

MultiSpeak®2 - A Framework For Real-Time Utility Software Integration

Gary A. McNaughton, P.E., *Member, IEEE*

Martin E. Gordon, P.E., *Senior Member, IEEE*

Abstract – Effective integration of software has been an elusive goal for electric utilities for many years. Integration, particularly among operations applications – such as supervisory control and data acquisition (SCADA), automated meter reading (AMR), outage management (OMS), and load management (LM) systems – has been particularly difficult. Such integration, where possible at all, has often required costly custom programming. The expense and complexity of custom integration has often prohibited electric distribution utilities from achieving successful interaction among operations applications.

The MultiSpeak® Initiative is a collaborative effort by software vendors, sponsored by the Cooperative Research Network of the National Rural Electric Cooperative Association. The Initiative has developed a specification that defines how a wide variety of application software provided by different vendors can be integrated in a platform-, database-, and operating system-independent manner. This is done using a real-time messaging framework and standard information systems protocols.

This paper discusses the implementation of the application interfaces and the real-time messaging framework. It also discusses how the vendor-developed interfaces are tested for compliance with the specification.

Index Terms – Enterprise-wide integration, application integration, data integration, software interfaces, MultiSpeak®.

I. BACKGROUND

Integration of data-intensive automation applications has been an on-going challenge for electric utilities of all sizes. However, integration has been especially difficult for small distribution utilities, which often do not have the information technology resources that are available to larger companies. A number of approaches have been tried to address these concerns, but typically resulted in costly, custom application interfaces that were an on-going maintenance concern for both vendors and cooperatives [1], [2].

The National Rural Electric Cooperative Association (NRECA) created the MultiSpeak® Initiative in collaboration with software vendors serving the electric utility industry. The purpose of the effort is to address the need for cost-effective, integrated software applications to serve the business needs of electric distribution utilities, such as rural electric cooperatives. The Initiative has addressed this need by developing a specification that defines (i) the data that typically need to be exchanged between software applications

of various types, (ii) interfaces to exchange such data, and (iii) a communications framework to implement the interfaces.

Early experience with the MultiSpeak effort was described and documented in a previous paper by the authors [3] and in the first version of the data exchange specification that resulted from the collaborative efforts of the participants in the MultiSpeak Initiative [4]. A more recent paper by the authors [5] describes the development of the application interfaces and the messaging framework used in the most recent version of the MultiSpeak specification (MultiSpeak2) [6].

This paper extends these previous works by describing how software provided by different vendors may be used to create an integrated automation system and how interfaces developed in accordance with MultiSpeak2 are tested for compliance with the specification.

II. MULTISPEAK APPROACH

MultiSpeak2 is based on simple standard technologies such as extensible markup language (XML) and standard communications protocols. Thus it is anticipated that some utilities can develop their own MultiSpeak interfaces where they are not available from the utility's software vendors.

MultiSpeak2 integration assumes that each software application is equipped with an interface that marshals the required information from the native data structure, converts those data into XML data packets, and transports those packets in the form of predefined messages. The receiving application is responsible for unbundling the message, parsing the XML and taking the appropriate action with the resulting data objects. Fig. 1 illustrates this approach for data moved between a customer information system (CIS) and a geographic information system (GIS).

The MultiSpeak specification is based on a model of data ownerships and data exchanges that is intended to implement common business processes in electric distribution utilities. The MultiSpeak specification includes the detailed definitions of over 850 individual data objects, encompassing virtually every data item of interest to distribution utilities. Interfaces are defined to include the list of data objects to be exchanged, a determination of which data fields within an object are mandatory to support the underlying business processes, and a determination of what modes of communication are required to accomplish the data exchange. Data exchanges are defined between functions, or theoretical capabilities of application software products. Table 1 describes the software functions currently supported in MultiSpeak2. Each function is referred to by a function name; for instance CB is the function name for the customer billing database function.

G.A McNaughton is with Cornice Engineering, Inc., Pagosa Springs, CO 81147 USA (e-mail: gmcnaughton@frontier.net)

M.E. Gordon is with the National Rural Electric Cooperative Association, Arlington, VA 22203 USA
(email: martin.gordon@nreca.coop)

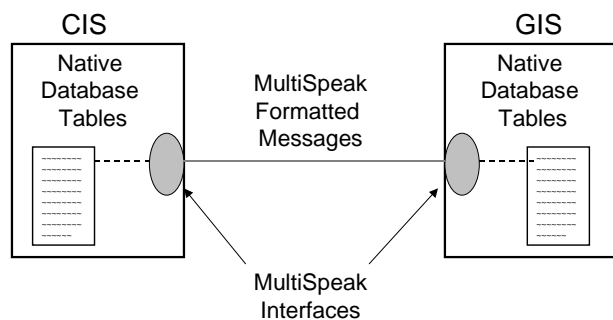


Fig. 1. MultiSpeak approach to common interfacing.

TABLE 1
MULTISPEAK FUNCTION DEFINITIONS

Function	Function Definitions
CB	Customer Billing Database
CD	Connect/Disconnect/Power Limitation
CH	Call Handling
CRM	Customer Relationship Management
DGV	Dynamic GIS Viewer
EA	Engineering Analysis
FA	Finance and Accounting
GIS	Geographic Information System
LM	Load Management
LP	Load Profiling
MR	Meter Reading
OA	Outage Analysis
OD	Outage Detection
SGV	Static GIS Viewer
SCADA	Supervisory Control and Data Acquisition
Staking	Automated Staking

Fig. 2 illustrates the interfaces supported in MultiSpeak2. Note that each box in the process model (Fig. 2) represents a software *function* (as defined in Table 1), not a software *application* or product. Any specific product may encompass multiple functions and hence full support of the MultiSpeak specification for that product would require support for all of the defined interfaces for all of the software functions implemented. The numbers listed in ovals are the interfaces as identified in the MultiSpeak Version 2.2 specification. Shaded ovals indicate interfaces that were defined previously in the MultiSpeak Version 1.1 specification [4]. Unshaded ovals denote interfaces that are new in Version 2.2 of the specification [6]. The designation “optional” in Fig. 2 indicates that the associated interface is not required for compliance with the specification. Although support for an optional interface is not required, it is recommended that if such an interface is implemented, it be done in the defined manner.

III. MULTISPEAK DATA TRANSFER MODES

MultiSpeak interfaces are based on a messaging paradigm. Applications wishing to exchange information send and receive messages that implement one or more of the following communications modes: (i) batch, (ii)

request/response (RR), and (iii) publish/subscribe (PS). Both the RR and PS modes are used in MultiSpeak2 to implement real time communications between automation applications.

For the purposes of MultiSpeak and this paper, communications are defined to be real time if no intentional time delay is injected into the communication; data are sent as soon as they are available. This is sometimes referred to as “soft” real time communications, in contrast with “hard” real time, in which communications must be completed within a specified time period. Although some applications deployed in utilities, notably SCADA, require hard real time communications among internal program functions, soft real time is adequate for integration among the application types supported in MultiSpeak2, including SCADA.

In batch communications, data are sent periodically in groups. Typically, batch communication is chosen either (i) where it is necessary to initially load replicated data in an application from another software package that is the primary holder of that data, or (ii) where timely data synchronization is not critical to the underlying business process. However, batch interfaces can be used to implement many tightly-coupled business processes in utilities provided the data exchanges are sent automatically, sufficiently frequently, and the messages sizes are manageable so that time spent parsing the XML is kept to a minimum.

Request/response communications is achieved in a point-to-point, synchronous manner in which one application (the client) requests information of the other (the server); those data are subsequently returned in a response message. Request/response messaging is particularly effective for applications where the client wishes to query the server in real time rather than storing a duplicate copy of data maintained on the server. The request/response communication mode is used by the Hypertext Transfer Protocol (HTTP), the basic protocol used in web site and web service implementations.

Publish/subscribe communications occur when one or more subscribers (clients) request to be notified of changes in data stored on the publisher (server). PS may be point-to-point or point-to-multipoint, depending on the number of subscribers to changes in a specific piece of data. Request/response messaging is adequately supported by transport mechanisms such as HTTP over TCP/IP or TCP/IP sockets connections. On the other hand, PS requires an additional defined messaging policy layer to facilitate point-to-multipoint messaging using only point-to-point message protocols. This might be required for instance if a geographic information system needs to send information about the same changes in facility data to both an engineering analysis and an outage management system. An additional function of the messaging policy layer is to provide separate communications channels for (i) published data and (ii) administrative messages, such as those used to support subscription, discovery, and ping activities. Fig. 3 illustrates schematically the function of the messaging policy layer for publish/subscribe communication.

Each software application that supports publish/subscribe communications is expected to provide a means to maintain a subscription list. That list includes information about which subscribers have requested notification of changes to which data objects made available by the publisher. In addition, the

publisher is required to provide a mechanism to (i) detect changes in the data objects to be provided under the

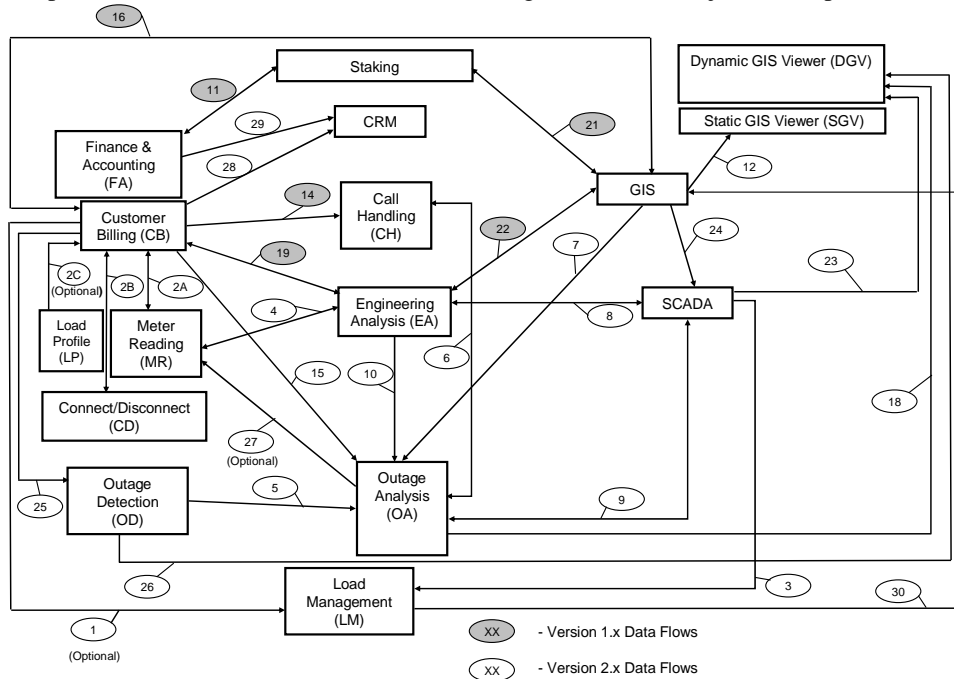


Fig. 2. MultiSpeak version 2.2 process model.

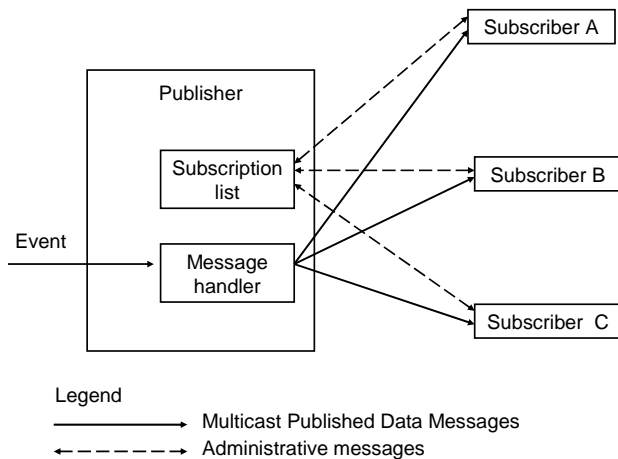


Fig. 3 MultiSpeak2 messaging policy layer for the publish/subscribe communication mode.

subscriptions (“events”), (ii) generate the appropriate message to publish those changes, and (iii) issue that message to all subscribers that have expressed interest in the modified data. It should be noted that, although MultiSpeak defines the specific point-to-point interfaces outlined in Fig. 2, it is also possible for a software application to use the publish/subscribe communications mode to obtain data in generic ways not originally considered in the specification. For instance application software for which no specific interfaces have been defined may subscribe to data published by functions included in Fig. 2, just as if a defined interface existed. This flexibility is likely to permit utilities and software vendors to make creative uses of MultiSpeak2 that were not anticipated by the authors of the specification. One such use would be the creation of

management reporting “dashboard” applications, which would subscribe to a variety of data presented by MultiSpeak-compliant applications and format the data in ways that support meaningful management action.

IV. IMPLEMENTATION OF MULTISPEAK INTERFACES

Although the MultiSpeak specification is transport-neutral, the MultiSpeak Initiative vendors initially have chosen to support two means to carry messages: (i) the Simple Object Access Protocol (SOAP), using the SOAP document mode, carried over HTTP, and (ii) TCP/IP sockets connections. The Initiative has issued two suggested practice documents that recommend how MultiSpeak should be implemented to avoid potential miscommunications among otherwise compatible software applications, one for SOAP implementations [7] and one for TCP/IP sockets connections [8].

Fig. 4 illustrates the detailed implementation of the PS messaging policy layer for SOAP data transmission. A SOAP client/server pair is required at the publisher as well as the subscriber application. Inspection of Fig. 4 indicates that two channels of communications are required for PS transactions. Dashed lines in Figs. 3 and 4 denote the administrative message events which occur on one communications channel; solid lines are published data events that are accomplished using the second channel. The numbers in Fig. 4 that are enclosed in boxes correspond to the administrative message events listed in Table 2; numbers in ovals in Fig. 4 correspond to the published data events outlined in Table 2.

A similar implementation for TCP/IP sockets has been defined that makes use of two socket connections between the publisher and subscriber, one for administrative messages and one for published data messages.

The MultiSpeak2 specification does not assume the existence of any message queue or other middleware to

accomplish the messaging policy layer described in the previous section. Since the goal of the MultiSpeak Initiative is

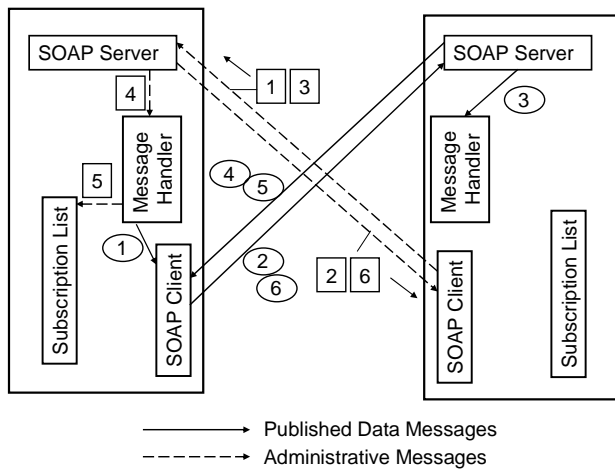


Fig. 4. Publish/Subscribe messaging implementation.

TABLE 2
PUBLISH/SUBSCRIBE MESSAGE EVENTS

- Administrative Message Events:
- 1 Subscriber issues a SubscriptionIDRequest message.
 - 2 Publisher issues a SubscriptionIDResponse.
 - 3 Subscriber issues a SubscribeRequest.
 - 4 Publisher receives and handles SubscribeRequest.
 - 5 Publisher updates subscription list.
 - 6 Publisher responds with a SubscribeAck.
- Published Data Events:
- 1 Publisher generates PublishedData message.
 - 2 Publisher issues PublishedData message.
 - 3 Subscriber receives and handles PublishedData message.
 - 4 Subscriber responds with PublishAck message.
 - 5 If required by the publisher, the subscriber returns a MessageConfirmation.
 - 6 If the subscriber sent a MessageConfirmation, the publisher returns a MessageConfirmationAck

to make cost-effective interoperable software available to electric distribution utilities, and the cost of message queue middleware is typically prohibitive for the target utilities, each vendor of MultiSpeak-compliant software is expected to provide the required messaging policy layer capabilities without assuming the existence of middleware to provide such functionality. In this mode, a publisher component would be expected to provide point-to-multipoint messaging using multiple point-to-point messages, each of which could easily be accommodated using either SOAP or sockets.

As a result, it should be possible for a utility to completely implement the capabilities illustrated in Fig. 2 using MultiSpeak-compliant software without any separate messaging middleware component. Although this implementation would be simple and cost-effective for the utility, it is achieved at the expense of increasing the complexity of each software product for which PS communications has been defined in the MultiSpeak2 specification. Additionally, such a point-to-point implementation lacks the capability to provide guaranteed

delivery, message persistence or message auditing, a function that would assist in debugging communications errors should they arise.

Although MultiSpeak2 has been defined in such a way that messaging middleware is not required for a functional implementation, MultiSpeak-compliant applications may also be installed using an integration server or message queue component. Installed in this manner, the integration server and the software applications that need to exchange data create a “hub and spoke” configuration, illustrated in Fig. 5. In this configuration, each software function (illustrated by the boxes ringing the integration server in Fig. 5) needs only maintain a single subscription for each type of data (that with the server) and send a single message to the integration server for each data event. The integration server would maintain the subscriptions with individual software functions and provide guaranteed delivery to all subscribers along with message persistence and message auditing.

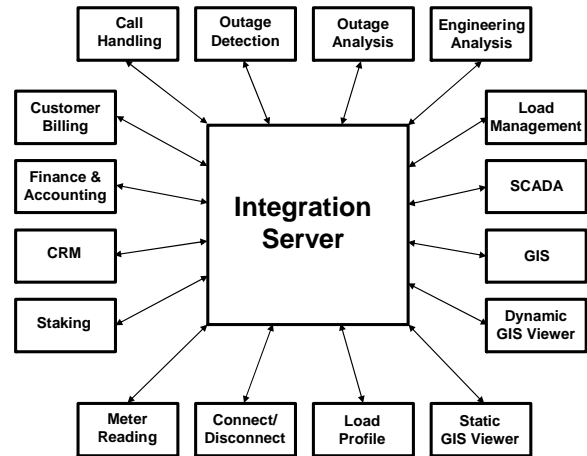


Fig. 5. Hub and spoke configuration using an integration server.

V. COMPLIANCE TESTING

The MultiSpeak Initiative maintains a compliance testing program to assure potential purchasers of software products designed in accordance with MultiSpeak2 that the product performs in compliance with the specification. Since the specification deals with data exchange and not the internal functioning of software, compliance tests focus on the proper exchange of data and do not certify any aspects of the internal functioning of the software product under test. Compliance testing is provided by independent parties that do not provide products of the type being tested.

The MultiSpeak2 specification is compiled in several parts, one of which is an XML schema that contains the data and message definitions. A special schema has been developed for each data flow direction for each defined interface. The purpose of these schemas is to establish the minimum requirements to support the assumed utility business processes that use the data flowing on the associated interfaces. Sample XML messages have been created for all of the message types defined in the specification, for each interface. Vendors may

use the sample data and sample messages to develop their integration modules.

Compliance tests include ensuring that (i) the application under test sends all of the required messages and data types expected of the supported software functions, (ii) all such messages validate with the appropriate XML schemas, and (iii) the application properly responds to messages sent to it. Compliance testers expect the vendor of the software under test to either display data received in a user interface, or to demonstrate that the internal database of the application under test has been updated appropriately.

Tests of interfaces using batch communications can be achieved by the exchange of appropriately formatted XML files, and hence do not require special hardware or testing platforms. However, certification of a real time interface requires a test bed that simulates the performance of the MultiSpeak function with which the software under test is expected to integrate. Fig. 6 schematically illustrates the use of the MultiSpeak compliance test bed. The test bed consists of an integration server designed to (i) accept and validate MultiSpeak messages, (ii) store the data, if any, resulting from those messages in a database, and (iii) generate and deliver the appropriate messages to the application under test.

For instance, if the software under test provides the GIS function, and is being tested for compliance on the GIS-EA interface (number 22 in Fig. 2), then the test bed accepts messages from the GIS, validates those messages and responds exactly as a MultiSpeak-compliant engineering analysis product would under the same circumstances. The integration server accepts all of the messages defined in the MultiSpeak specification, regardless of whether they use the batch, request/response, or the publish/subscribe communication mode.

VI. CONCLUSIONS

The MultiSpeak Initiative has developed a generally-applicable specification that vendors and utilities may use to develop standardized interfaces between commonly-applied software products. The specification defines a data dictionary and a messaging framework. Defined interfaces use a combination of batch, request/response, and publish/subscribe communications modes as appropriate to support typical business processes in electric distribution utilities.

Request/response and publish/subscribe modes are used to achieve real time integration. Such interfaces may be implemented using either document mode SOAP or TCP/IP sockets. A messaging policy layer has been developed to support the special needs of publish/subscribe communications.

Utilities may deploy MultiSpeak2-compliant applications without the need of applying special middleware or integration servers. However, the addition of a messaging server streamlines the configuration required during installation of the MultiSpeak-compliant software. In addition a server also provides the advantages of message persistence, guaranteed delivery and message logging and auditing.

For more information about developments of the MultiSpeak specification, a complete listing of the participating vendors and a list of currently compliant software products, see the MultiSpeak web site at www.multispeak.org. A user's guide [9] that describes the implementation of MultiSpeak2-compliant interfaces from the utility perspective is also available from the web site.

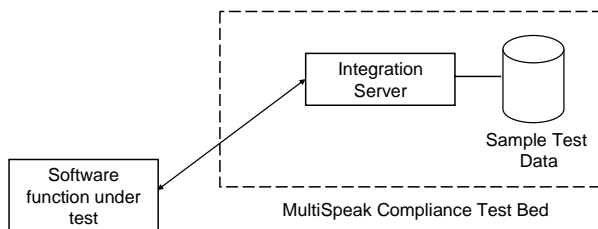


Fig. 6. MultiSpeak compliance testing.

In the example listed above, the software under test would be a GIS product and the integration server would be configured to act as an engineering analysis product. The GIS product would be expected to send complete information necessary to create an engineering model for the electrical utility system stored in the GIS (using both batch and request/response modes as defined in the specification). The integration server would check the data sent and return both load flow and short circuit analysis results appropriate for the system sent it, in both batch and publish/subscribe formats, as defined by the specification. The compliance tester would also ensure that the GIS under test was able to subscribe to the appropriate data objects and properly support all of the administrative messages required by MultiSpeak2

VII. REFERENCES

- [1] Gary A. McNaughton and Warren P. McNaughton, *Enterprise-Wide Data Integration in a Distribution Cooperative, Volume 1 – Executive Overview*, Arlington, VA: National Rural Electric Cooperative Association, 1996.
- [2] Gary A. McNaughton and Warren P. McNaughton, *Enterprise-Wide Data Integration in a Distribution Cooperative, Volume 2 – Technical Considerations*, Arlington, VA: National Rural Electric Cooperative Association, 1996.
- [3] Gary A. McNaughton and Martin E. Gordon, “Common Interfaces for Enterprise Integration – Experience With NRECA’s MultiSpeak™ Specification”, *Proceedings of the 2001 Rural Electric Power Conference*, New York, NY: Institute of Electrical and Electronics Engineers, Inc., 2001.
- [4] MultiSpeak Initiative Participants, *MultiSpeak Data Exchange Specification Version 1.1*, Arlington, VA: National Rural Electric Cooperative Association, 2000.
- [5] Gary A. McNaughton and Martin E. Gordon, “Development of a Real-Time Framework for Enterprise Integration – NRECA’s MultiSpeak@2 Specification”, accepted for publication in the *Proceedings of the 2004 Rural Electric Power Conference*, New York, NY: Institute of Electrical and Electronics Engineers, Inc., 2004.
- [6] MultiSpeak Initiative Participants, *MultiSpeak® Version 2.2 Specification*, Arlington, VA: National Rural Electric Cooperative Association, 2003.
- [7] MultiSpeak Initiative Participants, *MultiSpeak® Version 2 SOAP Implementation Guidelines*, Arlington, VA: National Rural Electric Cooperative Association, 2003.
- [8] MultiSpeak Initiative Participants, *MultiSpeak® Version 2 TCP/IP Sockets Implementation Guidelines*, Arlington, VA: National Rural Electric Cooperative Association, 2003.
- [9] Gary A. McNaughton and Warren P. McNaughton, *MultiSpeak® 2 Users Guide*, Arlington, VA: National Rural Electric Cooperative Association, 2003.

VIII. BIOGRAPHY



Gary A. McNaughton (M’ 1976) is the Vice President and Principal Engineer for Cornice Engineering, Inc. He received a B.S.E.E. degree from Kansas State University in 1976 and an M.S.E.E. degree from the University of Colorado in 1980. Prior to joining Cornice in 1995 he worked as a Plant Electrical Engineer for Union Carbide, at the Oak Ridge Gaseous Diffusion Plant, at Oak Ridge, TN, as a Transmission Planning and Protection Engineer for Colorado-Ute Electric Association, a generation and transmission cooperative, located in Montrose, CO, and as Staff Engineer, Manager of Engineering, and Assistant General Manager for Engineering and Operations for La Plata Electric Association, in Durango, CO. Mr. McNaughton is a registered professional engineer in the State of Colorado and a member of IEEE.



Martin E. Gordon (M’ 1966, SM’ 1988) is a Senior Program Manager for Electrical Systems Research for the Cooperative Research Network of the National Rural Electric Cooperative Association (NRECA), in Arlington, VA. Mr. Gordon received his B.S.E.E. in 1967 from the University of Massachusetts, in Amherst, and his M.S. in Operations Research/Administration in 1975 from the George Washington University, in Washington D.C. Prior to joining NRECA, Mr. Gordon worked in the transmission and distributions branches of the Electric Standards Division of the Rural Electrification Administration, for the Engineering Department of the Edison Electric Institute, and for the International Business Machines Corporation (IBM) in the areas of computer component and digital design. Mr. Gordon has also served as a consultant in the electric power industry and as a Lieutenant in the U.S. Navy where he was an assistant project manager for research and development in the Naval Ship Engineering Center. Mr. Gordon is a registered professional engineer in the District of Columbia and a senior member of IEEE. He has also served as chairman of the Washington, D.C. section of the IEEE Power Engineering Society.