Generell recepthanterare för processindustri

av

Jesper Dahlgren

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Sammanfattning

Prevas har i sina kunduppdrag haft behov av en enklare generell hanterare för processrecept som skickar receptparametrar(t ex rotationshastighet på en blandare eller temperatur i en ugn osv) till en Programmable Logic Controller. Mjukvaror för detta finns på marknaden men är antingen för komplexa eller kostsamma. För att lösa detta problem utvecklades en Windows Service. Denna rapport beskriver kort hur utvecklingen gick till och sedan mer djupgående hur detta system är uppbyggt och arbetar. Det utvecklade systemet ger Prevas en unik recepthanterare utvecklat efter deras behov som även kan modifieras i framtiden om så behövs utan att behöva förlita sig på externa utvecklare.

Nyckelord: PLC MES OPC DA DBMS C#.NET Windows Service

Abstract

Prevas has in its customer contracts had a need for a simpler general manager for process recipes which send recipe parameters (eg rotational speed of a mixer or temperature in a furnace, etc.) to a Programmable Logic Controller. Software for this is available on the market but are either too complex or costly. To solve this problem a Windows service was developed. This report briefly describes how development occurred and then in more depth how this system is structured and operates. The developed system gives Prevas a unique manager for process recipes developed according to their needs which also can be modified in the future if needed without having to rely on external developers.

Key words: PLC MES OPC DA DBMS C#.NET Windows Service

Tillkännagivande

Denna examensrapport skulle inte vara möjlig utan vägledning och stöd av Johan Frithiof och framför allt, André Berg

Jesper Dahlgren

Innehåll

1 Inledning

1.1 Bakgrund

Arbetet utförs åt Prevas AB. Prevas är en konsultfirma med stor kompetens inom industriell IT & automation, teknisk produktutveckling och inbyggda system. Prevas mål är att hjälpa sina kunder utveckla sin verksamhet och göra dem mer konkurrenskraftiga på marknaden genom framtagning av smartare produktionsmetoder och produkter, se [1] för mer information om Prevas.

Prevas har i sina kunduppdrag haft behov av en enklare generell hanterare för processrecept som skickar receptparametrar(t ex rotationshastighet på en blandare eller temperatur i en ugn osv) till en PLC. Mjukvaror för detta finns på marknaden men är antingen för komplexa eller kostsamma. I tidigare projekt har recept programmerats in i olika PLC:ers minnen, problemet med detta är det begränsade minnet i enheten och när detta är fullt måste nyare recept skrivas manuellt in varje gång de ska köras. Det nya systemet eliminerar minnesproblemet och ger Prevas ett eget program som kan anpassas utifrån deras behov.

1.2 Syfte och mål

Nyttan med en generell recepthanterare är att den kan användas i många olika projekt och är anpassningsbar till flera typer av PLC:er. Vad som uppnås är en enklare utvecklingsprocess för kommande projekt och således sparar tid/pengar.

I detta projekt ska en Windows Service kallad PREvas reCIpe System (PRECIS) utvecklas i C#.NET som skall kommunicera med en OPC-server som i sin tur sköter kommunikationen mot PLC via respektive driver. Anledningen till att det specifikt skall vara en Windows Service och inte t ex. en konsolapplikation är att programmet ska kunna köras på en server där ingen användare finns inloggad. Val av OPC-server och databashanterare görs tillsammans med Johan Frithiof och André Berg. Handskakning mellan PRECIS och PLC

skall utvecklas och testas. PRECIS kommer att hämta data som skall skickas till PLC från en databas. På kontoret i Malmö finns en Siemens S7-300 PLC tillgänglig för att arbeta och testa mot, därför är det viktigt att undersökta OPC-servrar har stöd för denna PLC. För att säkerställa PRECIS funktionalitet måste systemet testas vilket är det sista som görs i projektet förutom denna rapport. Detta innefattar alla normalfall och felhantering t ex. om uppkoppling mot OPC-server eller databas inte finns.

1.3 Frågeställningar

En av de viktigaste komponenterna i detta projekt är kommunikationen mellan PRECIS och PLC. Hur kommunikation/handskakning ska ske är vad detta arbete skall besvara.

Vilka frågor som ska besvaras/vad som ska göras och vilka eventuella kriterier det ska baseras på:

1. Vilken OPC-server ska användas? Kriterier:
    a. Låg kostnad
    b. Existerande kunskap hos Prevas Malmös anställda/användningsvana ska finnas
    c. Stort urval av drivrutiner ska finnas, viktigast för projektets genomförande är att en drivrutin för Siemens S7-300
2. Vilken databashanterare skall användas? Kriterier:
    a. Låg kostnad
    b. Existerande kunskap hos Prevas Malmös anställda/användningsvana ska finnas
    c. Enkel att hantera/använda
3. Hur ska PRECIS-server utvecklas?
    a. Hur ska handskakning genomföras?
    b. Hur ska databasen designas?

1.4 Avgränsningar

Systemet som helhet kommer använda sig av ett webbgränssnitt som kommer utvecklas av André Berg i samband med eller efter examensarbetet. Se Fig. 1

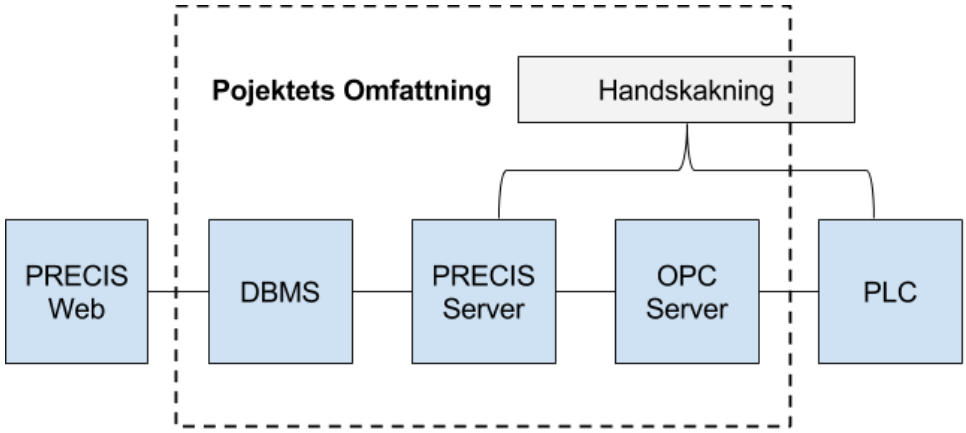

Fig. 1. Visuell beskrivning av projektets omfattning

Ytterligare avgränsningar är att projektet, på begäran av Prevas, skulle utvecklas i C#.NET och PRECIS Server utgöras av en Windows Service.

2 Teknisk bakgrund

I detta kapitel beskrivs den tekniska bakgrund som behövs för att förstå de olika komponenterna och projektet som helhet.

2.1 OPC

Open Platform Communications (OPC) är en standard för mjukvarugränssnitt som tillåter Windows-applikationer kommunicera med industriell hårdvara på ett säkert och pålitligt sätt och The OPC Foundation ansvarar för utveckling och underhåll av denna standard. OPC är uppbyggt av server/klient-par där servern hanterar kommunikation till PLC och avkodar denna åt klienten. Klienten kan t ex. vara ett Human Machine Interface (HMI, motsvarar Graphical User Interface, GUI). OPC Classic där OPC Data Access ingår är begränsad till Windows operativsystem medan det nyare OPC Unified Architecture är oberoende av plattform, se [2] och [3] för mer information om standarden OPC.

2.1.1 OPC Classic

Detta är ett samlingsnamn för att hämta processdata, alarm och historisk data. OPC Classic baseras på Microsoft Windows Distributed Component Object Model (COM/DCOM). OPC Classic består av följande standarder:

- OPC Data Access (DA)
- OPC Alarms & Events (AE)
- OPC Historical Data Access (HDA)

I detta projekt kommer dock endast standarden OPC DA utnyttjas. Se [4] för mer information om OPC Classic.

2.1.2 OPC Data Access

Detta är OPC Foundations specifikation för hur realtidsdata ska skickas mellan en datakälla och en datamottagare som t ex en PLC och ett HMI utan att de känner till varandras protokoll.

OPC DA Server/Client arkitekturen var den första arkitekturen som OPC Foundation definierade. Innan OPC DA fanns användes tillverkares egna protokoll och specialgjorda drivrutiner. Problemen med dessa var ofta hög kostnad, svårt att underhålla och varje typ av drivrutin behövde ofta konfigureras på ett specifikt sätt. När OPC DA kom kunde realtidsdatakällor paras/kopplas ihop med en datamottagare utan en specifik drivrutin för detta par. Se [5] för information om OPC Data Access.

2.2 OPC-Server

De flesta tillverkare av OPC-Servrar låter servern vara gratis men det som kostar är drivrutinerna. I projektet användes en Siemens S7-300 PLC som konfigurerades enligt specifikationen som beskrivs i bilaga A. Eftersom OPCDA är en framtagen standard av OPC Foundation kan vilken OPC-Server som helst användas så länge den stödjer OPC DA, se [2] för mer information. Detta innebär att val av OPC-Server inte hindrar framtida projekt från att använda andra OPC-Servrar för att uppfylla sina behov.

2.2.1 KEPServerEX5

Denna OPC-Server har använts i tidigare projekt på HöganäsBjuf där ugnsstyrningen uppdaterades. Gamla PLC:er byttes till nya Mitsubishi Q-serie PLC:er och använde då KepServer för att kommunicera med dessa.

Kostnad: $995, se [6] för mer information.

Stödda drivrutiner i paketet:

1. Siemens S7-200
2. Siemens S7-300
3. Siemens S7-400
4. Siemens S7-1200
5. Siemens S7-1500
6. Siemens S5
7. Siemens S5 (3964R)
8. Siemens S7 MPI

2.2.2 TOP Server

Denna OPC-Server har använts i ett projekt hos Knauf Danogips för att kommunicera med gamla Telemecanique PLC:er för att rätta till ett fel som fanns i befintlig kommunikation.

Kostnad: $995, se [7] för mer information.

Stödda drivrutiner i paketet:

1. S7 Ethernet Master
2. S7 Ethernet Slave (Unsolicited)
3. S7-200 PPI/PPM Serial
4. S5 AS511 Serial (PG Port)
5. S5 3964/3964R Serial
6. S7-MPI

2.2.3 Matrikon

Denna OPC-Server har inte använts i tidigare projekt hos Prevas Malmö.

Kostnad: 900 €, se [8] för mer information.

Stödda drivrutiner i paketet:

1. Hilscher Netlink MPI
2. Siemens CP243 Comm module
3. Siemens CP343
4. Siemens CP343 Comm module
5. Siemens CP443 Comm module
6. Siemens PLCs
7. Siemens S7-1200
8. Siemens S7-200
9. Siemens S7-300
10. Siemens S7-400

2.3 Windows Service

En Microsoft Windows Service, tidigare kallat NT Service är en exekverbar applikation som tillåter körning under en väldigt lång tid. Dessa tjänster kan startas automatiskt vid datorns uppstart och kan pausas och startas om. Denna typ av applikation passar bra till servrar där funktionalitet behövs en längre tid och där applikationen inte ska kunna störas eller ”krocka” med en inloggad användares aktivitet. Applikationen skapas och skrivs i Microsoft Visual Studio. Koden kontrollerar vilka kommandon som kan skickas till tjänsten och vad som utförs vid varje kommando. De kommandon som kan skickas är ”Startar”, ”Pausar”, ”Återuppta” och ”Stoppar”. Egna kommandon kan även definieras. Tjänster kan hanteras i den inbyggda Service Control Manager som automatiskt startas vid systemstart eller via en klass kallad `ServiceController`, se [9] för mer information om Windows Services.

2.4 C#.NET

C# är ett typsäkert objektorienterat programmeringsspråk som använder sig av .NET Framework. Med C# kan bland annat Windows services, Klient-Server applikationer, XML Web services och databasapplikationer skapas. Programspråket stödjer inkapsling, arv och polymorfism. Language-Integrated Query (LINQ) ingår vilket möjliggör hämtning av data i form av frågor (queries) från många olika källor (inkl. databashanterare) som liknar Structured Query Language (SQL). C# använder sig av en skräphanterare (garbage collector) vilket ställer mindre krav på programmeraren, se [10] för mer information om C#.NET.

3 Metod

I detta kapitel beskrivs enligt vilken modell projektet genomfördes och mer ingående hur varje fas gick till.

3.1 Arbetsmetod

Projektet inleddes med en enklare förstudie för att bestämma vilken DBMS och OPC-Server som skulle användas. Arbetet genomfördes med en iterativ metod där implementation och testning skedde jämlöpande. Hur arbetet genomfördes visualiseras i Fig. 2 och kan beskrivas på följande vis:

1. Välja OPC-Server, DBMS och hitta OPC-Client
2. Testa OPC-Client (utforska om den går att använda) och designa PLC-MES gränssnitt (handskakning), se 4.3 och 5.2
3. Designa databas, se 4.4 och 5.3
4. Implementera PRECIS-Server, se 5.4
5. Testa implementation
6. Vid designfel, modifiera handskakning/databas. Vid buggar i koden, modifiera koden. Repetera steg 5
7. Sluttest, se bilaga A

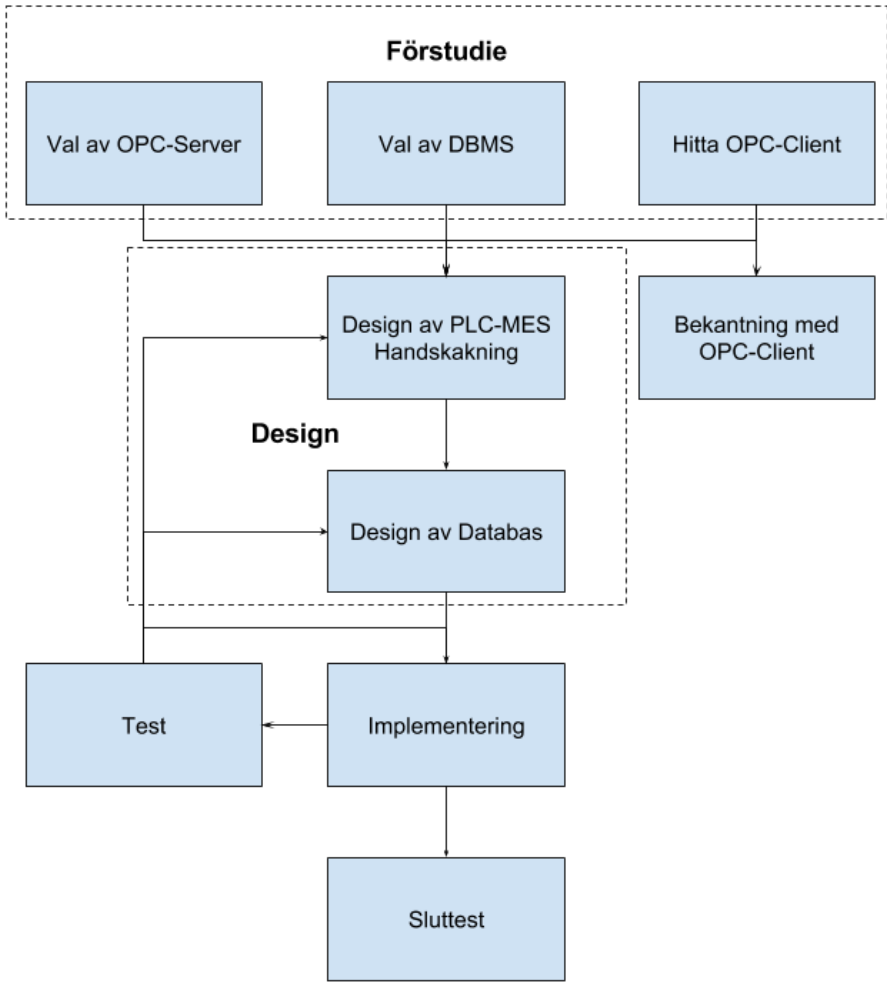

Fig. 2. Arbetsmetod, pil indikerar riktning som arbetet kan röra sig i.

Test kan innebära både informell testning enligt hela eller delar av bilaga A eller enkel testning under implementeringens gång. Denna metod valdes då programmering (utifrån egen erfarenhet) aldrig blir perfekt första försöket och problemen blir oftast synliga under kodning, därför är det bra att kunna stega tillbaka och göra om design vid behov.

3.2 Förstudie

Syftet med förstudien var valet av OPC-Server och DBMS som skulle göras. Det fanns inga prestandakrav sedan tidigare men låg kostnad för dessa var att föredra. Därför blev förstudien inte särskilt omfattande utan användes som en enklare grund att utgå från. Kandidaterna till DBMS kan tyckas vara godtyckliga men de togs fram på grund av tidigare erfarenhet av dessa. Då tidigare erfarenheter av OPC-Servrar inte fanns valdes kandidaterna utifrån förslag från en Prevas-anställd. När kandidaterna väl hade valts jämfördes de utifrån de olika kriterierna specificerade i 1.3.

3.3 Design

Denna fas inleddes efter valen av OPC-Server och DBMS gjorts. Projektets syfte definierat i 1.2 gav upphov till den initiala behovsanalysen för handskakning och databasen, se 4.3 och 4.4. Trots att denna fas är förutsättning till implementeringen så är det ändå meningen att kunna gå tillbaka till designfasen och göra de anpassningar som behövs.

3.4 Implementation

När designen ansågs vara tillräckligt bra skiftade fokus till mestadels programmering. Detta innebar realisering av Fig. 4. Under denna fas utvecklades även testspecifikationen jämlöpande. Efterhand som buggar i koden eller brister i designen upptäcktes fick dessa åtgärdas genom att stega tillbaka till designfasen för att sedan implementera ändringar och testa på nytt. Även addering av testfall till bilaga A gav upphov till förändring av kod och omtestning.

3.5 Sluttest

Detta test var det absolut sista som skedde i projektet (förutom rapportskrivning). Det genomfördes som ett blackbox-test enligt bilaga A där resultaten verifierades av en PLC-programmerare på Prevas genom att läsa av resultaten i PLC:en och loggar som skrevs. Dessa tester skedde även inofficiellt för att upptäcka eventuella kvarstående buggar innan det officiella testet genomfördes.

3.6 Källkritik

I detta projekt är det viktigast att informationen som hittas är korrekt och uppdaterad enligt senaste versioner av programvara mm. Tillförlitlighet baseras till stor del på följande frågor:

- Vem har publicerat informationen/driver hemsidan? Har författaren eller hemsidan officiell status?
- Vad är dennes relation till innehållet? Har denne intresse i att informationen är korrekt?

Referens [1] är pålitlig då det ligger i företagets intresse att uppge korrekt information om sig själv.

Referens [2] och [4] är mycket pålitliga eftersom de är skrivna av OPC-Foundation och beskriver OPC-standarden skapad av OPC-Foundation. [3], [5], [6], [7] och [8] är skriven av försäljare till programvaror som använder OPC-standarden vilket även har hög tillit (en försäljare bör för gott rykte utge korrekt information om sina produkter).

Referens [9], [10], [11] och [12] är mycket pålitliga då de är skrivna av Microsoft och beskriver applikationstyp och programmeringsspråk som är skapat av Microsoft som troligen innehar den mest korrekta informationen om detta.

4 Analys

Detta kapitel förklarar de olika valen av komponenter som gjordes och design av MES-PLC gränssnitt, databas och PRECIS-Server utifrån behovsanalyser.

4.1 Val av OPC-Server

KEPServerEX5 valdes för detta projekt. Som tidigare nämnt spelar det ingen roll vilken som väljs då alla tre stödjer OPCDA och priset skiljer sig inte mycket för en Siemens Driver Suite mellan dem. KEPServerEX5 valdes efter rekommendation av en Prevas-anställd. Dock köptes inte en licens utan en demoversion (begränsad till 2 timmars körtid) av KEPServerEX5 med alla deras drivrutiner.

4.2 Databashanterare

Denna del av projektet var inte enormt viktig och därför lades inte mycket tid ner på att bestämma vilken DBMS som skulle användas.

Det finns många databashanterare men kandidaterna för detta projekt var sådana som är gratis och där tidigare erfarenheter fanns.

Kandidaterna:

1. MySQL
2. PostgreSQL
3. SQL Server Express

Utav dessa valdes SQL Server Express. Anledningen till detta är att denna DBMS används ofta hos Prevas, den är lätt att använda med Visual Studio och Prevas hanterar många projekts databaser via SQL Server Management Studio. På grund av detta är det onödigt att frångå vad Prevas använder i vanliga fall för att vara enhetlig i utvecklingen.

4.3 Design av MES-PLC Gränssnitt

För att PRECIS-server och en PLC ska kunna kommunicera med varandra behöver de:

1. Varsin tag.
2. En PLC behöver kunna skicka begäran för nedladdning av värden till sina receptparametrar. Den behöver även beskriva för vilken produkt denna begäran avser.
3. PRECIS-server behöver kunna meddela om begäran är godkänd eller inte godkänd.
4. Om den inte är godkänd behöver den också kunna meddela varför den inte är godkänd.

Denna ”analys” användes som grund för design av MES-PLC Gränssnittet (Handskakning), se 5.2 för resultatet.

4.4 Design av relationsdatabas

Poängen med denna recepthanterare är att kunna byta värden på parametrar vid behov på ett recept. Detta behövs vid t ex. om en ny produkt ska produceras. Detta innebär att:

1. Ett recept behöver grundläggande information om receptet och taggar för kommunikation (handskakning) med en PLC, detta innebär att ett ”recepthuvud” bör finnas.
2. Sedan behövs parametrarna för receptet med information om vilken adress i PLC:en dessa ska vara i och information om enhet t ex. rotationshastighet i rpm eller flöde i m3/s och vad som är tillåtna värden på parametern (hanteras i webbgränssnittet).
3. Vilka produkter som hanteras av systemet måste finnas med.
4. Sist behövs de faktiska värdena på parametrarna och för vilken produkt de avser.

Dessa behov kan visualiseras enligt Fig. 3. Se Fig. 5 för den fullständiga lösningen.

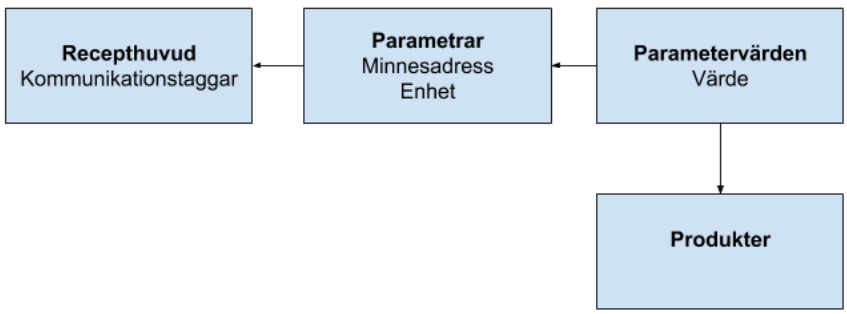

Fig. 3. Ingående design av databas, pil pekar mot den tabell en referens finns.

4.5 Design av PRECIS-Server

Programflödet för PRECIS-Server byggde på designen av databasen, se 4.4 och 5.3. Den kan beskrivas enligt TaBELL V, TaBELL VI, TaBELL VII, Fig. 4 och på följande vis:

1. RecipeService skapas och tilldelas RecipeId samt Version.
2. Kommunikationstaggar (se TaBELL I) och receptets alla parametrar hämtas.
3. Gå till ”Idle”
4. Om Request 0 kommer:
    a. Nollställ alla parametrar och kommunikationstaggar
    b. Gå till steg 3
5. Om Request 1000 kommer
    a. Hämta begärd produkt från PLC
6. Om produkt ogiltig eller receptet är ofullständigt
    a. CommandTag = Request Deined (145), se TaBELL II
    b. ErrorTag = 10 eller 30, se TaBELL IV
    c. Nollställ alla parametrar och begärd produkt.
7. Om produkt är giltig och receptet är ofullständigt
    a. Hämta parametervärden för den specifika produkten
    b. Sätt parametervärdena i PLC
    c. CommandTag = Request Confirmed (140), se TaBELL II

Symboler för programflödet ser ut på följande vis:

1. Rundad rektangel: Start eller paus i koden/programflödet
2. Rektangel: Process, vad som sker beskrivs i rutan
3. Romb: Beslut, handling bestäms utifrån något kriterium

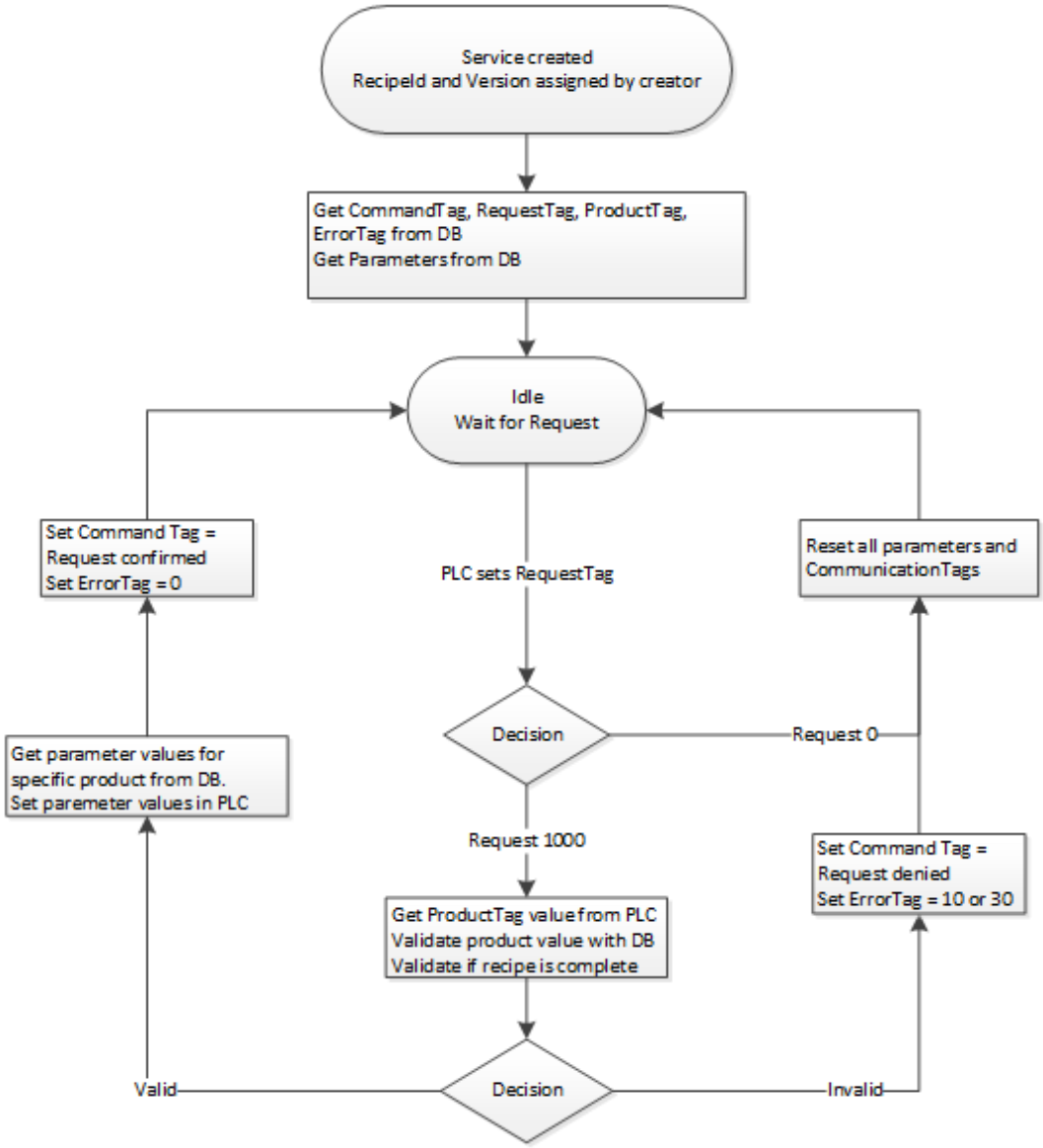

Fig. 4. Flödesschema för PRECIS-server

5 Resultat

I detta kapitel beskrivs de olika komponenterna som bland annat design av MES-PLC gränssnitt och databas mm. i den slutgiltiga produkten och hur de arbetar samt några kodexempel visas och beskrivs.

5.1 OPC-Client

För att kunna kommunicera med en OPC-Server från en Windows Service behövs en OPC-Client. Denna klient kan vara av olika typer, nämligen OPC Data Access (DA) eller OPC Unified Architecture (UA). DA är en äldre variant som har större stöd (större urval av drivrutiner) för olika PLC:ar och mer dokumentation tillgänglig, se 2.1.2. En OPC DA valdes för det stora utbudet av drivrutiner och att den är väletablerad på marknaden. Eftersom PRECIS-server skulle utvecklas som en Windows Service, dvs. köras på plattformen Windows så var det inget problem att OPC DA är begränsad till Windows. Den specifika klient som användes tillhandahölls av en anställd på Prevas. Tyvärr fanns ingen specifikation för denna klient tillgänglig så inlärning av denna skedde genom ”trial and error”.

5.2 MES-PLC Gränssnitt

Kommunikationen mellan MES-systemet och PLC-systemet sker via OPC och ett antal signaler, eller taggar, se TaBELL I. Denna specifikation för handskakning är ett underlag för konfigurering av en PLC. Denna specifikation gavs till en PLC-programmerare på Prevas som konfigurerade en Siemens S7-300 PLC enligt specifikationen för att kunna möjliggöra testning av programmet. Tillägg av signaler och deras olika koder kan göras i framtiden efter behov.

5.2.1 Signalutbyte

En PLC måste ha följande taggar for att kunna kommunicera med PRECIS-server.

TABELL I. SIGNALER PRECIS HANTERAR

Signal	Beskrivning	Typ	Riktning
CM	Command	Real	MES->PLC
RQ	Request	Real	PLC->MES
EC	Error code	Real	MES->PLC
Pxx	Parameter xx	Real	MES->PLC
Q01	Request parameter 01	Real	<->

”Request” är begäran från PLC och ”Request parameter 01” innehåller data som PRECIS-server behöver för att kunna behandla en begäran, se TaBELL III.

”Error code” används för att meddela PLC vad som är fel om något har gått fel, se TaBELL IV.

”Parameter xx” är ett samlingsnamn för alla parametrar receptet innehåller. ”Parameter xx” tilldelas det värde som är specificerat i databasen för den parametern.

5.2.2 Commands

Följande är olika kommandon med en associerad kod som PRECIS kan skicka till en PLC. Det finns endast två stycken för detta projekt men fler kan implementeras i framtiden vid behov.

TABELL II. KOMMANDON PRECIS KAN SKICKA

Kod	Betydelse
140	Request confirmed
145	Request denied

5.2.3 Requests

Följande är begäran med en associerad kod som PRECIS kan skicka till en PLC.

TABELL III. BEGÄRAN PRECIS KAN HANTERA

Kod	**Betydelse**
1000	Ladda ner alla parametrar

Programkod för hantering av begäran:

Förklarad kod hittas via {x} i Kodstycke 1:

PLCRequest anropas om värdet på RequestTag ändras.

1. Värdet för RequestTag hämtas
2. Om lastRequest inte == 0 och request inte == 0
    a. Alla parametrar rensas, clearParameters().
3. Annars körs handleRequest(request)
4. Om request = 0
    a. Inga fel, sätt errorCode = 0
    b. Sätt CommandTag = 0, setCommandTag(0)
    c. Rensa parametrar, clearParameters()
5. Om nedladdning av parametrar lyckas
    a. Sätt CommandTag = RequestConfirmed, setCommandTag(140).
6. Om nedladdning av parametrar misslyckas
    a. Sätt CommandTag = RequestDenied, setCommandTag(145).
7. lastRequest = request.

Kodstycke 1. Hantering av begäran

```
//In RecipeService.cs
private void PLCRequest(object sender, ValueChangedEventArgs e)
        {
            var values = this.group.ReadItems(this.items);
            var clientHandle = this.deviceMappingList.Where(y =>
y.Value == "RequestTag").First().Key;
            var requestTag = this.items.Where(x => x.ClientHandle ==
clientHandle).First();
            float request = requestTag.GetValue<float>();{1}

            if (lastRequest != 0 && request != 0)
            {
                clearParameters();{2}
            }
            else{
                handleRequest((int)request); {3}
            }
            lastRequest = (int)request; {7}
        }

private void handleRequest(int request)
        {
            Logger.Write(RecipeId + "." + Version + ": Request " +
request + " recieved", "Information", 0, runtimeId,
TraceEventType.Information, RecipeId + "," + Version);
            switch (request)
            {
                case 0: {4}
                    errorCode = 0;
                    SetCommandTag(0);
                    clearParameters();
                    break;
                case 1000:
                    if (this.setParameterValues())
                    {
                        this.SetCommandTag(RequestConfirmed); {5}
                    }
                    else
                    {
                        this.SetCommandTag(RequestDenied); {6}
                    }
                    break;
            }
        }
```

5.2.4 Error codes

Följande är felkoder som PRECIS kan skicka till en PLC som sedan en operatör kan använda för felsökning.

TABELL IV. FELKODER PRECIS KAN GENERERA

Kod	**Betydelse**
10	Produkten existerar inte
30	Receptet är ofullständigt

5.2.5 Nedladdning av värden (RQ=1000) där inga fel inträffar

Följande tabell är ett flöde som beskriver kommunikationen mellan MES och PLC samt kommunikationens riktning.

TABELL V. KOMMUNIKATION MELLAN MES OCH PLC

MES	**Riktning**	**PLC**	**Kommentar**
	<-	RQ = 1000 Q01 = ProductId	Skicka alla parametrar
Pxx = xx **CM = Request Confirmed (140)**	->		Alla begärda värden skickade
	<-	RQ = 0	Nollställer request
CM = 0 **Q01 = -1** **Pxx = -1**	->		Nollställer commando

5.2.6 Nedladdning av värden (RQ=1000), Error Code (EC=10)

Följande tabell är ett flöde som beskriver kommunikationen mellan MES och PLC samt kommunikationens riktning.

TABELL VI. KOMMUNIKATION MELLAN MES OCH PLC

MES	**Riktning**	**PLC**	**Kommentar**
	<-	RQ = 1000 Q01 = ProductId	Skicka alla parametrar
CM = Request denied (145) **EC = 10**	->		Alla begärda värden skickade
	<-	RQ = 0	Nollställer request
CM = 0 **EC = 0** **Q01 = -1** **Pxx = -1**	->		Nollställer commando

5.2.7 Nedladdning av värden (RQ=1000), Error Code (EC=30)

Följande tabell är ett flöde som beskriver kommunikationen mellan MES och PLC samt kommunikationens riktning.

TABELL VII. KOMMUNIKATION MELLAN MES OCH PLC

MES	**Riktning**	**PLC**	**Kommentar**
	<-	RQ = 1000 Q01 = ProductId	Skicka alla parametrar
CM = Request denied (145) **EC = 30**	->		Alla begärda värden skickade
	<-	RQ = 0	Nollställer request
CM = 0 **EC = 0** **Q01 = -1** **Pxx = -1**	->		Nollställer commando

5.3 Relationsdatabas

Detta kapitel beskriver hur databasen är uppbyggd som helhet och sedan mer ingående vad varje tabell innehåller. Se 4.4 för den ingående behovsanalysen av databasen.

5.3.1 Design av relationsdabas

Alla tabeller innehåller information om vem som senast ändrade något och när det skedde. Fig. 5 visar hur relationsdatabasen är uppbyggd och är en realisering av Fig. 3

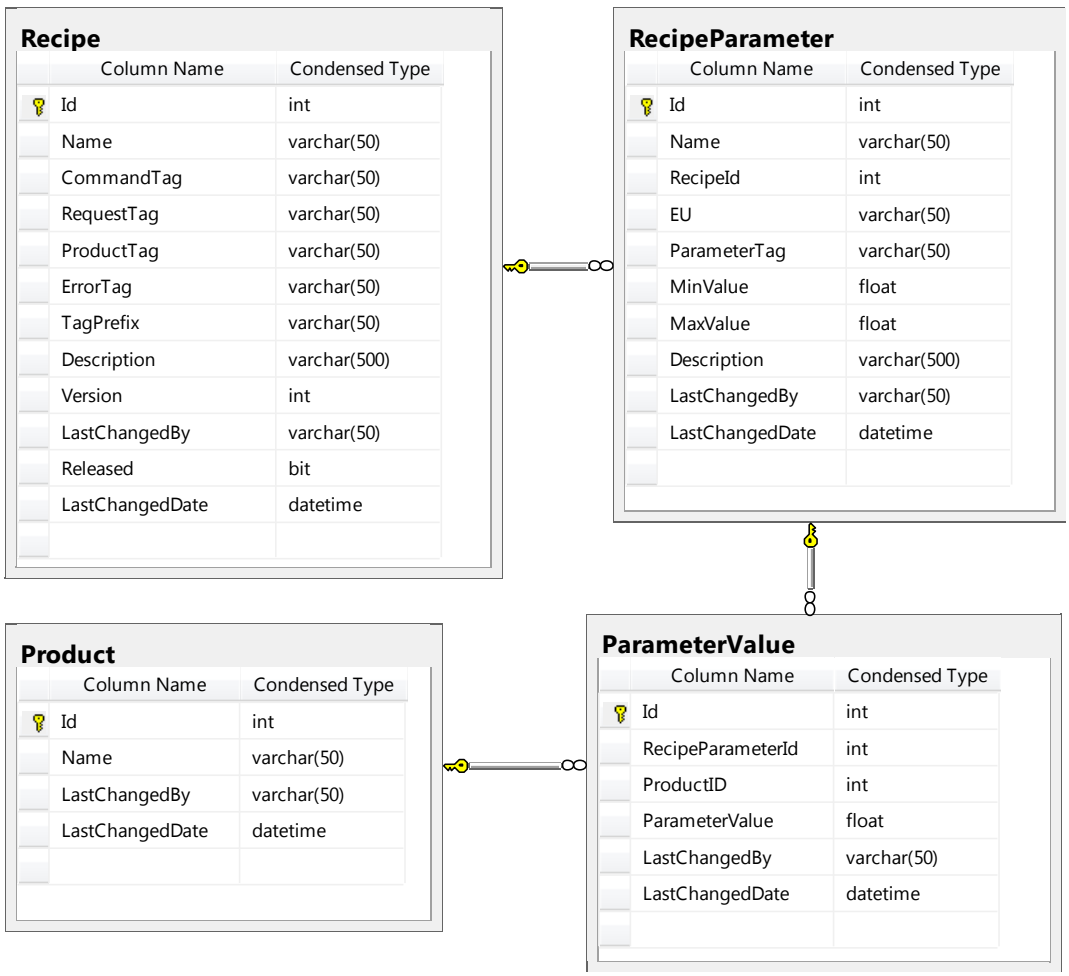

Fig. 5. UML-Diagram av relationsdatabasen genererat av SQL Management Studio.

5.3.2 Recipe

Denna tabell agerar som ”huvud” till de andra receptrelaterade tabellerna. Den har taggar för handskakning mellan PLC och MES, dessa är ”CommandTag”, ”RequestTag”, ”ProductTag”. TagPrefix används för adressering i KepServerEX. För att skriva till ”DB100,DINT100” i en PLC behövs kanal och namn på enheten i KepServerEX. Den fullständiga adressen är då t ex ”Channel1.Device1.DB100,DINT100”, prefixen kommer då bli ”Channel1.Device1.” så användaren slipper skriva in den fullständiga adressen på alla taggar.

5.3.3 RecipeParameter

Denna tabell definierar en parameter med namn, enhet, vilken adress i PLC:en den har och sedan vilket recept den tillhör.

5.3.4 Product

Denna tabell finns för att kunna ha olika parametervärden för samma recept, t ex. om ”lättmjölk” eller ”mellanmjölk” ska produceras med samma recept men olika parametervärden.

5.3.5 ParameterValue

Här ligger de faktiska parametervärdena och dessa hittas med kombinationen av ”ProductId” och ”RecipeParameterId”.

5.4 PRECIS-server

PRECIS-server består av flertalet Windows Services som kallas RecipeService och en HostService som hanterar dessa. Detta visualiseras i Fig. 6.

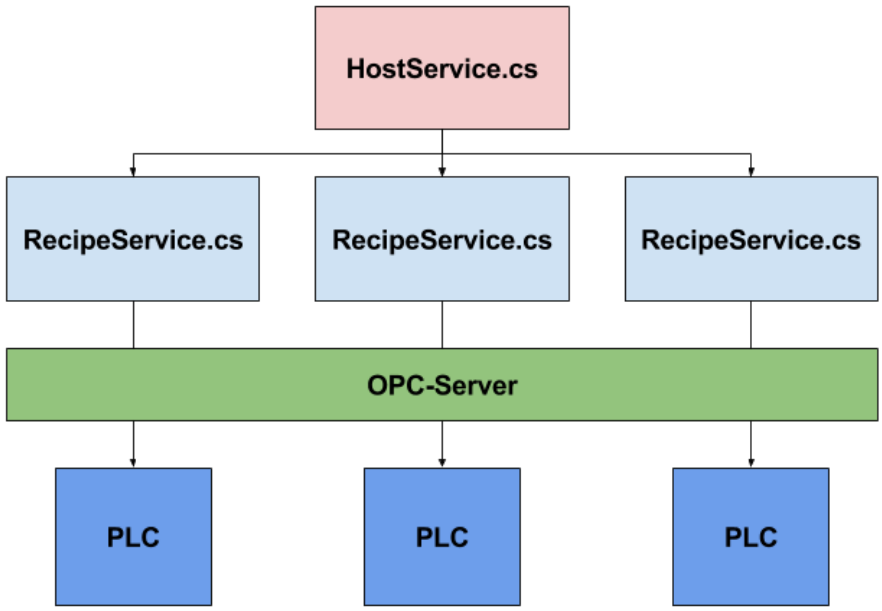

Fig. 6. Programuppbyggnad för PRECIS, data från RecipeService till PLC går via OPC-Server

5.4.1 HostService

När HostService skapas hämtar den alla frisläppta recept från databasen och för varje recept skapas en RecipeService som tilldelas RecipeId och Version.

5.4.2 RecipeService

Denna service hanterar begäran gjord av en PLC. Vid en korrekt begäran och om det begärda receptet är fullständigt laddar den ner parametervärden till PLC:en den är kopplad till, se Kodstycke 1 och Fig. 4.

5.4.3 Server refresh

I HostService finns en timer som efter ett tidsintervall (60 sekunder) genomför en uppdatering av sina RecipeServices, se Kodstycke 2. Den kontrollerar alla existerande RecipeServices om de inte längre är frisläppta, om någon inte är det stoppas den. Finns det något recept i databasen som är frisläppt men inte körs av HostService laddas receptId ner och en ny RecipeService startas.

Programkod för Server Refresh:

Förklarad kod hittas via {x} i Kodstycke 2:

1. För varje RecipeServices körs Reinit()
2. Om den blir stoppad så raderas den.
3. Alla recept hämtas från databasen.
4. Alla recept i databasen jämförs med alla recept i HostService för att se om de finns i HostService
5. Om den inte finns skapas en ny RecipeService.
6. Alla RecipeServices i HostService som inte är startade startas.

Kodstycke 2. Serveruppdatering

```
//In HostService.cs
        private void OnTimedEvent(Object source,
System.Timers.ElapsedEventArgs e)
        {
            for (int i = 0; i < recipeServices.Count();i++)
            {
                recipeServices.ElementAt(i).Reinit(); {1}
                if (recipeServices.ElementAt(i).IsStopped)
                {
                    recipeServices.RemoveAt(i); {2}
                }
            }
            var db = new PRECIS.Server.Models.PRECISEntities();
            var recipes = db.Recipes.Where(x => x.Released); {3}
            bool exists = false;
            foreach(var r in recipes)
            {
                foreach(var rs in recipeServices)
                {
                    if (rs.RecipeId == r.Id && rs.Version ==
r.Version)
                    {
                        exists = true; {4}
```

```
                        break;
                    }
                }
                if (!exists)
                {

this.recipeServices.Add(this.unityContainer.Resolve<IRecipeServ
ice>(new ParameterOverride("recipeId", r.Id), new Param
                eterOverride("version", r.Version))); {5}
                }
            }
            foreach (var rs in recipeServices)
            {
                if (rs.IsStopped)
                {
                    rs.Start(); {6}
                }
            }
        }
//In RecipeService.cs
public void Reinit()
        {
            var db = new PRECIS.Server.Models.PRECISEntities();
            var released = db.Recipes.Where(x => x.Id ==
RecipeId).FirstOrDefault();

            if (released == null || !released.Released)
            {
                this.Stop();
            }
        }
```

6 Slutsats

Projektet blev lyckat där en PLC kan konfigureras enligt innehållet i en databas och på så vis kan inställningar för olika produkter snabbt göras utan större ansträngning. Se bilaga B för källkoden till RecipeService.cs. Det utvecklade systemet ger Prevas en unik recepthanterare utvecklat efter deras behov som även kan modifieras i framtiden om så behövs utan att behöva förlita sig på externa utvecklare och produkter lika mycket som tidigare.

6.1 Svar på frågeställningar

Vilken OPC-server ska användas?

KEPServerEX5 valdes för den används i tidigare projekt och blev rekommenderad av en anställd hos Prevas Malmö. Dock som tidigare nämnt spelar valet av OPC-Server ingen roll så länge den stödjer OPCDA. Den kan bytas ut efter behov som t ex. om en drivrutin för en PLC är billigare hos en viss tillverkare.

Vilken databashanterare ska användas?

Valet av DBMS blev SQL Server Express för sin låga kostnad (gratis) och eftersom inget prestandakrav fanns var den avgörande faktorn den befintliga användningen av SQL Server Express hos Prevas.

Hur ska PRECIS-server utvecklas?

Utifrån en enklare behovsanalys byggdes handskakningen på ett signalutbyte där PRECIS-server skickar kommandon och PLC gör begäran, se 4.3. Databasen byggdes också på en behovsanalys och resultatet blev ett ”recepthuvud” som har parametrar kopplade till sig där sedan varje parameter kan ha ett specifikt värde för varje produkt systemet kan behandla, se 4.4.

6.2 Rekommendationer

För att PRECIS skulle kunna bli bättre måste den utvecklas vidare. Detta kan göras på följande sätt:

1. HostService och RecipeService flyttas ut till två basklasser där tillägg av funktionalitet kan göras genom att utvidga dessa klasser. För mer information om klasser, se [11].
2. Uppkopplingen till databasen lyfts ut till ett Interface (se [12]) och på så vis kan flera olika databashanterare användas och bytas ut på ett snabbt och smidigt sätt.
3. Stöd för hantering av batch och order för att få större kontroll på produktionen.

Med dessa förbättringar kan PRECIS bli ett mycket kraftfullare verktyg som kan anpassas till fler situationer och kunder till Prevas.

7 Terminologi

DBMS	Database Management System
MES	Manufacturing Execution System
MES-PLC Gränssnitt	Handskakning
PLC	Programmable Logic Controller
OPC	Open Platform Communications
OPC-AE	OPC Alarms & Events
OPC-DA	OPC Data Access
OPC-HDA	OPC Historical Data Access
OPC-UA	OPC Unified Architecture
SQL	Structured Query Language
UML	Unified Modeling Language

8 Referenser

[1] ”Om Prevas,” Prevas AB, [Online]. Available: http://prevas.se/innovation_for_growth_prevas.html. [Använd 29 April 2016].

[2] ”About: What is OPC?,” OPC Foundation, [Online]. Available: https://opcfoundation.org/about/what-is-opc. [Använd 13:27 18 April 2016].

[3] ”What is OPC?,” Cogent Real-Time Systems, [Online]. Available: http://www.opcdatahub.com/WhatIsOPC.html. [Använd 3 Maj 2016].

[4] ”Classic,” OPC Foundation, [Online]. Available: https://opcfoundation.org/about/opc-technologies/opc-classic. [Använd 03 06 2016].

[5] ”OPC Data Access (OPC DA) Versions & Compatibility,” MatrikonOPC, [Online]. Available: http://www.matrikonopc.com/opc-server/opc-data-access-versions.aspx. [Använd 3 Juni 2016].

[6] ”Siemens Suite for KEPServerEX,” Kepware, [Online]. Available: https://www.kepware.com/products/kepserverex/suites/siemens-suite. [Använd 13 11:08 April 2016].

[7] ”TOP Server: Siemens Suite,” SoftwareToolbox, [Online]. Available: https://www.softwaretoolbox.com/topserver/html/SiemensSuite.html. [Använd 11:26 13 April 2016].

[8] ”OPC-drivers: OPC Siemens S7,” MatrikonOPC, [Online]. Available: http://www.matrikonopc.com/opc-drivers/opc-siemens-s7-plc/base-driver-details.aspx. [Använd 11:04 13 April 2016].

[9] ”Introduction to Windows Service Applications,” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/d56de412(v=vs.110).aspx. [Använd 2 Juni 2016].

[10] ”Introduction to the C# Language and the .NET Framework,” Microsoft, [Online]. Available: https://msdn.microsoft.com/enus/library/z1zx9t92.aspx. [Använd 3 Maj 2016].

[11] ”Inheritance (C# Programming Guide),” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ms173149(v=vs.100).aspx. [Använd 9 Juni 2016].

[12] ”Interface (C# Reference),” Microsoft, [Online]. Available: https://msdn.microsoft.com/sv-SE/library/87d83y5b.aspx. [Använd 09 Juni 2016].

9 Bilagor

A Testprotokoll för PRECIS-server

A.1 Förutsättningar

För att kunna genomföra dessa tester behövs:

1. En PLC konfigurerad enligt 4.3
2. En databas konfigurerad enligt modellen i 5.3.1 samt uppkopplad till denna
3. En KEPServerEX5 som har uppkoppling mot PLC:en.
4. Huvudprogrammet PRECISserver.
5. .NET Framework 4.5 installerat
6. Programvara för att skicka begäran mm. från PLC, KEPServerEX5 har en inbyggd ”quickclient”.

A.2 Allmänt

Det finns två typer av testning: **N** och **F**.

N är testning av normalfall.

F är testning av felhantering.

A.3 Uppstart

Test ID	Test typ	Beskrivning	Förväntat resultat	Ok
1.1	**N**	Vid programstart hämtas alla ComTag till programmet	Alla ComTag hämtas till RecipeService.cs och sparas i en intern datastruktur med samma värden som i databasen	Ok
1.2	**N**	PLC har ingen request aktiv	Inget laddas ner till PLC	Ok
1.3	**N**	PLC har aktiv request utan att svar har erhållits	Parametrar laddas ner till PLC	Ok
1.4	**N**	PLC har aktiv request samt att svar har erhållits	Parametrar laddas ej ner till PLC	Ok
1.5	**N**	PLC har ingen aktiv request men CommandTag är aktivt	Parametrar laddas ej ner till PLC, CM nollas	Ok
1.6	**F**	Uppkoppling mot OPC-server finns inte	ComTag och parametrar laddas ner till RecipeService men kommunikation till PLC saknas (Uppkoppling återfås när OPC-server startas igen)	OK
1.7	**F**	Uppkoppling mot databas finns inte	Alla recipeServices stoppas och startas igen vid nästa serveruppdatering om uppkoppling finns då.	Ok

A.4 Nedladdning av parametrar till PLC

Test ID	Test typ	Beskrivning	Förväntat resultat	Ok
2.1	**N**	PLC begär alla parametrar (RQ 1000) med fullständigt recept och giltig produkt	Alla parametrar sparas i PLC med samma värde som i databasen	Ok
2.2	**N**	PLC skickar RQ 0	Parametrar, CM och Q01 nollställs	Ok
2.3	**F**	PLC begär alla parametrar (RQ 1000) med fullständigt recept men ogiltig produkt	EC 10 skickas till PLC	Ok
2.4	**F**	PLC begär alla parametrar (RQ 1000) med ofullständigt recept men giltig produkt	EC 30 skickas till PLC	Ok

A.5 Serveruppdatering

Test ID	Test typ	Beskrivning	Förväntat resultat	Ok
3.1	**Normal**	Ett recept är inte längre frisläppt	Dess RecipeService stängs ner och raderas från HostService vid en serveruppdatering.	Ok
3.2	**Normal**	Ett recept har blivit frisläppt	En ny RecipeService skapas i HostService och startas vid en serveruppdatering	Ok

A.6 Operativsystem

Test ID	Test typ	Beskrivning	Förväntat resultat	Ok
4.1	**Normal**	Kör programmet på Windows 7 64 bit	Programmet har full funktionalitet	Ok
4.2	**Normal**	Kör programmet på Windows Server 2008 64 bit	Programmet har full funktionalitet	N/A

B Programkod för RecipeService.cs

```
// ----------------------------------------------------------
----------
// <copyright file="RecipeService.cs" company="Prevas AB">
//     Copyright (c) Prevas AB. All rights reserved.
// </copyright>
// ----------------------------------------------------------
----------
namespace Precis.Server.Services
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using PRECIS.Server.ResourceAccess.Opc;
    using Microsoft.Practices.EnterpriseLibrary.Logging;
    using System.Diagnostics;
    /// <summary>
    /// This class handles downloading of parameters to the PLC.
The communication is done via OPC.
    /// </summary>
    public class RecipeService : IRecipeService
    {
        #region Fields
        /// <summary>
        /// A reference to the client.
        /// </summary>
        private readonly IClient client;

        /// <summary>
        /// This dictionary contains the mapping from a OPC
items client handle and the device id used in our database.
        /// </summary>
        private readonly Dictionary<int, string>
deviceMappingList = new Dictionary<int, string>();

        /// <summary>
        /// Contains all the signals used for communication with
the OPC.
        /// </summary>
        private readonly List<Item> items = new List<Item>();

        /// <summary>
        /// A reference to the group that all the items will be
added to.
        /// </summary>
        private IGroup group;
```

```
        #endregion

        #region Constructors and Destructors

        /// <summary>
        /// Initializes a new instance of the <see
cref="BunkerService"/> class.
        /// </summary>
        /// <param name="client">The client.</param>
        /// <param name="recipeId">The recipe to run.</param>
        /// <param name="version">The recipe version.</param>
        public RecipeService(IClient client, int recipeId, int
version)
        {
            this.client = client;
            this.RecipeId = recipeId;
            this.Version = version;
            this.lastRequest = 0;
            errorCode = 0;
            firstRun = true;
            runtimeId =
int.Parse(String.Concat(recipeId.ToString()+"0",
Version.ToString()));
        }
        //----------------------------------------
        // for logging --> Logger.Write(RecipeId + "." +
Version" + ": <<message>>", "Information", 0, runtimeId,
TraceEventType.Information, RecipeId + "," + Version);
        //
        //----------------------------------------
        #endregion

        #region Properties

        /// <summary>
        /// Gets a value indicating whether this instance is
started.
        /// </summary>
        /// <value>
        ///   <c>true</c> if started; otherwise, <c>false</c>.
        /// </value>
        public bool IsStarted { get; private set; }
        public int RecipeId { get; private set; }
        public int Version { get; private set; }
        private int lastRequest;
        private int errorCode;
        private bool firstRun;
```

```
        private static String TagPrefix;
        private static int runtimeId;
        private static int RequestConfirmed = 140;
        private static int RequestDenied = 145;

        /// <summary>
        /// Gets a value indicating whether this instance is
stopped.
        /// </summary>
        /// <value>
        ///   <c>true</c> if stopped; otherwise, <c>false</c>.
        /// </value>
        public bool IsStopped
        {
            get
            {
                return !this.IsStarted;
            }
        }
        #endregion
        #region Public Methods
        /// <summary>
        /// Starts this instance.
        /// </summary>
        public void Start()
        {
            this.client.Connect();

            this.group = this.client.AddGroup("PrecisServer",
true);
            Logger.Write(RecipeId + "."+ Version+": Starting
recipe service", "Information", 0, runtimeId,
TraceEventType.Information, RecipeId + "," + Version);

            var db = new PRECIS.Server.Models.PRECISEntities();
            var recipe = db.Recipes.Where(x => x.Id ==
RecipeId).FirstOrDefault();
            TagPrefix = recipe.TagPrefix;
            this.AddParameters();
            this.AddCommunicationTags();

            this.IsStarted = true;
        }



        /// <summary>
        /// Stops this instance.
```

```
        /// </summary>
        public void Stop()
        {
            Logger.Write(RecipeId + "." + Version + ": Stopping
recipe service", "Information", 0, runtimeId,
TraceEventType.Information, RecipeId + "," + Version);
            this.IsStarted = false;
            this.group.RemoveItems(items);
            this.RemoveOpcItems();
        }
        public void Reinit()
        {
            try {
                var db = new
PRECIS.Server.Models.PRECISEntities();

                var released = db.Recipes.Where(x => x.Id ==
RecipeId).FirstOrDefault();
                if (released == null || !released.Released)
                {
                    this.Stop();
                }
            }
            catch (Exception e)
            {
                Logger.Write(RecipeId + "." + Version + ": " +
e.Message, "Information", 0, runtimeId, TraceEventType.Error,
RecipeId + "," + Version);
            }


        }
        #endregion
        #region Private Methods
        /// <summary>
        /// Sets the CommandTag in the OPC.
        /// </summary>
        private void SetCommandTag(int CommandCode)
        {
            var clientHandle = this.deviceMappingList.Where(y =>
y.Value == "CommandTag").First().Key;
            var CommandTag = this.items.Where(x =>
x.ClientHandle == clientHandle).First();
            CommandTag.Value = CommandCode;
            clientHandle = this.deviceMappingList.Where(y =>
y.Value == "ErrorTag").First().Key;
            var ErrorTag = this.items.Where(x => x.ClientHandle
== clientHandle).First();
```

```
            ErrorTag.Value = errorCode;
            if (CommandCode == 145){
                Logger.Write(RecipeId + "." + Version + ":
Command code " + CommandCode + " set", "Information", 1,
runtimeId, TraceEventType.Error, RecipeId + "," + Version);
            }
            else if (CommandCode != 0) {
                Logger.Write(RecipeId + "." + Version + ":
Command code " + CommandCode + " set", "Information", 0,
runtimeId, TraceEventType.Information, RecipeId + "," +
Version);
            }
        }
        /// <summary>
        /// Adds all tags handling communication to the OPC
        /// </summary>
        private void AddCommunicationTags()
        {
            try {

                var db = new
PRECIS.Server.Models.PRECISEntities();
                var recipe = db.Recipes.Where(x => x.Id ==
RecipeId).FirstOrDefault();

                var item = this.group.AddItem(TagPrefix +
recipe.CommandTag, false);
                this.items.Add(item);
                this.deviceMappingList.Add(item.ClientHandle,
"CommandTag");
                Logger.Write(RecipeId + "." + Version + ":
CommandTag added:" + recipe.CommandTag, "Information", 0,
runtimeId, TraceEventType.Information, RecipeId + "," +
Version);
                item = this.group.AddItem(TagPrefix +
recipe.ProductTag, false);
                this.items.Add(item);
                this.deviceMappingList.Add(item.ClientHandle,
"ProductTag");
                Logger.Write(RecipeId + "." + Version + ":
ProductTag added:" + recipe.ProductTag, "Information", 0,
runtimeId, TraceEventType.Information, RecipeId + "," +
Version);
                item = this.group.AddItem(TagPrefix +
recipe.ErrorTag, false);
                this.items.Add(item);
                this.deviceMappingList.Add(item.ClientHandle,
"ErrorTag");
```

```
                Logger.Write(RecipeId + "." + Version + ":
ErrorTag added:" + recipe.ErrorTag, "Information", 0, runtimeId,
TraceEventType.Information, RecipeId + "," + Version);
                item = this.group.AddItem(TagPrefix +
recipe.RequestTag, true);
                this.items.Add(item);
                this.deviceMappingList.Add(item.ClientHandle,
"RequestTag");
                item.ValueChanged += PLCRequest;
                Logger.Write(RecipeId + "." + Version + ":
RequestTag added:" + recipe.RequestTag, "Information", 0,
runtimeId, TraceEventType.Information, RecipeId + "," +
Version);
                Logger.Write(RecipeId + "." + Version + ":
Communication Tags Added", "Information", 0, runtimeId,
TraceEventType.Information, RecipeId + "," + Version);
            }catch(Exception e)
            {

                Logger.Write(RecipeId + "." + Version + ": " +
e.Message, "Information", 0, runtimeId, TraceEventType.Error,
RecipeId + "," + Version);
                this.Stop();
            }
        }
        /// <summary>
        /// Adds all parameters to the OPC
        /// </summary>
        private void AddParameters()
        {
            try {
                var db = new
PRECIS.Server.Models.PRECISEntities();
                var parameters = db.RecipeParameters.Where(x =>
x.RecipeId == this.RecipeId);
                foreach (var p in parameters)
                {
                    var item = this.group.AddItem(TagPrefix +
p.ParameterTag, false);
                    this.items.Add(item);

this.deviceMappingList.Add(item.ClientHandle, p.Name);
                }
                Logger.Write(RecipeId + "." + Version + ":
Parameters Added", "Information", 0, runtimeId,
TraceEventType.Information, RecipeId + "," + Version);
            }catch(Exception e)
            {
```

```
                Logger.Write(RecipeId + "." + Version + ": Error
adding parameters", "Information", 0, runtimeId,
TraceEventType.Error, RecipeId + "," + Version);
            }
        }
        /// <summary>
        /// Checks if the productId is valid
        /// </summary>
        private Boolean ProductIsValid(int productId)
        {
            try {
                var db = new
PRECIS.Server.Models.PRECISEntities();
                var product = db.Products.Where(x => x.Id ==
productId);
                if (product.Count() == 0)
                {
                    return false;
                }
            }catch (Exception e)
            {
                Logger.Write(RecipeId + "." + Version + ": Error
validating product", "Information", 0, runtimeId,
TraceEventType.Error, RecipeId + "," + Version);
                return false;
            }
            return true;
            }

        private int setParameterValues()
        {
            var values = this.group.ReadItems(this.items);
            var clientHandle = this.deviceMappingList.Where(y =>
y.Value == "ProductTag").First().Key;
            var productTag = this.items.Where(x =>
x.ClientHandle == clientHandle).First();
            var product = productTag.GetValue<int>();

            if (!ProductIsValid(product))
            {
                Logger.Write(RecipeId + "." + Version + ":
Product: "+ product + " invalid", "Information", 1, runtimeId,
TraceEventType.Error, RecipeId + "," + Version);
                this.errorCode = 10;
                Logger.Write(RecipeId + "." + Version + ": Error
Code " + errorCode + " set", "Information", 1, runtimeId,
TraceEventType.Error, RecipeId + "," + Version);
                return 145;
```

```
            }
                var db = new
PRECIS.Server.Models.PRECISEntities();
                var Parameterlist = db.RecipeParameters.Where(x
=> x.RecipeId == RecipeId);

                var query =
                   from RecipeParameter in db.RecipeParameters
                   where RecipeParameter.RecipeId == RecipeId
                   join paramValue in db.ParameterValues on
RecipeParameter.Id equals paramValue.RecipeParameterId
                   where paramValue.ProductId == product
                   select new { paramValue.RecipeParameterId,
paramValue.ProductId };

                int parameterValuesCount = query.Count();
                int parameterCount = Parameterlist.Count();
                if (parameterCount != parameterValuesCount)
                {
                    Logger.Write(RecipeId + "." + Version + ":
Recipe incomplete for Product: " + product, "Information", 1,
runtimeId, TraceEventType.Error, RecipeId + "," + Version);
                    errorCode = 30;
                    Logger.Write(RecipeId + "." + Version + ":
Error Code " + errorCode + " set", "Information", 1, runtimeId,
TraceEventType.Error, RecipeId + "," + Version);
                    return 145;
                }
            foreach (var rp in Parameterlist)
            {
                var name = rp.Name;
                var value = rp.ParameterValues.Where(x =>
x.ProductId == product && x.RecipeParameterId ==
rp.Id).First().ParameterValue1;
                clientHandle = this.deviceMappingList.Where(y =>
y.Value == name).First().Key;
                var tag = this.items.Where(x => x.ClientHandle
== clientHandle).First();
                tag.Value = value;
            }
                this.errorCode = 0;
            Logger.Write(RecipeId + "." + Version + ": Parameter
Values downloaded", "Information", 0, runtimeId,
TraceEventType.Information, RecipeId + "," + Version);
            return 140;
```

```
        }
        private void handleRequest(int request)
        {
            if (!firstRun)
            {
                Logger.Write(RecipeId + "." + Version + ":
Request " + request + " recieved", "Information", 0, runtimeId,
TraceEventType.Information, RecipeId + "," + Version);
                switch (request)
                {
                    case 0:
                        errorCode = 0;
                        SetCommandTag(0);
                        clearParameters();
                        break;
                    case 1000:
                        int command = this.setParameterValues();
                        this.SetCommandTag(command);
                        break;
                }
            }
            else if(firstRun)
            {
                var clientHandle =
this.deviceMappingList.Where(y => y.Value ==
"CommandTag").First().Key;
                var CommandTag = this.items.Where(x =>
x.ClientHandle == clientHandle).First();
                int CM = CommandTag.GetValue<int>();
                if (request == 0 && CM != 0)
                {
                    this.clearParameters();
                    this.SetCommandTag(0);
                }else if(request == 1000 && CM == 0)
                {
                    int command = this.setParameterValues();
                    this.SetCommandTag(command);
                }
                firstRun = false;
            }
        }
        private void clearParameters()
        {
            try {
                var db = new
PRECIS.Server.Models.PRECISEntities();
                var list = db.RecipeParameters.Where(x =>
x.RecipeId == RecipeId);
```

```
                var clientHandle =
this.deviceMappingList.Where(y => y.Value ==
"ProductTag").First().Key;
                var tag = this.items.Where(x => x.ClientHandle
== clientHandle).First();
                tag.Value = -1.0;

                foreach (var rp in list)
                {
                    var name = rp.Name;

                    clientHandle =
this.deviceMappingList.Where(y => y.Value == name).First().Key;
                    tag = this.items.Where(x => x.ClientHandle
== clientHandle).First();
                    tag.Value = -1.0;
                }
                Logger.Write(RecipeId + "." + Version + ":
Parameter Values Reset", "Information", 0, runtimeId,
TraceEventType.Information, RecipeId + "," + Version);
            }
            catch(Exception e)
            {
                Logger.Write(RecipeId + "." + Version + ": Error
clearing parameters", "Information", 0, runtimeId,
TraceEventType.Error, RecipeId + "," + Version);
            }


        }
        private void PLCRequest(object sender,
ValueChangedEventArgs e)
        {
            var values = this.group.ReadItems(this.items);
            var clientHandle = this.deviceMappingList.Where(y =>
y.Value == "RequestTag").First().Key;
            var requestTag = this.items.Where(x =>
x.ClientHandle == clientHandle).First();
            float request = requestTag.GetValue<float>();

            if (lastRequest != 0 && request != 0)
            {
                clearParameters();
            }
            else{
                handleRequest((int)request);
            }
```

```
            lastRequest = (int)request;
        }
        /// <summary>
        /// Removes all the added items from the OPC.
        /// </summary>
        private void RemoveOpcItems()
        {
            this.group.RemoveItems(this.items);
            this.client.RemoveGroup("PrecisServer");
            Logger.Write(RecipeId + "." + Version + ": Removing
OPC items", "Information", 0, runtimeId,
TraceEventType.Information, RecipeId + "," + Version);
            this.client.Disconnect();
        }
        #endregion
    }
}
```