Developing a Common Language for IED Communication in the Substation

Developing a Common Language for Intelligent Electronic Device Communication in the Substation

Dr. W. Premerlani, R. Mitchell, Dr. I. Ali, Dr. T. Saulnier
GE Corporate Research & Development
Schenectady, NY

M. Adamiak
GE Protection & Control
Malvern, PA

J. Melcher
EPRI
Palo Alto, CA

Introduction

From the introduction of the Intelligent Electronic Device (IED), we have had the ability to communicate with and extract information from these devices. The ability to intelligently communicate gives added value to the IED and as such, has hastened their implementation. As time has gone on, we have watched communities of these devices sprout up in the substation - typically with no concerted attempt to inter-communicate much less interoperate. As a result, a veritable "Tower of Babble" has arisen inside the substation. Attempts to date to achieve some semblance of common communication have focused around "Rosetta Stone" solutions whereby a translation module of software is located between the IED and host computer. Although this technique achieves today's goals, translation hardware is usually required and creation of the translation module can be costly and time consuming. Another aspect is that revisions and generations of new IED's have become a frequent occurrence demanding constant "stone cutting" and "chipping" of translation communication software. Newer IED designs implement faster communication rates, have more data to communicate, and are capable of performing some programmable logic functions.

In view of future capabilities and a continuing proliferation of IED's in the substation, a cry has come from the utility community to create a framework for not only common communication but an architecture that will provide for interoperation. Interoperation implies the ability to "plug and play" and also to be able to "share" data and functions. As an example, a protective relay may be required to provide a "check synchronism" function which requires the magnitude and phase angle comparison of two voltages. The relay performing the function may have intrinsic access to only one voltage. The other voltage may be available from another device in the substation . An interoperable system could then negotiate for access to the other voltage and as such, avoid all the overhead involved in direct wiring.

Integration Benefits

As substation integration becomes a reality, there are numerous benefits that can be realized. With data sharing, wiring between devices can now be minimized. Distributed data acquisition now becomes the foundation of substation integration. Traditional hardware devices such as the Remote Terminal Unit and the Digital Fault Recorder now become primarily functional entities that draw on other IED's for their data. Interoperation permits distributed functionality, that is, the data and/or the decisions needed for a particular function may reside among multiple IEDs.

Such changes in the substation design paradigm can be measured both quantitatively as well as qualitatively. Quantitatively speaking, substation integration has the potential for the following savings [1]:

- Elimination of the station fault recorder & wiring
- Elimination of the station Sequence of Events recorder & wiring
- Minimization of RTU wiring

- Minimization of Breaker Wiring

Qualitatively, the integrated system brings with it:

- Reduced O&M through "Real Time" condition monitoring
- 100% redundancy in fault recording
- Rapid fault location
- Integrated Protection & Control

and many others.

Requirements Document Creation

Through the funding of EPRI and in conjunction with numerous IEEE Working Groups and the MMS Forum, work has begun on a top down design to define the requirements for an integrated Protection, Control, and Data Acquisition communication system. The requirements document (open to the public for review and comment) is intended to be the foundation of an open protocol definition that will focus on peer to peer communication in the substation and is expected to have extensions to other areas of power system communication.

The software architecture is based on the International Standards Organization (ISO) seven layer Open System Interconnect (OSI) model for communication protocols [2]. This model breaks a protocol down into 7 independent functional entities that can be linked together (depending on the functional requirements) to create a protocol definition. Some of the layers of interest are as follows.

The bottom layer is know as the "Physical" layer and defines how one connects into the system. For example, do I connect a fiber optic cable or a pair of copper wires. The second layer is the Data Link Layer which defines how the data is packaged. The third layer in known as the network layer which defines, in a multi-path environment, how a piece of data gets from device A to device Z.

The top or seventh layer, and the focus of of this paper, is known as the Application layer. It is at this level at which the user or user's program interfaces with the communication protocol and ultimately other IED's. It is also at this level where the greatest challenge lies as the development of a common language is required here in order to interoperate among the various IED's. The challenge here can be equated to communication through the spoken language. As a simple model, the basic building blocks of most languages are nouns and verbs. Combinations of these nouns and verbs express requests, issue commands, and exchange information. The reason we can communicate together is because we have all learned the same nouns and verbs in the same language and can turn to a dictionary to describe the words we do not understand.

The requirements definition process mandated some technique whereby the various information items and functions of the IED's could be described and where that description could be shared by all. A software design technique known as Object Oriented Methodology (OOM) has been adopted as the fundamental tool to be used in the overall design process. This paper presents a basic description of the concept of Object Modeling and details aspect of existing protocols that make use of various aspects ot this tool.

II What is an Object-Oriented Methodology? (OOM)

An object-oriented methodology is a software system development process that organizes software as a collection of discrete objects that incorporate both data structure and behavior. The term object-oriented is used to describe both the process and its output. A software system that combines data structure with behavior, that is what the object does, is said to be object-oriented. Object-oriented methodologies improve the quality of software systems, and have been used commercially to produce several types of object-oriented products, including languages, libraries, database management systems, graphical interfaces, and simulators, for example, as well as a large number of applications in a wide range of problem domains. An object-oriented methodology is also well suited to communications in general, and communications between intelligent electrical devices (IED's), in particular. Experience has shown that there are several benefits to an object-oriented methodology:

- Reduced life cycle cost. Clear documentation, improved system design, and software reuse reduce the overall system life cycle costs.

- Traceability. Software development is a seamless process. The models derived from analysis of customer requirements are carried forward and permeate subsequent development steps.

- Better design. The system is clearly represented in terms of real-world entities and behavior. Application experts and system developers have a common basis for discussion.

- Better selection. Models are even useful for purchased software. Models can aid understanding of vendor software and enrich the basis for selecting the best product.

- Coherence. Careful modeling improves the quality of the thought processes that are embodied in a design.

- Communication. Models promote communication between developers and customers. Models bring important names and application concepts to the fore so that they can be defined and understood by all parties.

- Extensibility. Software organized about an object-oriented theme parallels the real world and is flexible with respect to changes in requirements. This is in contrast with a procedural approach.

- One uniform paradigm. The notion of an object uniformly applies to programming code and database code, analysis and design.

- Improved quality of data. Rigorous modeling improves the quality of the data. Many constraints can be woven into the fabric of a model.

An object is a concept, abstraction, or thing with crisp boundaries and meanings for the problem at hand [1]. Important characteristics of objects include identity, classification, polymorphism, and inheritance. Identity means that objects can be distinguished from one another. Switches, relays, and transformers are examples of objects. Two identical switches can be distinguished from each other, for example, so that a request to open one of them is not intended to open the other one. Objects have attribute values and may be related to other objects. For example, a switch could be in the open position and be controllable by a certain relay. Objects also have behavior,
specified by the operations that may be applied to them. For example, the "close" operation may be applied to a switch. By classification we mean that objects that have common attributes and behavior are grouped together into
a class. Thus all switches from the same manufacturer with the same model number are in the same class. Polymorphism is a term used in object-oriented methodologies that means the same operation can be applied to different classes of objects with appropriate results. For example, the "close" operation is polymorphic on the classes switch, recloser, and breaker because each can respond to a request to "close" in a perfectly appropriate way. Inheritance is the sharing of attributes and behavior based on a hierarchy. A class can be broadly defined and then refined into superclasses. Inheritance is used to describe a taxonomy of classes. For example, there are many kinds of relays, and we could use inheritance relationships to describe their taxonomy.

Other important object-oriented concepts include abstraction, encapsulation, combining data and behavior, and reuse through inheritance. By abstraction we mean focusing on the essential aspects of objects. Abstraction is one of the most powerful features of object-orientedmethodologies. Through abstraction it is possible to focus on concepts rather than implementation. The trap that is easy to fall into is losing sight of the essentials by confusing design with implementation. By rushing intoimplementation it is easy to become overwhelmed by details before understanding essential concepts. The abstraction process in object-oriented methodologies avoids this trap by focusing on concepts before implementation via a concise, implementation independent representation of system structure and behavior.

Encapsulation, which is also called information hiding, means that the internal details of an object are hidden from the external interface to the object. Only those details that are needed to interact with an object are published. Operations on an object provide a clean separation between internal and external views of the object. The external view of an object is what services the operations perform. The internal view of an object is how the services are implemented. The behavior of the "close" operation is visible in the external view of a switch object. The details of how this is accomplished is hidden from the external view.

Object-oriented software also combines data and behavior into a single hierarchy. In nonobject-oriented approaches there is a separate hierarchy of data structure and procedures. By combining data with behavior, the responsibility for the details of an operation is shifted from the procedure to the object. For example, in an object-oriented application it is not necessary for a print procedure to deal with the details of each object that can be printed. Instead, each object knows how to print itself. This simplifies maintenance, because it is not necessary to update procedural code for printing when a new class is added. Instead, if the new class needs to support the print operation, then the print operation is implemented for it.

This leads to the concept of reuse through inheritance. It is often possible to define a new class as a specialization of an existing class, and share the implementation of common operations. For example, if we find that there is a taxonomy of switches, it may be possible to implement the "close" operation on the superclass.

The abstraction power of object-oriented models provides a neutral representation that could be implemented in a variety of ways, including programs, databases, grammars, and communications protocols, for example. We have also found it useful to construct object-oriented models of concepts as an aid to understanding.

The concepts used in constructing object oriented models have natural interpretations in a number of domains of discourse. Yet, a model itself is simpler and more

fundamental than its realization in a particular implementation, because implementations often require extra detail. There is usually more than one way to implement a particular concept. For example, the concept of inheritance is implemented in different ways in object-oriented programming languages, relational database management systems, and communications. In object-oriented languages there is direct support for inheritance by declaring a class to be a subclass of another class. In relational database systems, there is no direct support for inheritance, and there are several ways for mapping an inheritance hierarchy onto database tables. An object model can also be interpreted as representing a grammar. In that case, inheritance can be interpreted as alternate production rules. However, in most communications protocols, there is no direct support for inheritance. In some protocols, such as MMS, there are constructs that can be used to implement inheritance.

In addition to an object model, some object-oriented methodologies also include a dynamic model and a functional model. The object model describes the static structure of the objects in a system. The dynamic model specifies the control aspects of a system. Finally, the functional model describes the structure of computations. Of the three models, the object model is most relevant to system communications, and has a natural interpretation in that context. Important constructs in an object model include objects, classes, attributes, methods, associations, and generalizations. In this paper, we will use the OMT notation, one of the more popular notations for object models [3]. Another notation that is also being used to model IED's is Coad's notation [4].

An object is a concept, abstraction, or thing with crisp boundaries and meaning for applications. Each object exists and can be distinguished from other objects. A particular circuit breaker is an example of an object. Another example of an object is a particular real time data point. Objects can be real objects or software objects. In some cases there is a correspondence between the two. For example, a circuit breaker is a real object controlled by an IED. Inside of an object-oriented IED there would probably be a software object that implements the control logic for the circuit breaker and which communicates with other IED's. Both kinds of objects are valid elements in an object model, depending on the purpose of the object model. Objects can also be classified as passive objects or active objects. An active object can perform a service. A circuit breaker is an example of an active object. A passive object, such as a measurement, conveys information.

A class is an abstraction of a group of objects with similar properties, common behavior, similar relationships to other objects, and common semantics. We say that an object is an instance of a class. As we shall see, an object can be an instance of more than one class through an inheritance relationship between classes. Thus, all measurements are instances of the class, Measurement.

Classes may have attributes and methods. An attribute is a named property of a class that describes a value held by each object of the class. For example, the class Tag could have the attributes date, type, and description. A method is a service provided by objects in a class. For example, trip is a service provided by the class CircuitBreaker.

One notation [3] for representing classes in object models is shown in Figure 1. A class is drawn as a rectangle

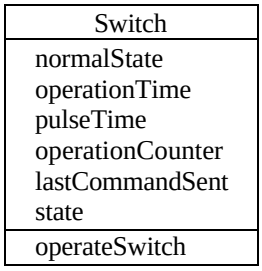

Figure 1 Example of a class

with three sections. The name of the class is placed in the top section, the attributes in the middle section, and the methods in the lower section.

Another object-oriented concept is that of a link, which is a physical or conceptual connection between objects. In a programming language, links are sometimes implemented with pointers, but links should not be confused with pointers. Links are a logical abstraction of pointers, and may be implemented in a variety of ways, particularly when you consider that they can be interpreted in the context of databases and communications as well as programming language. A link between objects indicates that those objects are related in a meaningful way.

An association is an abstraction of a group of links with common semantics, just like a class is an abstraction of a group of objects. One representation of an association between the classes Company and Person is shown in Figure 2.

The association in Figure 2 shows several features of associations using the OMT notation, including association names, role names, and an indication of multiplicity. In the example, the association name, indicated by a name by the middle of the association, is "Works-for". This indicates that a link between a particular

company and person indicates that the person works for the company. A role name is a name on an end of an association, indicating the role that the class on that end plays with respect to the association. A role is an end of an association. In the example, Company plays the role of employer, and Person plays the role of employee. Association names and/or role names are needed when the semantics of an association is not absolutely clear from the associated classes. We show both an association name and role names in the example for illustration. It is not necessary to use both.

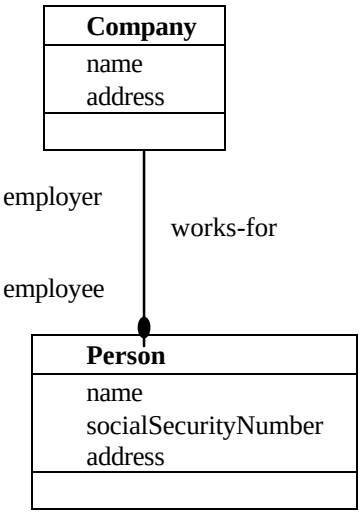

Figure 2 Example of an association

In this example, the association is between two classes, which we call a binary association, and which is commonly found in practice. Associations between more than two classes may also occur, but are less commonly found.

Also shown in Figure 2 is one representation for indicating multiplicity. By multiplicity, we mean the permissible number of objects on one end of an association that can be linked to a single instance of an object on the other end. In this example we are assuming that every person has exactly one employer and that a company can have any number of employees. The straight line at the employer role of the association indicates exactly one. The solid ball at the employee role of the association indicates any number of employees. Not shown in this example, a hollow ball Indicates an optional link.

Aggregation is a special form of an association which indicates a assembly-part relationship in which an instance of the assembly contains an instance or instances of the part. We use a diamond on the assembly end of the association [1]. Thus, in Figure 3, a tap changer is an optional part of a transformer.

Generalization is a relationship between a class, called the superclass, and one or more variations of the class, called the subclasses. The superclass holds common attributes, operations, and associations. Each subclass adds its own unique attributes, operations, and associations. A subclass inherits the attributes, operations, and associations of its superclass. An example is shown in Figure 4, using the OMT notation. Generalization is indicated with a triangle with the peak pointing towards the superclass. A number of phases are usually carried out in an iterative, seamless fashion in applying an object-oriented methodology. The phases in OMT are conceptualization, analysis, design, and implementation [1]. During conceptualization, models are prepared of the problem domain. These models can be shared among several applications. Analysis refines the model to represent the application requirements. During design, strategies are developed for mapping the models to an implementation.

III How is OOM Applied to Communicating IED's?

An object-oriented methodology can be applied to the entire software development process to foster interoperability. The result is not just a communications standard, but a system architecture as well. The first step in the process is to simply apply an object-oriented methodology to construct standard object models of IED's, without focusing on the details of communications. These models standardize the external interfaces and behavior of IED's and will need to be validated by a standardization process. Because of their abstract nature, these models are easy to grasp at a glance, and provide a unified representation to serve both programming and communications. A portion of an object model at this level of abstraction is shown in Figure 5.

The next steps add layers of detail, with different kinds of detail added for programs, databases, and communications. In the case of communications, we recommend an abstract communications model for an intermediate level of abstraction that could still span a variety of communications protocols. The model should support object definitions, including attributes as well as methods. For example, an abstract syntax notation could be used to indicate communications design decisions. A mapping of the object model in Figure 5 to this intermediate level is shown in Figure 6. Finally, at the lowest level of representation, the abstract syntax notation is mapped to a particular communications protocol, such as MMS, for example.

An object-oriented methodology can be applied in an integrated software development process which

simultaneously addresses programming, databases, and communications. A single, high level representation helps the developer conquer complexity. Encapsulation reduces complexity by hiding design and implementation details. Layers of detail are added through a unified strategy for mapping between models at different levels of abstraction. We will now look at the interpretation of the constructs of an object model in the context of communications.

There are several interpretations for objects and classes. The usual interpretation is that an object is a software object inside an IED. An IED may contain many objects. Each object is an instance of a class in the object model for that type of IED. The attributes of the software object are the same as those of the class. Each software object provides the services corresponding to the methods of its class. A message with arguments may be sent to an object requesting a particular service. In addition, messages can be used for direct access to an attribute, depending on whether or not the attribute is private. A private attribute can be accessed only through its services.

Another interpretation for some objects and classes is that they are real. Both real and software classes can appear in a single object model. Real classes are often added to a model to provide a context for understanding the model. For example, classes representing real objects such as physical transformers or circuit breakers may appear in an object model, and may participate in associations to software classes. In many cases there is both a real class and a corresponding software class. Thus, there could be both a class for a physical circuit breakers as well as for the software objects for controlling them.

We also distinguish between passive and active classes. An active class is one with methods for providing services. A passive class as attributes only. Passive objects may be required in some cases as the arguments passed with a service request or may be returned by a service. In the context of communications, polymorphism is interpreted as an abstraction of services. Thus, although the details of opening a disconnect switch are different from that of tripping a circuit breaker, at a high level of abstraction we can view the two services as being similar. Inheritance relationships are interpreted as meaning each subclass inherits the attributes, methods, and associations of all of its ancestors in the inheritance hierarchy.

In the context of communications, an association between two classes indicates that it is possible to create, delete, and traverse links between objects in the associated classes. For example, in Figure 7 the class Controller is associated with the class VirtualDevice in a one-to-many association. This means it should be possible to provide the following services:

- Given a controller object, find all associated virtual devices.
- Given a virtual device, find its associated controller.
- Link a virtual device to a controller.
- Unlink a virtual device from a controller.

We will say a few words about implementation of associations, because implementations are often confused with associations. The problem is that there are many ways to implement a given association, too many to explain here. For example, in addition to the commonly used approach of mapping an association as an embedded pointer, embedded object, collection of pointers, or collection of objects, in one or both ends of the association, it is also possible to map an association to an object itself, with appropriate services. Association implementation strategies should be selected in mapping from an object model to an intermediate level of abstraction. Avoid placing implementation decisions in the object model because it introduces unnecessary detail, defeating the goal of abstraction.

Aggregation is another construct in an object model that has meaning in the context of communications. In addition to the properties an services of an association relationship, aggregation indicates a part-assembly relationship. In the case of passive objects, aggregation is commonly implemented by including part objects in an assembly object whenever the assembly object is passes as an argument or returned from a service. The difference between associations and aggregations is that, properly used, aggregations form a directed graph with no cycles, while there are no restrictions on the use of associations. For communications, the multiplicity of aggregation is never many-to-many.

In addition to classes that are directly in the application problem domain, two other types of classes may appear in an object model: generic classes and metaclasses. A generic class is one that can contain more than one kind of data. A metaclass is one that describes another class. For example, in Figure 8 the classes RealTimeDataPoint, Real, State, and Discrete are generic classes. These classes could hold just about any kind of data. Although generic classes are quite useful, we should point out that, by themselves, generic classes do not specify a standard, and must be accompanied by a companion standard to achieve interoperability. For example, in order for an application to interact with RealTimeDataPoint in a meaningful way, it must have access to an enumeration of the valid values for the attribute PointName, and must know what to do about each different value. Metaclasses, which are classes that

describe other classes, are useful in constructing systems that are self descriptive. For example, instances of metaclasses could be used describe the object model of an IED, and make it available in electronic form as an online service. Object models containing metaclasses are called metamodels. A portion of a metamodel for describing object models is shown in Figure 9.

So far, our discussion has been focused on new IED's. Legacy systems and devices must also be considered, because it may be desired to install a new IED in a legacy system or legacy IED in a new system. It is not likely that legacy devices will comply with the new standards. Many legacy devices treat all objects as passive and do not provide object-oriented services. We expect that in either case a gateway will be needed for protocol translation. A related problem is that of reusing existing IED designs. We do not expect manufacturers to port all of their designs to a new standard in an overnight process. Mapping existing designs to object models is a reverse engineering process. We expect that it will be applied gradually over time to convert existing designs to work in an object-oriented architecture.

"VERB" IMPLEMENTATIONS

1. BACnet Application Layer

The application layer specification for BACnet (Building Automation and Control network) contains specifications for modelling the communication aspects of devices and also application functions for transmitting and manipulating this information over the communication channel. Though BACnet attempts to use an object-oriented methodology for specifying the application layer, the specification for the attributed of the data elements found in a device - device model- and the specification of the methods - application services - which can be applied on these data elements are specified independently.

BACnet has attempted to insulate the specification of the device model and application service from its implementation in terms of the bits and byte representation on the network. It uses the ASN.1 abstract syntax notation and BER for encoding messages. The encoding scheme is not purely ASN.1. Explicit encoding is described for encoding the header information for the application layer.

We will briefly describe the description for the Device Model and then of the application layer services in BACnet. Even though BACnet specification is not object-oriented, we have used OMTool notation to organize the BACnet application layer specification in an object-oriented manner. OMTool provides a good graphical representation of the information organized using object-oriented methodology.

2. Device Modelling Approach

The approach taken by BACnet towards modelling devices is to represent devices as a collection of objects. BACnet uses an object oriented methodology for the application layer. BACnet provides specifications of classes and mechanisms for capturing the relationships (or associations) between these classes. The objects in a device is an instance of a class. This enables one to model the communication specific features of most physical devices like fan, air-conditioners etc found in the building automation environment.

We now consider the object-classes specified in BACnet for the example presented in the previous section. For greater clarity and ease of understanding, for the objects we have only included here the required attributes (properties) of the classes. Optional parameters have only been included when necessary. The reader should refer to the protocol document to obtain the exhaustive list of attributes of the class. In most cases the names of attributes provides a very good idea of its definition.

BACnet provides 18 different object-types or object-classes to represent the communication specific information at devices. There is no organization of the classes specified in BACnet. Even though it is obvious that there is inherent interconnection and a hierarchical organization to the classes this has not been used, as the class design methodology is not truly object-oriented. Some of these object-classes have a large number of properties common to them. For example, BACnet's BinaryInput, BinaryOutput and BinaryValue objects share a large number of common attributes and in fact should be subclass of a super-class called Binary.

All objects in BACnet have a key attribute uniquely identifies the object Object_Identifier. The Object_Identifier is encoded by 3 bytes which carry information about the class and the instance number for the object. There are two other attributes which are common to all the classes : Object_Name and Object_Type.

In order to better explain the philosophy used by BACnet, we have introduced new classes and a hierarchical structure to organizing the classes. There are three levels at which the classes defined by BACnet can be organized. These levels also correspond to the level of detail or information which one can obtain about the device.

At the highest level, one obtains boiler-plate information about the device's communication organization. This is shown in Figure ___ . At this level one can obtain

information about the vendor manufacturer, protocol information and names of the other objects contained in the device. The class which captures this information in BACnet is called the Device class.

At the second level are defined classes to categorize all the data elements in the device, as shown in Figure __ . The different classes are:

1. Variable : All objects which are variables belong to this class. This class further has three subclasses called analog, binary and multi-state based on the type of variable.

2. File

3. Program

4. Calendar

5. Management : This class represents objects which are used to manage other objects. These are the subclasses:

i. Command : Objects in this class are used to change the value of a group of properties in a set of objects.

ii. EventEnrollment : The objects in this class contain information to manage events in BACnet system. It provides a connection between the occurrence of an event and transmission of notification message to one or more recipients.

iii. Group : A group object represents a collection of other objects and one or more of their properties. A group object enables one to specify a large number of other objects in a shorthand manner.

iv. Recipients : The recipient object contains information required for the distribution of event notifications. It is used to form a link between the different kinds of events and list of destination devices which should receive the notification. The event notifications can be sent to different destinations based on the time-of-day or day-of-week.

v. Schedule : Schedule object is used to describe a periodic schedule for writing specified values to specified attributes of objects. This schedule can also contain exceptions. The intent of this is not for real-time periodic transfer of messages as required during process control.

6. Loop : A loop object represents the externally visible characteristics of any form of feedback control loop.

7. Proprietary : BACnet allows any vendor to define a proprietary object, as long as it satisfies some compliance conditions. This class is use to represent these objects.

At this second level of detail, the detailed internal structures of the major classes is not shown. Also not captured are the associations between the different classes.

The detailed structure of the classes is captured in the third level of detail. The discussion of this level of detail is out of scope for this paper.

3. Application Services

Five areas of application services are defined in BACnet.

1. Alarm and Event Services

2. File Access Services

3. Object Access Services

4. Remote Device Management Services

5. Virtual Terminal Services

These services are shown in Figure ___. Here we will not go into the details of the services. An idea of what objects are expected to support which services is provided in the OMT diagrams. There are two very interesting services, which are defined in BACnet : Who-Is and I-Am. All objects are expected to support these services. These services are used to locate objects in the system. Since all objects support these services, one does not need to have a localized object-directory to find objects in the system. This substantially reduces the logic required to do object-directory management.

Networked devices require a common language in order that they may interoperate with each other. This common language is often referred to as a common application language (CAL) and relies on the definition of an application layer protocol. There are three critical elements of acommon application language. First, the communicating devices must share a common defintion of the information elements that may be exchanged between them. Second, these devices must share a common syntax for describing the messages that are interchanged. Finally, these devices must share a common definition of the services or behavior that can be requested across the network.

In general, we see two approaches to developing these application layer definitions. A common approach is to combine the definition of behavior and information into

the structure of application layer messages that are passed between devices. Although this approach typically improves system performance since there are fewer steps in interpretation of the messages exchanged, it sacrifices extensibility in the sense that the semantics of a given message are tied directly to its structure and that structure has been fixed by the defintion. Another approach is to provide an encoding step between the defintion of semantics and the structure of the message. This is the approach used by application layer standards such as MMS. In fact, MMS is really only a framework within which an application layer protocol definition may be developed.

The need to define consistent information and behavior at the application layer lends itself naturally to the object-oriented paradigm found in some programming languages. Within this paradigm, objects define both data and action. The MMS framework defined in ISO 9506-1 and ISO 9506-2 employs some aspects of the object-oriented paradigm. MMS defines abstract objects such as variables, events, domains, and journals. For each abstract object type, attributes are defined that provide information about the object. MMS also defines an abstract set of operations called services that may be performed on objects of a particular type. Unlike the object-oriented paradigm used in some programming languages, these services are not strictly part of the objects themselves nor is there an inheritence hierarchy as one would traditionally find in C++ for example.

The definition of a common application language using MMS requires that the abstract defintions found in MMS must be mapped into the particular domain of interest. This mapping corresponds to the development of what is referred to as a companion standard. A companion standard defines objects, attributes, and operations that are consistent with the MMS framework abstractions but which are relavent to a particular application domain. A particular companion standard definition need not employ all of the abstractions defined in the MMS standard but it must be consistent with the MMS framework. A modelling framework such as that provided by OMT (Object Modelling Tool) may be employed in this process. A companion standard must also define how the objects and operations that have been defined are mapped into messages communicated between devices. A number of encoding strategies have been proposed including BER (Basic Encoding Rules) and PER (Packed Encoding Rules). In general, these rules define how information is mapped into the bytes that actually form the messages communicated between devices. In choosing the encoding rules for a particular application it is important to realize that there is a trade-off that must be made betweenthe power of the encoding scheme and the performance and memory required in its implementation.

OOM provides a tool whereby the “nouns” and “verbs” that describe an IED and its functions can be created or “abstracted”. The “nouns” or the information contained within the IED are known as the “attributes” of the object and the “verbs” or what the IED can do to the data are known as the “methods”. For example, a relay will make measurements of voltage and current and compute watts and vars. The attributes for this one aspect of a relay would be: Volts, Amps, Watts, and Vars. Subsequently, one of the methods would then be “Compute”.

In establishing the groundwork for abstracting the numerous attributes and methods of the substation IED’s, a model of the model or a “meta” model was created. This “meta” model defines data that would be present in any type of IED. There is a standard diagram that is used to construct the object model which is illustrated in Figure 2 via the “meta” model for an IED. The top line is the name of the object being described which in this case is a Virtual Device object. The middle section is the “attribute” list and the bottom section is the “methods” list. Clearly, there are more attributes and methods needed to describe, for example, a relay object. The beauty of the object modeling approach, however, is that various attributes can be grouped in classes and then linked back to the base model. This technique allows the addition of new attributes without having to re-do the definitions and assignments of the previously defined attributes.

Work is now in progress in the MMS Forum and the IEEE Power System Relay and Substation Committees to define standard or public object definitions. Common items such as Voltage, Current, Watts, Vars, etc. can very readily be agreed upon as far as a standard definition goes. It is inevitable, however, that each manufacturer’s IED will have attributes that are new or unique to that IED. These “vendor specific” attributes, being otherwise unknown to anything else in the system, need a mechanism to define what they are.

As such, the concept of “self defining data” was included in the requirements document. In response to the standard query “who are you?”, an IED would be required to download its object definition, complete with a data dictionary, to define any “vendor specific” attributes. In this manner, an RTU function could automatically query all IED’s in the substation and compile a standard list of the information it is required to obtain.

Implementation

The current institutional efforts to define structure for the new digital universe must be accompanied by the rapid development of implementation vehicles if the industry is

to advantage, rather than succumb to the new technology. If the current standards work is to have more than historical value it must be accelerated to synchronize with the pace of technological change, be viewed as a real time consensus definition of best practice, and be connected to an implementation path embraced by the utility industry.

Utilities are driven by the need for increased productivity in the emerging competitive environment, while at the same time are undergoing personnel reductions which weaken their ability to define and implement integrated automation systems to improve productivity. Functionally fragmented institutional structures are evolving to support these integrated system goals, but the trauma of such major changes further inhibits internal solutions.

Figure 3 illustrates the principal elements of substation integration and automation systems. The integration function has, in today's environment, typically moved outside of the utility box. The Integrator's challenge is to define best fit / best value solutions tailored to the specific utility's goals. In this consulting role the Integrator is expected to develop specifications for open systems with maximum flexibility for growth. His role can be expanded to identify the supplier of products and services, layout the program, manage the program, or take turnkey responsibility, as well as providing ongoing support and services.

The IED suppliers provide protection or monitoring based intelligent digital devices which are performance / cost optimized in a rapidly changing competitive market. The IED's also provide the data acquisition and control interface with the power system at the substation.

The Software suppliers provide custom software interfacing, drivers and control functions to integrate the IED's and provide the required system functions. The Integration Equipment includes the substation control and interfacing equipment such as Programmable Logic Controllers (PLC's) Personal Computers (PC's), Remote Terminal Units (RTU's), and associated communication equipment.

The value to the utility of being able to implement open systems which meet current and future requirements with interoperable elements is obvious. The alternative of a single supplier providing all elements integrated into a closed system has generally not been acceptable to the utility industry. Several other approaches are emerging which include:

- Combinations of the Software supplier and the Integrator
- The IED supplier with the Software supplier
- The integration Equipment supplier with the Integrator

and so on. In the absence of at least an appropriate de facto standard embraced by all the often competing elements, the solutions will continue to be somewhat cumbersome, inflexible and costly.

The two essential aspects of realizing the benefits of the application of digital technology to substation and power system automation are: First; a fast track commitment by the utilities to sponsor and support the definition of open standards for integration; and Second; the development and utilization of Integrators who effectively implement these emerging standards.

One example of the possible ways in which this could be accomplished would be the utilization of existing utility organizations such as EPRI to establish system integrators which, because of their utility sponsorship, would implement the preferred standards, represent the utilities, and strongly influence the support of the related hardware and software suppliers. Various other proactive utility initiatives would appear to be worth considering. The screens for such initiatives might include:

- support by a significant number of utilities
- a possible equity position by the sponsoring utilities
- a strong linkage with the continuing "standards" activity
- a strategy that allows customizing for individual utilities
- operational control at the Integrator with an advisory board representing the sponsoring utilities
- the assignment of key utility personnel to the Integrator during a project.

The alternative of waiting to see what becomes available does not look very attractive for the utility.

Conclusions

The search for a common communication platform in the utility industry is becoming acute being driven by the proliferation of communicating Intelligent Electronic Devices. A “top down” approach to solving the communication problem has been sponsored by EPRI. This effort has resulted in the creation of a requirements document that is open to the public for review. A summary of the basic requirements is presented. The heart of the requirements document is the use of Object Oriented Methodology as the tool to create a common IED communication language. This effort will only be effective if there is a concerted effort between the various industry players to quickly bring this technology to practice.

References

1. M. Adamiak, E. Weintraub, J. Schnegg, J. Burger, *The Integration of Protection, Control, and Monitoring is a High Voltage Substation*, Proceedings of the 46th Annual Texas A&M Conference for Protective Relay Engineers, April 12-14, 1993.

2. ISO 7498 - *OSI Basic Reference Model*

3. Rumbaugh et al, *Object-Oriented Modeling and Design*, Prentice-Hall, 1991

4. Peter Coad and Ed Yourdon, *Object-Oriented Analysis*, 2nd edition, Prentice-Hall, 1991