The Benefits of Component object-based Supervisory System Application Development versus Traditional HMI Development

By Steven D. Garbrecht
Product Marketing Manager
White Paper Summary

There are several fundamental differences between Component object-based and traditional Tag-based Human Machine Interface (HMI) and Supervisory Control and Data Acquisition (SCADA) products. This paper explains the differences between the two methodologies and shows that savings of 90% in development can be achieved with a Component object-based SCADA product, and explains how to calculate the savings of one versus the other.

Introduction

Component based software architectures have been around for many years in the commercial computing world. Object-oriented development has also been in use for some time. Both are only now starting to make their way into the Process Control and SCADA software community. In fact, the only product available today from the major automation software vendors that combines both is the Industrial Application Server. The Industrial Application Server is part of the FactorySuite A² Product Line from the Wonderware® Division of Invensys PLC.

Definitions:

Object-Oriented Programming: Object-oriented programming (OOP) is a programming language model organized around "objects" rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data. The programming challenge was seen as how to write the logic, not how to define the data. In contrast, Object-oriented programming takes the view that what we really care about are the objects we want to manipulate rather than the logic required to manipulate them.

Component: In object-oriented programming and distributed object technology, a component is a reusable program building block that can be combined with other components in the same or other computers in a distributed network to form an application. Examples of a component include: a single button in a graphical user interface, a small interest calculator, an interface to a database manager. Components can be deployed on different servers in a network and can communicate with each other for needed services. A component runs within a context called a container. Examples of containers include pages on a Web site, Web browsers, and word processors.

Therefore Component Object-based refers to a system that uses an object-oriented development workflow to produce a componentized application that can be distributed across any number of computing resources. Because the linkage from component to component and from the development environment to the run-time environment is maintained, incremental changes can be made without having to shut down the entire system.
Tag-Based Versus Component Object-based Systems

Tag-based Supervisory Systems

From the inception of PC-based HMI and Supervisory products, the development of data access, scripting, alarming and data analysis has been based on the concept of tags. While simple and very portable from one project to another, a tag-based environment has the downfall of a flat namespace, with no inherent ability to link elements together into more intelligent structures with built in relationships and interdependencies. Global changes to a tag database are typically done externally to the development environment, as a text file or in tools like Microsoft Excel, and the re-imported into the application. Reuse in a tag-based system is commonly instituted through dynamic or client-server referencing, which allows a common graphic to be created, and then a script is executed to switch the tags being viewed in run-time. Furthermore, because of the flat structure of the application, changes need to be sought out and analyzed as to the effect on the rest of the application.

Object-Oriented Systems

Component-based and Object-oriented development in the Information Technology (IT) world originally referred to tools which aimed to release the developer from mundane, repetitive program tasks, while at the same time maximizing re-use through the development of common components.

As you would expect, these tools are not an exact fit for the industrial environment. For one thing, systems integrators and production engineers are typically not computer programmers. Furthermore, there are some key architectural differences between IT and production automation applications. For example, general information technology applications typically involve database access from non-deterministic, forms-based interfaces that accomplish things like on-line banking, business reporting, HR Management, financial accounting or static information look-up. Conversely, plant intelligence, production management or supervisory control applications involve the accessing of real-time data in PLCs, performing sophisticated calculations to determine flows and production numbers, and displaying this real-time data in graphics intensive client environments or analysis tools, and also writing to and reading from production and operational related databases. The two environments are different enough to dictate that Component object-based tools be purposely built for one setting or the other. Component object-based architecture used with Industrial Application Server is designed to help industrial customers with development, management and maintenance of larger, more complex supervisory systems.
The following table compares component object-based versus tag-based architecture.

<table>
<thead>
<tr>
<th></th>
<th>Component object-based architecture</th>
<th>Tag-based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Development</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Application Structure</strong></td>
<td>Hierarchical-</td>
<td>Flat-</td>
</tr>
<tr>
<td></td>
<td>Creation of components using object oriented workflow methodology</td>
<td>Monolithic instances of software, running on one to many machines as separate 'applications'</td>
</tr>
<tr>
<td><strong>Runtime</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Application Structure</strong></td>
<td>Hierarchical-</td>
<td>Flat-</td>
</tr>
<tr>
<td></td>
<td>Components representing physical devices, running coordinated in different locations when required</td>
<td>Monolithic instances of software, running on one to many machines as separate 'applications'</td>
</tr>
<tr>
<td><strong>Graphics Development</strong></td>
<td>Done last</td>
<td>Done first</td>
</tr>
<tr>
<td><strong>Scripting</strong></td>
<td>Developed in object templates as part of a component</td>
<td>Developed separately, linked to a graphical representation</td>
</tr>
<tr>
<td><strong>Promotion of standards</strong></td>
<td>Strictly enforced</td>
<td>Not strictly enforced</td>
</tr>
<tr>
<td><strong>Global Application Changes</strong></td>
<td>Propagated from object templates</td>
<td>Ability to distribute and exchange/enhance components as required</td>
</tr>
<tr>
<td></td>
<td>Ability to distribute and exchange/enhance components as required</td>
<td>Based on graphics or changed in tools like Excel</td>
</tr>
<tr>
<td><strong>Data Represented by</strong></td>
<td>Logical Constructs such as a physical device (i.e. Valve, pump) or logical device (i.e. PID Loop or Calculation) as Objects and components</td>
<td>Graphical Devices as Objects, Tags</td>
</tr>
</tbody>
</table>
Benefits for Manufacturing Applications

In Component object-based SCADA, application objects contain the aspects or parameters associated with the device they represent. For example, a valve object can contain all the events, alarms, security, communications and scripting associated with a device. Objects do not just represent plant equipment. They can also model common calculations, database access methods, Key Performance Indicators (KPIs), condition monitoring events, Enterprise Resource Planning (ERP) data transfer operations and many other tasks that you want the plant information system to do. Because these operations are modular, it is easy to add them to any and all parts of the application. For example, if there is a standard way your organization calculates and initiates a maintenance work order for a pump, you can encapsulate this function as an object and use it with any pump in the application.

Manufacturing applications typically have a number of common Components. These include common types of:

- Plant devices and equipment
- Operating procedures
- Process measurements
- Calculations
- Graphics displays

This allows a cookie cutter approach, where typically small software programs are developed as objects/code modules that can be stamped out and joined together to form an application. Most automation vendors have this capability today. Where a Component object-based SCADA System is different is that after the cookies are stamped out, you can change the stamp, and all of the cookies you already made are automatically changed.
1. Replication is the Creation of Components from Object Templates
2. Change Propagation is accomplished by changing the Object Template, which can cause any or all of the associated components to change.

This functionality is possible because when a SCADA package is truly Component object-based, it operates as a parent-child relationship, in which parent object templates are developed and then components are replicated or instantiated from the parent objects. Now all of the children are tied back to the parent, so a change in the parent can be replicated to all of the children. This is an extremely powerful development capability in that:

- Application creation is optimized by using object templates to automatically generate components (replication)
- Project changes are easily accommodated by making changes in the object template and having the components inherit the changes via change propagation
- Ongoing system changes and expansions are easier and more cost effective because of automated replication and change propagation

Component object-based doesn’t mean “Hard-to-Use”

Most of the functions in the ArchestrA Integrated Development Environment are drag-and-drop, click-to-select or fill-in-the-text box. The premise is to develop it once, and use it many times in the application. In most cases, Component object-based development is easier than modifying scripts line-by-line. In addition, the number of syntax and run-time errors is minimized because the tool enforces system specific rules.
Object-oriented graphics with SCADA

The phrase “Object-oriented” graphics has been used with SCADA since the early 1990’s. It is generally used today to refer to the ability to build graphics and draw pictures based on classes or a hierarchy. Object-oriented graphics allow you to build a symbol and replicate it across a screen or HMI application, and then have visual changes made to all the similar symbols at the same time. While this is useful functionality, SCADA applications require more than graphics. For example, much of the work that goes into a supervisory application is for things like:

1. Alarm Monitoring
2. Animation Scripts
3. Security Scripts
4. Supervisory Scripts
5. Historical data storage
6. Integration with other applications and SQL Databases
7. Event Detection
8. Flow and movement calculations
9. Device integration

In order to fully realize the benefit of a Component object-based architecture, a SCADA System today needs to depict all of these things, along with the graphics as objects.
Development: Object-based versus Tag-based Architecture

As with many new approaches to system development, using a Component object-based SCADA package is different when compared to tag-based SCADA applications. The following paragraphs explain some of these differences.

Tag-Based Systems

From the inception of PC based HMI and SCADA software, users have built operator graphics and linked them to tags, which represented addresses in a PLC or a control system. The concentration was on the computer and the software application. The steps below are an example of how a traditional tag-based SCADA application is developed.

1. A new HMI application is created on a single computer.
2. Windows or displays are created for the application.
3. Graphics are created for the windows.
4. Tag definitions are imported from the PLC or manually configured.
5. Alarm and Event Detection Scripts are defined for each tag.
6. Tags are linked to graphic elements.
7. Graphics animation scripts or links are created.
8. Input/Output (I/O) Tags are defined and linked to the application.
9. If the application is to be deployed in a client-server environment, the application is re-architected to centralize alarming, event detection, history archiving, graphics and IO servers.
10. Changes to the system require shutting down the application, making changes to the many scripts and tag database references to enable the new functionality, and reloading the new HMI application on each workstation.

Object-based Architecture

The Industrial Application Server and the ArchestrA Integrated Development Environment (IDE) have brought a new era to SCADA Software development through the ability to create a complete plant device model. The developer is removed from the complexities of the computing environment and allowed to concentrate on “modeling” how the production facility is laid out and the different manufacturing cells and processes that comprise plant-wide supervisory control. Once the plant model is captured, it is easy to implement supervisory control functions. A small investment in creating object templates yields big results in engineering productivity. The ten easy steps to creating a supervisory application using the Industrial Application Server are:
1. A site survey is conducted to understand the layout of the manufacturing operation or process.
2. A list is developed of similar pieces of equipment. Distinct areas of operation are also identified.
3. Object templates are configured for each common device or component in the facility. This process sets up the standards for the supervisory application and for any applications that are created in the future.
4. Device object templates can be contained within each other to build-up a more complicated device.
5. Device object templates have attributes which represent real I/O available in the PLC or control system. These attributes are then linked to the I/O through Device Integration Objects.
6. The application can then be assembled by using a simple drag and drop capability inside of the IDE.
7. Components are then assigned to security groups.
8. The model created in the IDE can now be deployed to the computers that will host the application.
9. Graphics are configured using InTouch®, the world’s most popular HMI software package.
10. Once the application is developed, maintenance of the system is easy. Changes made to Object Templates can be propagated to the Child Components linked to the Object Templates.
Life-Cycle Savings: Areas

The Life-Cycle Savings associated with any SCADA development tool can be categorized into four basic areas, as illustrated in the following table.

<table>
<thead>
<tr>
<th>Savings Area</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Development Savings Related to Application Generation</td>
<td>This represents the savings that result from the time saved when users develop applications by defining object templates once, and then generating components from those templates multiple times.</td>
</tr>
<tr>
<td>Initial Development Savings Related to Application Changes</td>
<td>This represents the development savings gained through the ability to propagate changes from Object Templates to all the components derived from those Templates. When multiple application changes are requested during development, the savings can really add up.</td>
</tr>
<tr>
<td>Maintenance Savings throughout the System’s Life-Cycle</td>
<td>Using a distributed system significantly reduces maintenance costs through the ability to remotely monitor, change and deploy software to all nodes in the network. This is especially important for geographically distributed networks because users can save both time and money by eliminating the need to travel to each site for maintenance or upgrades.</td>
</tr>
<tr>
<td>Savings Across All Sites</td>
<td>These savings result from reusing the Templates and applications created for this project on other projects. Companies use this to drive standards in their projects, which is particularly beneficial for systems integrators, OEM machine builders and facility operators.</td>
</tr>
</tbody>
</table>

Example:

Let’s take a simple example to show where the savings come from when developing SCADA systems using Component object-based technology. For example, let’s take a plant supervisory application that has, among other things, 27 Double Seat Valves, each having 6 I/O points of interest that will be monitored. These are I/O points in the PLC that define the operation of this valve. In a traditional SCADA System, 162 tags (27 valves times 6 I/O per valve) would be created. In Component object-based SCADA, a common valve object template is created and components that represent each individual valve are instantiated or replicated from that object template. Now let’s say that it takes 0.4 hours per tag to develop the application using a traditional, tag-based SCADA system. This is not including graphics or PLC control logic development. Let’s also estimate that it takes 2 hours to develop a valve object template and an additional 20% more (or 0.4 hours) per component instance to customize each individual valve in the application.
Device Example

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Number of Instances</th>
<th>I/O per Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double Seat Valve</td>
<td>27</td>
<td>6</td>
</tr>
</tbody>
</table>

Individual Estimations

<table>
<thead>
<tr>
<th></th>
<th>Tag-based</th>
<th>Component object-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tags Needed= 162</td>
<td>Templates Needed= 1</td>
<td>Development = 2 Hours per Object Template + 20% more per Component Instance</td>
</tr>
<tr>
<td>Development= 0.4 Hours per tag</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Remember that an Object Template encapsulates scripting, security, alarming, events, history configuration, and device communications. In a tag-based system, all of this needs to be programmed using additional memory tags. Now let's do the math.

Initial Development Effort

<table>
<thead>
<tr>
<th></th>
<th>Traditional, Tag-based HMI</th>
<th>Component object-based SCADA</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>162 Tags * 0.4 Hours per Tag = 64.8 Hours</td>
<td>(2 Hours per Object Template * 1 Template) + (27 Valve Instances * 0.4 hours per instance) = 12.8 Hours</td>
<td>52 Hours or 80%</td>
<td></td>
</tr>
</tbody>
</table>

This is an impressive savings, even if you estimate half of this number or 40%. Now suppose that a change is requested by the user that affects 10% of the application. In a Tag-based SCADA product, it can be assumed that 10% of the effort of the original development would be applied to make the change. With Component object-based SCADA, because of the parent-child relationship between objects and components, the 10% change effort only needs to be applied to the object template. Therefore, the additional savings can be calculated like this:

Application Change Effort

<table>
<thead>
<tr>
<th></th>
<th>Traditional, Tag-based HMI</th>
<th>Component object-based SCADA</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>64.8 Hours * 10% Change = 6.48 Hours</td>
<td>2 Hours per Object Template * 10% Change= 0.2 Hours</td>
<td>6.28 Hours or 96%</td>
<td></td>
</tr>
</tbody>
</table>
What to look for when selecting a SCADA system

Component object-based development provides a number of advantages in development and maintenance of the application, but what about the computing architecture that hosts the SCADA System? The technical aspects of the system must also be considered, including:

- Does the development tool provide a realistic model of plant equipment and manufacturing, areas, processes or production lines?
- Can it easily be integrated with network security and allow centralized security configuration?
- Does it provide flexible device connectivity and the ability to interface to all the devices in the plant?
- Does it provide centralized diagnostic utilities?
- Can the application scale from a single node to many nodes without having to re-architect the application?
- Can HMI applications be remotely deployed to computers across the network?
- Does it provide a unified namespace which allows tag browsing across the entire PLC network, both in run-time and in off-line development?
- Can computing loads be distributed across multiple computers if needed?
- Does the system provide cost effective redundancy using Commercial Off-the-Shelf (COTS) Hardware?
- Is the alarm subsystem distributed?
- Is historical archiving defined during HMI development or is a separate tool required?

All of these are things that a SCADA package should and can do today.

Wonderware Industrial Application Server and SCADA

Engineering productivity has often been elusive in the development of supervisory and SCADA applications. This is further compounded when multiple development groups are involved. The Industrial Application Server (IAS) helps immensely in this area, with a Component object-based approach to application development which allows device Object Templates to be configured once and re-used throughout an application. Also, a central repository for application development and an integrated development environment allows several people to modify and build an application simultaneously.

Funding required to change out your existing control system is hard to justify these days. In many cases production equipment and the control hardware that runs the process is as efficient as it will get. Changing needs that come to light are best addressed by using software located on networks on top of the control system. These include asset
Benefits of Component object-based Supervisory System Application Development

management, ERP integration, plant-wide optimization and production reporting applications. This is where the Industrial Application Server works great as an application integration platform, across other systems as well as across FactorySuite A² products. The Industrial Application Server provides a data integration capability and the communications security needed to tie all of your islands of automation together. With over 800 device interfaces, Wonderware can connect to almost any device in your plant.

System Lifecycle Savings: Planning for the future

The day a supervisory control system is commissioned, requests begin coming in for enhancements to its functionality. This is where a Component object-based system and centralized configuration database really helps. The Industrial Application Server allows for a standard device object template to be modified, and changes can be distributed globally to individual nodes and processes in the system. If a line or process expansion is needed, existing configuration and development work can be easily replicated and modified to fit the new need. All changes to the existing configuration are tracked and logged. Engineers can incorporate on-line help to any object Template and document the systems as it is being developed, so people who do modifications in the future can more easily change the application.

Calculate Your System Life-cycle Savings

To help customers understand how their specific supervisory control or SCADA project can benefit from the use of the Industrial Application Server and ArchestrA Technology, Wonderware has developed the FactorySuite A² Life-Cycle Savings Estimator. Please visit the following web site to calculate the savings for your specific project:

http://fsestimator.wonderware.com

References

The following documents provide additional information on how to implement a component object-based architecture using the Industrial Application Server. They can be located on the Wonderware Tech Support Site at http://www.wonderware.com/support/

- FSA² Deployment Guide
- ArchestrA IDE User’s Guide
Contact Wonderware or your local Distributor for information about software products for industrial automation.

Wonderware Corporation • 26561 Rancho Parkway South, Lake Forest, CA 92630
Tel: (949) 727-3200 • Fax: (949) 727-3270
www.wonderware.com

© 2004 Invensys Systems, Inc. All rights reserved. No part of the material protected by this copyright may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, broadcasting, or by any information storage and retrieval system, without permission in writing from Invensys Systems, Inc.

Invensys; Wonderware; ActiveFactory; ArchestrA; DT Analyst; FactorySuite; FactorySuite A2; InBatch; InControl; IndustrialSQL Server; InTouch; InTrack; QI Analyst; SCADAAlarm; SuiteLink; SuiteVoyager; WindowMaker; WindowViewer; WonderWorld; “Every system in your plant, working in concert.”; the Visualize, Analyze, Optimize symbols; and “Visualize, Analyze, Optimize” are trademarks or service marks of Invensys plc, its subsidiaries and affiliated companies. All other brands and product names may be the trademarks or service marks of their respective owners.