An XML Model for Use Across Heterogeneous Client–Server Applications

Suvendi Chinnappen-Rimer and Gerhard P. Hancke, *Senior Member, IEEE*

***Abstract*—Applications that use directory services or relational databases operate in a client–server model, where a client requests information from a server, and the server returns a response to the client. These client–server applications typically have a specific message protocol that is unique to that application. Systems with multiple client–server applications require that there are separate client programs that individually communicate with their respective server programs. A need exists to access information from heterogeneous systems in a standard message request–response format. A generic eXtensible Markup Language (XML) model was developed to obtain data from diverse measurement systems. The objective of this paper is to describe the XML model that abstracts the differences in the underlying heterogeneous client–server message formats and provides a common XML message interface. The XML messages are parsed through a common XML gateway that decides to which application server to forward the messages. The generic XML messages are translated to the correct application server format before being sent to the application server.**

***Index Terms*—Client, eXtensible Markup Language (XML), generic, message, server.**

I. Introduction

INTERNET technologies are increasingly being used to monitor and manipulate remote electronic devices. One area that is receiving widespread attention is the use of Internet protocols in field area networks (FANs) to facilitate services such as remote monitoring, control, and maintenance [1]. A FAN connects nodes that are the access points that are responsible for data acquisition and its short-term storage [2]. The aim of this paper is to provide an eXtensible Markup Language (XML) model to create a standardized set of interfaces, which are interoperable with a variety of client software and hardware [2].

In a typical traditional client–server environment, applications operate on a request–response mechanism, i.e., the client applications request information from a server application using either a private or a public network to transport messages (requests and responses) between the client and server applications. Each server application required the use of a separate client interface.

Manuscript received May 3, 2007; revised January 18, 2008. This paper was presented in part at the International Symposium on Virtual Environments, Human–Computer Interfaces, and Measurement Systems, Lugano, Switzerland, July 2003.

The authors are with the Department of Electrical, Electronic, and Computer Engineering, University of Pretoria, Pretoria 0002, South Africa (e-mail: gerhard.hancke@up.ac.za; g.hancke@ieee.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIM.2008.920027

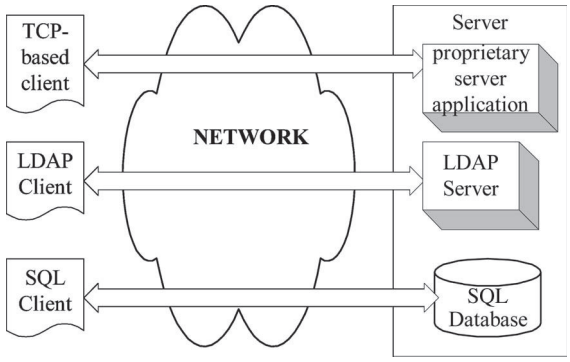

Fig. 1. Traditional client–server architecture.

For example, consider a typical client–server architecture that uses the Internet as the message transport network. The system has the following applications/services running on it:

1) a directory service [e.g., Lightweight Directory Access Protocol (LDAP)];
2) a Transmission Control Protocol (TCP)-based proprietary server program;
3) a relational database server.

The traditional architecture would require separate client programs that access each of the server or database applications, as shown in Fig. 1.

If a new server application is added, each client needs to load another client program that can access the server's data. The need to install specialized clients on workstations increases the complexity of maintaining a system and reduces the flexibility to introduce new protocols into a system. Any addition of new software may require the addition of new shared libraries that may complicate or interfere with existing applications (such as stability, versions, etc.). It is preferable to deal with changes at a single server than having to update multiple clients. The placement of an application-layer gateway between the field bus and the IP-based network [3] results in a networking solution that increases the server-side complexity while decreasing the client-side complexity.

The XML is evaluated as a solution to the problem of accessing data across diverse client–server applications through a common message model. Fig. 2 provides an overview of the proposed system architecture using XML as a common message format between multiple client–server applications.

This paper is structured as follows. Section II provides a brief overview of the background and context in which this research was undertaken, as well as a brief overview of current research focusing on using XML as a common message interface. In

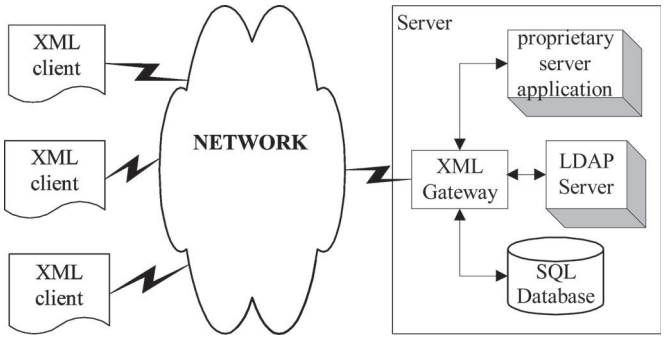

Fig. 2. Generic message gateway using XML.

Section III, the similarities between client–server applications are described. In Section IV, the design of the XML model is described. In Section V, the various solutions considered in identifying the type of application server to which to send a message are highlighted. In Section VI, the various alternatives are evaluated. Section VII provides a brief overview of the actual implementation. In Section VIII, the results of using XML as a generic message interface instead of the native protocols are analyzed.

II. Background and Related Work

One of the problems experienced by electricity providers in many countries is that of meter tampering and the intimidation and coercion of human meter readers to change their measurements. This paper is part of a larger work that is attempting to remotely obtain electricity usage readings (among other types of measurements, such as heat, gas, or water) [1]–[3]. Small electricity measurement devices with internal microcomputers are connected to a proprietary field bus. These devices are controlled from a central server that communicates with these devices.

This paper focuses on the need to allow multiple users to remotely access these data (from various sources) without requiring specialized software to be installed on their computers. The research described in this paper forms part of a project that provides a secure gateway between the Internet and a private field bus network [1]–[3]. The Internet is used to provide remote access to the field bus network data. The gateway provides access to data in FAN node using either a proprietary TCP-based protocol, an LDAP server, or a database server.

The objective is to develop a common message interface to send requests and receive responses from different client–server-type applications. XML was used to develop the common message interface because the interpretation of XML data is entirely up to the application that processes it. A generic model was developed to obtain data from diverse measurement systems. The type of client–server applications considered send requests and receive responses in text-based format and has mechanisms to read, modify, insert, and/or delete data.

This paper and related research is not focused on Web Services. XML is a data description language being developed and independently standardized by Web Services. XML is currently being used for Web Services as well as other applications that are not related to Web Services.

Attempts to solve the problem of restructuring and reformatting of data as it passes from a software tool or process to another predate the widespread use of the Internet that started in the mid 1990s. Blattner *et al.* [4], [5], in a study on generic message translation, attempted to solve the problem by providing a visual interface that can create a mapping between fields in different message types that specifies which fields have similar semantic content. The papers of Blattner *et al.* were published before the introduction of XML into the computing landscape. However, in their paper, the authors conclude that some sort of "parser generator" must be constructed to take descriptions for data specifications and create a "translator" between systems.

Since the introduction and standardization of XML, several studies have been undertaken on the feasibility of using XML as a means to either provide an interface between legacy applications and the Internet or describe data in a standard format.

A study to use XML as the wrapper interface in migrating legacy applications to the browser-based Internet platform conducted by Bi *et al.* [6] focused on understanding the functionality of the legacy system and the user interaction in the legacy system. Bi *et al.* developed a thin web-based client to interact with the legacy system that was wrapped within an XML application. The focus of the paper was legacy applications, but it demonstrated the effectiveness of using XML to interact with multiple legacy applications.

The work of Peinl and Mitschang [7] investigates transforming independent autonomous data sources into a common XML format to provide an integrated communication platform for mobile applications.

Both works acknowledge the advantages provided by using XML as the data modeling and exchange mechanism between applications (clients) and information sources (servers).

The work of Law [8] describes an attempt to use XML and LDAP messages to describe the network database schema, with the intention that modification of the database information could be achieved without taking down the whole system. The author reached the conclusion that XML's extensibility results in increased flexibility in setting up data content according to different applications and/or different vendors. This conforms to the results of our research, which shows that if the main elements are defined in a common schema (during the design of the XML document), the subelements within the schema can be extended according to the specific needs of the application.

However, the use of XML for generic messaging does not come without some disadvantages, i.e., slower processing speed caused by the additional overhead of using XML to transform and parse messages. Bhoedjang *et al.* [9], in their study of distributed data structures, conclude that the performance measurements of applications that run application-specific codes are faster than those that use generic message-passing software.

The possible advantages of using XML (such as scalability, interoperability, and portability) to describe instrumentation

data for use in developing a virtual laboratory are described by Bagnasco *et al.* [10]. The XML was used to describe each instrument that could be remotely controlled. The authors concluded that initial experiments were promising and that use of XML plays an important role in environment definition and process maintenance.

III. Identification of Similarities Between Heterogeneous Applications

To design the model, the similarities of different client–server applications were identified, and using these similarities, a common messaging system using XML as the command interpretation language was developed.

In typical client–server applications, the following similarities can be identified.

1) The client requires connection information about the server, such as the host name on which the server resides, the port number on which the server is listening, the context or directory in which the server is located, the data source name and database-specific driver details, etc.
2) The client has to provide the server with authentication details, so that the server can verify that the client has access to the information that the server will be able to provide. This is for security reasons so that rogue client applications that may have malicious intent are not allowed to gain access to the information that the server provides. The typical authentication information required is a user name, a user password (or access code), and the role of the user (i.e., some users may have more privileges to information than other users).
3) The messages sent between the client and the server are of the request–response type. The message sent from a client is a request, which contains some specific command name used in the application. The message sent from the server is a response to the specific command.

IV. XML Model

The model was designed by initially focusing on how messages are sent and received between a client and the server. For example, a typical client–server application using TCP as the message transport mechanism will function as follows.

1) A server will stay in a listen state, which means that the server application listens for input data on a specific port.
2) The client will send a connection request to the server.
3) The server will validate the client's authentication details.
4) If the client's authentication details are valid, the server will inform the client that the connection is accepted.
5) Otherwise, the server will inform the client that the authentication details are invalid and that the connection will not be established.
6) The client can send a message (command) to the server.
7) The server will process the request and return a response to the client.
8) After the client has processed all requests, it informs the server that the connection will be closed.
9) The client then closes the connection.
10) The server remains in the listen state, in case it has other clients that are still connected to it or may want to connect to it.

From the above description and using the similarities identified in the previous section, an XML model that includes the following elements was developed.

Connection Information, i.e.,

```
<connection-info>
  <connection-url> ... </connection-url>
</connection-info>
```

Authentication Information, i.e.,

```
<authentication-info>
  <auth-name> "..." </auth-name>
  <auth-code> "..." </auth-code>
  <auth-role> "..." </auth-role>
<authentication-info>
```

Request message with a specific command

```
<request>
  <command> "..." </command>
    ... [command parameters] ...
</request>
```

Response message to a specific command

```
<response>
  <command> "..." </command>
    ... [response details] ...
</response>
```

The command names and parameters in the messages sent between the client and server element's value can be the specific command name used in a server application. The XML schema does not specify what the child elements of the command element should be. The reason is that each client–server application has specific commands with specific fields. Each element is stored as an element value pair in a hash table that is sent to the relevant server application. The value of the command element is irrelevant, because the XML gateway application assumes that the server application to which the command value is sent will be able to correctly interpret and process the command.

The advantages of not specifying the elements within a command element are as follows.

1) Each client–server application can retain their original command names without having to create a generic set of command names that applies to similar types of commands between the different applications.
2) Server applications with different commands and command parameters that were not considered in the original design can use the same schema.
3) The XML gateway program that processes the XML document does not need to know what the required child elements of the command element are.

V. Routing Messages to the Correct Application Server

After parsing the XML message received from a client application, the XML gateway requires a method to determine the correct application server to which to send the messages. The following three methods were considered:

1) using a unique address location mechanism;
2) using a specific XML element;
3) trying the first available (connected) server application specified in the connection URL.

These three possible options are described below.

A. Option One

Many request commands require (as an input parameter) some sort of unique address or field that indicates where the data can be located. This address may be a directory location, a database table and/or field name, a file name, a network node and file name, and many other possible location-type variables.

Option one requires that a user prefix the unique data location value with a unique code that identifies the server application for which the command is intended. The server ID string is prefixed to the start of the data location value and separated from the actual data location value by a dot (“.”), i.e., in a similar format to the structure of an IP address. The address content is used in the implementation to determine to which server application to send the message. The first part of the address before the end punctuation point (left side of the address taken as beginning of address) indicates the type of server application. For example, if the message address is “SQL.SYS.GATEWAY@DataHostAddress!gateway,” then the characters before the first punctuation point are “SQL.” This indicates that the request is intended for the database server specified in the connection URL.

B. Option Two

Option one works well where the command requires input data that can be used to identify the application server to which the request is intended. However, some commands may not require any input data.

To solve this problem, option two considers including an additional element (the protocol element) to the XML model that will identify the server application that is a child element of the request element. This means that all commands within the request element will be directed to the identified server application, i.e.,

```
<request>
  <protocol> LDAP
  <command> read </command>
  <command> modify </command>
  </protocol>
</request>
```

This approach requires that the client application specify, in the creation of the XML document, the server for which server the requests are intended.

C. Option Three

Option three attempts a connection to each server application specified in the connection URL. The request is sent to the first server application that is available and the result returned to the client.

VI. Analysis of the Three Options

Option one uses a unique identifier to determine the server application. The advantage of this approach is that a single user interface can be used to send and receive messages to multiple application servers. The drawback is that the user has to enter a unique application identifier when entering in the location of the data.

Option two uses an additional element in the XML request document that describes the application server for which the message is intended. The advantage of this solution is that the user is not required to specify a unique identifier per command request. However, the disadvantage is that there has to be a separate user interface per application server so that when it formats a message request, the client application adds the type of application server for which the message is intended as the appropriate protocol element value.

Option three works well if there is only one server application up at the time and if it happens to be the server application to which the client intended the request to be sent. However, if these exact requirements are not met, then the flaws in this solution become apparent. For example, if more than one server application is available, the XML gateway may send the request to the incorrect server application. Therefore, option three is not considered as a viable solution.

The XML model uses the dual approach of options one and two. This provides us with the flexibility of not requiring that all commands have a data location identifier element or that all requests require a protocol element.

Option two is suitable for batch-type interactions, where the server identifier can be passed as an input parameter to the batch

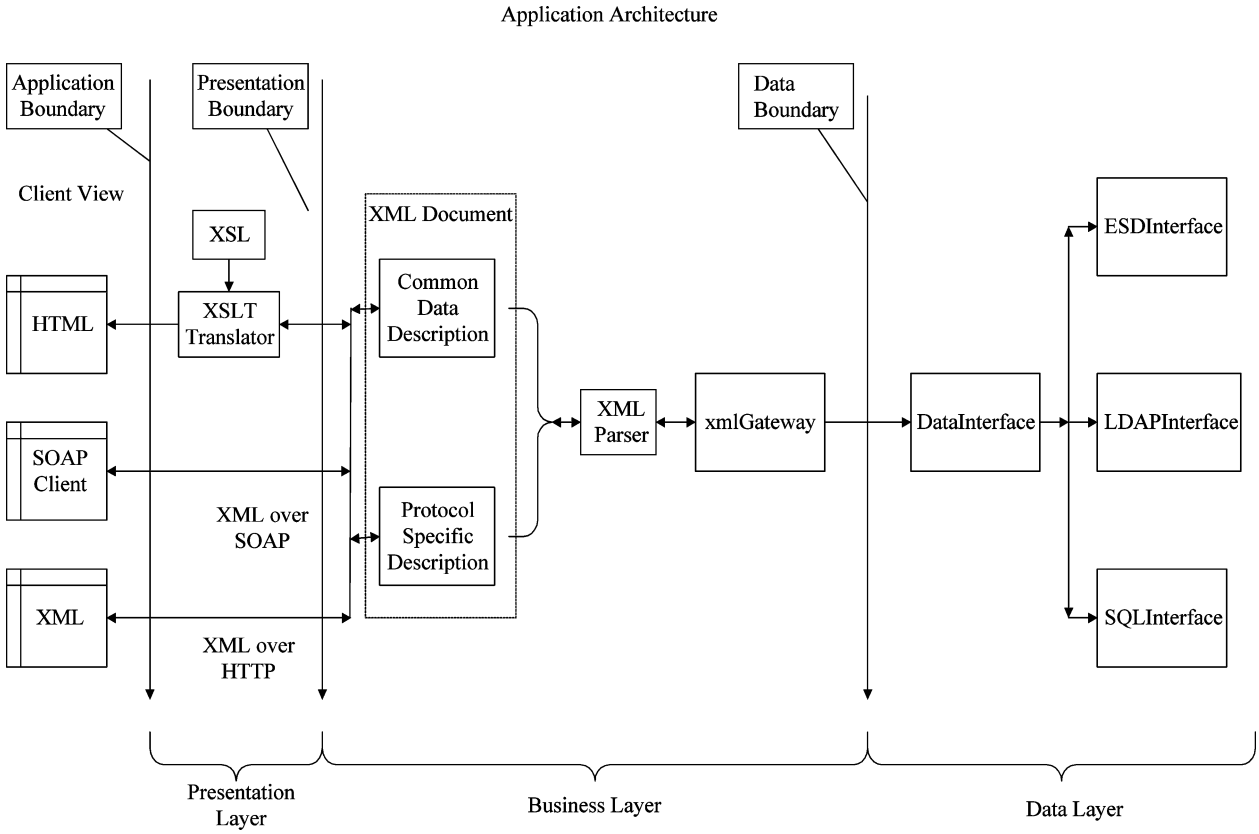

Fig. 3. Application architecture.

application when it is run. This negates the need for separate client applications on the client workstation.

Option one is suitable for user interfaces such as web browsers, where a common user interface can be used to send messages to multiple server applications.

VII. Implementation

The application architecture consists of the following major components:

1) the presentation layer: user interface consisting of HTML and JavaServer Page (JSP) forms;
2) the business layer: processing and interpretation of XML document;
3) the data layer: provides access to specified data source.

Currently, the business and data instructions are processed on a single server machine, but the data sources can be moved to another server without impacting the application. In addition, most of the presentation preprocessing is done on the server so that the client is presented with only HTML-type forms that require user input or that display the server response to a request command. The application architecture is shown in Fig. 3.

The proposed XML model was used as the messaging mechanism for the three systems described in the introduction, i.e., a directory service, a proprietary TCP server program, and a relational database server.

A browser-based user interface to the server applications was implemented for option one. The user was required to enter the parameters required to process a specific command. The user interface approach is suited for option one because the server identifier can be prefixed to an input data field's value. A transforming language [i.e., eXtensible Stylesheet Language Transformation (XSLT)] was used so that each server's response could be specifically formatted for that application.

The XML messages are directly sent to the XML gateway to test option two. The server identifier is parsed as an input parameter to the batch program. The batch program uses this input value when creating the XML document to send to the XML gateway. The XML document includes the additional protocol element enclosing the server identifier.

The server application is written in Java, and the server uses the Linux operating system. The software occupancy requirement on the client side is small, i.e., they require support for either a web browser (option one) or a small batch program (option two). The batch program need not exceed 20 000 B. This means that the client application is not restricted to being a computer workstation but can be a mobile device such as a personal digital assistant. The design does not require any XML programs to be stored on the actual meter readers. Data are requested from the meters and stored in different data sources on the server. The current server-side software occupancy requirement for the XML application is less than 130 kB. The main resource users are the data sources. The data sources need

TABLE I
TIME DIFFERENCE FOR CLIENT–SERVER APPLICATION WITHOUT XML AND WITH XML

Command	Proprietary Application (msec)	Database Application (msec)	Directory Server (msec)
READ (no-XML)	12	19	141
WRITE (no-XML)	5	19	138
READ (XML)	522	406	495
WRITE (XML)	172	228	430

TABLE II
MEMORY USAGE DIFFERENCES FOR CLIENT–SERVER APPLICATION WITHOUT XML AND WITH XML

Command	Proprietary Application (bytes)	Database Application (bytes)	Directory Server (bytes)
READ (no-XML)	53288	10768	16176
WRITE (no-XML)	52528	11856	15584
READ (XML)	483496	193928	514184
WRITE (XML)	216640	175400	242280

not run on the same server. To run all three data sources on the same server (a web server), the application requires 256 MB of random access memory.

VIII. EVALUATION OF XML VERSUS NON-XML MESSAGES

Option one proved effective in reducing the need for separate client interfaces. In addition, by using a transforming language such as XSLT, each server's response can be specifically formatted for that application without increased complexity on the client side.

Option two is effective when used within a batch client application, as it reduces the need for different client programs while ensuring that requests are sent to the appropriate server application.

The performance of XML versus non-XML messages was assessed, and the results were analyzed.

Table I shows the result of the time taken (in milliseconds) to send and receive a non-XML message and an XML message to each of the different server applications. Similar read- and write-type commands are used across the different client–server applications.

From the results, it is clear that non-XML-type messages provide better performance results than XML-type messages. The additional time is due to the construction of messages into XML format and the parsing of the XML messages to determine the type of command and application server.

The memory usage of XML versus non-XML messages was assessed, and the results were analyzed. Table II shows the amount of total occupied memory used when transmitting non-XML messages versus memory usage when transmitting XML messages to each of the different server applications. Similar read- and write-type commands are used across the different client–server applications. Because the XML messages have a larger footprint in terms of additional message descriptors and use of the parser, they will require more memory. It should be noted that the values in Table II are also dependent on the underlying data source and how it handles memory resource allocation and recovery.

The time taken to process a message does not increase in a directly proportional manner as the number of messages increase, i.e., the time taken for two messages is not double the time taken for one message. As shown in Fig. 4, the time per XML message, as the number of messages increases, tends to a constant level per request/command.

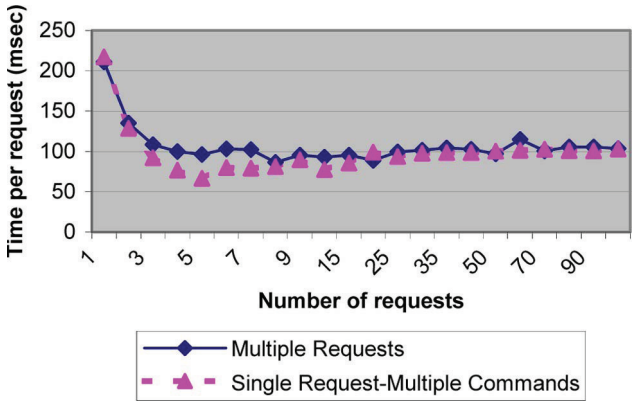

Fig. 4. Time taken to process an XML command as the number of commands increase.

The time taken to process a single request with multiple commands within the request element is slightly smaller then the time taken to process multiple request–command messages. This is because connection info, request, and other higher order elements do not have to be parsed for each command. Because the number of elements preceding a command element is small, the time taken before reaching the command element is relatively small. Therefore, it can be concluded that the XML parser is efficient and may be able to handle larger XML documents with minimal additional performance cost.

There is clearly a tradeoff between the amount of time it takes to send a request and receive a response and the flexibility of providing a common message interface to multiple applications.

The following advantages of using XML in the implementation were identified:

1) standard message format for multiple applications, i.e., XML is application independent;
2) common gateway handles requests to multiple application servers;
3) flexibility in extending the XML document structure to incorporate new application servers with minimum additional changes to existing coding infrastructure;
4) additional security benefit of having only one access point [HyperText Transfer Protocol (HTTP) port] to multiple applications made available to external networks.

The following disadvantages of using XML in the implementation were identified:

1) slower response times, leading to decreased performance;
2) increased central processing unit usage;
3) increased memory resource usage.

It should be noted that these measurements were done on a single computer, i.e., no network transmission overheads affect the results. The choice of the right type of message must take into account the size of the message and the transmission delays of sending data across a network.

IX. Conclusion

A common message model for sending and receiving messages between heterogeneous server applications has been designed and implemented using XML as the data description language. Similarities between client–server applications were identified and used as the basis for defining the common XML elements in the schema.

After the design was implemented and tested, the performances of XML and non-XML messages were evaluated. As expected, the increased verbosity of XML results in a larger footprint that requires more processing time and resources. This means that any implementation using XML has to carefully weigh the benefits of flexibility, extensibility, and standard message formats against reduced performance.

XML does not appear to be suitable for applications that require high-speed, real-time responses. However, client applications that use the Internet to obtain server information from multiple applications will benefit from reduced client-side complexity. Server applications that serve large client bases and, therefore, require smaller resource allocation per request may not be scalable because of the integration with XML. The reduced performance levels from using XML mean that it does not scale to handle large numbers of concurrent client requests.

If the applications are run in batch mode, where the need for fast (i.e., microsecond and nanosecond) responses is not important, then the solution is useful. As long as the time delay is not too long in human (user) terms, then the additional response times caused by the XML footprint are negligible. Note that, in this implementation, the time delay for responses is less than or is within seconds. However, if the data sources are located on different servers, the time delay will be dependent on the network performance.

Therefore, it can be concluded that when used to encode messages in a standard format for use in client–server-type environments, XML can provide significant advantages. However, where performance and memory usage requirements are important, it is not a feasible solution.

References

[1] P. Palensky and T. Sauter, "Security considerations for FAN-Internet connections," in *Proc. IEEE Int. Workshop Factory Commun. Syst.*, Sep. 6–8, 2000, pp. 27–35.
[2] M. Lobashov, G. Pratl, and T. Sauter, "Implications of power-line communication on distributed data acquisition and control system," in *Proc. IEEE Conf. Emerging Technol. Factory Autom.*, Sep. 16–19, 2003, vol. 2, pp. 607–613.
[3] T. Sauter, M. Lobashov, and G. Pratl, "Lessons learnt from Internet access to fieldbus gateways," in *Proc. IEEE 28th Annu. Conf. Ind. Electron. Soc.*, Nov. 5–8, 2002, vol. 4, pp. 2909–2914.
[4] M. Blattner, L. Kou, J. Carlson, and D. Daniel, "A visual interface for generic message translation," in *Proc. IEEE Workshop Vis. Lang.*, 1988, pp. 121–126.
[5] M. Blattner and L. Kou, "A user interface for computer-based message translation," in *Proc. IEEE 22nd Annu. Hawaii Int. Conf. Syst. Sci.*, 1989, vol. IV, pp. 43–51.
[6] Y. Bi, M. E. C. Hull, and P. N. Nicholl, "An XML approach for legacy code reuse," *J. Syst. Softw.*, vol. 61, no. 2, pp. 77–89, Mar. 2002.
[7] P. Peinl and B. Mitschang, "Towards an integrated systems approach for mobile traveller applications," in *Proc. IEEE 1st Int. Conf. WISE*, Jun. 19/20 2000, vol. 1, pp. 491–496.
[8] K. L. E. Law, "XML on LDAP network database," in *Proc. IEEE Can. Conf. Electr. Comput. Eng.*, 2000, pp. 469–473.
[9] R. Bhoedjang, J. Romein, and H. Bal, "Optimizing distributed data structures using application-specific network interface software," in *Proc. IEEE Int. Conf. Parallel Process.*, 1998, pp. 485–492.
[10] A. Bagnasco, M. Chirico, and A. M. Scapolla, "XML technologies to design didactical distributed measurement laboratories," in *Proc. 19th IEEE Instrum. Meas. Technol. Conf.*, May 21–23, 2002, vol. 1, pp. 651–655.

Suvendi Chinnappen-Rimer received the B.Sc. degree in electrical engineering in 1991 from the University of the Witwatersrand, Johannesburg, South Africa, and the M.Eng. degree in 2003 from the University of Pretoria, Pretoria, South Africa, where she is currently working toward the Ph.D. degree in computer engineering.

She is currently a Lecturer with the Department of Electrical and Electronic Engineering Science, University of Johannesburg. Her current research interests include aspects of wireless sensor and actor networks and the use of image processing in fault detection.

Gerhard P. Hancke (M'88–SM'00) received the B.Sc., B.Eng., and M.Eng. degrees from the University of Stellenbosch, Stellenbosch, South Africa, and the D.Eng. degree from the University of Pretoria, Pretoria, South Africa, in 1983.

He is a Professor and the Chair of the Computer Engineering Program and Research Group for Computer Networks and Security, Department of Electrical, Electronic, and Computer Engineering, University of Pretoria. His research interests are in wireless sensors and actuators networks, and he extensively partakes in collaborative research programs with international research institutions.