Using Object-Oriented Technologies to Solve Substation Automation Dilemma

Jim Y Cai, *Member, IEEE,* and H. Lee Smith, *Life Senior Member, IEEE*

***Abstract*-- Traditional Substation automation/integration systems involve huge numbers of man-hours to commission and maintain, even for similar systems. Using object-oriented technologies can reduced this commissioning time by around 80%. This has been demonstrated by field experience. The real-time information processing objects are composed of interaction-GUI parameters, real-time data, and functional processing. The defined objects – classes can be re-used among all the projects. Commissioning a new project is just to create instances of classes by "drag and drop". No more individual point/tag definition or verification is needed. These steps were done once when class was created. Changes made to class will change all its instances automatically. The classes can be inherited from, referred to, and embedded into others classes. Simple naming convention makes data usage instances link to data source instances automatically.**

***Index Terms*—Substation integration and automation, Object-oriented methods, Real-time systems,**

I. Introduction

THE traditional real time processing system is built on point/tag concepts, each point/tag represents a basic real-time information item in the system. Based on this concept, first step of building a real time application – substation integration system or a control center SCADA master, is definition and creation of the real time database. Most of us are familiar with this task of defining each point/tag of each I/O type of very IED/RTU and the virtual/calculated points, then creating interactive part GUI to database– graphics with correct real time data linkage. In addition to using some special tools to build the calculation/control logic definitions. After all the creation, the major task is still ahead. To verify all the point/tag definitions one by one are correct and have the correct dynamic linking to the graphics. The only relief we could receive from these database definition procedure is to use some kind of spreadsheet or text file editing with import capability to avoid typing the similar text again and again. For graphics creation, some basics component can be grouped together as template to save some time.

An object-oriented approach might be the right way to go. As we all know there are no completely identical projects in the world. But in most cases, we do have similar projects with similar elements. For example, in substation automation/integration applications, each power company will want all the breakers with the same voltage level to behave in exactly the same manner: same alarming logic, same graphical representation, and same operator prompt to trip/close. So conceptually, we want to treat this breaker as a basic object or brick.

II. Object Structure for real-time information processing

We may get really confused and frustrated if we try to exactly define an object-oriented system/software package. As real time information processing application engineers, what we look for is re-use or duplication. These basic tested objects can be re-used in the same project, in different projects individually without testing being required in new projects.

This paper presents an objected-oriented real time information processing solution implemented on Linux that is already in operation in the field.

A. Generic Object Structure

Let's first clarify some terms we use. The term "object" is a generic term as a concept. "Object" in this paper or our solution does not specifically have any strict definition. We use the term "object" to describe the implemented approach. Each object has three basic elements, data, method (functional processing), and interaction to the GUI (graphic user interface). We use "class" to represent an object or say strict definition of objects. The term "class" has same definition as in C++ language. Or you can say, class is a template.

The term "Instance" is the utilization of a class. The instances are composed of partially real time information/database. So "Object" here refers to class and instance.

As we mentioned above, the object is composed of three parts: real time data, functional processing, and user interaction.

The real-time data are values and definitions of analog, digital, etc. By "drag and drop", you create an instance of the

J. Y. Cai is with Doubletree Systems, Inc. – a XJ group company (e-mail: jimcai@dsius.com).

H. L. Smith is with HLS Consultant Services, USA (e-mail: HLSConsultant@aol.com).

object, which is identical to creation of traditional real-time database. Then the values and attributes of analogs, digitals

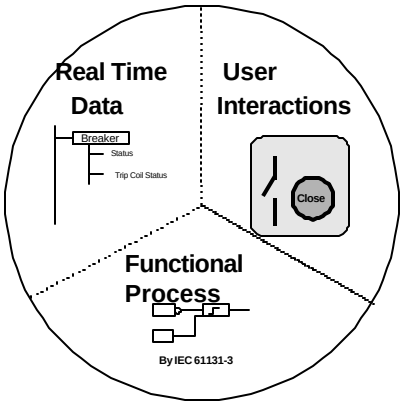

Figure 1. Object Structure

will be updated accordingly.

Fig. 1. Object Structure for Real time Information Processing

B. Real-time Data

A class could have hundreds of instances in a project. In some instances some data could be shared with all instances or whole system. Some data specific in each instance can't be shared among other instances. We call the shared data server data. The non-shared data is called client data. In the class builder tool, there are two panels for each class: one is titled "server data" and other is titled "client data".

Both server and client resource members can be all kinds of data types including elementary data types like integer, class type, reference types, and alarm type.

Server data has access authorization levels to make data variables private, friendly, and global. Addressing format for the variable is xxx.yyy. . .vvv. The first xxx is the instance name of this object, which must be unique system-wide. The last vvv is variable name. Between two are the names of class or referred instance. So we can address a breaker status in a substation by Sub1.Bay1.Breaker.status1. All the characters in the names can be in different language character sets.

C. User Interaction - GUI

User interaction defines how the object is displayed on the screen including static background. The dynamic update rate and how the object reacts to operator action by clicking or pressing some key are also defined.

D. Functional Processing with IEC 61131-3

Functional processing may be:

Inter-locking logic for a breaker operation,
Alarming for breaker status,
Simulation logic for testing and/or training purpose,
Special calculation,
Or algorithms.

Functional processing can be created by following IEC61131-3 ladder logic language and/or by using the system proprietary scripting language. The functional processing can be executed continuously or on demand. This feature is also called soft PLC (Programmable logic controller) function and the system will guarantee that all the continuous logic execution finishes in a timely manner.

III. IMPLEMENTATION

Real time information processing always consists of real time data acquisition, and real time data presentation. The real time data acquisition provides interface to field devices – IEDs. While the real time data presentation provides user interaction interfaces and displays. Objects that provide interfaces to IED are data source objects. Data usage objects are objects that provide user interaction interfaces. An example is used to explore the idea in the following section. There is one 220KV bay in a substation, which has a breaker and switches with one multi-function protective relay IED. There are two data source objects for this bay, one for the bay controller IED and one for the protective relay. And there is one data usage object - the bay object, which may be derived from the breaker and switch class. The object defines the graphics display, the pop-up windows, interlocking, alarming, etc.

A. Data Source Objects – IED definitions

Data Source Objects contains basically three functions:

Protocol interpretation;
IED interface's control, settings, and diagnostics;
Data point definitions.

Protocol interpretation is a special processing function of the object to interrupt the communication message by following the defined architecture. For each IED interface, GUI is definitely needed to control, monitor and diagnose the interface such as start/stop polling, reset the channel, etc. The point list will define variables associated with all the points as well as data types. Obviously, those data are server data so the other objects can access them accordingly.

For the above protective relay IED, a class is created, which will share the same protocol interpretation function with the other classes. The protective relay classes could have their own GUIs (like using their IED picture/LEDs as its own background). The IED point list will include, for instances, OC_Activated, Breaker_Trip, Breaker_Status, Breaker_Trip_Coil, GroundingSwitch_Status. All the points can be imported via a text file, which could be generated by IED configuration tool. Definition file in the future IEC61850-6 substation configuration language could be used. When an instance BAY1_IED is created from the protective relay class, then I/O points can be addressed as

BAY1_IED.OC_Activated,
BAY1_IED.Breaker_Trip,
BAY1_IED.Breaker_Status,

BAY1_IED.Breaker_Trip_Coil,
BAY1_IED.GroundingSwitch_Status.

Classes for different I/O configurations of IEDs with same protocol can be easily created by only editing the I/O definition text file.

And also all the engineering scaling, ranges can be defined in the class. When data source class is created in house, full testing and documentation can be done in house. The class encapsulates all the information needed. Then the class can be used across projects with peace of mind.

B. Data Usage Objects – One-line Diagram

A substation bay on a one-line diagram consists of breakers and switches. The basic requirement is that the dynamic text and graphics should be updated automatically. It should have appropriate pop-up/prompts for the operator's interaction, alarming when analog values go out of limits and digital status change state. The object definition should also include event logging associated with the bay, bay level interlocking which monitors and controls operates breakers and switch operations.

To create this bay class, the class variables are defined as:
$Instance_IED.OC_Activated,
$nstance_IED.Breaker_Trip,
$Instance_IED.Breaker_Status,
$InstanceIED.Breaker_Trip_Coil,
$Instance _IED.GroundingSwitch_Status.

Here "$Instance" is a substitution command. When an instance gets created with this class, they will be replaced with instance name. Please note that the class variable names are the same as in the IED class. By using those variables, the dynamic update, interlocking logic can be implemented. By using the class builder tool, graphics as well as all the pop-ups/prompt cab be created as you wish. Using the built-in function block classes compatible with IEC61131-3, interlocking logic is easily implemented. During this creating process, the source of the variables is ignored

Then by drag and drop, the bay instance by name is created with name Bay1. The system will automatically link the bay variables to the IED variables.

C. Objects Linking – Naming Convention in Class Building

Still we need naming convention to accomplish automatic linking between the data source and the data usage objects. Variable name of both data source and data usage classes must be the same so all the instances created with the classes can be automatically linked.

D. Object Reuse

Reuse is the method for reducing the commissioning effort. Object reuse is accomplished through:

Classes and instances,
Inheritance-type embedded classes,
Instance referencing,
Mirroring instances.

Class and instances is basic reuse architecture. Each class can contain fully tested and field proven data definitions, function processing and GUIs, documentation as well as class version information. The class is a fully encapsulated element. When class is being copied among projects, it keeps all the information and relationships intact. In most cases, a modification to the class will automatically apply to all the instances.

The Inheritance-type embedded class feature makes the class more powerful and easy to manage. For example, bay class can be built up from switch and breaker classes. When you create an instance of a bay class, the switch and breaker instances will be created automatically.

Instance referencing is a linkage to other instances and a mechanism of accessing objects.

The mirroring function makes sure that the same instance can be viewed at several places or on multiple screens without creating multiple instances.

E. Project Implementation Procedure

Object modeling analysis is the first step of the project implementation. The analysis can be at the industry/application level to define generic objects for substation integration/automation. The analysis can be at the project level to define generic objects for specific projects or customers. For substation integration/automation application, the following classes identified:

Data usage objects: single pole and three pole switches, single pole and three pole breakers, bay classes, bus classes, and transformer classes. Note only one switch class is needed for single pole switches to cover switches for all voltage levels, no matter what color and position orientation,

Advanced application classes (automatic transfer, load shedding, voltage/reactive control, etc.),

Data source objects: Protective relay IED classes, bay controller classes and meter IED classes.

As substation integration/automation system providers, several class libraries have been built:

1. IED class libraries to cover most commonly used IEDs like XJ's transmission line protection devices,
    a. WH801, transformer protection
    b. WBH801, bus protection WMH801,
    c. GE UR's T60, D60, L60
    d. SEL-421, etc.

    These IED classes can be used almost in all projects without any customization.

    2.One-line display class libraries. Usually, each customer has their unique requirements for the GUI and processing functions. Therefore a library must be maintained for specific user/utilities along with the baseline;

3. Advanced applications libraries. In most cases, these classes can be re-used without any customization.

Again, all the classes can be built, and fully tested in-house with supporting documentation. For a specific project, the classes from the class libraries can be re-used to create, test and document the project. If customization is needed, the class build tool is used to provide a built-in version control feature to track all the modifications.

F. Operating Experiences

As an example consider a 220KV substation with two 220KV lines, ten 110KV feeders, twenty 10KV feeders, two three-winding transformers. About 50 classes used. About 30 of classes are IED classes, 15 one-line diagram classes, and several advanced application classes. Approximately 100 turnkey substation integration/automation systems are delivered worldwide each year. Approximately 80% of the traditional commissioning time has been saved using the object definition approach.

IV. ACKNOWLEDGMENT

The authors gratefully acknowledge the contributions of Mr. Yonghua Zhou, Yonghua Liu, and Yong Wei of XJ Group Corporation for sharing their work.

V. BIOGRAPHIES

Jim Y Cai (M'1997) graduated from the Shanghai Jiao-Tong University, China and received B.Sc. and M.Sc in 1983 and 1985 respectively. His employment experience includes the China Computer Systems Engineering, Open Systems Control, Doubletree Systems, Inc. His special fields of interest include power system automation and control. Jim Y Cai is also member of IEEE PSRC and Technical Expert IEC TG57 Working Group 3.

H. Lee Smith, PE(LSM)BSEE, MSEE, PE in PA, Member Substation Committee, Data Acquisition, Processing Control Subcommittee, Technical Expert IEC TG57 Working Group 10, US Coordinator TC57 WG 10, 11, 12. Adjunct Professor Penn State University. Author over 40 technical papers and article including IEEE/PES Tutorial on Automation Systems, chapter on Remote Terminal Units. For the past twenty years, he was employed by two different automation suppliers in an executive staff position. Presently self- employed as Principal Consultant with HLS Consultant Services.