Practical Considerations in Application of UCA GOOSE

Practical Considerations

In

Application of UCA GOOSE

Mark Adamiak Drew Baigent
GE Power Management
King of Prussia, PA Markham, Ontario

Scott Evans
GE Corporate Research and Development
Schenectady, NY

Presented at :

Georgia Tech Relay Conference

May 3-5, 2000

Practical Considerations in Application of UCA GOOSE

Mark Adamiak Drew Baigent
GE Power Management
King of Prussia, PA Markham, Ontario

Scott Evans
GE Corporate Research and Development
Schenectady, NY

Introduction

The power and functionality of next generation microprocessors has led the way to next generation digital relays. One particular characteristic of these new processors is the tight integration of the processor with a high-performance communication controller. Given this new communication capability, an international effort has ensued to create a single communication protocol that advantages the capabilities of these new processors. That effort has centered around the Electric Power Research Institute's Utility Communication Architecture or UCA including a definition for the relay to relay communication of binary state data known as the GOOSE.

Development of UCA

The Utility Communication Architecture or UCA had its origins in 1990 as the framework for the ensemble of communication requirements that exist in the utility enterprise. It was at this time that utility managers were looking to consolidate communications among their planning, SCADA, metering, protection, and control departments. In attempting this consolidation, the cost of integration of diverse communication protocols was realized and a drive towards communication commonality was begun.

Since 1993, there has been a focus on the application of UCA in the substation. The process started with the creation of a requirements document that defined the communication requirements for the various functions inside a substation. The functional requirements document was followed by an implementation and evaluation phase that defined the "profile" of a communication structure for the substation. The profile that has been defined is shown generically in figure 1.

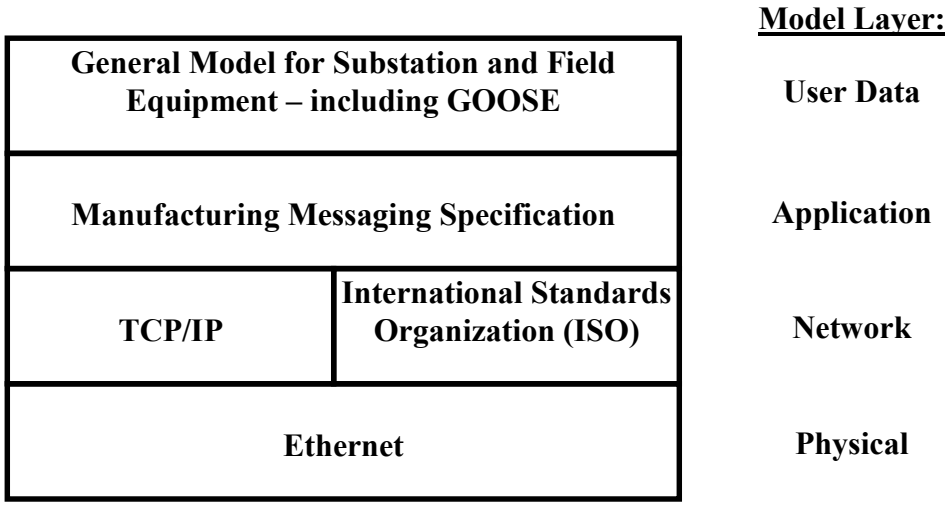

Figure 1
UCA Substation Profile

The goal in defining the "next generation" substation profile was to implement a high speed, networkable, peer to peer architecture using as many "existing" communications protocols as possible. In addition, the goal of user data interoperability required the definition of standard names for commonly used data objects.

The profile developed uses Ethernet for the Physical and Data Link layers. Although Ethernet is "non-deterministic" when operated in a "shared access" mode of operation (due to collisions), Ethernet technology has advanced to provide "switched" access which minimizes collisions. In addition, Ethernet provides a growth path to higher-speed Ethernet networks such as 100 MB and 1GB with 10 GB already defined.

For the "networking" layers, although the original goal was to stay within the realms of the International Standards Organization (ISO) standards, the popularity of the Internet dictated the inclusion of the TCP/IP networking layers. In November of 1999, the International Electrotechnical Committee (IEC) selected TCP/IP as the "mandatory" networking protocol for intra and inter substation communications and the ISO networking layers as optional. The inclusion of these networking layers makes data from the substation available over a utility intranet, WAN, or even the Internet.

For the Application or service layer, the Manufacturing Messaging Specification (MMS) was chosen. MMS provides a rich set of services to read, write, define, and create data objects. It is MMS and its ability to manipulate logical objects that differentiates this profile from all other existing profiles.

Lastly, the UCA substation profile defines standard "object models" for commonly used data elements. These standard models are defined in the document entitled: General Object Models for Substation and Field Equipment (GOMSFE). This standardization facilitates interoperability as any manufacturer who, for example, allows "Phase A - Gnd Voltage" to be externally visible, does so in a common manner.

Evolution of the GOOSE

One of the unique functional requirements identified for UCA was high-speed (goal of 4ms) device to ***multi***-device communications of simple binary state information. Inasmuch as sending multiple messages to multiple devices would incur an unacceptable time delay, an implementation was chosen that could send the same message to multiple devices simultaneously in a communication mode known as "multicast" (see Figure 2). The implementation of this function was done through the MMS information report service. The information report was used to deliver a binary object model (a collection of binary states of the device), known as the **G**eneric **O**bject **O**riented **S**ubstation **E**vent or **GOOSE**.

GOOSE works in a model type known as "Publisher / Subscriber". In this model, the sending device "publishes" the user-selected state bits in the device. Any device interested in any of the states of the publishing relay is programmed to "subscribe" to the publishing device's GOOSE message.

The GOOSE message is launched under one of two scenarios. The first scenario launches a GOOSE on a change of state of any of the binary variables in the message. The second scenario launches a GOOSE message on a user-selectable periodic basis. The reason for the latter scenario is that in absence of a state change, there is no way for the subscribing device to determine that the publisher is alive. When the subscribing device fails to receive an expected GOOSE message, it can declare the publisher as "dead" and set default states on the binary variables expected from the publisher.

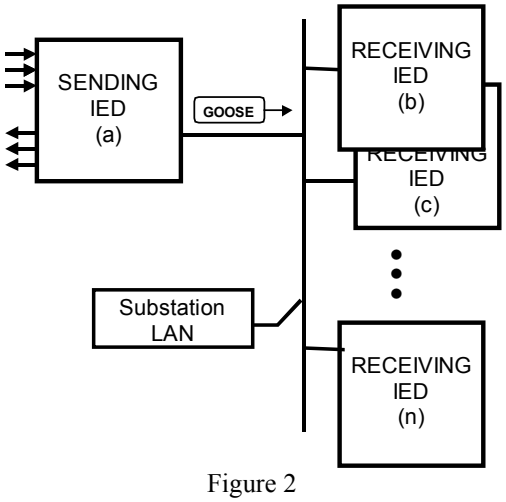

Figure 2
GOOSE Multicast Concept

One risk in the multi-cast GOOSE is the potential lack of assured and timely message arrival and processing due to "Best Effort" protocol quality of service (QOS) and the variable communications latency associated with a shared medium network. Coexistence of mission critical control signals with non-mission critical data causes variation in the QOS for the mission critical data that must be managed carefully. UCA GOOSE achieves reliability through the use of repeated unacknowledged messages. This paradigm produces specific challenges in that repeated messages add to the network load and device processing. Network collision problems can be met to some degree with switched routers. Message filters as low as possible in a device network stack can address device overload concerns by giving priority to mission critical messages related to a specific device and discarding others. Still, care must be taken to ensure that oscillography and other non-time critical data sent on the network are correctly managed at the application layer to prevent overloading the network or device buffers and delaying the control messages. In order for UCA GOOSE to be effective as a method to send mission critical data throughout the network, all of the above concerns must be addressed and enforced by all devices on the network.

The content of the GOOSE message can be broken down into three areas: a header, Dynamic Network Announcement (DNA) state information, and User State information.

The header contains operational information about the GOOSE message such as the name of the sending device, the time that the GOOSE message was constructed, and the maximum time until the next GOOSE message. The Max Time until next GOOSE is used to detect failure either in the sending device or in the network itself.

The Dynamic Network Announcement or standard bit pairs were designed to facilitate connection between devices. For example, if a line relay issues a "Trip" signal, any device subscribing to the line relay and that is interested in that particular signal can take action accordingly. There are 32 pre-defined bit pairs which can be seen in Appendix I. The concept is similar for the "User State" information except that the definition of the data is totally user defined. Data states are sent in pairs with (0,1) and (1,0) being the primary logical states, (0,0) being defined as a "transition" state, and (1,1) being undefined.

The delivery of the GOOSE object is through the connectionless ISO stack, which means that there is no specific destination address. As such, some address must be entered for the Ethernet receiver to resolve. One option is to require the user to create and enter a 48 bit address to define who the publisher is. This same 48 bit address would have to be programmed into each subscriber that wanted to receive a GOOSE message from that publisher.

A second option is to work with a **S**elf **M**entoring **A**nd **R**e-**T**raining or **SMART** GOOSE. Self Mentoring is the process of automatically defining and determining an address of the publisher and subscriber(s) based on a logical name. During set-up, the engineer programs every device in the station with a unique name and programs the receiving devices with a list of device names from which it should expect to receive data. On start-up of the network, the SMART GOOSE reads the Media Access Control (MAC) address of the Ethernet controller and uses this unique address as its source and destination address for GOOSE messages. Next, the Ethernet receiver in the receiving devices goes into "promiscuous" mode whereby all multi-cast messages are read and decoded. The name of the device sending the message is compared with the programmed list of "devices to listen to". If the names match, the receiving device stores the MAC address of the sending device in a high-speed hardware address comparator. Once all address / name matches have been made, promiscuous mode is turned off and all multi-cast messages are now captured based on a hardware address comparison.

The "Re-Training" part of the SMART GOOSE comes into play when a relay is taken out of service or a CPU module is exchanged or upgraded. When the CPU card is changed, the corresponding MAC address for the Ethernet card changes as each Ethernet controller in the world has a unique 48 bit address. SMART GOOSE (on the receiving side) recognizes that the GOOSE message it was expecting is missing. In this scenario, the receiving relay goes back into promiscuous mode – again searching for a message with the name of a desired device inside. Once found again, the new MAC address for the new CPU / Ethernet controller is stored in

the high-speed look-up table and the receiving relay once again turns off promiscuous mode and returns to normal operation.

As mentioned earlier, the goal for "max time on the wire" was less than 4ms. Achieving this time involves defining a number of system factors including internal GOOSE processing, communication speed, number of devices on the LAN, and LAN loading. Figure 3 shows an oscilloscope timing of a digital input on one relay and the closure of the output contact on a second relay. This time (7.8ms) includes processing of the digital input (including debounce), processing of relay logic, transmission through the communication stack, time on the wire, communication processing on the receiving relay and output contact execution time. Time on the wire for this application was in the microsecond time frame.

Generalized Relay Architecture

Given this new construct of device to device communications, there is now a need to be able to logically integrate data not only from one device but also from other logical devices in the network. As such, one can now look at a generalized device logical architecture to perform this integration function. This generalized architecture is shown in figure 4.

The architecture can be divided into three areas: inputs, combinatorial logic, and outputs. In the GOOSE enabled device, there are four sources of inputs. The first are the outputs from the various protection functions. The second is from the traditional "hard wired" digital inputs. Third, we can speak of "remote inputs" that come from other devices in the network. Lastly, there are "virtual inputs". These inputs are memory locations that can be set from external sources such as a Human Machine Interface. A good example of a virtual input is user flag that is set to block operation of a device or function.

Given the various input signals present in this next generation device, some sort of programmable logic is now needed to combine these signals into the functionality required by the user. In general, any given device may have multiple logic functions internally programmed – each with its own resultant output.

This leads to the third area of the generalized functional architecture which is the output structure. Each logic equation implemented will have an internal result or "virtual" output. It is desirable from a user's perspective to be able to time tag the virtual outputs in order to evaluate the proper operation and performance of the programmed logic. Virtual outputs can then be mapped to one of three places: First of all, virtual outputs can be mapped to a physical output contacts. Secondly, virtual outputs can be mapped to "remote" GOOSE outputs and sent to multiple other devices. Lastly, if the programmable logic permits, a virtual output can be fed back as an input to the programmable logic. This capability allows one to create a state machine in the device.

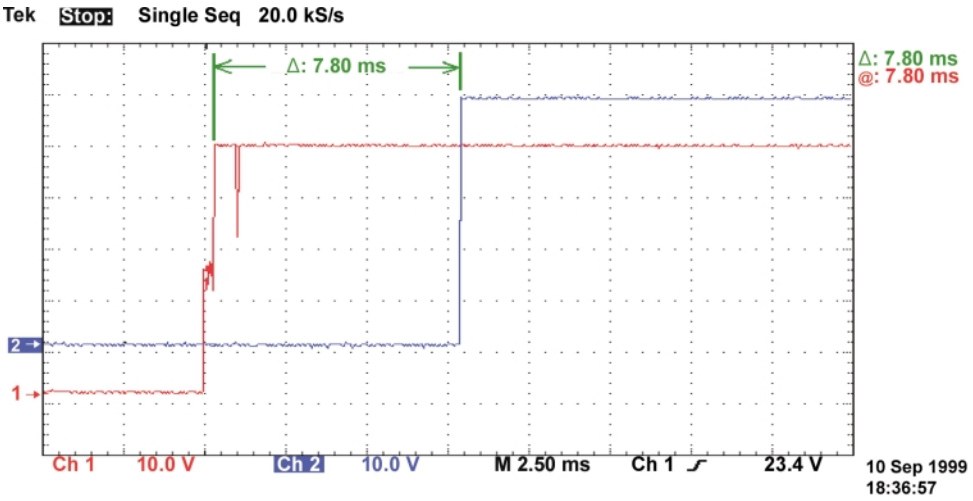

Figure 3
Oscilloscope Timing of Relay to Relay GOOSE Message

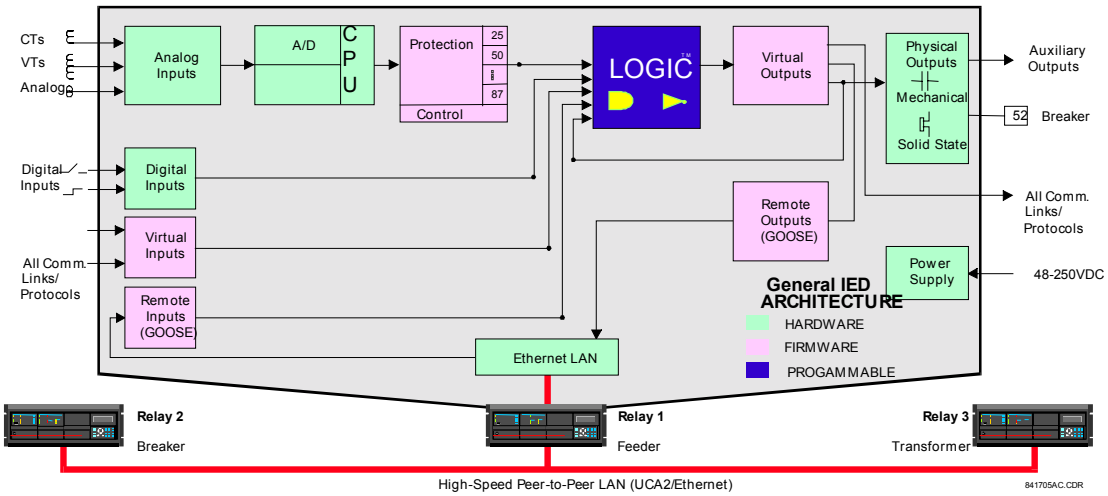

Figure 4
Generalized Device Logical Architecture

GOOSE Applications

Given GOOSE messaging in conjunction with a programmable logic architecture, there are now a limitless number of applications that can be implemented. This section looks at four of these possible applications:

Voting:

In the art and science of protection, part of the "art" is creating a balance between the security and dependability of a protection scheme. One technique that is used in mission critical application is the concept of voting (see figure 5). Voting says to issue a trip only if 2 out of 3 relays say trip. To implement this function, relay 1 needs to get trip information from relays 2 and 3, relay 2 needs to get trip information from relays 1 and 3, etc. Each relay is required to implement the voting logic internally and each relay has to subscribe to the others trip message.

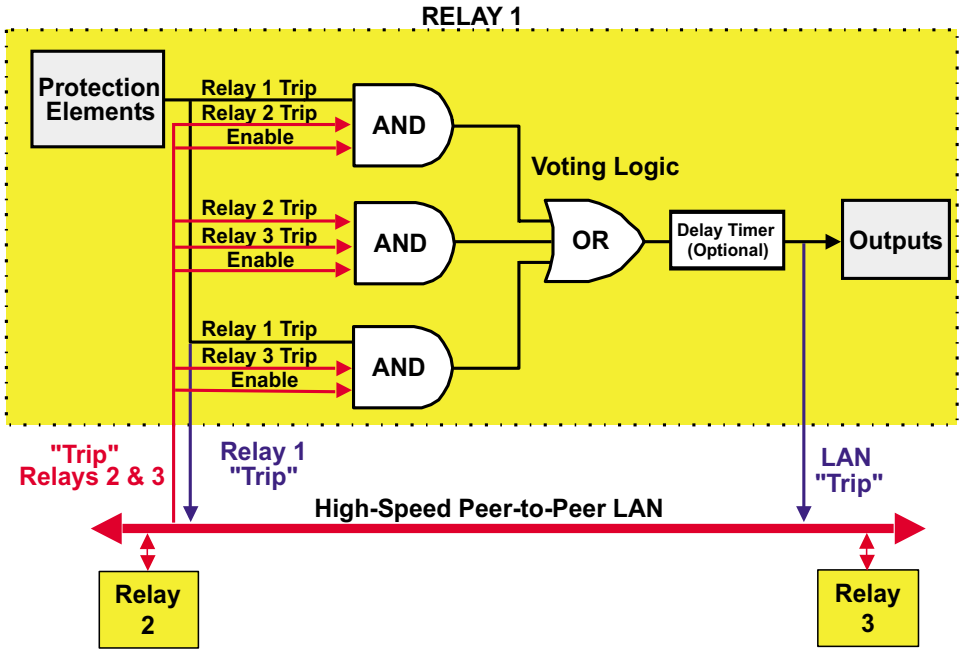

Figure 5
Relay Voting Scheme using GOOSE Messaging

Since each relay has the voting logic internal to itself, failure of a single relay does not fail the voting scheme.

What is interesting to note is the performance of the scheme under a "dead GOOSE" scenario – that is – when a particular relay fails to send a GOOSE message in the allotted time frame. When this happens, a default state is assigned to the expected data. Depending on how the default is set by the engineer, the voting scheme will default to be more secure or more dependable. For example, if relay 2 fails and the trip input to relay 1 is defaulted to "no trip", for a trip to occur, both relays 1 and 3 must issue a trip thus defaulting to a secure state. On the other hand, if the failed input from relay 2 is set to "trip", the system defaults to a dependable state in that if either relay 1 or relay 3 say trip, the system will trip.

Bus Blocking

In many distribution station applications, an incoming feeder has a breaker feeding a bus with multiple feeders exiting from the bus (see figure 6). Typical protection calls for an instantaneous overcurrent function that is coordinated with the overcurrents on the underlying feeders. Coordination typically can be translated as "time delay". Application of GOOSE messaging between the underlying feeders and the incoming feeder breaker can optimize this situation.

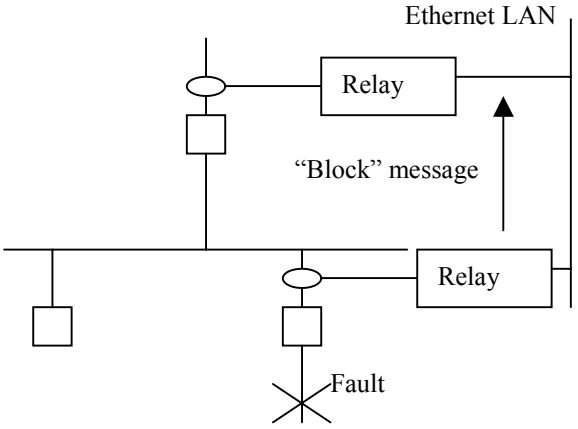

Figure 6
Bus Breaker Blocking

When any of the feeders detects an overcurrent, it is to send a "block" message to the incoming feeder telling it not to trip. Such communication can speed up protection for bus faults and add security for feeder faults.

Load Shedding

Typical load shedding applications in a substation require the addition of a separate under frequency relay followed by wiring from the load shed relay to any breakers to be tripped under an under-frequency condition. Reality is that most breakers in a substation are connected to the tripping output of at least one relay in a substation. Connecting these relays via an

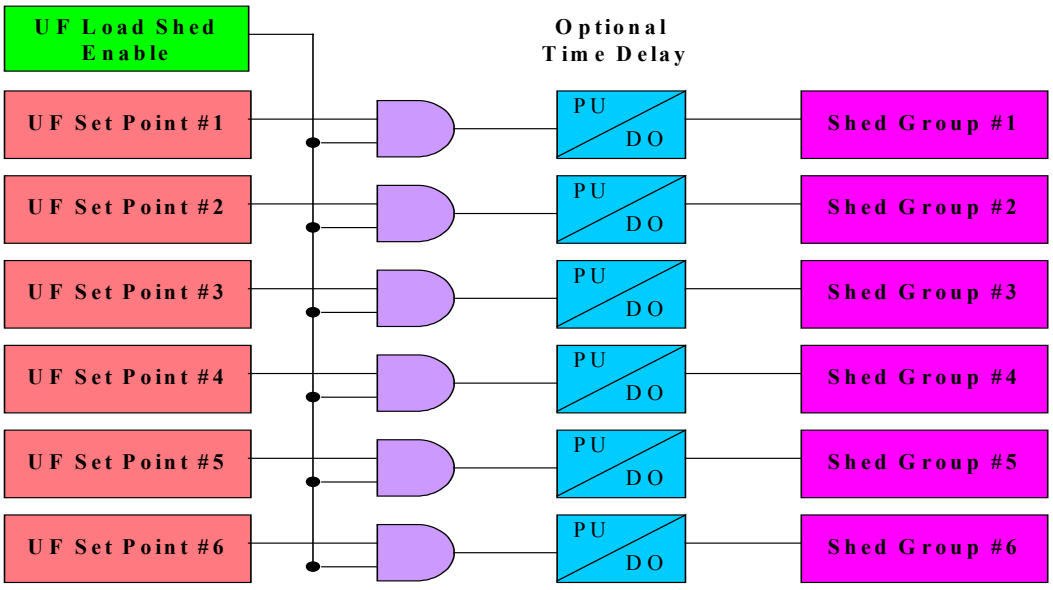

Figure 7
Distributed Load Shed

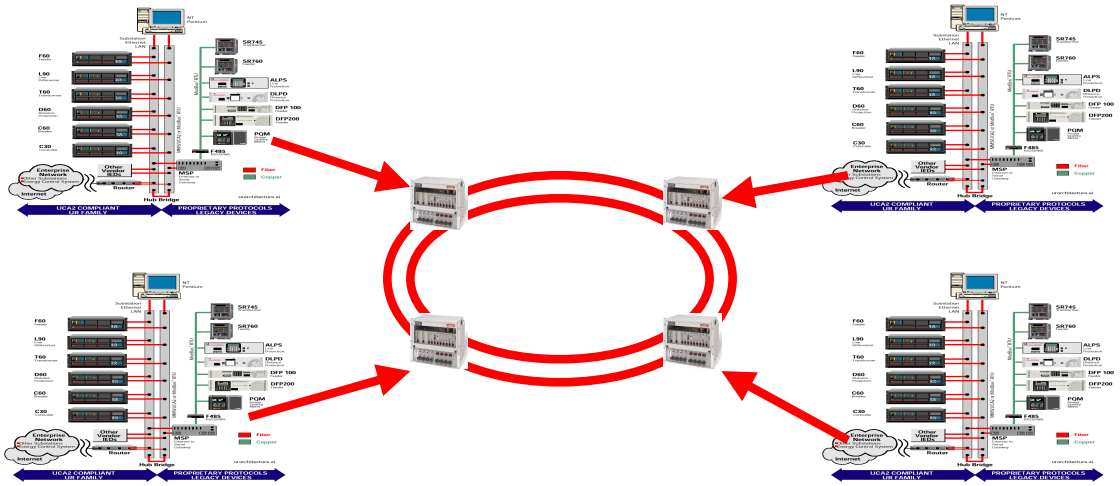

Figure 8
Wide Area GOOSE (WAG)

Ethernet network, load shed becomes a GOOSE message to trip the appropriate breaker (figure 7). With some additional logic, the engineer could actually create a rotating schedule of loads to shed. Clearly, a restoration scheme could be created in similar manner. Since this scheme could be loaded into any relay, redundancy is also easy to implement.

Wide Area GOOSE (WAG)

Although the GOOSE message is not routable, a number of SONET multiplexors are capable of "bridging" GOOSE messages between substations over Ethernet. With this configuration, one is able to perform functions such as transfer tripping, pilot based protection, high speed data transfer, and in general, have a foundation for next generation power system control. Caution need be taken when implementing such a scheme as too many GOOSE messages can clog the network.

Conclusions

Next generation microprocessors in next generation digital relays have provided the foundation for next generation communication protocols and applications. The Utility Communication Architecture has provided the relay engineer with a new high-speed binary tool in the form of the Generic Object Oriented Substation Event or GOOSE. With this tool, relay engineers will be able to create advanced protection schemes more easily with no additional wiring.

Bibliography

1. General Object Model for Substation and Feeder Equipment (GOMSFE). Version .9, January 14, 1999
ftp.sisconet.com/EPRI/UCA2.0/
gomsfe9.zip

Appendix I

Dynamic Network Announcement
Bit Pair Definitions
From GOMSFE .9, January 14, 1999

DNA is a single message that conveys all genetically required protection scheme information regarding an individual IED. This message uniquely reports the status of the devices in the IED to it's peers per the enrollment list. The table below defines the use of DNA for protection messages

Bit #	Bit Pair	Bit Order	00	01	10	11
		Value	0	1	2	3
		Definition	State	State	State	State
0,1	1	OperDev	Normal	Trip	Close	Invalid
2,3	2	Lock Out	Invalid	Normal	LO	Invalid
4,5	3	Initiate Reclosing	Normal	Cancel	Auto Reclosing	Invalid
6,7	4	Block Reclosing	Normal	Cancel	Block	Invalid
8,9	5	Breaker Failure Initiate	Normal	Cancel	Initiate	Invalid
10,11	6	Send Transfer Trip	Normal	Cancel	Set	Invalid
12,13	7	Receive Transfer Trip	Normal	Cancel	Set	Invalid
14,15	8	Send Perm	Normal	Cancel	Send Perm	Invalid
16,17	9	Receive Perm	Normal	Cancel	Receive Perm	Invalid
18,19	10	Stop Perm	Normal	Cancel	Stop Perm	Invalid
20,21	11	Send Block	Normal	Cancel	Send Block	Invalid
22,23	12	Receive Block	Normal	Cancel	Receive Block	Invalid
24,25	13	Stop Block	Normal	Cancel	Stop Block	Invalid
26,27	14	BkrDS	Between	Open	Closed	Invalid
28,29	15	BkrPhsADS	Between	Open	Closed	Invalid
30,31	16	BkrPhsBDS	Between	Open	Closed	Invalid
32,33	17	BkrPhsCDS	Between	Open	Closed	Invalid
34,35	18	DiscSwDS	Between	Open	Closed	Invalid
36,37	19	Interlock DS	Invalid	Non Interlock	Interlock	Invalid
38,39	20	LineEndOpen	Between	Open	Closed	Invalid
40,41	21	Mode	Test	Offline	Available	Unhealthy
42,43	22	Event	Invalid	Normal	Event	Invalid
44,45	23	Fault Present	Invalid	Clear	Present	Invalid
46,47	24	Sustained Arc	Invalid	Normal	Present	Invalid
48,49	25	Downed Conductor	Invalid	Normal	Downed	Invalid
50,51	26	Sync Closing	Normal	Cancel	Initiate	Invalid
52, 53	27	Reserved				
54, 55	28	Reserved				
56, 57	29	Reserved				
58, 59	30	Reserved				
60, 61	31	Reserved				
62, 63	32	Reserved				