Component Based Rapid OPC Application Development Platform

Jouni Aro

Prosys PMS Ltd, Tekniikantie 21 C, FIN-02150 Espoo, Finland

Tel: +358 (0)9 2517 5401, Fax: +358 (0) 9 2517 5402, E-mail: jouni.aro@prosys.fi, http://www.prosys.fi

KEY WORDS: OPC, RAD, SCADA, data acquisition, framework

ABSTRACT

This paper describes a component based design for building OPC applications rapidly in a standard object oriented application development environment.

The described design makes up a component based platform that enables rapid development of industrial applications. In addition to OPC, the platform helps to deliver device and production data through other standard communication channels, such as ODBC, HTTP, SOAP, etc. It enables development of flexible, efficient and reliable applications that take full advantages of the OPC communication level, without the need to learn the communication standard or the common pitfalls in OPC communications. It enables the developer to concentrate on the application level, such as supervisory control and data acquisition (SCADA), production management systems (PMS), manufacturing execution systems (MES), asset management, product life cycle management, eService, etc.

The platform integrates into the industry leading rapid application development (RAD) tools, Borland Delphi and C++ Builder, which are used to build efficient and feature rich applications to Microsoft Windows and Linux operating systems, including Microsoft .NET. The platform connects and uses the best practices and components available for those environments, supporting rapid development of data acquisition, data analysis, full featured user interfaces, database access and web services, in addition to the OPC communications. It can be enhanced to enable OPC Server functionality, which will enable leveraging application data to upper level industrial systems, for example enterprise resource planning (ERP) and plant resource management (PRM) systems.

The platform is based on variable components which are used to model all data related to measurements, control, analysis etc. The variables use multicast signals to notify the so called connectors through variable links, which are used in other platform components to enable data transfer into different external systems in a generic way, including OPC Servers, SQL databases, user interface components, analysis components, web clients, external systems, etc.

In addition to the variables, which keep the current status and configuration up to date, the platform design includes components for historical data, alarms and events, reports and recipes.

1 INTRODUCTION

OPC /3/ has changed the world. It has made it possible to use device and equipment data in PC applications without the need to write vendor specific device drivers, etc. However, this is only the beginning: it will still require a lot of work before the applications are built, because software development is complicated and a typical OPC application needs to do a lot more than just communicate with an OPC Server.

It is well-known that estimating the amount of work required to develop new applications is very difficult /2/. Keys to faster and more reliable development are, among others, platforms and frameworks that consist of pre-tested components that work well together – as well as with other components and frameworks. To enable fast prototyping, the components should enable click and set configuration of the basic functionality. This may not restrict application development, however. Therefore, it is best to build the platforms into generic application development environments, such as Java, .NET, C++, Visual basic, Delphi, etc. Of course, some vendor specific environments (standard control centre applications) may enable the same power, but in practice, flexibility is limited to the capabilities built on that product.

Now that OPC communications are available to most field devices, the next step is to add more layers on top it. These layers will enable application development to focus on the actual application logic instead of learning to cope with the OPC communication details. Best approach is to create generic frameworks for industrial applications, where OPC plays just the minor role of field device communicator.

2 KEY REQUIREMENTS

OPC Application may be anything that uses OPC communications. In this context, it is restricted to an OPC Data Access (DA) Client application, which uses data from an OPC DA Server for process monitoring, device diagnostics, etc. /3/ In addition, the application may need to provide the information to higher level clients as an OPC DA Server, but most likely, it will use other communication standards to visualize, analyze, store and distribute the data. The requirements for most OPC Applications are specified in the following sections (See Figure 1).

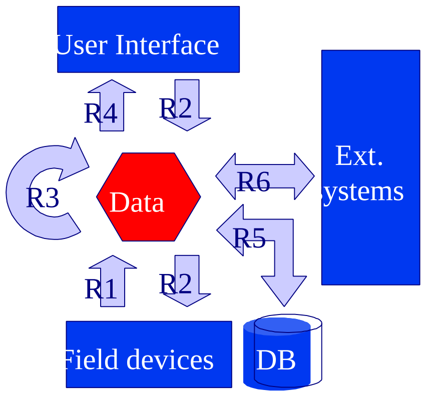

Figure 1. Target requirements for a typical OPC Application.

2.1 R1 - Data Acquisition

The application acquires data from any field device. The data is values, time stamps and qualities of the respective process measurements and other data items in the field device memory or sensor output.

The data acquired from field devices is typically very raw. It may require type conversions, scaling, etc. to get the data into human readable format: tag it with the correct names, engineering units and context.

2.2 R2 – Control

The application sends control signals to the field devices. The control data is normally setpoints and limit values corresponding to the production targets, so called supervisory control. Even though you cannot guarantee a certain response time, it may be possible to use a PC application to process control. However, it is normally better to use dedicated PLCs or other hardware for active control – and use the PC for higher level tasks.

2.3 R3 – Analysis

The PC application is a perfect location for basic as well as advanced data analysis. Analysis creates higher level information from the raw data acquired from the devices: signal comparisons, device status analysis, signal processing, classifications, etc.

2.4 R4 – Visualization

After all, the final target is to show the relevant data on screen for system operators, production managers, etc. The visualization may occur at the local screen or on a remote system or even in email. In this context, we concentrate on building screens that help to monitor the process online. In many applications, the main target is just to get the acquired data visualized.

2.5 R5 - Data Persistence

It is seldom enough to just show the current status: the trends and reports must be permanent, the events must be stored securely and we may also want to use databases to keep our production recipes, orders, etc. Therefore, it is a fundamental aspect of this kind of platform to enable usage of any databases for storage of data measured from production and for data used to control production.

2.6 R6 - Data Distribution

The final requirement for OPC Applications is to distribute the acquired or analyzed data to remote systems: external applications, remote monitors, email, etc. This may even be the main purpose of some applications, which function as plain data servers. Standard protocols for data distribution are all OPC standards, ODBC, HTTP, SOAP, etc.

2.7 Data

The data of an OPC Application typically includes: 1) current process status, including the value, quality and time stamp of variables (process measurements, equipment states, control signals) and their properties (limits, set points, values); 2) Historical trends; 3) Alarms and events; 4) Reports; 5) Recipes.

2.8 Static / Dynamic configuration

The applications may be configured statically and dynamically. Static configurations are used for example in control centers, which are configured to model the signals of the monitored production line. Dynamic configurations are needed for generic applications that adapt to different environments: for example, to monitor the status of all field devices installed in a factory. The requirements for dynamic configuration are, of course, much heavier than for a static configuration. Static configurations are much easier to test and assure to function in the fixed environment. Rapid development is not really possible or even desired in dynamic environments, except for throw-away prototyping /2/.

3 PLATFORM DESIGN

To fulfill the requirements of section 2, the following design is proposed (See Figure 2). The core of the applications are the Variable components (TPsVar) which are used to model all process variables. The VarLinks enable multicast notifications of changes and a generic read/write access to all variable data. All other functionality is implemented as Connectors, which synchronize variable data with external data sources, for example the OPC Connector copies data between the Variables and OPC Items.

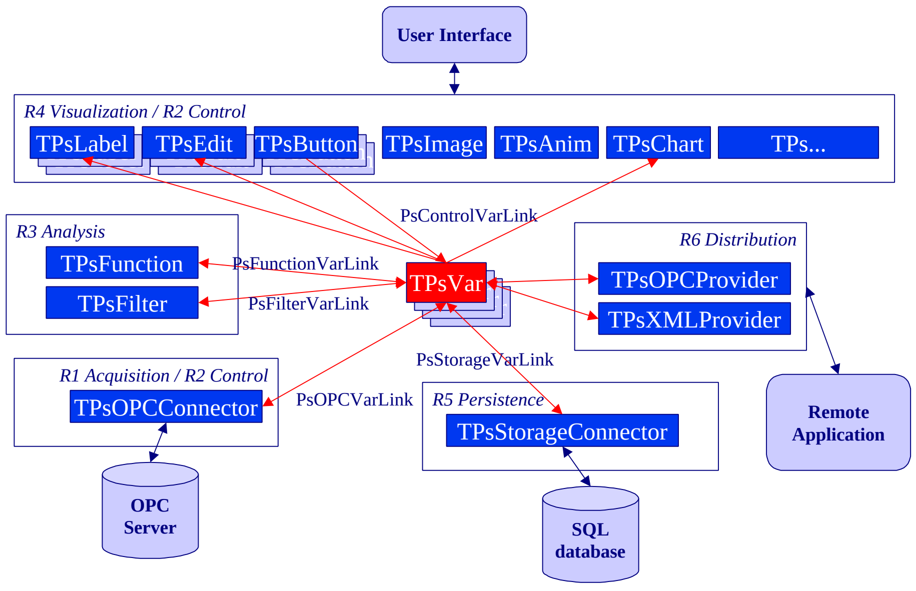

Figure 2. Platform design with components grouped according to requirements. *TPs* is used to denote the objects of the framework. The objects and components are included in Prosys Sentrol, which is a product of Prosys PMS Ltd.

This design follows common software design guidelines and principles, namely the Model-View-Controller pattern /1/. Its strength is also in the independence of the components: the Connectors depend on the Variables and VarLinks, but not on each other. The base classes which encapsulate common behavior also support scalability by enabling new specializations to be derived in a straight forward manner.

3.1 Variables

The variables are used for modeling the process signals: measurements, actuator positions, status bits, etc. There are different variable types for different signal data types: Boolean, float, string, state, array, etc. The complete class hierarchy is presented in Figure 3.

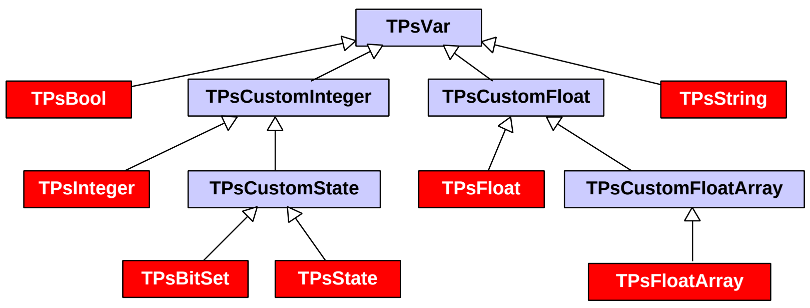

Figure 3. Variable class hierarchy. TPsVar and TPsCustomX types are generic base classes with common functionality shared by the descendant classes.

The variables include different properties, called VarProps. For example, for TPsFloat these are 1) the signal name and engineering units, 2) current value, 3) set point and alarm limits 4) status and 5) statistics (minimum, maximum, mean, standard deviation of value since last reset). Other variable types may have these and/or other VarProps as suitable.

For each VarProp the component keeps the value, quality and timestamp. Quality may be good, uncertain or bad. Timestamp corresponds to the last change time in UTC.

The VarProp values are accessible both as native data types and as variants.

3.2 VarLinks

Other components connect to the variables using VarLink objects. One VarLink attaches to a single VarProp of a single variable. For example, the user interface component TPsLabel simply listens to changes of a VarProp of a certain variable, and changes the contents of the label text to show the current value of the respective variable property. Buttons and Edit boxes also use the VarLink to change the value of the VarProp.

The VarLink enables shared functionality in all components that need to use the variable data either for reading or writing values to it. All property values are accessed as variant data, which enables data type independent functionality. This means that the components can use all property values similarly, whether the data is actually string, float, integer, Boolean or even array data.

In addition to the value, the link is used to access the quality and timestamp of the respective VarProp.

As you can see, all functionality in Figure 2, except for the variable itself, is implemented using VarLinks. The links are specialized for different purposes. For example, the OPCVarLink adds OPCItem specific properties to the generic VarLink.

3.3 Connectors

The Connectors are components that synchronize data between variables and other data sources, such as database tables, OPC Items, etc. They are actually collections of specialized VarLinks. Each VarLink defines the correspondence between a single VarProp and an external data item.

The Connectors simply listen to changes in the variables or in the external data – or both – and copy values respectively to keep them in sync.

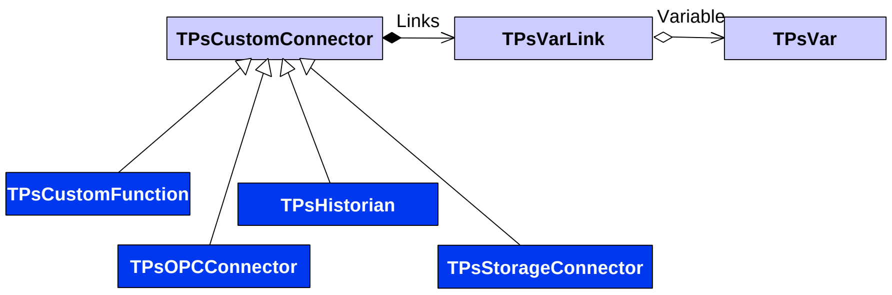

Figure 4. The connectors use VarLinks to configure correspondence between variable and external data. Most of the required functionality can be achieved with different connectors. Every Connector actually uses a specific VarLink, e.g TPsHistoryVarLink or TPsOPCVarLink.

A lot of functionality can be implemented as Connectors. For example, StorageConnector inserts or updates new rows to database whenever variable values change – or copies data from the database to the variables; function calculates an output whenever an input changes, etc.

3.4 OPC Connector

The OPC Connector is a Connector that is specialized to synchronizing the Variables with OPC Items. It is a collection of OPCVarLink objects. The OPCVarLink adds properties related to an OPC Item into VarLink. The OPC Connector itself functions as an OPC Group (Data Access 1.0 or 2.0x): it registers OPC Items corresponding to the links into the OPC Server and either reads the values synchronously using a timer or subscribes for asynchronous data changes from the server.

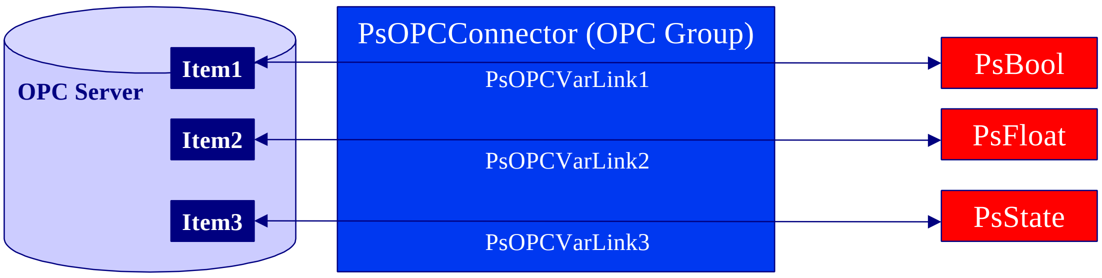

Figure 5. The OPC Connector synchronizes the variables with OPC Items.

The OPC Connector uses a transaction buffer, where it stores every data change from the server before copying the values to the variables. It keeps the quality and timestamp of every change, and the variables are updated using those, so in fact it won't matter, if there is a delay before the variables are updated. The queue may get filled up, if the server sends a number of data changes before the connector has time to copy them to the variables. The connector ensures, however, that the variables eventually get every data change – so that they can also notify other components, for example the Historian, of every change.

The OPC Connector also takes care of writing values to OPC Items when the respective Variables are requested to change. The next section (Concept of Execution) explains this in more detail.

In Figure 2, there is also an OPC Provider, which will provide the variable data to external applications as an OPC DA Server. The design of that will not be described here, however.

Concept of Execution

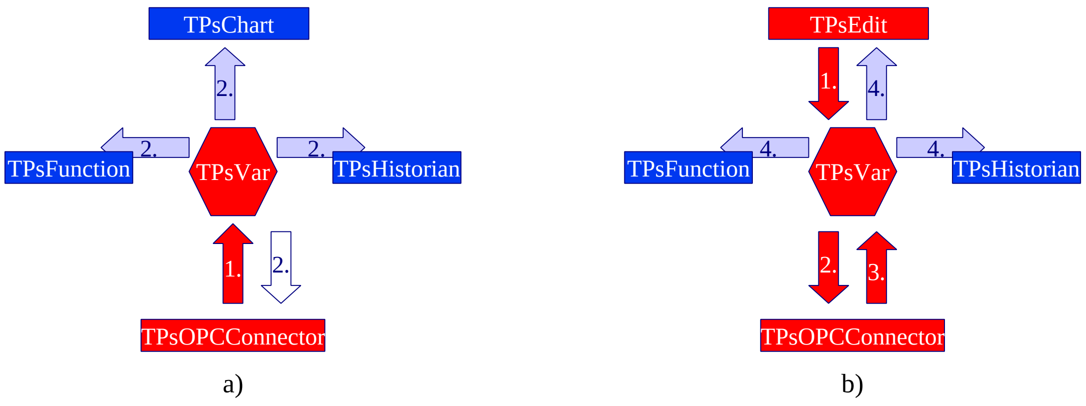

Figure 6. The data flow a) from the OPC Connector to other components, b) from the user interface to the OPC Connector. The OPC Connector is a data owner, i.e. it is responsible that the value, quality and timestamp of the Variables are equal to the actual values in the field devices.

The variable validates the values that are written to it and notifies other components of the changes. There are two aspects of this. You can rely on the validity of the data, when it is received from the field device. But, if a change is requested to the values from the user interface, for example, the new value must be written to the field device before it is valid in the application. The field device *owns* the data, because that is the real origin of the measurements and target of the control values. The variables should reflect the state of the field devices in every case. Of course, when the field device is not accessible, the quality of the respective variable must reflect this, too.

In the platform, the OPC Connector functions as a *data owner*, which is responsible for the validity of the value, timestamp and quality of each variable that it updates. Whenever the connector (Figure 6a. 1.) writes new values to the variables, these are used unconditionally, and (2.) all other components are notified by OnChange event through their VarLinks about the change.

On the other hand, (Figure 6b 1.) when a change is requested from the user interface, (2.) the OPC Connector, reacts to an OnChanging event from the variable by copying the value to the field device. (3.) After the value is written successfully, the connector updates the variable, and (4.) the variable notifies other components.

SCADA components

SCADA applications typically include trends, alarms, recipes and reports. These can all be implemented using the proposed design as Connectors. The Historian monitors changes in variable values and stores the values to database. Alarms and events are handled by the EventLog. Recipes are collections of values, set points or alarm limits that must be written to the variables. Reports are collections of values or statistics measured during a certain production interval, e.g. for a roll or batch.

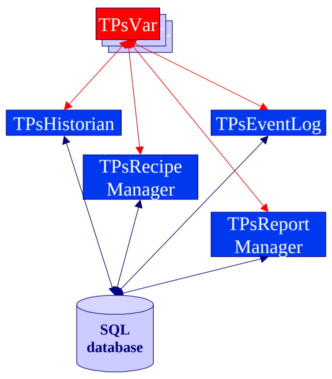

Figure 7. SCADA functionality is achieved with specialized connectors.

CASE STUDIES

The platform has been used in real world applications for several years. Two, quite different case designs demonstrate the applicability with different field devices and OPC Servers.

Metso ND800FF Client

The target was to build a data viewer for the Metso ND800FF intelligent valve controller for Foundation Fieldbus. The application is used as an interactive diagnostic data viewer connected to the Foundation Fieldbus through suitable OPC Servers. The valve controller has ca. 50 variables: Boolean, float and float array types.

The application logic was mostly built using the variable and OPC Connector components, and the retrieved data is visualized using the label and chart components that connect to the variables.

Application building was very straight forward and the developers did not need to concentrate on the actual OPC communications, except for learning how different OPC Servers actually support the standard. For example, in this context it was not possible to use synchronous and asynchronous communications at the same time, so only synchronous communications were used.

The application browses the OPC Server address space automatically to locate the Metso ND800FF devices and the respective FF function blocks. It then connects the variables to the OPC Items of a selected device and retrieves the values automatically from the OPC Connector. Each server requires custom coding for the automatic address space browsing, due to different styles of hierarchy and OPC Item ID formats.

FieldCare ä Condition Monitoring

FieldCare is a product of Metso and Endress+Hauser. It is a complete field device configuration tool, enabling plug-in applications that extend the functionality. FieldCare Condition Monitoring is a plug-in application that monitors the diagnostic data of the field devices of a complete factory online 24 hours 7 days a week.

The application uses HART Foundation OPC Server and Softing Profibus DPV1 OPC Server to retrieve diagnostic data asynchronously from any HART or Profibus PA device connected to the network.

The application includes an internal database, which is used for system configuration and historical device data. The application locates and identifies all devices available in the OPC Servers connected to the system and initializes the variable data model according to the network topology. The application currently enables up to 1000 field devices with ca. 50 variables each (Boolean and float datatypes) to be monitored online.

The device variables are used to determine the status of each field device and the current status is reported on a HTML web service. The variable trends are also stored in the database, and they are attached to the reports.

The application design was much more complicated than the ND800FF Client, due to the highly dynamic configuration requirements. The variables and OPC Connectors are used extensively and removed the need for OPC communication specific development. However, the different OPC Servers require special treatment in several aspects. In addition to the differences in the address spaces, the server behavior mostly varies in conventions regarding the qualities of the OPC Items, especially during communication failures.

In this context it would also be useful to be able to prioritize the devices and variables, but it is often very obscure how the server polling frequency can actually be affected. Changing the OPC Group Update Rate does not guarantee that the server actually polls the devices respectively.

Our experience is that it is important to get contact to the server developers, to be able to find out the actual server functionality and to get the details right.

CONCLUSION

Common requirements for typical OPC Applications were defined and a design proposal was presented. The design is based on software components that function as a framework. The design makes up a platform that enables new OPC Applications to be built rapidly and reliably using generic pre-tested components for the basic communication tasks.

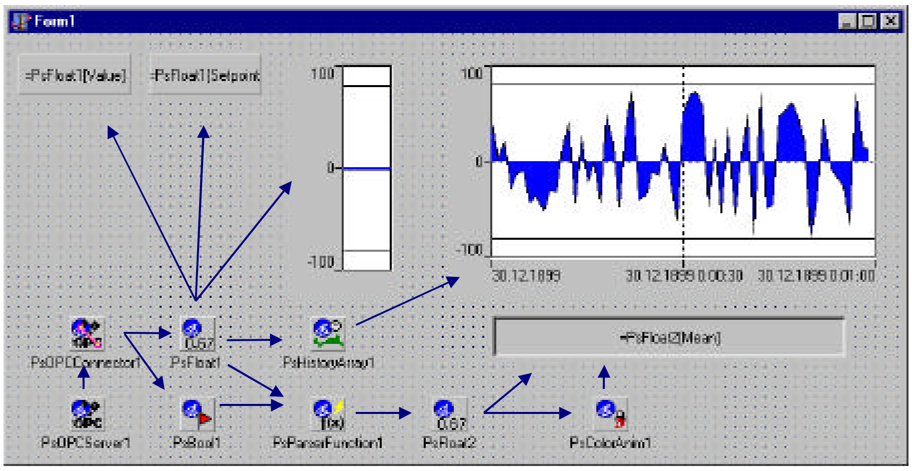

Figure 8. Prosys Sentrol integrates the components into Borland Delphi & C++ Builder. The sample application is built simply by connecting the components in the Borland Form Designer.

The components are available as a commercial product, Prosys Sentrol™ /4/, which is available for Borland Delphi™ and C++ Builder™ application development tools. Borland Delphi and C++ Builder have demonstrated a very powerful component based application development for Microsoft Windows for a decade and recently also for Linux and .NET. Prosys Sentrol integrates OPC seamlessly into the Borland development environment.

The platform has been used in real world applications for data acquisition, device diagnostics and production monitoring. The component set will be extended to enable even more flexibility and connectivity in future.

REFERENCES

/1/ Fowler M.: Patterns of Enterprise Application Architecture, Addison-Wesley, 2003, pp.330-332.

/2/ Steve McConnell: Rapid Development: Taming Wild Software Schedules, Microsoft Press, 1996, ISBN: 1556159005

/3/ OPC Foundation: OPC DA 2.05a Specification, 2001-12-17, http://www.opcfoundation.org/

/4/ Prosys PMS Ltd: Prosys Sentrol 2.0, http://www.prosys.fi/