

A Secure Approach to Distributed Internet-Enabled Metrology

Åsmund Sand, Harald Slinde, *Member, IEEE*, and Tor A. Fjeldly, *Fellow, IEEE*

Abstract—This paper aims to explore ways of securely operating electrical instruments via the Internet. A system is presented, which allows a person connected to a local area network (LAN) to operate multiple electrical instruments located at other LANs via a public web server on the Internet, regardless of firewalls, proxy servers, or source network address translators that separate them. A full system test has been performed, and data from the test are later presented. The solution is based on Microsoft .NET Remoting, which is part of the Microsoft .NET Framework. A viable application is Internet-enabled calibration, and possible users are national measurement institutes and calibration laboratories. The solution is general and does not depend on the underlying network connections involved as long as hypertext transfer protocol over secure sockets layer is supported.

Index Terms—Calibration, distributed measuring systems, Internet, Internetworking, measurement.

I. INTRODUCTION

A. Internet-Enabled Calibration

INTERNET-ENABLED calibration has many definitions. In this paper, the term is used to describe the process of controlling and monitoring a calibration session directly via the Internet. The unit under test (UUT) is not moved from the owner's laboratory. Instead, a calibrated standard, e.g., an electrical multimeter, is transferred to the laboratory from a reference laboratory [e.g., a national measurement institute (NMI)], and the calibration process is then controlled directly by the reference laboratory.

There are many advantages to this approach. Since the UUT is never moved, the uncertainty associated with the transport of the UUT is eliminated. The reference laboratory often has much better understanding of the transport uncertainty of the calibrated standard.

Because the UUT is calibrated in the laboratory where it is normally used, the calibration conditions are more compatible with the normal working conditions. In traditional calibrations, the reference laboratory has a control of the calibration conditions in their own laboratory but has less insight into the normal working environment of the instrument.

With Internet-enabled calibration, the downtime of the UUT is minimized, and the UUT will be occupied only during the

calibration process. Many businesses cannot afford to have long downtimes when expensive equipment is to be calibrated.

In recent years, quite a few works on Internet-enabled calibration and distributed measurement architectures have been published. Most of these have a clear client-server perspective using hypertext transfer protocol (HTTP)/HTTP over a secure socket layer (HTTPS). Either the client controls the instruments connected to a web server [1]–[8], or the client downloads the procedures from a server to be run locally [9], [10]. Common to all of them is that the calibration or measurement procedure must be run at the computer where the instruments are connected.

Some systems also use symmetrical protocols to obtain bidirectional communication between the server and the client [11] (although they may not communicate bidirectionally through all firewalls and proxy servers, which might only support HTTP/HTTPS).

In this paper, we present a system where a person at one local area network (LAN) may control and monitor the instruments located at another LAN, as if these were connected locally at the former. A viable application is Internet-enabled calibration, as described, where a third-party authority controls and monitors the calibration process directly at a customer's laboratory from the authority's computer. The calibration procedure may be run on the authority's computer, at the server, or locally at the customer's computer. This would be difficult to implement for the aforementioned systems.

A full system test was performed, where two multimeters were sent to an external customer of Justervesenet (JV). An electrical calibrator, which is owned by the customer, was then partly calibrated using the two multimeters. The data from this test are presented in Section VIII.

B. Challenges

Direct, bidirectional, and secure communication between the two PCs, which are located on different LANs separated by the Internet, is challenging for many reasons. The Internet is a highly insecure network, and firewalls and proxy servers are required to separate a LAN from the exposure to unwanted intruders. There is also a shortage of public Internet Protocol (IP) addresses, which for some businesses requires the installation of source network address translators (SNATs). A SNAT maps the internal LAN addresses to the external Internet addresses. Because of this, all connections between a LAN PC and the outside of the firewall/proxy server/SNAT usually have to be initialized from the LAN PC. To establish direct communication between the two PCs in different LANs, a general solution

Manuscript received February 4, 2005; revised February 14, 2007.

Å. Sand and H. Slinde are with Justervesenet, the Norwegian Metrology Service, 2007 Kjeller, Norway (e-mail: aasmund.sand@justervesenet.no; harald.slinde@justervesenet.no).

T. A. Fjeldly is with the University Graduate Center, 2027 Kjeller, Norway, and also with the Norwegian University of Science and Technology, 2027 Kjeller, Norway (e-mail: torfj@unik.no).

Digital Object Identifier 10.1109/TIM.2007.895595

would be to set up a persistent connection from both PCs to a public server, which binds the two connections together in a so-called Meet-in-Middle configuration. All communication signals between the two LAN PCs pass through this server.

There are several security issues to attend to, such as confidentiality, integrity, and availability. The Internet is an open network, which allows general insight into all the traffic flowing between the connected computers. It is therefore necessary to use encryption when confidential information is exchanged. Access control and authentication to prevent unwanted access to private resources is also needed. Availability, at least during a remote instrument-control process, is critical.

II. MIDDLEWARE

A. Object-Oriented Middleware (OOM)

To provide direct method invocation between the network-separated application domains, the so-called middleware is required to transfer information about the method call over the network. OOM is a communication technology, which enables data objects to communicate across networks. There are several OOM technologies available, including among others Common Object Request Broker Architecture (CORBA) [12], Java Remote Method Invocation (RMI) [13], and Microsoft .NET Remoting [14]. These technologies use interface/proxy-based method invocation over Transmission Control Protocol/IP (TCP/IP) networks. They act as black boxes to the users and handle all network-related issues involved in the method-invocation process.

For two PCs on different LANs to communicate directly across the Internet, they need to maintain a persistent connection to some shared public state object, which forwards method calls in both directions. The OOM technologies mentioned use a shared object on a public server to which clients connect. The TCP connections may be kept active between the remote object and each client during a whole session. After the connections are set up, each client generates a local-proxy representation of the remote object. This proxy object has the same interface as the real server object, and clients who need to call a method on the server object call the corresponding method on the local proxy instead. The proxy then forwards the method call via the underlying OOM system across the network to the real object and handles the method reply. This way, the clients may use the remote server object as if it were located locally. To realize bidirectional remote method invocation, thus allowing the server object to invoke methods directly on each connected client, the server object also needs to have a local-proxy representation of the client objects.

Fig. 1 illustrates these processes. When the server object needs to call a method on a connected client, the default behavior for the OOM technologies mentioned is to try to establish a new TCP connection to the client and transfer the method call using this new connection. This behavior is efficiently stopped by the client-side firewall, and the bidirectional remote method invocation therefore fails. While CORBA and Java RMI are quite rigid with regard to modifying the communication channels and communication protocols used, the .NET Remoting architecture allows developers to customize the communication

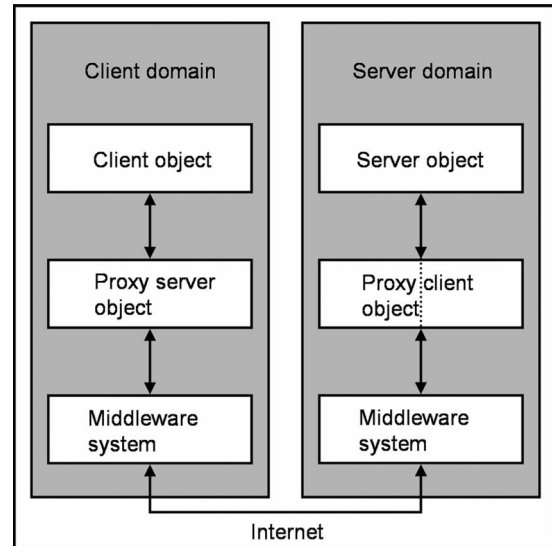


Fig. 1. General OOM-system architecture. The client object calls methods on the server object via the local proxy server object. The server object may also call methods on the client object via the local proxy client object. The underlying middleware system transfers the method calls across the Internet.

channels in such a way that the server object reuses the existing TCP or HTTP connections initialized by the clients and, therefore, is permitted a passage through the client-side firewalls. When a public web server hosts the remote object, the clients usually have no problems setting up the HTTP connections from behind the company firewall or proxy server.

To provide this functionality, some restrictions will dictate the choice of platform. Software developed for .NET may only be run on computers with the .NET Framework installed. Traditionally, this means that the computers involved must run the Windows platform. Platform independent versions of the .NET Framework are under development [15], but these have yet to be tested for the present system. This way, the .NET Framework could work similarly to the Java Runtime Environment.

B. Microsoft .NET Remoting

Microsoft .NET Remoting is a relatively new OOM middleware technology, which enables applications in different application domains to communicate, as described previously. The .NET Remoting structure supports not only the native TCP and HTTP channels but also the custom-made channels. The latter is particularly important when it comes to the callback methods from the server object to the client. When using the native TCP or HTTP, the remote server object needs to open a new TCP or HTTP connection to the client, if callback methods on a client are to be invoked. However, this is efficiently stopped by the client-side firewall, SNAT, or proxy server, as shown before, and is not a solution. This is a limitation of the native TCP and HTTP channels in .NET Remoting, which are both one-directional, but it is not a problem of the .NET Remoting technology as such.

In the system presented, using a custom-made channel, each client initially sets up two HTTP connections to the remote server object: one for outgoing traffic and one for incoming traffic. The remote server object then uses the latter for callback

methods. By using a licensed open-source product from GenuineChannels [16], this can be realized quite easily. As in a regular HTTP communication process, the clients use HTTP requests, and the server uses HTTP responses for communication. The client initially sets up two keep-alive HTTP connections to the server, which are called SENDER and LISTENER. The LISTENER connection is in a constantly pending HTTP request mode (with a timeout value of choice). The client uses the SENDER connection when calling a method on the server. The server, on the other hand, uses the LISTENER connection when calling a method on the client.

When the client calls a method on the local proxy server object, the underlying .NET Remoting system wraps the call in a message object and sends it in an HTTP request to the server using the SENDER connection. On the server, the .NET Remoting system unwraps the method call and calls the correct method on the server object. It then sends the method reply back on the LISTENER connection in an HTTP response.

When the server calls a method on the local proxy client object, it wraps the method call in a message object and uses the client's pending HTTP request to send an HTTP response, which is containing the message object, on the client's LISTENER connection. After the client receives this HTTP response, the underlying .NET Remoting system unwraps the message object and calls the correct method on the client object. It then sends back the method reply in an HTTP request on the SENDER connection. Thus, bidirectional method invocation is obtained, without inefficient client polling, and standard security protocols, like HTTPS, can still be used.

Microsoft's web server, which is the Internet Information Service (IIS), hosts the remote server object, and all security features of the IIS architecture are available. The custom-made channel supports the IIS default security services, encryption, and compression for faster transfer of data. To set up the two connections, the clients must be able to connect to the remote server on port 443 secure sockets layer/transport layer security, which most firewalls and proxy servers allow today.

III. INSTRUMENT CONTROL

The system implemented uses the Virtual Instrumentation Software Architecture (VISA) [17] for instrument control through a preinstalled dynamic linking library (DLL) file, which is the `visa32.dll`. VISA is a standardized interface operating between the control application and the instruments. It may control VME eXtensions for Instrumentation, general-purpose interface bus (GPIB), serial, or computer-based instruments, and makes the appropriate driver calls depending on the type of instrument used. The `visa32.dll` library contains several methods for communicating with hardware over the aforementioned hardware interfaces. In this project, only GPIB connections were considered. A small .NET component was developed to load this VISA library at startup. The component contains four methods: `FindResources`, `Read`, `Write`, and `Query`. The `FindResources` method is used to scan the client-side instrument bus for connected instruments. The `Write`, `Read`, and `Query` methods are used to send text messages to and from the instruments. The installed .NET client application

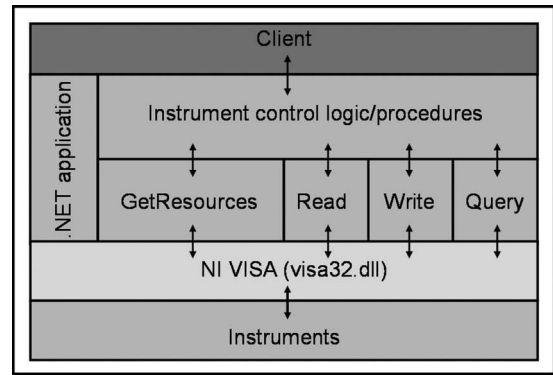


Fig. 2. Instrument-control architecture locally with the client. The .NET application loads a small library, which communicates with the connected instruments via the VISA interface. After loading the library, the .NET application is able to communicate with the instruments.

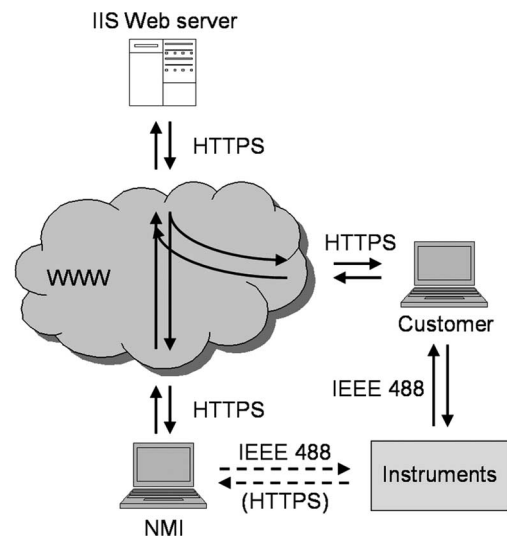


Fig. 3. Internet-enabled calibration overview. The controlling NMI authority may control the customer's instruments via the web server. To the authority, it seems like the instruments are connected locally at the authority's PC. The underlying middleware system hides the real communication path, which goes via the server.

loads this component at startup and may then call the contained methods as needed. This way, all implementations concerning the measurement setup and measurement logic are done in the .NET environment. The instrument-control logic could be placed locally on the server or on the controlling client. The client-side instrument-control architecture is shown in Fig. 2.

If the middleware system is running as described previously, it is quite easy for a client to control another client's instruments. The middleware system is transparent to the clients, and to the controlling client, it appears as if the other client's instruments are connected locally.

In the system presented, an NMI authority takes control of a customer's PC-connected instruments and performs the measurement via the Internet. This is shown in Fig. 3.

IV. SECURITY

When using the Internet for general instrument control and, particularly, when offering callback methods from the server object directly to the connected clients, several security

concerns arise. If unwanted intruders could gain access to the remote object and fake the identity of a reference-laboratory authority or replay old method calls, serious damage could be done at the customer side. The customers must therefore be absolutely confident that they are communicating with a real authority. The authority must also be confident about the customer's identity, e.g., when the Internet-enabled calibration is performed. The information sent across the network should be impossible to alter or replay, at least not without detection.

The .NET Remoting architecture has no default security features, but it is prepared for custom security implementation. By hosting the remote object in Microsoft IIS, the system can leverage all the security features provided by the IIS architecture.

To develop and implement a good security model from scratch is complicated, and if high-level security is required, a better solution would be to use a well-known and well-tested security protocol such as HTTPS. The current solution operates over HTTPS with server and client authentication, using X.509 certificates signed by a trusted certificate authority. When using certificates in the authentication process, it is virtually impossible to fake the identity of a customer computer, authority computer, or the server. In addition, username and a password are required to authenticate the specific persons who operate the customer and authority computers. This is how the authentication process is performed in many online banking services. The client certificates must be distributed to the customers and authority in a secure fashion. If a laptop is sent out to the customers in advance, the certificates should be installed before sending. The customers must be thoroughly educated in how certificates should be handled and stored.

To improve the availability of the system, one approach could be to let the dedicated servers, e.g., Kerberos, take care of the authentication process to avoid overloading the application server. This will also make the system more resistant to denial-of-service attacks. In an Internet-enabled calibration system, there would be very few users at any given time, and the application server would then have free resources to handle unwanted connection attempts.

The clients involved are regarded as trusted once authenticated.

V. PERFORMING MEASUREMENTS

The programming language Microsoft C#, which is pronounced C Sharp, was used to develop most parts of the system. This programming language supports runtime compilation of source code: a feature which is used when performing remote measurements.

When a client connects to the server and the client's instrument bus is scanned, a special instrument-wrapper object is generated on the server and on the client for each connected instrument. These instrument wrappers act as proxy instruments for the real ones, e.g., when calling a write method with a certain input string parameter in an instrument wrapper on the server, the parameter is automatically forwarded across the Internet to the correct client instrument. If, on the other hand, the write method is called on an instrument wrapper locally on the client, the parameter is sent to the instrument directly.

TABLE I
CHANGE IN ac CURRENT AFTER ROUNDTRIP TRANSPORT
FOR THE TWO MULTIMETERS

Frequency (HZ)	Nominal (mA)	Change (mA) Multimeter 1	Change (mA) Multimeter 2
45	3.29	-0.000150	-0.000100
1000	3.29	-0.000150	-0.000120
10000	3.29	-0.000190	-0.000120
1000	0.19	-0.000009	-0.000002
1000	190.00	-0.005700	-0.001600
1000	1000.00	-0.064000	-0.016000

Measurement and calibration procedures are stored in a central database as source code, following a certain format. These procedures make use of the instrument-wrapper interfaces, which are implemented by the instrument-wrapper classes, so that the same procedure could be used for a wide variety of instruments, as long as their corresponding instrument-wrapper class implements the correct interface.

In an Internet-enabled calibration scenario, a reference laboratory picks a specific calibration procedure from a database based on which instrument-wrapper classes are currently associated with the client. The measurement data used in the calibration are stored in the database as extensible markup language. The procedure, with the data, could then be sent to the client to be compiled and run in runtime. The procedure could also be compiled directly on the server and run from there. Then, the instrument-wrapper classes would be responsible for transferring the method calls across the Internet to the correct instrument at the client. If the procedure makes rapid calls to the client's connected instruments, it may be preferable to send the procedure to the client to be run locally due to the potential network congestion problems. If the procedure makes relatively infrequent instrument calls or performs heavy calculations in between instrument calls, it might be a good approach to run it directly from the server or some other fast PC connected to the server. There is no limitation on which PC the procedure could be run as long as the PC has a persistent connection to the remote server object.

The connected authority may also perform low-level instrument control on a connected client by calling the instrument-wrapper class methods directly on the server. This way, the client's instruments appear to be connected locally at the authority PC. This is an important feature of the system, which lets an authority expert use his or her expertise to bear on problems or errors with the instruments. This would be a difficult task to implement as an automatic procedure.

VI. SCALABILITY AND MODULARITY

New measurement procedures may be added to the dedicated database server at runtime, and they can then be utilized by operators directly. Any instrument with a bus supported by VISA can be connected to the system. The system is therefore very scalable with regard to performing measurements and communicating with instruments. By using OOM, the whole distributed system can be treated as one object-oriented application comprising concepts like encapsulation, inheritance, and polymorphism. This preserves and strengthens the modularity

TABLE II
DETAILED DESCRIPTION OF EACH STEP PERFORMED BY THE CUSTOMER AND THE OPERATOR IN THE MEASUREMENT PROCESS

Actor	Performed action	Detailed results
Customer	Starts client application	The HTTP channel is initialized between client and server computers
Customer	Calls the “Login” method with a username and a password	The middleware transports the method call and the parameters to the server where they are validated
Customer	Calls “Register instruments” at server	The middleware transports the local instrument connection information to the server where it is registered
Operator	Starts client application	The HTTP channel is initialized between client and server computers
Operator	Calls the “Login” method with a username and a password	The middleware transports the method call and the parameters to the server where they are validated
Operator	Calls the “Get instruments” method	The middleware transports the method call to the server and returns a list of connected instruments
Operator	Checks the instruments by sending individual instrument commands	Each instrument command is transported to the server, where they are forwarded to the correct client and instrument. The results are returned to the operator in a similar way.
Operator	Selects and runs a measurement procedure	A list of possible measurement procedures, based on the connected instruments, is sent to the operator from the server. The operator then decides where to run the procedure, either locally or at the remote instrument computer. The measurement is then compiled and initiated.
Operator	Receives measurement results	After the measurement has finished running, the results are returned to the operator as a list of measurement points (each associated with a specific time and other meta information). The data is shown in a graph and written to a text file for future processing.

of the system and, thus, enables the addition, removal, or alteration of components without affecting the rest of the system. One could potentially increase the scalability of the system by supporting more instrument buses or instruments with their own hardware cards. This could be solved by adding a module to the system, which would handle adding support for the hardware similar to the module responsible for adding measurement procedures.

VII. COMPETING SYSTEMS

An earlier version of the current system (iMet v1.0) has been compared to a similar system (iGen) at the National Physical Laboratory (NPL) in England [18]. iMet v1.0 did not support the dynamic addition and utilization of measurement procedures, and the measurement procedures in use had to be implemented beforehand in the instrument communication software.

The two compared systems differed mostly in architectural design. For iGen, the operator used the instrument-connected computer directly to control the measurement process, and all measurement procedures were downloaded from a database server and run as scripts during runtime. For iMet v1.0, the operator and the instruments were generally separated by the Internet, and complex communication software had to be used. This made iMet v1.0 more network dependent than iGen, but it enabled the operator to be located anywhere (where an Internet connection was available). For iGen, the operator needed to travel to the instruments.

The systems also differed in that iGen could store measurement procedures as scripts in a database. As mentioned, iMet required that the measurement procedures were installed in the applications beforehand. This made iGen much more scalable with regard to running different measurements and adapting to changing measurement situations.

VIII. IMPLEMENTED TEST SYSTEM

JV implemented the proposed system and tested it in a realistic setup with two connected computers (one located at JV and one located at an external customer of JV), with firewalls at both locations and a public web server. The process went as follows: In order to calibrate one of its electrical calibrators, the customer first contacted JV, which then sent two calibrated precision digital multimeters to the customer. Two multimeters were used to investigate the effects of the transport. The customer is then connected to the public server object and registered its three connected instruments, thus creating three instrument-wrapper objects on the server and locally as described previously. The operator at JV is also connected to the server and obtained a reference to the customer and to the three connected instrument wrappers.

The operator then did some initial checks on the instruments by sending native IEEE488 (GPIB) commands via the server object to the instruments. In the test calibration, the operator first picked a specific measurement procedure from the database and downloaded it locally, where it was compiled and run. The procedure contained methods to measure five quantities, which are the ac and dc current, the ac and dc voltage, and the resistance, for several values and frequencies. When the procedure was completed, the measurement data were shown graphically to the operator and stored in a text file locally. The measurement data were later analyzed, the multimeters were recalibrated at JV, and a report was generated and sent to the customer. The analysis and recalibration results showed that both digital multimeters had changed during transport. It was hard to say if this was due to transport or drift, because the calibration history of the multimeters was not well known. An example of the changes observed is shown in Table I. Note that the changes in the multimeters were well within the specifications of the electrical calibrator.

The calibration process described does not depend on which instrument types are connected. The only requirements are that the instrument types are registered in the server's database, that the instrument-wrapper classes for the three instrument types exist, and that the calibration procedures and calibration points for these specific instrument-wrapper combinations are available.

A summary of the main steps in an Internet-enabled measurement is shown in Table II.

IX. CONCLUSION

The solution presented is general in the sense that it does not depend on the types of network connections involved (e.g., via proxy server, SNAT, and dial-up) but does require that the computers involved install Microsoft's .NET Framework. As shown before, implementations of the .NET Framework on other platforms than Windows are available.

When performing the Internet-enabled calibration at a customer's site, the computers involved could be an NMI authority PC, an NMI server, and an NMI laptop, where the latter is to be sent to the customer with the required measurement standards. This is similar to the procedure in NPL's iVR project [19]. This way, the reference laboratory has full control over all computers in the system, and the aforementioned .NET Framework criterion could be easily fulfilled. If the customer's security policy does not allow external computers to be introduced into the customer's LAN, a dial-up connection could also be used, which is similar to the NPL approach [20]. Although several security problems at the customer's site could be solved this way, the bandwidth on dial-up connections is generally lower than the direct connections from a LAN. Using dial-up connections is not suitable when long-term connections are required, which is due to costs.

It is of crucial importance to find a suitable transfer standard before using the proposed system in the Internet-enabled calibration.

REFERENCES

- [1] F. Pianegiani, D. Macii, and P. Carbone, "An open distributed measurement system based on an abstract client-server architecture," *IEEE Trans. Instrum. Meas.*, vol. 52, no. 3, pp. 686–692, Jun. 2003.
- [2] M. Bertocco, F. Ferraris, C. Offelli, and M. Parvis, "A client-server architecture for distributed measurement systems," *IEEE Trans. Instrum. Meas.*, vol. 47, no. 5, pp. 1143–1148, Oct. 1998.
- [3] W. Winiecki and M. Karkowski, "A new Java-based software environment for distributed measuring systems design," *IEEE Trans. Instrum. Meas.*, vol. 51, no. 6, pp. 1340–1346, Dec. 2002.
- [4] M. Bertocco and M. Parvis, "Platform independent architecture for distributed measurement systems," in *Proc. IMTC*, May 2000, vol. 2, pp. 648–651.
- [5] K. Michal and W. Wieslaw, "A new Java-based software environment for distributed measurement systems designing," in *Proc. IMTC*, May 2001, vol. 1, pp. 397–402.
- [6] H. Ewald and G. Page, "Client-server and gateway-systems for remote control," in *Proc. IMTC*, May 2003, vol. 2, pp. 1427–1430.
- [7] T. A. Fjeldly and M. S. Shur, *Lab on the Web, Running Real Electronics Experiments via the Internet*. New York: Wiley, 2003.
- [8] S. Kolberg and T. A. Fjeldly, "Internet laboratory with web services accessibility," in *Proc. Adv. Technol.-Based Edu.: Towards Knowledge-Based Soc., 2nd Int. Conf. m-ICTE*, A. M. Vilas, J. A. M. Gonzalez, and J. M. Gonzalez, Eds., Badajoz, Spain, Dec. 2003, vol. 3, pp. 1700–1704.

- [9] R. A. Dudley and N. M. Ridler, "Traceability via the internet for microwave measurements using vector network analyzers," *IEEE Trans. Instrum. Meas.*, vol. 52, no. 1, pp. 130–134, Feb. 2003.
- [10] D. Ives, G. Parkin, J. Smith, M. Stevens, J. Taylor, and M. Wicks, "Report to the national measurement system directorate, Department of Trade and Industry—Use of Internet by calibration services: Demonstration of technology," National Physical Laboratory (NPL), Teddington, U.K., NPL Rep. CMSC 49/04, Mar. 2004.
- [11] A. Carullo, M. Parvis, and A. Vallan, "Security issues for internet-based calibration activities," in *Proc. IMTC*, May 2002, vol. 1, pp. 817–822.
- [12] *The Common Object Request Broker Architecture*. [Online]. Available: <http://www.corba.org/>
- [13] *Java Remote Method Invocation*. [Online]. Available: <http://java.sun.com/products/jdk/rmi/>
- [14] *Microsoft .NET Remoting*. [Online]. Available: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/hawkremoting.asp>
- [15] *The Mono Project*. [Online]. Available: <http://www.go-mono.com>
- [16] *GenuineChannels*. [Online]. Available: <http://www.genuinechannels.com>
- [17] *Virtual Instrumentation Software Architecture*. [Online]. Available: <http://www.ni.com/visa/>
- [18] A. Sand, M. Stevens, and G. Parkin, "Internet-enabled calibration: An analysis of different topologies and a comparison of two different approaches," *IEEE Trans. Instrum. Meas.*, to be published.
- [19] D. Rayner. (2003). *Survey of International Activities in Internet-Enabled Metrology*, p. 9. [Online]. Available: http://www.npl.co.uk/ssfm/download/documents/cmssc21_03.pdf
- [20] R. Barker and G. Parkin. (2003). *Use of the Internet for Calibration Services—Protecting the Data—Final Rep.*, p. 17. [Online]. Available: http://www.npl.co.uk/ssfm/download/documents/cmssc28_03.pdf



Åsmund Sand received the M.Sc. degree in physics from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 2000 and the Ph.D. degree on Internet-enabled metrology from NTNU's University Graduate Center (UniK), Kjeller, Norway, in 2007.

He is currently working on issues concerned with applying the Internet to electrical metrology and calibration at Justervesenet, the Norwegian Metrology Service, Kjeller.



Harald Slinde (M'94) was born in Trondheim, Norway, in 1962. He received the M.Sc. degree in applied physics from the Norwegian Institute of Technology, Trondheim, in 1986. He received the Ph.D. degree from Imperial College, London, U.K., in 1993.

From 1987 to 1989, he worked at the Center for Industrial Research, Oslo, Norway (currently, a part of the Sintef group), as a Research Scientist. From 1989 to 1993, he was a Research Fellow at Imperial College. Since 1994, he has been at Justervesenet, the Norwegian Metrology Service, Kjeller, Norway. Here, he was responsible for building up the low-frequency ac voltage and current activity. He is a qualified assessor for calibration laboratories within the dc low-frequency area for Norwegian Accreditation. Since July 1996, he has also been the Group Leader for the electrical and time/frequency activities at Justervesenet. His current main research interests are in ac-dc difference, ac voltage measurements, and Internet-enabled metrology. He is representing Justervesenet in the Consultative Committee for Electricity and Magnetism and the Euromet Electricity Technical Committee.



Tor A. Fjeldly (M'85–SM'88–F'00) received the M.Sc. degree in physics from the Norwegian Institute of Technology, Trondheim, Norway, in 1967 and the Ph.D. degree from Brown University, Providence, RI, in 1972.

From 1972 to 1994, he was with Max-Planck-Institute for Solid State Physics, Stuttgart, Germany. From 1974 to 1983, he worked as a Senior Scientist at The Foundation for Scientific and Industrial Research at the Norwegian Institute of Technology research organization. Since 1983, he has been on

the faculty of the Norwegian University of Science and Technology (NTNU), Kjeller, Norway, where he is a Professor of electrical engineering. He is currently with NTNU's University Graduate Center. He was the Head of the Department of Physical Electronics at NTNU, and he also served as an Associate Dean of the Faculty of Electrical Engineering and Telecommunication. From 1990 to 1997, he held the position of Visiting Professor at the Department of Electrical Engineering, University of Virginia, Charlottesville, and from 1997 to 2003, he was a Visiting Professor at the Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY. His research interests have included the fundamental studies of semiconductors and other solids, development of solid-state chemical sensors, electron transport in semiconductors, modeling and simulation of semiconductor devices, and circuit simulation. He has written about 200 scientific works, including several books and book chapters. He is also a codeveloper of the circuit simulator Automatic Integrated Circuit Modeling Spice. Since 1998, he has been a Co-Editor-in-Chief of the *International Journal of High Speed Electronics and Systems* and a Co-Editor of the book series *Selected Topics in Electronics and Systems*, both with the World Scientific Publishers, Singapore.

Dr. Fjeldly is an elected member of the Norwegian Academy of Technical Sciences.