Objektsorienterad proview

Design

Revision: 04 10 07 Claes Sjöfors
Version: V4.1-1

SSAB Oxelösund

Inledning

Objektsorientering har länge dominerat programmeringstekniken genom språk som c++ och java. På senare år har objektsorientering även blivit en trend inom styrsystemvärlden. Proview har redan från början haft en objektsorienterad design, genom att databasen är uppdelad i objekt på vilka sedan t ex plc-metoder opererar. Men för att kunna utnyttja fördelarna med objektsorientering saknas några viktiga element

- möjligheten att bygga upp objekt av andra objekt.
- arv av attribut och metoder.
- skuggning/överlagring av attribut och metoder.

Låt oss titta närmare på ett exempel, en ABB frekvensomformare AS800. Idag konfigurerar man den med en PlantHier, och under den diverse signal-objekt och ett PlcPgm med styrprogrammet. I en mer objektsorienterad konfigurering skapar man en AS800 klass, som innehåller signalobjekten i form av attribut. Plcstyrningen läggs i ett sammansatt funktionsobjekt som kan kopplas till ett AS800 objekt. Modul och kanal objekt för profibus bakas ihop i ett io objekt som även detta kopplas till AS800 objektet. Det man har vunnit är

att det nu finns en klass, dvs en mall för AS800 konfigureringen, och den här mallen kan man bygga vidare på med objektsbild, symbol för översiktsbilder, hjälptext, URL till datablad etc. Skapar man en instans av AS800 får man allt detta på köpet.

Min förhoppning är att detta ska förändra vårt sätt att bygga styrsystem. Arbetet kommer i stor utsträckning att vara inriktat på att bygga klasser för de i systemet ingående komponenterna, och dessa klasser får sedan andra projekt tillgång till. Har någon tillverkat en generell AS800 klass kan övriga projekt använda denna, och principer som inkapsling och återanvändning av kod kan utnyttjas även i proview. Skulle man behöva funktion utöver det som finns in AS800 klassen använder man arvs-funktionen och skapar en subklass.

Attrref istället för objid

De attribut som idag är av typen objid måste i många fall kunna peka på ett objekt som är ett attribut i ett annat objekt, och därmed måste de refereras med en AttrRef istället.
Objid typen byts till AttrRef i bl a föjande objekt:

- SigChanCon i signaler och kanaler.
- Object eller XxObject i GetXx/StoXx/SetXx/ResXx/CStoXx.
- I GetXp/StoXp/SetXp/ResXp/CStoXp ersätts två attribut, Object eller XpObject och Parameter (String80), med en AttrRef.
- ResetObject i PlcPgm och NMpsCell objekt.
- DefGraph och DefTrend i signaler och $PlantHier.

Symboliska attributreferenser

Det finns behov att i klassdefinitonen kunna kunna referera från attributreferenser i en klass till attribut i en annan klass (eller i samma klass), dvs att specificera klassen och attributet, men lämna objektsidentiteten öppen. Objektsidentiteten ersätts sedan av någon enhet, typ plc't eller io-hanteringen.

Koden i ett template plc-program behöver t ex kunna referera till ingångar, utgångar och interna variabler i funktions-objektet. Dessa referenser måste kunna programmeras i klassdefinitionen, medan det är först vid instansieringen som objekts identiteten blir känd. I detta fallet ersätts den symboliska referensen med en verklig referens vid kompileringen av instansen.

Symboliska referenser implementeras med en ny typ av vrep, wb_vrepref, som instansieras i tre instanser för symbolerna $PlcHost, $PlcConnect och $IoConnect. Dess uppgift är kunna översätta symboliska AttrRef'ar till namn, och vv.

I en symbolisk AttrRef läggs vid för symbolen i Objid.vid och klassidentiteten för den klass som refereras i Objid.oix. Offset och Size markerar det attribut inom klassen som refereras på normalt sätt.

En symbolisk AttrRef översätt till `'Symbol':'klass'.'attribut'` där klass är namnet på klassen där kolonet efter volymsnamnet ersatts med '-', t ex

```
$PlcHost:CVolVWXN1T-Class-AS800.Frig.ActualValue
```

Klasser i klasser

Klasser i andra klasser implementeras på följande sätt.
Ett attribut kan även vara en klass genom att TypeRef i $Attribute är en cid (klassidentitet) och genom att Class-biten sätts i Flags.

Ett objekt som är ett attribut i ett annat objekt kallas ett attributobjekt.

Arv

Arv implementeras genom att första attributet har superklassens klassidentitet och namnges ”Super”.
Vid översätting namn ->attrref och attrref->namn presenteras superklassens attribut som om de tillhörde subklassen.
Om subklassen innehåller ett attribut med ett namn som redan finns definierat i superklassen, döljs superklassens attribut. Det kan dock refereras genom att explicit ange Super i attributnamnet, t ex `A1-B1-C1.Super.Status`
Även metoder ärvs från superklassen , och kan överlagras av metoder i subklassen med samma namn.

Template objekt

Värden i templateobjekt specifieras enligt namngivning nedan. Template objekt ärver inte template-värden från superklasser.

```
Object Template
  Body RtBody
     Attr Super.Temperatur = 10
     Attr Frigwb.Description = “Frigivnining”
     Attr Bredd = 25
  EndBody
EndObject
```

Interna referenser i Template objektet, dvs om en attribut referens pekar på ett attribut i det egna objeket, är tillåtna. Referenserna förblir interna vid instansiering av objekt.

Namn

Ett attributnamn kan innehålla flera attribut led, separerade med punkt.
Om t ex objektet VWX-UGN-Ventil innehåller ett attribut av klassen Di, ÖppenGL, blir namnet för ActualValue för denna di:

```
VWX-UGN-Ventil.ÖppenGL.ActualValue
```

Antalet attribut segment kan var max 20.

wb_load

Wb_load parsern måste kunna tolka attribut namn med flera led.

Structfiler

Structfils genereringen görs för både c och c++.
C++ filen namnges .hpp och klassen pwr_Class_’klassnamn’.

Eftersom arv inte finns i c visas en superklass så som den ligger i databasen som ett element med namnet ’Super’ först i structen.

```
typedef  struct {
  pwr_sClass_MyClass    Super;
```

```
  pwr_tFloat32          Radie;
} pwr_sClass_AnotherClass;
```

I c++ definieras samma klass så här

```
class pwr_Class_AnotherClass : public pwr_Class_MyClass {
 public:
  pwr_tFloat32      Radie;
};
```

Funktionsobjekt

Om en klass ska innehålla kod som ska exekveras i plc't, gör ett funktionsobjektet som en separat klass. Denna funktionsobjekts klass innehåller ett template PlcPgm, och ett attribut 'PlcConnect' av typen AttrRef som innehåller kopplingen till huvud-objektet. Även huvudobjektet innehåller ett PlcConnect attribute som pekar på funktions-objektet.

I plc koden måste man kunna hämta och sätta värden på attribut i funktions-objektet och i huvud-objektet. Detta görs genom symboliska attributreferenser. I plc-koden hämtas resp sätts värden på in och utgångar med Get och Sto/Set/Res objekt som innehåller symboliska attributreferenser till $PlcHost. Vid kompileringen ersätts Objid i dessa attributreferenser med oid för funktions-objektet. På samma sätt hämtas resp sätts värden i huvud-objektet med symboliska attributreferenser till $PlcConnect.

Vid editeringen av plc-programmet i klasseditorn, kopplas Get/Sto/Set/Res objekt genom att attribut i template-objektet för funktions-objektet eller huvud-objeket väljs ut.

Profibus IO

Om signalerna i ett objekt hämtas från profibus görs en speciell ProfibusIO klass. Denna kopplas till huvud-objektet genom att huvudobjektet innehåller attributet IoConnect av typen AttrRef. Denna koppling sker med en IoConnect metod som finns i $AttrRef klassen. Kopplingen mellan signaler i huvud-objektet och kanaler i profibus-objektet görs i template huvud-objeket med symboliska attributreferenser ($IoConnect). Dessa symboliska referenser löses upp vid initieringen av io't i runtime (eller i connect-metoden).

AttrRef

I Flags, adderas bitarna Object, ObjectAttr och Array, som markerar att attrrefen pekar på ett helt objekt, ett attribut som är en klass, resp ett attribut som är en array.

Dessutom adderas biten Shadowed, som markerar att attributet skuggas av ett annat attribut i en subklass. Om Shadowed är true ska attrref'en presenteras med fullständigt namn där namnleden '.Super' inte döljs.

Objektsbilder

Objektsbilder skapas som förut med en ge-graph med samma namn som klassen, i vilken kopplingar till attribut i objektet görs genom att objektsnamnet ersätts med "$object".

Ge-editorn för en objektsbild startas från klasseditorn meden metod i $ClassDef objektet. Kopplingar till attribut i objeketet görs genom att man väljer ut attribut i template-objeket, och aktiverar connect-funktionen (ctrl/dubbelklick MB1) i ge-editorn. Namnet på template-objektet ersätts då med strängen "$object".

Grafiska symboler

Man ska kunna skapa subgrafer för klassen, där dynamiken är förprogrammerad och kopplad till olika attribut i klassen. Det ska räcka med att koppla ihop en subgraf med en instans av klassen.

Detta kräver att man kan lagra avancerad dynamik i en subgraf, med symboliska kopplingar. Eventuellt krävs också en annan typ av istansiering av subgrafer, om man vill kunna bygga subgrafer av andra subgrafer med för programmerad dynamik.

Metoder i utvecklingsmiljön

Meny metoder i utvecklingsmiljön ärvs från superklass till subklass och en metoder i subklassen överlagrar metoder i superklassen om namnet är lika.

Däremot ärvs inte create och adopt metoderna. Dessa anropas inte heller för superklasser eller för attributobjekt.

Metoderna för ett attributobjekt begränsas till menymetoder som är specifika för klassen eller superklasser till klassen (de metoder som finns i $object är inte tillämpliga på attributobjekt).

Metoder i Xtt

Meny metoder i xtt ärvs från superklass till subklass och en metoder i subklassen överlagrar metoder i superklassen om namnet är lika.

Menymetoderna opererar även på attributobjekt.

Eftersom man nu kan baka in t ex DsTrend objekt i klassen, och DefTrend attributet för diverse signaler i klassen ska peka på ett internt DsTrend objekt, bör man kunna göra denna koppling i klassdefinitionen's templateobjekt. Detta kan ske antingen med ett symboliskt namen ($Self) eller genom att referenserna görs inom templateobjektet och interna referenser behålls vid instansieringen.

IO hantering

Vid initieringen av io't måste alla signaler hittas, även de som ligger som attributobjekt. SigChanCon är av typen AttrRef och både signaler och kanaler kan vara attributobjekt. Symboliska namn av typen $IoConnect ska lösas upp.

Trace

Trace ska kunna lösa upp symboliska namn av typen $PlcHost och $PlcConnect.

Korsreferenser

Även signaler som ingår i andra objekt måste ingå i korsreferenserna.
Hur hittar men dem ?? GetClassList i urvecklingsmiljön ?

Klassvolymer i databas

Klassvolymer lagras idag som wb_load filer. Eftersom konfigureringen av ett projekt mer och mer kommer att förläggas till klassvolymerna skulle det kännas säkrare att kunna lagra dessa i en databas.

Idag fungerar klasseditorn så här.

- Vid uppstart läser man in wb_load filen för klassvolymen i en vrepwbl. Vid inläsningen beräknas offset och storlek på attribut, template objekt skapas. vrepwbl'en har sig själv som metavolym för att kunna tolka referenser till interna typer och klasser.
- Volymen importeras till en vrepmem i vilken editeringen sker. Editeringen styrs av speciella syntaxregler. Även vrepmem'en har sig själv som metavolym.
- Vid lagring, skrivs vrepmem'en ut på en wb_load fil. Vrepmem'en töms och den nyskapade wb_loadfilen läses in och importeras pss som vid uppstart.

Här skulle man kunna behålla vrepmem som editor men ersätta vrepwbl'en med en vrepdb. Vrepdb'n måste då göra allt det som vrepwbl'en gör idag: beräkna storlek och offset på attribut, skapa och konvertera template objekt, lösa upp referenser till interna typer och klasser.