The Benefits of Component Object-Based SCADA and Supervisory System Application Development

By Steven D. Garbrecht, Marketing Program Manager for Infrastructure and Platforms

Table of Contents

1. Overview 3
2. Introduction 3
3. Definitions 3
4. Tag-Based vs. Component Object-Based Systems 4
   - 4.1. Tag-Based Systems 4
   - 4.2. Component Object-Based Systems 5
   - 4.3. Comparing the Two Systems 5
5. Benefits for Manufacturing Applications 6
6. Develop Once, Re-Use Many Times 8
7. Object-Oriented Graphics in the SCADA Environment 9
8. Development: Tag-Based vs. Object-Based Architectures 9
   - 8.1. Tag-Based Architecture 9
   - 8.2. Object-Based Architecture 10
9. Lifecycle Savings 11
10. What to Look for When Selecting a SCADA System 13
11. SCADA Systems Featuring the Industrial Application Server 14
12. System Lifecycle Savings and the Industrial Appplication Server 14
13. Calculate Your System Lifecycle Savings 14
14. References 15

1. Overview

There are several fundamental differences between SCADA and supervisory HMI (human-machine interface) products that are based on component-object application development and traditional tag-based development. This white paper differentiates the two methodologies and illustrates how developing SCADA and supervisory HMI products with component objects can result in an ***80% cost reduction*** over developing them using a traditional tag-based approach. It also explains how to calculate the savings.

2. 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 communities.

In fact, the only product available today from the major automation software vendors that combines both is the Industrial Application Server from Wonderware, a business unit of Invensys Systems, Inc.

3. Definitions

The following definitions will frequently appear throughout this white paper.

- **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 re-usable program building block that can be combined with other components in the same or other computers in a distributed network to form an application. Components can be deployed on different servers within a network and can communicate with each other to perform services. Examples include:
    - A single button in a graphical user interface,
    - A small interest calculator and

    - An interface to a database.

- **Container** – Components run within the context of a container. Examples of containers include:
    - Web servers,
    - Application servers and
    - Database servers.

- **Component Object-Based** – This term 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.

- **Replication** – This term refers to the creation of components from object templates.

- **Change Propagation** – In object-oriented programming, this term refers to the process of changing an object template once and then selectively distributing that change throughout an application, multiple applications, or an entire system.

4. Tag-Based vs. Component Object-Based Systems

4.1. Tag-Based 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. Although this approach can be simple and portable from one project to another, a tag-based environment uses a flat namespace, which is a shortcoming because individual elements cannot be linked or organized 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 then re-imported into the application. Re-use in a tag-based system is commonly instituted through dynamic or client-server referencing. The system creates a common graphic containing scripts that switch the tags in runtime. Because the application structure is flat, the user must then change each tag in the system and analyze how the change affects the rest of the application.

4.2. Component Object-Based Systems

The concept of component-based or object-oriented development originated in the information technology (IT) world. Its goal was to provide tools that would 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, IT applications typically involve accessing databases from non-deterministic, forms-based interfaces that accomplish things like online banking, business reporting, HR management, financial accounting or static information look-ups.

Conversely, plant intelligence, production management or supervisory control applications involve acquiring real-time data in PLCs; performing sophisticated calculations to determine flows and production numbers; displaying real-time data in graphics-intensive client environments or analysis tools; and/or 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. The component object-based approach offered by the Wonderware Industrial Application Server's underlying ArchestrA software architecture is designed for industrial customers who develop, manage and maintain supervisory systems.

4.3 Comparing the Two Systems

The following table contrasts component object-based and tag-based architectures.

	Component Object-Based Architecture		**Tag-Based Architecture**	
	Development	**Runtime**	**Development**	**Runtime**
Application Structure	Hierarchical – Components created using object-oriented workflow methodology	Hierarchical – Components represent physical devices and can coordinate with components in different computers	Hierarchical – Graphical content sometimes created using object orientation	Flat – Monolithic instances of software run on one/multiple machines as separate "applications"
Graphics Development	Performed last	N/A	Performed first	N/A
Scripting	Developed in	N/A	Developed	N/A

	Component Object-Based Architecture		Tag-Based Architecture	
	object templates as part of a component		separately, linked to a graphical representation	
Standards	Strictly enforced	N/A	Not strictly enforced	N/A
Global Application Changes	Propagated from object templates	Components can be distributed, exchanged or enhanced	Based on graphics or changed using tools like Excel	Require recompilation of the application
How Data Is Represented	Logical constructs such as physical devices (e.g., valves or pumps) or logical devices (e.g., PID loops or calculations) are represented as objects and components	N/A	Graphical devices are represented as objects or tags	N/A

5. Benefits for Manufacturing Applications

In component object-based SCADA applications, application objects contain 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 don't just represent plant equipment. They can also model common calculations, database access methods, key performance indicators (KPIs), condition-monitoring events, ERP data-transfer operations and a variety of other production and performance management tasks.

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.

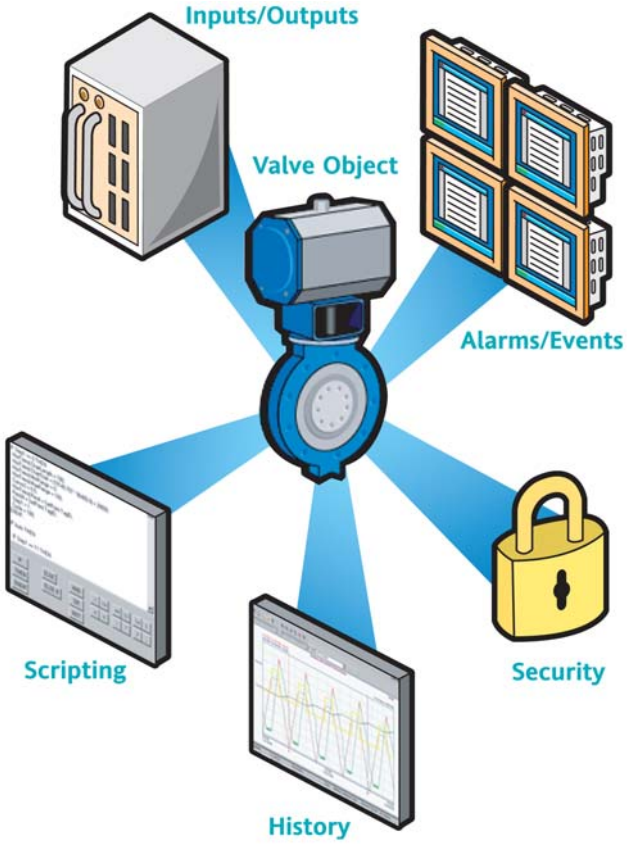

Fig. 1: An object template contains valuable information about Alarms/Events, Security, History, Scripting and Inputs/Outputs.

Manufacturing applications typically have a number of common components including common types of:

- Plant devices and equipment;
- Operating procedures;
- Process measurements;
- Calculations; and
- Graphical displays.

Component object-based development facilitates a cookie-cutter approach, in which small software programs can be developed as objects/code modules, stamped out and joined together to form an application. Some 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.

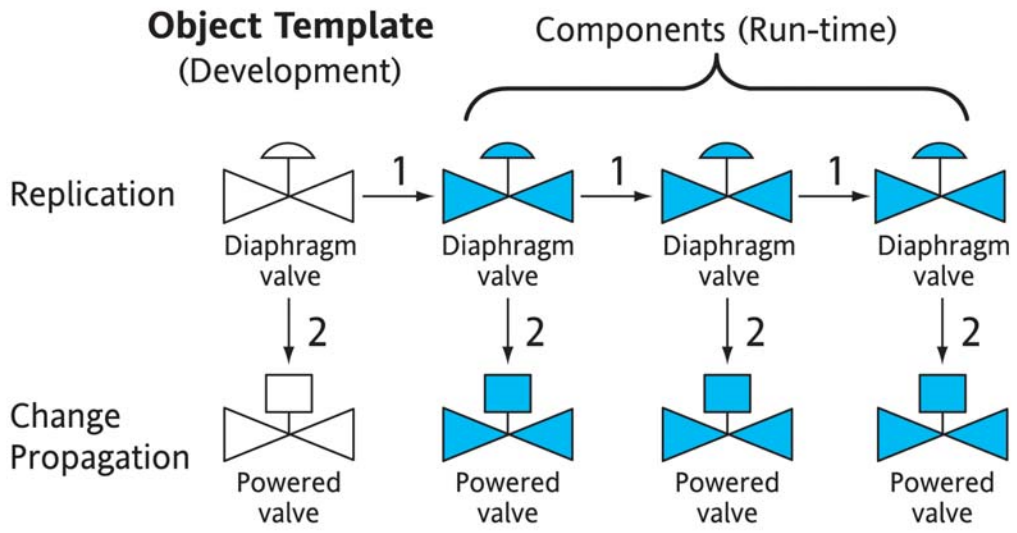

Fig. 2: Replication of Objects and Change Propagation

The first row of Fig. 2 shows the replication of an object representing a diaphragm valve. Replication is the process during which components are created from object templates. The next row illustrates a change (manual to powered) being propagated throughout all the valves. Change propagation is accomplished by changing the Object Template, which can cause any or all of the associated components to change.

Replication and change propagation are possible because, when a SCADA package is truly component object-based, it operates within a parent-child relationship. The "parent" object templates are developed first. Then components are replicated or instantiated from the parent objects. These "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

6. Develop Once, Re-Use Many Times

The Industrial Application Server's object-oriented approach to development is easier than development involving heavy programming. The software's ArchestrA Integrated Development Environment (IDE) enables the use of simple drag-and-drop, click-to-select

or fill-in-the-text box techniques. In most cases, this approach is easier than modifying scripts line-by-line. In addition, the number of syntax and run-time errors is minimized because the IDE enforces system-specific rules. What's more, users can develop objects once and then re-use them many times in the current application as well as future projects.

7. Object-Oriented Graphics in the SCADA Environment

The term "object-oriented graphics" has been used in the SCADA environment since the early 1990s. Today, this term usually refers to the ability to build graphics and draw pictures based on classes or a hierarchy. Object-oriented graphics enable users to build a symbol and replicate it across a screen or HMI application. They can then edit the symbol and easily distribute the changes to all the similar symbols at the same time.

While this is useful functionality, SCADA applications require more than graphics. Much of the development work that goes into designing supervisory applications is spent on creating items or functionality such as:

- Alarm monitoring;
- Animation scripts;
- Security scripts;
- Supervisory scripts;
- Historical data storage;
- Integration with other applications and SQL databases;
- Event detection;
- Flow and movement calculations; and
- Device integration.

In order to fully realize the benefits of a component object-based architecture, a SCADA system today needs to depict all of these things, along with the graphics as objects.

8. Development: Tag-Based vs. Object-Based Architectures

8.1 Tag-Based Architecture

From the inception of PC-based HMI and SCADA software, users have built operator graphics and linked them to tags that represented addresses in a PLC or a control system. The concentration was on the computer and the software application.

The steps below represent the typical development of a traditional tag-based SCADA application:

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. Graphical animation scripts or links are created.
8. 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 I/O 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.

8.2 Object-Based Architecture

The Industrial Application Server and its underlying ArchestrA 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 empowered to concentrate on modeling how the production facility is laid out. The developer can focus on the different manufacturing cells and processes that comprise plant-wide supervisory control.

After 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 of similar pieces of equipment is created. 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 create 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 (DI 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 Wonderware's InTouch HMI software.
10. Once the application is developed, system maintenance is easy. Changes made to object templates can be propagated to the child components.

9. Lifecycle Savings

The lifecycle savings associated with any SCADA development tool can be categorized into four basic areas, as illustrated in the following table.

Savings Area	Explanation
Initial Development Savings Related to Application Generation	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.
Initial Development Savings Related to Application Changes	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.
Maintenance Savings throughout the System's Lifecycle	Using a distributed system significantly reduces maintenance costs through the ability to remotely monitor, change and deploy software to all HMI 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.
Savings across All Sites	These savings result from re-using the templates and applications created for this project on other projects. Companies use this to drive standards in their projects. This is particularly beneficial for system integrators, value-added resellers (VARs), original equipment manufacturers (OEMs), machine builders and facility operators.

Here's a simple example to show where the savings come from when developing SCADA systems using component object-based technology. 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 * 6 I/O per valve) would be created. In component object-based SCADA systems, 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

Device Type	Number of Instances	I/O per Instance
Double seat valve	27	6

Individual Estimations

Tag-Based	Component Object-Based
Tags needed = 162	Templates needed = 1
Development = 0.4 hrs. per tag	Development = 2 hrs. per Object Template + 20% more per Component Instance

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

Traditional, Tag-Based HMI	Component Object-Based SCADA	Savings
162 tags * 0.4 hrs. per tag = 64.8 hrs.	(2 hrs. * 1 Object Template) + (27 Valve Instances * 0.4 hrs per instance) = 12.8 Hours	52 hrs. or 80%

This is impressive savings -- even if you estimate half of this number, or 40%!

Now suppose that a customer requests a change that affects 10% of the application. Using tag-based SCADA software, it's reasonable to assume that 10% of the effort spent on the original development would be applied to make the change.

Alternatively, using component object-based SCADA software such as the Wonderware Industrial Application Server, the 10% change effort only needs to be applied to the object template -- because of the parent-child relationship between objects and components.

In this situation, the additional savings can be calculated like this:

Application Change Effort

Traditional, Tag-based HMI	Component Object-Based SCADA	Savings
64.8 hrs. * 10% change = 6.48 hrs.	2 hrs. per Object Template * 10% change = 0.2 hrs.	6.28 hrs. or 96%

10. What to Look for When Selecting a SCADA System

Component object-based development provides a number of advantages in the development and maintenance of applications, 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 and production lines?
- Can it easily be integrated with network security and enable 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 re-architecting the application?
- Can HMI applications be remotely deployed to computers across the network?
- Does the development tool provide a unified namespace that facilitates tag browsing across the entire PLC network, both in run-time and in off-line development?
- Can computing loads be distributed across multiple computers?
- Does the system provide cost-effective redundancy using commercial off-the-shelf (COTS) hardware?
- Is the alarm sub-system distributed?
- Is historical archiving defined during HMI development or is a separate tool required?

A modern SCADA system should be able to do all of the above.

11. SCADA Systems Featuring the Industrial Application Server

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 Wonderware Industrial Application Server helps immensely in this area, with a component object-based approach to application development that facilitates the configuration of device object templates that can be re-used throughout an application. Also, a central repository for application development and the ArchestrA IDE enable several people to modify and build an application simultaneously.

These days, it's hard to justify the funding required to change out an existing control system. 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. This applies to asset management, ERP integration, plant-wide optimization and production-reporting applications.

The Industrial Application Server works very well as an application integration platform, across other systems as well as across other Wonderware products. It provides the data integration and communications security needed to transform islands of automation into one powerful and robust system. Because Wonderware offers more than 800 device interfaces, the Industrial Application Server can connect to virtually any device in a SCADA environment.

12. System Lifecycle Savings and the Industrial Application Server

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 help. The Wonderware Industrial Application Server facilitates modifications to standard device-object templates, 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 online help to any object template and document the systems as they are developed, making it significantly easier to update applications.

13. Calculate Your System Lifecycle Savings

Wonderware offers a tool to help people estimate their potential system lifecycle savings when using the Wonderware Industrial Application Server and ArchestrA technology for a specific supervisory control or SCADA project. Please visit the following website to view it: http://fsestimator.wonderware.com.

14. References

The following documents provide additional information on how to implement a component object-based architecture using the Wonderware Industrial Application Server and ArchestrA technology.

- Wonderware's FactorySuite A^2 Deployment Guide
- ArchestrA IDE User's Guide

These documents are both available on the Wonderware Technical Support website at www.wonderware.com/support/.

Contact Wonderware or your local Wonderware Distributor for more information on industrial automation and information software solutions.

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

© 2002, 2004, 2006 by Invensys Systems, Inc. All rights reserved. No part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording or otherwise), or for any purpose, without the express written permission of Invensys Systems, Inc.

Invensys, Wonderware, ArchestrA, InTouch, FactorySuite and FactorySuite A2 are trademarks 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.

Part No. 15-0162; 3/06