

DEVELOPMENT OF A REAL-TIME FRAMEWORK FOR ENTERPRISE INTEGRATION – NRECA’S MULTISPEAK®2 SPECIFICATION

Gary A. McNaughton, P.E.
Member, IEEE
Cornice Engineering, Inc.

Martin E. Gordon, P.E.
Senior Member, IEEE
National Rural Electric Cooperative Association

Abstract - Enterprise application integration has been an elusive goal for 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 small utilities from achieving successful interaction among operations applications.

The MultiSpeak® Initiative is a collaborative effort of software vendors sponsored by the Cooperative Research Network of the National Rural Electric Cooperative Association. During 2000, the Initiative developed a specification for common data interfaces among back-office applications widely applied at distribution cooperatives. The Initiative has developed extensions to that specification to include: (i) SCADA, AMR, OMS, LM, and customer relationship management interfaces and (ii) a messaging framework to support real-time communications among the supported applications, both operations and back-office. This paper discusses the development of the operations support application interfaces and the real-time messaging framework. It also discusses how these specification enhancements can facilitate cost-effective application integration in small utilities.

Key Words – Enterprise-wide integration, application integration, data integration, software interfaces, MultiSpeak®.

BACKGROUND

Integration of data-intensive automation applications has been an on-going challenge for electric utilities of all sizes. However, integration has been especially vexing for small utilities, which often do not have the information technology resources that are available to larger companies. The need for the efficient work processes that can result from integrated automation is typically greater at small utilities than at larger ones because of their leaner staffing levels. 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 small electric utilities, such as rural electric cooperatives. The Initiative has addressed this need by defining (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.

The use of standardized interfaces enables small electric utilities to obtain lower cost integration for their automation systems. As a result, such utilities will be able to improve staff efficiency and accomplish more for their customers with their existing staffing levels. In addition, the nature of the MultiSpeak interfaces permits the co-ops to focus their resources on improving utility operations rather than on costly maintenance of software 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].

This paper describes an enhancement to these early efforts that (i) addresses shortcomings of the MultiSpeak Version 1.1 specification, (ii) extends that specification to include interfaces for operations

applications, and (iii) includes near real-time communications using several widely-adopted communications methodologies.

MULTISPEAK APPROACH

A basic philosophy of the MultiSpeak Initiative is that vendors should be able to write a single, common interface to facilitate communication with another type of software. Such an interface should not vary substantially regardless of the vendor of the application with which their product is to integrate. Changes in the structure of one vendor's software should not require changes in the agreed upon interface. The chosen approach should be independent of computing platform or database structure. Finally, the approach should use common data definitions and agreed-upon data flows.

These requirements limit somewhat the means that can be used to achieve integration. Any scheme that requires native database access, such as structured query language or open database connectivity (ODBC), will not achieve the goals of the group. The approach chosen is for each vendor to write and maintain an interface that marshals the required information from their native data structure, converts those data into extensible markup language (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. Figure 1 illustrates this approach.

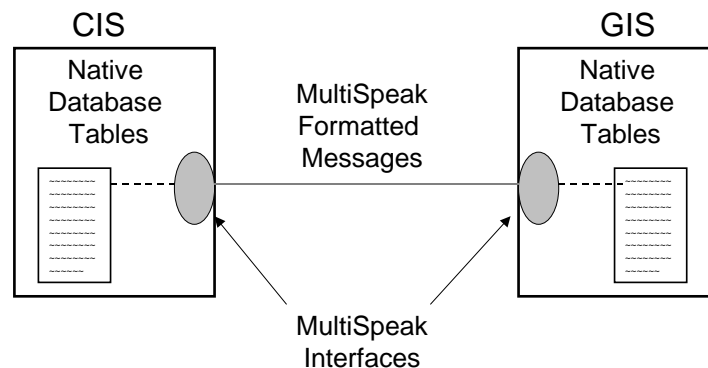


Fig. 1. MultiSpeak approach to common interfacing.

The MultiSpeak Initiative initially focused on five back office software applications: (i) customer information systems (CIS), (ii) geographic information systems (GIS), (iii) engineering analysis (EA), (iv) interactive voice response systems (IVR), and (v) automated staking. Version 1.1 of the MultiSpeak Specification [4] defined seven interfaces and developed a data dictionary for the information that could meaningfully be exchanged among these applications. Only batch file transfers were defined in Version 1.1. In addition, no provisions were made for incremental updates or for deletion of data. Furthermore, the transfer of graphical data relied on proprietary graphics protocols.

CHANGES IN MULTISPEAK VERSION 2.2

The MultiSpeak Version 2.2 specification (MultiSpeak2) [5] extends the coverage of the specification to include (i) supervisory control and data acquisition (SCADA), (ii) automated meter reading (AMR), (iii) outage management, (iv) load management, and (v) customer relationship management systems. Version 2.2 carries forward provisions for batch transfers where such make sense to support utility business processes, but also includes a messaging framework to support real-time integration. Both the batch and real-time messaging capabilities can support incremental updates and deletions with a high degree of granularity. The messaging framework is capable of supporting both request/response and publish/subscribe communications modes. Furthermore, the exchange of graphical data makes use of the Geography Markup Language (GML), an open-source, XML-based means to describe graphical features, which has been developed by the OpenGIS® Consortium, Inc. [6].

The MultiSpeak specification consists of four parts:

1) Process Model. The MultiSpeak process model identifies (i) the software functions inherent in the types of applications covered by the specification, (ii) data ownerships, (iii) data flows required to support integration between pairs of software functions, and (iv) the communications modes necessary to best implement the identified data flows.

2) Messaging Framework. MultiSpeak2 is implemented using a messaging paradigm. A comprehensive set of messages has been developed to facilitate the required batch or real-time communications.

3) Integration Scenarios. The combination of (i) the defined business processes and data definitions from the process model and (ii) the message structures included as part of the messaging framework, constitutes a set of interface definitions. Such definitions are also termed integration scenarios for information exchange.

4) XML Schema. The data objects and message components are encapsulated in data packets structured as extensible markup language (XML) elements. The actual data object definitions and the details of permissible message structures are documented in the form of an XML schema file.

APPLICATION TO FUNCTION MAPPING

The interfaces established in this specification are defined based on information flows between software *functions*, not individual software *applications*. This was done in order (i) to give flexibility to model a number of different product types and (ii) to minimize duplication in the specification.

It was important to give the specification the flexibility needed to model a number of different product types. The automated meter reading application is illustrative of that need. Some AMR systems have the capability to support two-way communications between the AMR server and a meter, and hence can query individual meters and implement service connect and disconnect remotely. AMR systems using one-way communications technologies usually cannot support such functions. However, both systems support meter reading functionality. Figure 2 shows examples of AMR systems that support different combinations of software functions. In that figure, AMR Product A supports only the meter reading software function, whereas AMR Product B supports the meter reading, load profile and remote connect and disconnect software functions. Thus MultiSpeak was defined to enable vendors to choose which software functions they wish to support in their applications but to give the utility software user a clear and unbiased means to evaluate the levels of compliance of competing products.

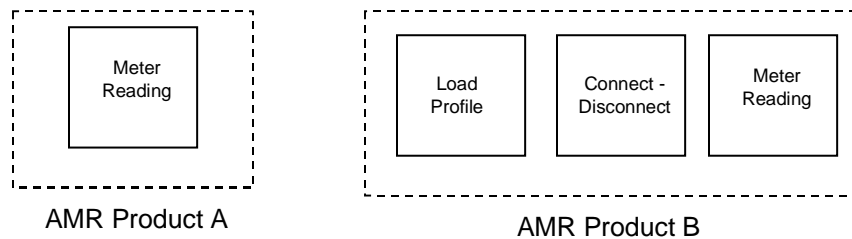


Fig. 2. Example of applications and supported software functions.

It was also important to eliminate duplication of interface definitions. An example of this situation is detection of outages. Detected outages can be logged using (i) a database application employed by customer service representatives to enter information taken from customer phone calls, (ii) an interactive voice response system that logs customer phone calls, (iii) meters reporting to an AMR system, or (iv) a system that deploys dedicated outage detection modules installed at the customer location. Regardless of the source of the outage notification, the data that needs to be communicated with another software application is identical. Hence, it would lead to redundant interface definitions if each of these systems had its own unique interface.

Table 1 describes the software functions currently defined by MultiSpeak. Each function is referred to by a function name; for instance CB is the function name for the customer billing database function.

TABLE 1
MULTISPEAK FUNCTIONS AND FUNCTION DEFINITIONS

| Function | Function Definitions |
|----------|--|
| CB | Customer Billing Database. This function includes a customer information database, customer billing and accounting for electrical usage. |
| CD | Connect/Disconnect/Power Limitation. This function controls remote switches or power limiting devices installed at customer services. |
| CH | Call Handling. This function manages in-coming and out-going calls. |
| CRM | Customer Relationship Management. This function enables tracking customer care activities for individual end-use customers. |
| DGV | Dynamic GIS Viewer. This function accepts and displays dynamically changing data in a geographic context. It is the intention of a dynamic GIS viewer to display changes in the status of data with no intentional time delay. Examples of this function are Automatic Vehicle Location or near-real-time display of outages. |
| EA | Engineering Analysis. EA is treated as a single function although it typically includes all engineering analysis functions, such as voltage drop and fault study capabilities. |
| FA | Finance and Accounting. This function includes the corporate accounting, accounting for time and materials, and work order accounting. |
| GIS | Geographic Information System. MultiSpeak treats GIS as a single function, although a GIS typically includes map editing/creation tools, printing tools and database editing/query tools. Any given vendor's GIS <i>application</i> may also include the capabilities of the dynamic GIS viewer function. |
| LM | Load Management. The load management function accepts information about required power system curtailments and manages those curtailments by communicating with remote devices, such as load control switches. |
| LP | Load Profile. The load profile function exchanges information about how metered loads change over time by saving information about such loads on a periodic basis. |
| MR | Meter Reading. This function collects information from remote meters, typically revenue meters, and presents it to other systems for analysis. |
| OA | Outage Analysis. The outage analysis function accepts outage information from Outage Detection (OD) sources. Such information is used to: (i) assist a human dispatcher to determine which power system devices have likely operated to create the observed pattern of outages, and (ii) facilitate outage reporting. |
| OD | Outage Detection. The outage detection function is broadly defined for the purposes of MultiSpeak; it includes any means by which information about outages is stored on a computerized server. |
| SGV | Static GIS Viewer. This function accepts and displays information in a geographic context. This function is used to display information that is not rapidly changing; hence real-time response is not necessary. |
| SCADA | Supervisory Control and Data Acquisition. SCADA provides status detection, logging of analog information, and control of remote power system equipment. |
| Staking | Automated Staking. This function provides field design and cost estimation capabilities. |

MULTISPEAK2 PROCESS MODEL

The MultiSpeak specification is based on a model of data ownerships and data exchanges that is intended to implement common business processes in small electric utilities. These relationships are outlined in Table 2. The first column of Table 2 ("Interface Number") indicates a data flow number for reference. The next column ("Linked Functions") indicates which two functions are linked on this interface. The third column ("Data Exchanged") summarizes the types of information that are to be exchanged. The column entitled "Direction" establishes the ownership of the data, that is to say, which function is the primary repository of the associated data and has the ultimate responsibility for its accuracy in the context of this defined interface. For instance, for interface 1, the owner of the customer data to be exchanged is the customer billing (CB) function and the consumer of the data is the load management (LM) function. This is implied by the designation "CB>LM" in the "Direction" column. The final column ("Communications Requirements") outlines the modes of communication that are defined for each interface. Three modes are used, batch (B), request/response (RR), and publish/subscribe (PS).

Batch communication implies that the function that owns the data sends information periodically in groups. In this mode there usually is a delay between the time that new information is available and when it is sent. It should be noted that even batch messaging could appear to an end user be "real time" if the messages are kept short and are sent sufficiently frequently.

TABLE 2
VERSION 2 MULTISPEAK PROCESS MODEL
DATA EXCHANGES AND COMMUNICATIONS REQUIREMENTS

| Interface Number | Linked Functions | Data Exchanged | Direction | Communications Requirements (*1) |
|------------------|------------------|--|----------------------------|----------------------------------|
| 1 (optional) | CB – LM | Customer information. | CB>LM | B or PS |
| 2A | CB-MR | Meter readings/characteristics, service status, phasing Customer, service, meter, and location info | MR>CB CB>MR | B, PS or RR B, PS or RR |
| 2B | CB-CD | Connect/disconnect and power limitation commands; customer, meter, service Acknowledge connect/disconnect, power limitation commands | CB>CD CD>CB | B, PS or RR B, PS or RR |
| 2C (optional) | LP-CB | Load profile data. | LP>CB | B or PS |
| 3 | SCADA-LM | Load shed schedules/commands, power factor correction signal | SCADA>LM | PS |
| 4 | MR-EA | Customer metered load, measurement, measurement device, phase (request by EA, respond by MR) Connectivity (location of meter electrically) | MR>EA EA>MR | RR RR |
| 5 | OD-OA | Status change event, device information Status (request by OA, response by OD) | OD>OA OD>OA | PS RR |
| 6 | CH-OA | Outage information, status, restoration information, estimated time to restoration (ETOR), call backs Is this service part of an existing outage? (CH request, OA responds) Revised call back status | OA>CH OA>CH CH>OA | PS or RR RR PS |
| 7 | GIS-OA | Connectivity file | GIS>OA | B |
| 8 | EA-SCADA | Analogs (volts, amps, MW, PF), device status (request from EA, respond by SCADA), temperature, time Connectivity | SCADA>EA EA>SCADA | RR B |
| 9 | SCADA-OA | Device status and transitions, SCADA analogs Presumed status of downstream devices | SCADA>OA OA>SCADA | PS or RR PS or RR |
| 10 | EA-OA | Connectivity file | EA>OA | B |
| 11 | FA-Staking | Work orders, units & materials Revised work orders, staked assemblies, pick list & CPRs (optional) | FA>Staking Staking>FA | B or PS B or PS |
| 12 | GIS-SGV | Existing facilities information and background graphics, customer, service | GIS>SGV | B (required) PS (optional) |
| 13 | | This interface has been intentionally omitted. | | |
| 14 | CB-CH | Customer, service, network & connect/disconnect, meter | CB>CH | B, PS or RR |
| 15 | CB-OA | Customer connect/disconnect, power limitation command Customer meter data, customer and service information | CB>OA CB>OA | B or PS B, PS or RR |
| 16 | CB-GIS | Customer, service, network, meter, transformer, pole, joint use, usage Transformer, pole, joint use, network | CB>GIS GIS>CB | B, PS or RR B, PS or RR |
| 17 | | This interface has been intentionally omitted. | | |
| 18 | OA-DGV | Outages to display | OA>DGV | PS |
| 19 | CB-EA | Billing account load information Connectivity | CB>EA EA>CB | B or RR B |
| 20 | | This interface has been intentionally omitted. | | |
| 21 | Staking-GIS | Existing facility information & background graphics Modified facility information & background graphics | GIS>Staking Staking>GIS | B or RR B or PS |
| 22 | EA-GIS | Connectivity Load flow and short circuit analysis results | GIS>EA EA>GIS | B or RR B or PS |
| 23 | SCADA-DGV | Device status and SCADA analogs | SCADA>DGV | PS or RR |
| 24 | GIS-SCADA | Connectivity & coordinates of nodes | GIS>SCADA | B |
| 25 | CB-OD | Customer, meter, and service information | CB>OD | B or PS |
| 26 | OD-DGV | Outage detection events, outage detection devices | OD>DGV | PS |
| 27 (optional) | OA-MR | Customers affected by outage, Outage events | OA>MR | PS |
| 28 | CB-CRM | Customers, services, transformers, meters, customer calls | CB>CRM | B |
| 29 | FA-CRM | Work orders | FA>CRM | B |
| 30 | LM-GIS | Load management devices | LM>GIS | B, PS |

Request/response communication implies that a client (data consumer) requests that a specific action be taken by the server (data provider); the server responds with a message outlining the results of that action. For instance, the client could request the current contents of a specific data item to be sent by the server. The server would respond with the required information or an appropriate error message. Request/response messaging is often used by the client in lieu of maintaining a separate copy of the subject data.

Publish/subscribe communication implies that the server (data provider) makes available (or publishes) data stored in its system to one or more clients, which subscribe to such data. Although publish/subscribe systems are often implemented using message queue middleware that supports point-to-multipoint messaging, this is not a requirement of MultiSpeak. The cost and complexity of messaging middleware is typically prohibitive in the small utility environment. Hence, MultiSpeak requires that a software function that acts as a publisher maintains a list of subscribers and sends multiple point-to-point messages to implement the equivalent functionality that would be offered by the message queue middleware.

Figure 3 illustrates the interfaces defined in Table 2. Note that each box in the process model (Figure 3) represents a software *function* (as defined in Table 1), not a software *application* or product. As illustrated in Figure 2 and discussed above for the AMR application, 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 defined interfaces shown in Table 2. Shaded ovals indicate interfaces that were defined previously in the MultiSpeak Version 1.1 specification. Unshaded ovals denote interfaces that are new in this version. The designation “optional” in either Table 2 or Figure 3 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.

MULTISPEAK MESSAGING FRAMEWORK

The MultiSpeak specification includes a comprehensive set of messages that can implement the batch, request/response, and publish/subscribe communications modes. Batch messages are simply a data packet wrapped with a pair of XML tags to denote the batch message.

Request/response messaging requires the use of two pairs of point-to-point messages, the Get/Show pair and the ModifyRequest/ModifyResponse (Modify) pair. The intention of the Get/Show messages is to permit the client to request data (in the form of a Get) from the server and to facilitate the return of the requested data in the form of a Show.

The Modify message pair enables data to be entered into one application where it makes sense from a business process perspective and to be updated in another application, even though the recipient is the defined “owner” of such data. For instance, customer telephone numbers could be updated from an outage management system while the dispatcher is talking with a customer reporting an outage, even though that data is normally owned by the CB function of a customer information system. Although this is a powerful capability, it could corrupt data on the server if used indiscriminately. Hence, the server application (the recipient of the ModifyRequest message) always has the option whether or not to change its stored data in response to such messages. Figure 4 illustrates the use of the request/response messages.

The publish/subscribe message set is by far the most involved of the communications modes; it requires a total of fifteen distinct messages, some of which may require point-to-multipoint delivery. Figure 5a illustrates a pair of messages that is used to communicate to a prospective subscriber (client) which information is available from a specific publisher (server). The client requests the information using a PublisherServicesRequest message; the server responds using a PublisherServicesResponse, which contains information about which (i) versions of the MultiSpeak specification, (ii) MultiSpeak functions (see Table 1), and (iii) items of information are supported by the publisher. Figure 5b shows the message set that is used to initialize a data subscription. Using the SubscribeRequest, a client can implement a subscription to any of the supported data items using flexible criteria chosen to limit the information returned to that necessary for the client’s application. Once a subscription is active, the publisher sends any changes in the requested data to the client using a PublishedData message, which the subscriber must acknowledge using a PublishAck message (see Figure 5c). If desired, the server can also require the client to confirm the receipt and proper posting of the received data to the client’s database using the MessageConfirmation message. Figure 5d shows a set of miscellaneous messages that (i) enable the client to request a subscription cease

(the UnsubscribeRequest/UnsubscribeAck message pair), (ii) enable the server to notify the client in a controlled manner that it is being unsubscribed unilaterally (the UnsubscribeNotice message), and (iii) a set of messages that enable each participant to check whether the other is still active (the SubscriptionPing message pair).

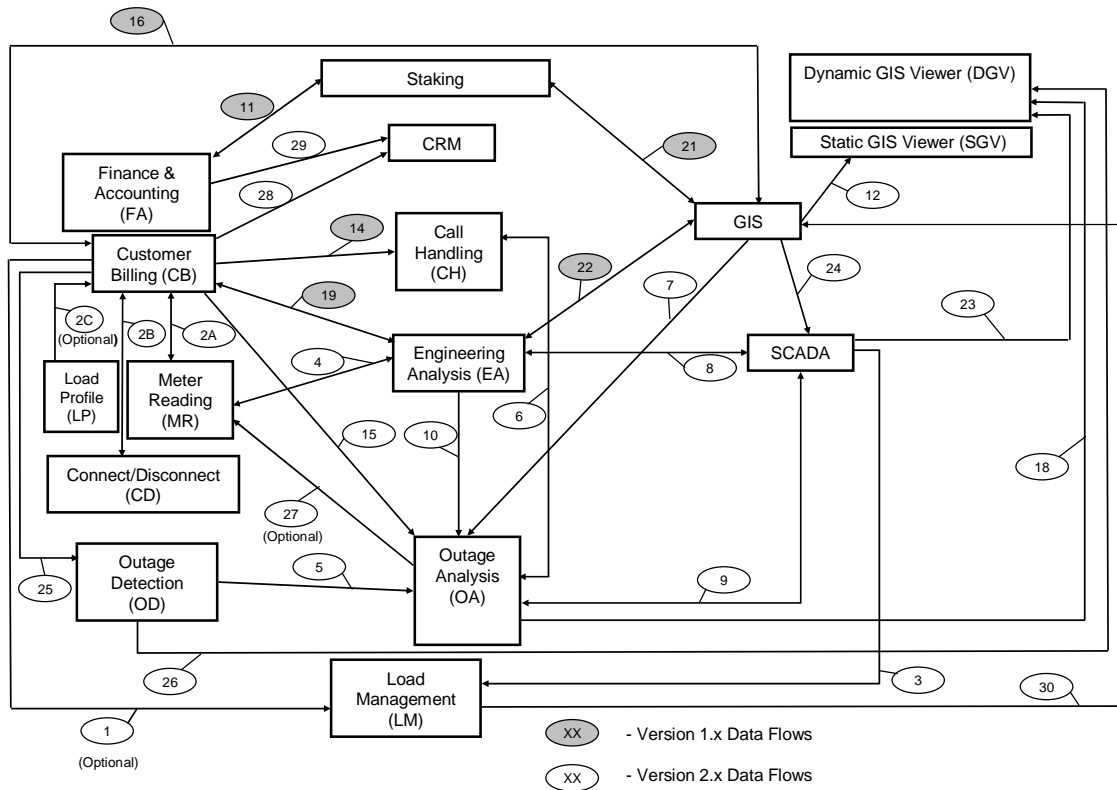
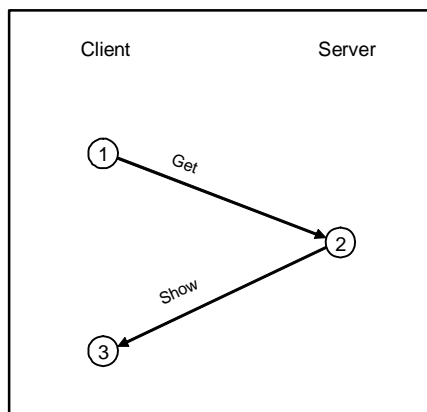
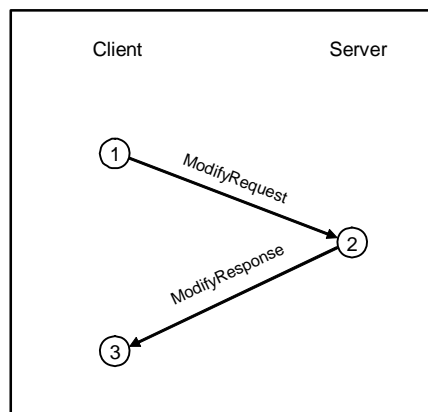


Fig. 3. MultiSpeak version 2.2 process model.



(a) Get/Show Message Pair



(b) ModifyRequest/ModifyResponse message pair

Fig.4. Request/response message set.

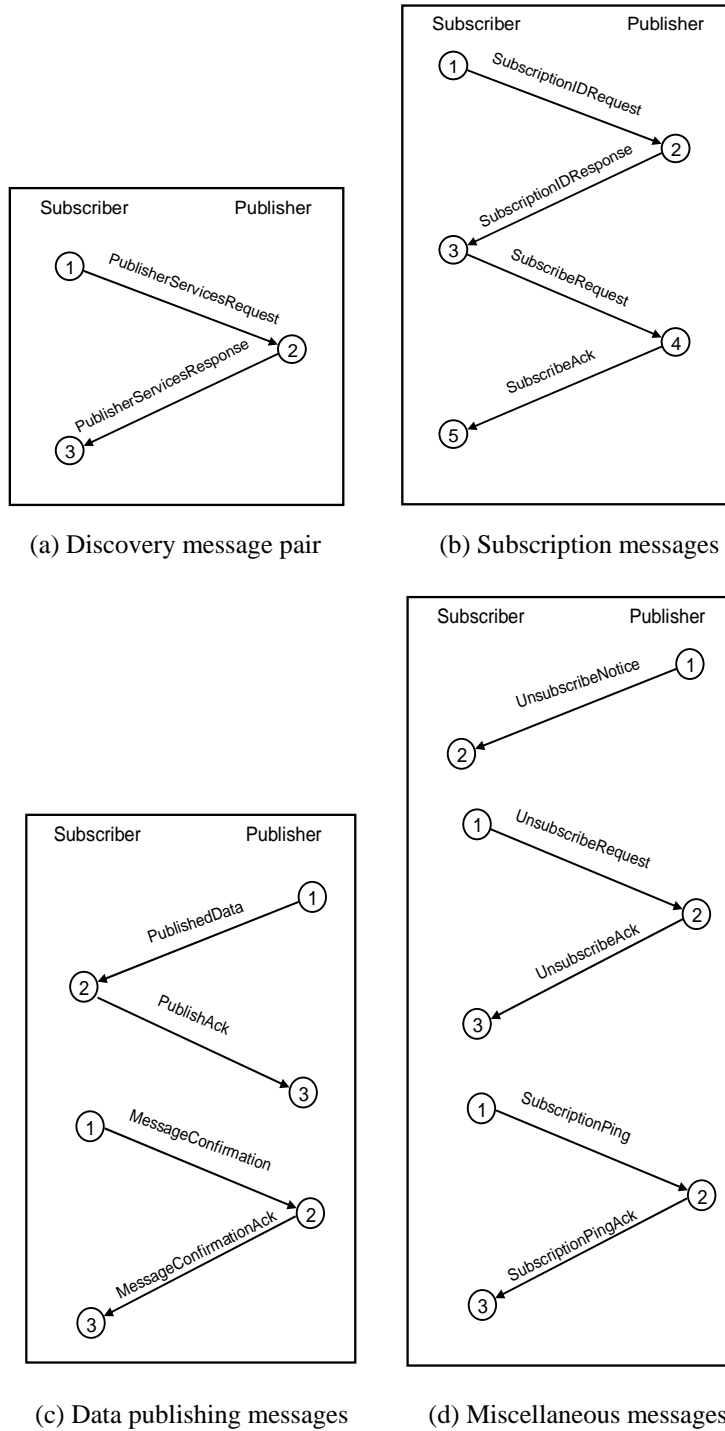


Fig.5. Publish/subscribe message set.

COMMUNICATIONS TRANSPORT OPTIONS

Communications transport options are means by which MultiSpeak messages may be exchanged between two MultiSpeak-compliant applications. Three communications transfer options are currently supported by the MultiSpeak Initiative, (i) file-based, (ii) SOAP, and (iii) sockets. Although support of multiple, incompatible transport options makes specification of MultiSpeak-compliant applications more

difficult for the software user, the inclusion of three options makes it possible for all participating vendors to choose a transport that applies to its software and computing platform. One vendor already offers a communications hub that provides interconnection of applications that use otherwise incompatible transports.

In the context of the MultiSpeak specification, a data transfer is termed real-time if it is completed as soon as possible after a change in data occurs; that is to say with no intentional time delay. Both the SOAP and sockets transfer options are considered to be real time. On the other hand, in file-based communications, data is typically buffered and sent in batches; hence it is sometimes called “batch” communications.

Table 3 indicates which of the three communications transport options (file-based, SOAP, and sockets) supports the three required communications modes defined in Appendix B (batch, request/response and publish/subscribe).

TABLE 3
COMMUNICATIONS MODES AND COMMUNICATIONS TRANSPORT OPTIONS

| Communications Mode | Communications Transfer Options | | |
|---------------------|---------------------------------|---------|------|
| | File-Based | Sockets | SOAP |
| Batch | X | X | X |
| Request/Response | | X | X |
| Publish/Subscribe | | X | X |

A brief description of each of the supported transfer options follows.

File-based communications. File-based transfer occurs when one software application writes data in the form of a file stored in a file space accessible to another application. The receiving software program subsequently reads the data from that file and takes the required actions based on the contents of the file. In the case of MultiSpeak messages, the file contains the data packet that needs to be sent between the two systems. The programs communicate asynchronously and no dedicated connection between them is required. File-based communications is analogous to an e-mail message.

SOAP communications. Simple Object Access Protocol (SOAP) is a means to exchange XML-formatted data using standard Hypertext Transfer Protocol (HTTP) delivered over a Transmission Control Protocol/Internet Protocol (TCP/IP) network. The MultiSpeak-formatted data packet, similar to the packet sent using either the file-based or sockets options, is encapsulated in a text wrapper, called a *SOAP envelope*, and sent with the same technology used to implement web servers. SOAP supports real-time data transfers. It requires both applications to be running and communicating at the same time, but a permanent connection between the two systems is not necessarily required. SOAP is analogous to the communication that occurs between a web browser and a web server.

Sockets communications. TCP/IP sockets are a well-established means for two programs to communicate in real time using a persistent connection. The sockets option relies on TCP/IP network protocols to stream the MultiSpeak data packet across this connection. Sockets connections are analogous to telephone conversations; both parties must communicate simultaneously using a connection that is maintained until communications is terminated by one party. The data payload is identical to that used in the SOAP option and nearly identical to that used in the file-based option.

A COMPARISON OF FILE-BASED AND REAL-TIME COMMUNICATIONS

Although file-based communications is thought by some as being cumbersome or insufficiently frequent to meet modern data exchange requirements, it should be remembered that file-based transfers that are automatically scheduled so that no human intervention is necessary, and that happen sufficient frequently, are indistinguishable from data exchanged in “real-time” to the end-user of the information. File-based transfer is simple and reliable. It may serve the complete needs for data exchange for some utilities, and is likely to play a role for some interfaces in most utilities.

Having said this, there are some business processes, for instance, outage management, that are inconsistent with the time delays inherent in file-based communications. For MultiSpeak interfaces that support such business processes, the real-time transfer options (SOAP or sockets) are more appropriate. Figure 6 indicates (i) which interfaces are defined in MultiSpeak 2 for both batch and real-time communications (those shown in solid lines in Figure 6) and (ii) which will support the underlying business processes only if sent in real time (those denoted by dashed lines), and hence are defined in MultiSpeak 2 for real-time transfers only.

Examples of business processes that may be adequately supported with batch communications include:

- Establishing a units and materials database in an automated staking system from a work order accounting system (MultiSpeak function=FA).
- Establishing an initial geographic display in a SCADA system database from a GIS.
- Updating a laptop GIS viewer from a GIS server.

Examples of business processes that require real-time transfers of data include:

- Displaying actual SCADA device status or outage status in a GIS.
- Intelligent outage and outage call handling, using SCADA device status.
- Maintaining up-to-date outage detection status for outage analysis.
- Performing engineering studies based on actual system conditions and loads.
- Queries of AMR units to determine status of a customer service.
- Sending real-time signals to load management switches.
- Real-time crew dispatch and crew location.

EXTENSIONS TO MULTISPEAK

MultiSpeak has been defined in such a manner that it can easily be extended to cover (i) new data objects, (ii) new fields in existing data objects, and (iii) data interfaces that are not currently included in the specification. The MultiSpeak object, which includes all objects defined in the specification, can be extended to include new objects using an extensions class. Furthermore, each currently defined object has its own extensions class so that data fields not currently part of the specification can be added. This flexibility enables software vendors to extend their interfaces to deal with unique circumstances or to implement features that differentiate their products without affecting the core standardization.

In addition, it should be noted that the inclusion of the request/response and publish/subscribe messages enables communications between programs in a more generalized way than is immediately obvious. Clearly, MultiSpeak has been defined to support a number of software types. However, programs for which no interface is defined, for instance work management, asset management or document imaging, may obtain data from any software application that acts as a server in any MultiSpeak defined interface by using the messages defined for communicating with that server. For instance, a work management system could obtain work orders requiring crews from the FA function or outages to be worked from the OA function. In addition, software that makes use of data objects defined in MultiSpeak in ways not covered by the specification may also use those data object definitions to minimize the work necessary to communicate with a function covered by MultiSpeak. For instance, a meter test board vendor could pass the meter nameplate information to a customer billing (CB) function using the currently defined meter object.

CONCLUSIONS

The MultiSpeak Version 2.2 specification provides a robust, flexible means for vendors to provide standardized interfaces for a wide variety of software products commonly applied in small electric utilities. The latest version of the specification includes support for operations software, such as SCADA, AMR, outage management, and load management as well as real-time inter-program communications. The MultiSpeak specification will reduce the time necessary for vendors to develop new software interfaces and, once MultiSpeak2 interfaces are in use, will reduce the costs of maintaining those interfaces. This will

speed time to market and reduce the development costs for vendors and hopefully reduce the costs of integrated automation systems for small utilities.

The participants in the MultiSpeak Initiative have designed the specification to handle the data exchanges most commonly used to support co-op business processes. However, if groups of vendors wish to provide extended capabilities that require data objects or data fields outside those defined in the specification, MultiSpeak can still provide a foundation that easily can be modified to incorporate those extensions.

For more information about developments of the MultiSpeak specification and a complete listing of the participating vendors, see the MultiSpeak web site at www.multispeak.org.

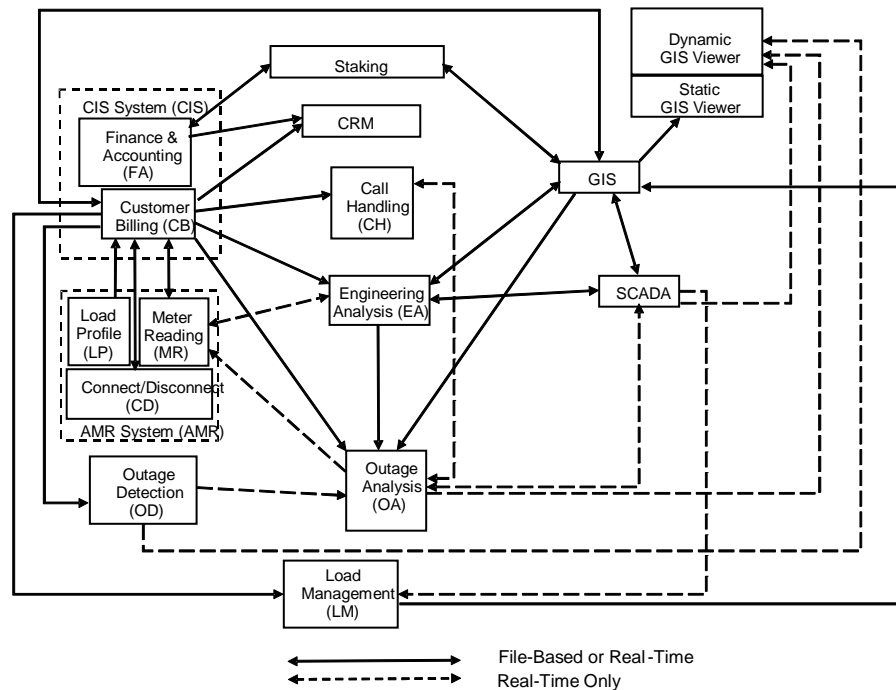


Fig. 6. Supported communications transfer options, by interface.

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 1.0*, Arlington, VA: National Rural Electric Cooperative Association, 2000.
- [5] MultiSpeak Initiative Participants, *MultiSpeak ® Version 2.2 Specification*, Arlington, VA: National Rural Electric Cooperative Association, 2003.
- [6] Open GIS Consortium, Inc., *OpenGIS® Geography Markup Language (GML) Implementation Specification, Version 2.1.2*, OpenGIS Consortium, www.opengis.org, 2002.
- [7] Gary A. McNaughton and Warren P. McNaughton, *MultiSpeak® 2 Users Guide*, Arlington, VA: National Rural Electric Cooperative Association, 2003.

BIOGRAPHY

Gary A. McNaughton 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 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.