information som lästes in flera gånger. Detta medförde att minnesåtgången blev betydande när antalet enhet i systemet steg. Därför gjordes formatet om till det som beskrivs i figur 4.8. Under strecket finns DUCType som innehåller information om en viss typ av enhet. Det finns bara en instans av klassen för varje typ av enhet. Över strecket finns all specifik information om en DUC, samt pekare till den information som är gemensam med andra enheter av samma typ. Överst i klassen finns ett fält med plats för alla variabler som finns i DUCen. Till detta fält hör ett lika stort fält med pekare till respektive variabler. Dessa används då trädstrukturen byggs upp i huvudfönstret. Här finns även en pekare till den DUCType som hör till DUCen. Varje variabel består av ett fält med alla delar av variabeln representerade. Här liksom tidigare finns ett fält med pekare till varje del av variabeln. En pekare till den gemensamma informationen i DUCType finns för att underlätta åtkomsten vid editering av värdena. Längst ner finns själva värdet lagrat i "Value" tillsammans med en pekare till den information som styr vad som är tillåtet att mata in.

Denna lösning sparar både minne och är smidig att använda när editeringen skall implementeras i programmet. Med hjälp av pekarna så slipper programmet en tidskrävande algoritm för att hitta rätt information. Minnesåtgången för varje enhet som läggs till systemet blir betydligt mindre än vid den tidigare beskrivna lösningen.

## 4.5.4 Editering

Editering sker, som tidigare nämnts, i en tabell med variablerna listade. Vid inskrivning av ett värde till en variabel så sker en kontroll gentemot typ och gränsvärden för variabeln. För att veta vilken variabel som har editerats i tabellen krävs någon form av hantering av de insatta variablerna. En lösning är att söka efter namnet på variabeln i den markerade enheten. Denna lösning fungerar bra, men är komplicerad och tar lång tid att utföra. Det krävs även en relativt komplex sökfunktion, eftersom namnet som presenteras i tabellen är av en punktnotation, krävs det att det riktiga namnet läses ut ur det presenterade. För att undvika denna sökning så finns det en lista som innehåller pekare till variabler, vilken visas i figur 4.9.

Varje rad i listan motsvarar respektive rad i tabellen. Om rad tre är markerad och editerad i tabellen så är rad tre i listan en pekare till motsvarande variabel. Med denna lösning så behöver programmet bara hålla reda på vilken rad i tabellen som är aktuell. Denna lösning utnyttjar lite mer minne men är så pass mycket lättare att använda så det överväger minnesåtgången.



**Figur 4.9:** Visar förhållandet mellan inmatat värde och gränsvärden vid editering av variabelvärden.

## 4.5.5 Standardvärden

Det finns en möjlighet att i programmet spara en standardinställning för en viss typ av enhet. Denna kan sedan laddas in i en annan enhet av samma typ och på så sätt effektivisera arbetet när flera enheter med liknande inställningar skall läggas till i systemet. Detta görs genom att en enhet av den aktuella typen laddas in och konfigureras på det bestämda sättet. Sedan sparas värdena på varje variabel i en textfil, med samma namn som enhetstypen. När en annan enhet skall laddas med samma värden, letar programmet efter en fil med samma namn som enheten.

Denna funktion realiserar i programmet genom att tabellens funktioner används. Det objekt som används till tabellen kallas i Delphi för **ValueListEditor**. Detta objekt har ett antal

funktioner som kan användas i olika situationer. En funktion gör om hela tabellen till en lista med strängar där varje rad motsvarar en variabel och har följande struktur:

**Variabelnamn = värde (ex nviOutdoorTemp=20)**

ValueListEditor har även en funktion som gör om en lista med strängar enligt ovan till en tabell. När en enhets värden skall sparas, laddas hela enheten i tabellen så att alla variabler finns med i tabellen. Efter det sparas tabellen till en fil på hårddisken. När sedan en enhet skall laddas med samma värden, laddas tabellen med hjälp av listan i filen. Sedan sker en kontroll av hela tabellen innan värdena sparas på rätt ställe i programmet.

Genom att värdena skrivs in när en enhet finns laddad i bakgrunden, sker kontrollen innan de sparas undan. Detta underlättar sedan vid inladdning då alla felaktiga värden redan har rättats till. Om definitionen av standardvärden skulle ske på annat sätt skulle kontrollen vara svårare att utföra innan värdena sparats. Med den valda lösningen sker kontrollen direkt vid inskrivningen av värdet.

#### 4.5.6 Variabelinitiering

Vid inläsning av en XIF-fil skapas det ett utrymme för varje variabel i minnet. Dessa fylls därefter upp med information från resursfilerna. Vid användning av egendefinierade variabler (UNVT), stämmer inte storleken som är angiven i XIF-filen överens med storleken i resursfilen. Detta skapar problem när inläsningen av data sker och det inte finns tillräckligt med initierad plats att lagra all nödvändig information. Eftersom den största delen av informationen hämtas från resursfilerna, föll valet på att initiera minnesutrymme efter vad som anges i dessa. Det finns en funktion, "GetNrElements", i ResourceReader som returnerar storleken på den eftersökta variabeln. Funktionen loopar igenom variabelträdet och räknar hur många element variabeln består av. Räkningen sker på ett sätt som gör att resultatet passar den form som programmet sparar informationen på. Därför anropas denna funktion vid initiering av varje variabel, även de som inte innehåller några egendefinierade delar. Genom att göra på detta sätt fungerar det som en garanti på att minnesutrymmet alltid stämmer överens med variabeln i resursfilen.

## 5 Resultat

PlantMaker är ett program som kommer att fungera som en brygga mellan ReginTechs två olika produktvärldar, SRDLink och LonWorks. Detta är en nödvändighet för att inte tvinga kunden till att bara välja en av produktvärldarna, utan tillåta kunden till att välja helt fritt. Det kommer även att bli lättare att uppdatera system genom att uppdateringar från båda världarna blir tillgängliga. För att det skall fungera helt ut så räcker det inte med PlantMaker utan det krävs ny hårdvara som sköter kommunikationen mellan SRDLink och LonWorks. En kommunikationsmodul är under uppbyggnad och kommer att finnas till hands då den färdiga produkten av PlantMaker finns till hands.

Förutom tidigare nämnda fördelar, kommer applikationen att vara till stor hjälp vid installation av större system. När antalet enheter i ett system ökar, kommer komplexiteten i kommunikation och antalet bindningar att öka. Det blir svårare för en installatör att hålla reda på alla in- och utgångar som skall kopplas ihop för att få önskad funktion på systemet. Här

kommer PlantMaker att vara till stor hjälp då det blir möjligt att sköta allt via en dator. I programmet går det att sköta installationen av systemet på distans, och den fysiska närvaron som behövs begränsas till uppsättningen av enheter och kablar. Även underhåll och omkonfigurering blir enklare då allt detta kan ske via datorn. För att kunna uppnå detta måste PlantMaker skapa och skicka installationsmoduler till respektive enhet i systemet. Det är främst denna funktion som återstår att implementera i programmet. Denna funktion ingår inte, enligt tidigare nämnda systemkrav i examensarbetet.

## 5.1 Implementering

För att göra det möjligt att utveckla programmet med de nödvändiga funktionerna, är programstrukturen viktig. För att kunna bygga ett system som är överskådligt och har så liten beroendegrad som möjligt mellan de olika delarna i programmet, så krävs det en grundstomme i strukturen. Valet föll på en Controller klass som fungerar som en brygga mellan användargränssnittet och det underliggande programmet. Denna lösning gör det möjligt att utvidga programmet, samtidigt som det låga beroendet mellan delarna bibehålls.

Lösningen med att använda XIF- och resursfiler har fungerat bra. Behovet av en stor databas med databaskopplingar och inläsningstider har försvunnit. Svarstiden på systemet har minskat drastiskt jämfört med Automate. Med PlantMaker tar det ca en sekund att läsa in en enhet jämfört med 15 – 20 sekunder hos Automate. Problemet med lösningen är att den är anpassad för LonWorks och kräver lite anpassning för att klara av protokollet SRDLink.

Strukturen på programmet har en klar uppdelning efter funktion, vilket gör det lätt att hitta i koden. Nackdelen är att det blir fler anrop mellan klasserna och mer data som skickas fram och tillbaka. Det som övervägde till funktionsuppdelningens fördel, var att personer som inte är insatta i koden så lätt som möjligt skall kunna hitta och ändra delar i programmet. Detta väger tungt eftersom programmet inte är klart och det kommer att ske en kontinuerlig utveckling av det den närmaste tiden.

Användargränssnittet har tyvärr inte testats fullt ut, beroende på att det saknas en del funktioner som gör att testet i dagsläget blir mindre värt ur utvärderings synpunkt. Vikten på användarvänligheten har beaktats under konstruktionsfasen och resultatet är en ren och enkel miljö. Endast det viktigaste visas på skärmen med en klar uppdelning mellan de olika delarna. För att få fram ett mer korrekt resultat krävs det ett ordentligt genomfört test av en eller flera framtida användare.

## 5.2 Utveckling

Som beskrivs i systemkraven, är programmet inte klart även om examensarbetet är det. De delar som faller inom ramen för examensarbetet är en del av ett större program. Delarna som sköter konstruktionen av installationsmodulerna och kommunikationen med enheterna kvarstår att implementera. När dessa delar implementeras kommer det att komma upp saker som gör att en ständig utveckling av användargränssnittet kommer att ske. Grunden i strukturen med de två träden och tabellen där inmatningen sker kommer att ligga kvar. Utvecklingen kommer främst att handla om ytterligare fönster med funktioner som inte direkt är kopplade till konfigurationen av enheter. Det kan t.ex. vara fönster där direktkommunikation med en enhet sköts, information om kommunikation etc.

Programmet kommer eventuellt kompletteras med en grafisk framställning av ett konfigurerat system. I så fall blir det möjligt att konfigurera bindningar mellan enheter grafiskt. Anledningen till att den grafiska delen ligger utanför den första implementeringsfasen är att den är svår att lösa i praktiken. Den är inte heller nödvändig för att få en fungerande applikation.

## 6 Slutsats

Projekt PlantMaker har lett fram till ett konfigureringsverktyg för att styra ett styrsystem bestående av en eller flera enheter. I programmet går det att lägga till och ta bort enheter, lägga in nya enhetstyper och konfigurera olika typer av enheter. Programmet stödjer konfiguration av två olika kommunikationsvärldar, SRDLink och LonWorks. I programmet går det att kombinera enheter från de två kommunikationsprotokollen i samma system.

Nytänkandet i förhållande till redan existerande produkter är användandet av XIF- och resursfiler. Detta gör att inläsningen går fort och behovet av resurskrävande databaser försvinner. Lösningen är också en förutsättning för att på ett effektivt sätt binda ihop de olika protokollen som finns inom ReginTech.

Examensarbetet är en del av ett större system som är projekterad till knappt ett år, vilket medför att det återstår utvecklingsarbete för att få fram en fungerande produkt. Det som är gjort är en grund som skall byggas ut till ett fullkomligt program. Grunden för användargränssnittet är gjort och basfunktionerna finns implementerade. Det krävs finputsning på användargränssnittet för att få det enhetligt med övrig programvara som är framtagen av ReginTech.

## Referenser

- [1] Delphi Programming. Zarco Gajic. Delphi.about.com (2002-03-20)
- [2] Echelon Corporation. [www.echelon.com](http://www.echelon.com) (2002-05-06)
- [3] ElectroTest Sweden AB: (2000) *Lonhandboken*. Första upplagan. Göteborg: Novum Grafiska AB. ISBN 91-631-0291-9
- [4] Larman, Craig: (1998) *Applying UML and Patterns*, Upper Saddle River: Prentice Hall. ISBN 0-13-748880-7
- [5] LonMark: (1999) *Resource File API Reference Guide*, Revision 2
- [6] LonMark: (2000) *Resource File Developer's Guide*, Revision 3
- [7] LonMark: (2000) *External Interface File Reference Guide*, Revision 4.0A
- [8] LonUser Sweden. [www.lonusersweden.org](http://www.lonusersweden.org) (2002-05-06)
- [9] Schürmann, Bernd: (1998) *Structure and Design of Building Automation Systems. New Information Technologies in Science, Education, Telecommunications and Business (IT+SE'98)*, Yalta-Gurzuff, Ukraine, 1998. URL: <http://galway.informatik.uni-kl.de/publications/> (2002-03-05)
- [10] ÅF-Elteknik AB: (1999) *Smarta hus – bra för miljön*, rapport 1999:1. URL: [http://miljoteknik.vinnova.se/rapporter\\_och\\_dokument.html](http://miljoteknik.vinnova.se/rapporter_och_dokument.html) (2002-02-15)

## Ämnesrelaterade webbsidor

[www.echelon.com](http://www.echelon.com)

[www.lonmark.org](http://www.lonmark.org)

[www.lonusersweden.org](http://www.lonusersweden.org)

[www.regintech.se](http://www.regintech.se)

[delphi.about.com](http://delphi.about.com)

## Ordlista

API	Application Programming Interface
BACnet	Building Automation and Control Network, Ett datakommunikationsprotokoll.
DLL	Dynamic Link Library
DUC	Data Under Central
EEPROM	Electrically Erasable Programmable Read-Only Memory
EHS	European Home System
EIA	the Electronics Industry Alliance
FND	Firm Neutral Datacommunication
ISO	International Standardization Organization
Lon	Local Operating Network
LonMark	Standardiserings-organisation för Lon-produkter
LonTalk	Kommunikationsprotokollet för LonWorks
LonWorks	Nätverksplattform för Lon-produkter
nci	Network Configuration variable Input
nco	Network Configuration variable Output
Neuron C	Händelsebaserat programspråk som har stora likheter med ANSI C
nvi	Network Variable Input
nvo	Network Variable Output
OSI	Open System Interconnection
RAM	Random Access Memory
ROM	Read Only Memory
SCPT	[skipit] Standard Configuration Property Types
SNVT	[snivit] Standard Network Variable Types
SRDLink	Styr- ReglerDUC. Kommunikationsprotokoll framtaget av ReginTech
UCPT	[ukeepit] User defined Configuration Property Types
UML	Unified Modeling Language
UNVT	[univit] User defined Network Variable Type
WordFIP	Fransk-amerikanskt kommunikationsprotokoll
XIF	External Interface File

# Appendix A

En genomgång av klasser och deras viktigaste funktioner och procedurer. Klasserna innehåller flera funktioner men dessa är bara för internt bruk inom klassen.

+ = public (synlig utanför klassen)  
- = private (synlig endast inom klassen)  
:typ = typen som funktionen returnerar.

**TAddDUCDlg** – Dialogfönster för tillägg av DUC till systemet.

- +Create – Skapar klassen.
- +Destroy – Tar bort klassen.
- +AddDUC(List: TStrings): String – Visar listan "List" i fönstret och returnerar namnet på vald typ.

**TAddDUCTypeDlg** – Dialogfönster för tillägg av enhetstyp till valbara enheter.

- +Create - Skapar klassen.
- +Destroy – Tar bort klassen.
- +ShowDialog(var List, AddList2, RemoveList2: TStrings): Boolean – Visar fönstret med namnen i "List" som inlagda typer, AddList2 och RemoveList2 innehåller tillagda resp. borttagna typer.

**TDUC** – Klass som representerar en enhet i systemet.

- +Create(DUCType2: TDUCType; Name2: String) – Skapar klassen.
- +Destroy – Tar bort klassen.
- +GetDUCTree(TreeView: TTreeView): TTreeView – Returnerar enhetens inre trädstruktur.
- +GetDUCTree(TreeView: TTreeView; Node: TTreeNode): TTreeVeiw – Returnerar en trädstruktur med Node som rot.
- +GetDUCType: TDUCType – Returnerar en pekare till DUCType.
- BuildDUCTree(TreeView: TTreeVeiw; Node: TTreeNode) – Bygger upp enhetens inre trädstruktur med "Node" som rot.
- BuildLeaf(ValueRecArray: Value\_Rec\_Array; P\_Array: PValue\_Rec\_Array; var Index: Integer; ParentNode: TTreeNode) – Bygger upp en trädstruktur av en variabel som läggs som ett löv till "ParentNode".

**TDUCManager** – Håller reda på enheterna i systemet.

- +Create(ResourceReader2: TResourceReader; Save2: TSave; Open2: TOpen) – Skapar klassen, med referenser till berörda klasser.
- +Destroy – Tar bort klassen.
- +AddNewDUCType(Name, FileName: String): Boolean – Läger till en DUCType till valbara enheter.
- +AddNewDUCType(Name: String): Boolean – Läger till en DUCType som finns lagrad på hårddisken.
- +AddNewDUC(Name: String): Boolean – Läger till en enhet av typen "Name" till systemet.
- +AddDUC(DUC: TDUC): Boolean – Läger till en kopia av en annan DUC till systemet.
- +UpdateDUCTypeList(AddList, RemoveList: TStrings): Boolean – Uppdaterar DUCTypeList.

- +ChangeDUCName(NewName: String): Boolean – Byter namn på en enhet.
- +RemoveDUCType(Name: String): Boolean – Tar bort en typ från valbara typer.
- +RemoveDUC(Name: String): Boolean – Tar bort en enhet från systemet.
- +CopyDUC(DUC: TDUC): TDUC – Kopierar en DUC.

**TDUCType** – Representerar en typ av DUC.

- +Create – Skapar klassen.
- +Destroy – Tar bort klassen.
- +DUCTypeToList(var List: TString) – Gör om en DUCType till en lista med strängar.
- +ListToDUCType(List: TString): Boolean – Gör om en lista med strängar till en DUCType.

**TEditFunctions** – Tar hand om kontrollen av inmatade värden.

- +Create(ValueListEditor2: TValueListEditor; ResourceReader2: TResourceReader) – Skapar klassen.
- +Destroy – Tar bort klassen.
- +DisplayValueInt(Value, ScaleA, ScaleB, ScaleC: Integer): String – Returnerar en sträng innehållandes värdet på variabeln.
- +ChechValue(TreeNode: TTreeNode; Lev: Integer): Boolean – Kontrollerar värdena på variablerna som ligger under TreeNode i trädstrukturen.
- +ChechValue(ValueRec: PValue\_Rec; DisplayName: String): Boolean – Kontrollerar värdet på en speciell variabel.
- +ShowDataInValueList(TreeNode: TTreeNode): TList – Visar alla variabler under TreeNode i ValueListEditorn samt returnerar en lista med pekare till varje variabel i minnet.
- +Resolution(ScaleA, ScaleB, ScaleC): Integer – Returnerar en sträng med noggrannheten uträknad utefter de olika skalorna.
- RemoveZero(Val: String; NrDec: Integer): String – Tar bort onödiga nollor.
- CheckInt(ValueRec: PValueRec; DisplayName: String): Boolean – Kontrollerar ett heltal.
- CheckFloat(ValueRec: PValueRec; DisplayName: String): Boolean – Kontrollerar ett flyttal.
- CheckEnum(ValueRec: PValueRec; DisplayName: String): Boolean – Kontrollerar en uppräkningsstyp.
- CheckBitF(ValueRec: PValueRec; DisplayName: String): Boolean – Kontrollerar en bitfield.
- CheckChar(ValueRec: PValueRec; DisplayName: String): Boolean – Kontrollerar en variabel med ASCII-tecken.
- ShowChar(ValueRec: PValue\_Rec; DisplayName: String) - Visar en variabel innehållandes ASCII-tecken i tabellen.
- ShowInt(ValueRec: PValue\_Rec; DisplayName: String) - Visar ett heltal i tabellen.
- ShowFloat(ValueRec: PValue\_Rec; DisplayName: String) - Visar ett decimaltal i tabellen.
- ShowBitF(ValueRec: PValue\_Rec; DisplayName: String) - Visar en bitfield i tabellen.
- ShowEnum(ValueRec: PValue\_Rec; DisplayName: String) - Visar en uppräkningsstyp i tabellen.

**TOpen** – Hämtar information från filer på hårddisken.

- +Create(App: String) – Skapar klassen.
- +Destroy – Tar bort klassen.
- +OpenProperties(Prop: TStrings): TStrings – Öppnar sparade egenskaper.
- +OpenDUCTypeList(List: TStrings): TStrings – Öppnar sparad DUCTypeList.
- +LoadDefaultValue(List: TStrings; FileName: String): TStrings – Hämtar standardvärden för en viss enhetstyp.
- +LoadDUCType(List: TStrings; FileName: String): TStrings – Hämtar en sparad DUCType.
- +InitMainWindow(Left, Top, Width, Height, Panel1Width, Panel2Width, Panel5Height: Integer): IniRecord – Öppnar initieringsvärden för huvudfönstret.
- +InitResourceFiles: TStringList – Initierar resursfilerna.

**TPropertyDlg** – Fönster för administrering av programmets egenskaper.

- +Create - Skapar klassen.
- +Destroy – Tar bort klassen.
- SaveProperties – Sparar egenskaperna.
- ShowProperties – Visar dialogrutan.

**TResourceFileDialog** – Dialogfönster för administrering av resursfiler.

- +Create - Skapar klassen.
- +Destroy – Tar bort klassen.
- +ShowResourceFileDialog(List: TList; var ResName: String; var AddList: TStringList) Boolean – Visar fönstret med den information som skickas med i "List".

**TResourceReader** – Läser information från resursfilerna.

- +Create(Path: String) - Skapar klassen.
- +Destroy – Tar bort klassen.
- +ReadVariables(TempDUC: TDUCType): TDUCType – Läser in all information som DUCType behöver.
- +OpenStandardTyp(Path: String) – Öppnar standard resursfilerna.
- +ReadVarInfo(var TypeRecArray: Type\_Rec\_Array; TYP\_No, Nr: Integer) – Läser in information till en variabel.
- +ReadResourceString(Sel, Index: Integer):String – Hämtar en språksträng från språkfilen.
- +ReadEnumeration(Min, Max, Sel, Index, TYP\_No: Integer):StringArray – Läser in valbara strängar till en uppräkningsstyp.
- +OpenResourceFiles(Path, Name: String; var tResFileRec: ResourceFileRec): Boolean – Öppnar resursfiler för egendefinierade resursfiler.
- +AddResourceFile(Path, Name: String): Boolean – Läger till en uppsättning resursfiler till listan över valbara resursfiler.
- InitVarInfo(var TypeRecArray: Type\_Rec\_array; tName: String) – Initierar en variabel.

**TSave** – Sparar information i filer p hårddisken.

- +Create(App: String) - Skapar klassen.
- +Destroy – Tar bort klassen.
- +SaveResourceFileList(ResList: TList): Boolean – Sparar listan över valbara resursfiler.
- +SaveProperties(Prop: TStrings): Boolean – Sparar egenskaper för programmet.
- +SaveDUCTypeList(List: TStrings): Boolean – Sparar DUCTypeList.
- +SaveDefaultValue(List: TStrings): Boolean – Sparar standardvärden för enheter.
- +SaveDUCType(List: TStrings; FileName: String): Boolean – Sparar en DUCType.

+SaveInitMainWindow(Left, Top, Width, Height, Panel1Width, Panel2Width, Panel5Height: Integer): Boolean – Sparar initieringsvärden för huvudfönstret.

**TUnvtDialog** – Dialogfönster för val av egendefinierad variabel.

+Create - Skapar klassen.

+Destroy – Tar bort klassen.

+Show(List: TStringList; VarName: String): Integer – Visar en lista över valbara egendefinierade variabler.

**TXifRead** – Läser in en XIF-fil till programmet.

+Create - Skapar klassen.

+Destroy – Tar bort klassen.

+ReadXif(Name2, FileName2: String): TDUCType – Skapar en DUCType och läser in dess information från XIF-filen.

**TMain** – Håller reda på allt som har med huvudfönstret att göra.

+Create - Skapar klassen.

+Destroy – Tar bort klassen.

-CopyNodeUnder(TreeView: TTreeView; SourceNode, TargetNode: TTreeNode; DeleteNode, Undo: Boolean) – Kopierar en nod i trädet som beskriver hela systemet.

-RemoveDUCFromList(Node: TTreeNode) – Tar bort en enhet från DUCList.

-SelectNodeTree1(Node: TTreeNode) – Väljer en enhet i huvudträdet.

-ShowDUCTree – Visar en enhets inre struktur i det andra trädet.

-SelectNode(TreeNode: TTreeNode)- Väljer en nod i det andra trädet.

-ShowDataInValueList – Visar variabler i tabellen.

-ClearValueList – Rensar tabellen.

-SaveValue: Boolean – Sparar ett värde i tabellen.

-SaveAllValues: Boolean – Sparar alla värden i tabellen.