Building robust persistent layer on .NET platform

WISZCZOR TOMÁŠ, ČERNOHORSKÝ JINDŘICH
Department of Measurement and Control
Faculty of Electric Engineering and Computer Science
VSB – Technical University of Ostrava
17.listopadu 15, 708 33 Ostrava-Poruba
CZECH REPUBLIC

Abstract: The article describes how metadata can be used for development of a robust basic IO like layer for a control system based on OPC DA. The article describes importance and basic features of metadata in the first part. In the second there is the practical example shows the creation of a persistent layer part. All code is written in C# language. Result layer simplifies working with OPC DA server. Items from OPC DA server can be mapped by persistent layer to objects in the code. The object "knows" how to send or load appropriate items to/from data fields or property fields. No other additional code is needed for creating a new object.

Key-Words: .NET, attribute, metadada, persistent, business object, OPC.

1 Introduction

In this article we very often talk about metadata. The metadata are data about data. They have got main utilization in database systems (DBS). DBS collects raw data. Storing data don't say about what they are, therefore DBS needs some metadata to express this information. For example metadata says which data express the name of person, the age of person and so on.

The metadata are important in programming languages as well. The classic example is COM (Component Object Model) technology. A COM component is a binary code in fact. The binary code can be called from various programming languages. The possibility is based on metadata. After COM component is written in a programming language then it can't be using in another programming language. Programmer must make description of component in description language for this reason. Ability of multilanguage calling is based on type library, which is written in IDL (Interface Definition Language). The IDL describes all components (classes, methods ...). Any language that wants to use a component must read information from the type library. Therefore the development tools or special functions of an operating system (OS) must understand IDL language. Most development instruments create special proxy object from the information collected from IDL about COM component.

The .NET framework provides advanced support for operation with metadata. CLR (Common Language Runtime), CLS (Common Language Specification) and CTL (Common Type System) allow that one class (written e.g. in Delphi) can be extended in C# language. This interaction in programming languages is unique and is based on using metadata, CLS and CTS as well.

Another important property is the possibility of extending the set of metadata within .NET platform. Programmer can define his own marks and after that he can work with these marks. This is very powerful feature of .NET framework. Web services use this principle for example [2], [3], [4]. Every class that provides methods marked as [WebMethod] (in C# language) can publish these methods on intranet/internet. Any client can call these methods after. .NET special marks – labeled as an attribute [1] – say how to handle them. However our attributes .NET does not know. Therefore the programmer has to develop some layer – environment – through witch our attributes will be processed and make available to .NET.

2 OPC persistent layer

The theory of attributes allows using effective attributes in data source – for example an OPC DA (Ole for Process Control) data source. An attribute establishes an association between element of code and any data source. A class can be connected to OPC DA server and properties or fields with a tag in OPC server by the same way. Thus, for example, a class can have two methods defined. One for loading tags to objects a second one for saving properties or fields to the OPC DA server. The scenario shows persistent objects, where all important data are in the OPC DA server.

Definition of new attribute is very easy and a principle of definition is in fig.1.

```
[AttributeUsage(AttributeTargets.Class)]
public class
     OPCServerAttribute:Attribute
{
  private string _opcsn;
  public OPCServerAttribute(string
     OPCServerName)
  {
       _opcsn=OPCServerName;
  }
  public string OPCServerName
  {
       get
       {
               return _opcsn;
       }
  }
}
```

Fig.1. Definition of a new attribute.

An attribute in .NET is a class de facto [1]. The class extends abstract class Attribute. Over the class programmer can preface usage of attribute. Fig.1. demonstrates the attribute that it can be used only by class. This using is defined by attribute AttributeUsage. .NET supports various elements of code. Definition of attribute for OPC DA’s tag is made the same way.

A second part consists in creation of a special layer for working with attributes defined by user. Any class that can use this principle must extend basic class of persistent layer. In the basic class programmer defines elaboration mechanism. If extended object calls refresh() or save() method, the persistent object parses object for our–defined attributes and makes appropriate actions according to what it finds. So persistent object works with metadata of object using so called reflex mechanism.

Fig.2. shows short code of save method in C#. It finds our attributes and post data to OPC DA server via OPC XML DA gateway.

```
public void Save()
{
  arItems.Clear();
  OPCServerAttribute osa =

t.GetCustomAttributes(typeof(OPCServerAttribute),
    false)[0] as OPCServerAttribute;
  if(osa!=null)
  {
    ro.ClientRequestHandle=osa.OPCServerName;
    foreach(FieldInfo fa in t.GetFields())
    {
      foreach(Attribute at in
            fa.GetCustomAttributes(
            typeof(OPCNameItemAttribute),false))
      {
        ItemValue itv = new ItemValue();
        itv.ItemName=((OPCNameItemAttribute)at).
                       OPCItemName;
        itv.Value=fa.GetValue(this);
        arItems.Add(itv);
      }
    }
    wril.Items =
  (ItemValue[])arItems.ToArray(typeof(ItemValue));
    rb=server.Write(ro,wril,true,out ril, out oes);


    if((oes!=null)&&(oes.Length>0))
    {
      string error_message="";
      for(int i = 0;i<oes.Length;i++)
      error_message=error_message+oes[i].Text+" ";
    }
    throw new Exception(error_message);
    }
  }
}
```

Fig.2. An example of Save() method.

The save() method from the persistent layer analysis a derived class. The analysis says what OPC DA server will be required and what fields will be associated with OPC DA server. All data are extracted from the class by reflex mechanism. The same principle is used in refresh() method.

A next source code (fig. 3.) shows definition of a new class that uses our persistent layer represented by OPCPersistendData class which implements refresh() and save() methods.

```
[OPCServer("KEPware.KEPServerEx.V4")]
public class OPCTest:OPCPersistenceData
{
  [OPCNameItem("Channel_1.Device_1.Bool_1")]
  public bool a1;
  [OPCNameItem("Channel_1.Device_1.Tag_1")]
  public short s1;
  [OPCNameItem("Channel_1.Device_1.Tag_2")]
  public short s2;
  [OPCNameItem("Channel_1.Device_1.Tag_3")]
  public short s3;
}
```

Fig.3. A new object is described by our attributes.

As parameter in constructor of attribute OPCServerAttribute programmer passes ProgID of OPC DA server and in attribute OPCNameItemAttribute (associate fields with tag in OPC server) a full name of tag (item).

This is advantage of the persistent layer. A programmer can create any class that it has got mapped any element of code to variable (tag) in OPC DA server. Fig. 3. demonstrated the solution. A public field a1 has got association with tag Channel_1.Device_1.Bool_1 within OPC DA server. If is called refresh() method then fields a1 gets an actual value and if is called save() method then value form a1 is stored in the OPC DA. It is very simple. The programmer need not write any code from read/write from/to OPC DA server.

3 Conclusion

The concept of metadata is useful not only for building development tools but also for creating robust framework not only for control systems. The programmer can establish associations with various data sources through attributes. A framework designed for new attributes processes these attributes by the reflex mechanism making appropriate actions accordingly.

The article describes fields/class association with OPC DA's tags. This scenario simplifies the using OPC DA servers. Only one piece of a code for read/write operations from/to OPC DA server is defined in one place. There is no need to write the new code for OPC DA partnership for another new class because older one can be reused (defined in persistent layer). Therefore every object must always extend the persistent object (as persistent layer). After each changing of work with OPC DA server it is necessary to make only a modification of the persistent layer – no modify the objects.

Persistent layer can provide a cache for tags. So refresh() and save() method need not communicate directly with OPC DA server for modification of tags in the server.

References:

[1] S. Robinson, *C# Programujeme profesionálně*, Computer Press, 2003.

[2] T. Wiszczor, J. Černohorský, S. Slíva, .NET technology in manufacturing and control, IWCIT'03 p.155-161, ISBN 83908409-7-9.

[3] T. Wiszczor, J. Černohorský, Mobile access to Xfactory from devices designed for OS PocketPC, PDS 2004, p.111–115, ISBN 83-908409-8-7.

[4] M. MacDonald, Microsoft .NET distributed applications: integrating XML web services and .NET Remoting, Microsoft Press, 2003.

Acknowledgement: The work and the contribution were supported by the project GAČR 102/05/0571 – Architectures of embedded system networks.