A Standardized Approach for Transducer Interfacing:

Implementing IEEE-P1451 Smart Transducer Interface Draft Standards

Kang B. Lee,
Leader, Sensor Integration Group
kang.lee@nist.gov
Telephone: (301) 975-6604

Richard D. Schneeman,
Computer Scientist, Sensor Integration Group
rschneeman@nist.gov
Telephone: (301) 975-4352

United States Department of Commerce
National Institute of Standards and Technology
Manufacturing Engineering Laboratory
Automated Production Technology Division
Gaithersburg, Maryland 20899 USA

Abstract

NIST researchers have developed a reference implementation and companion demonstration for this currently defined set of specifications to provide a concrete example of the IEEE P1451, Draft Standard for a Smart Transducer Interface for Sensors and Actuators. The reference implementation includes both hardware and software components that when integrated together yield an environment for illustrating complete P1451 functional aspects and capabilities. This document briefly provides an overview of both parts of the standard and more specifically how they relate to this demonstration. The reference implementation approach used as well as resources required are also discussed to familiarize the reader with the demonstration environment. Specific implementation issues are then discussed concerning the several main areas of the software and hardware components used in this implementation. The first software component, called *NCAPTool*, written in C++, provides a graphical user interface (GUI) -based Windows environment in which various functional aspects of the standards can be exercised. The second component is a dynamic link library (DLL), also written in C++, that provides an Application Programming Interface (API) to the P1451.1, *Draft Standard for a Network Capable Application Processor (NCAP) Information Model*. The third component provides the hardware necessary to illustrate a tangible implementation of the P1451.2, *Draft Standard for a Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats.* All three components together illustrate the integration of both P1451.1 and P1451.2 as well as providing a visual capability for demonstrating the standards' key functional aspects.

Keywords: actuators, application programming interfaces, interoperability, graphical user interface, network independent, portability, reference implementation, sensors, sensor interface, transducers.

Table of Contents

1. INTRODUCTION 4
1.1 TRANSDUCER DEVICE OVERVIEW 4
1.2 WHAT IS A SMART TRANSDUCER? 5
1.3 TRANSDUCER INTEGRATION PROBLEMS 5
1.4 A STANDARDS-BASED SOLUTION 5
2. IEEE P1451 OVERVIEW 5
2.1 P1451.1 - INFORMATION MODEL 6
2.1.1 Using Object-Oriented Design 7
2.1.2 Block Classes 7
2.1.3 Base Classes 8
2. 2 P1451.2 - TRANSDUCER TO MICROPROCESSOR INTERFACE 9
2.2.1 STIM 9
2.2.2 TEDS 9
2.2.3 Digital Interface 9
3. IMPLEMENTING THE OBJECT MODEL 10
3.1 CLASS IMPLEMENTATION 10
3.2 A LIBRARY-BASED IMPLEMENTATION 11
4. P1451.2 HARDWARE RESOURCES 11
5. DEMONSTRATION ARCHITECTURE 11
5.1 SYSTEM CONFIGURATION 11
5.1.1 Server Configuration 12
5.1.2 Client Configuration 13
6. DEMONSTRATING THE STANDARD 13
7. SUMMARY 14
8. REFERENCES 14
9. ACKNOWLEDGMENT 15

1. Introduction

Researchers at the National Institute of Standards and Technology (NIST) have developed a Reference Implementation of the proposed IEEE P1451 *Draft Standard for a Smart Transducer Interface for Sensors and Actuators* [1][2]. The standard defines a set of specifications minimizing the hardware and software problems associated with interfacing transducers to multivendor networks, multistandard buses, and a variety of microprocessor-based platforms. The set of specifications addresses these concerns by focusing on three key areas, including: **(1)** application-level portability for transducer-based software, **(2)** network-independent access for transducer-based applications, and **(3)** transducer interoperability using a plug-and-play approach to connecting transducers to a microprocessor platform and a network.

In order to highlight key aspects of the draft specifications, a demonstration in the form of a reference implementation was developed. The reference implementation provides NIST researchers and interested parties with:

- A standards-based software and hardware platform in which to provide a venue for the demonstration.
- A concrete proof-of-concept implementation in which to test the standards' effectiveness, provide useful feedback, and expedite the standards definition process through experimentation.
- A useful graphical software tool to demonstrate the standards by exercising key functional interfaces from the specification.

Before we begin a detailed examination of the reference implementation, we first need to frame our discussion with a brief description of the relevant parts of a networked transducer. In addition, the capabilities that make a transducer device *smart* also need to be briefly summarized.

1.1 Transducer Device Overview

A transducer is a device that converts energy from one domain into another. The resulting quantity is calibrated to minimize errors during the conversion process. A transducer is either a sensor (i.e., a device that senses pressure or temperature) or an actuator (i.e., a controllable device such as a valve or a relay). Figure 1 illustrates the components found in a typical transducer connected to a network.

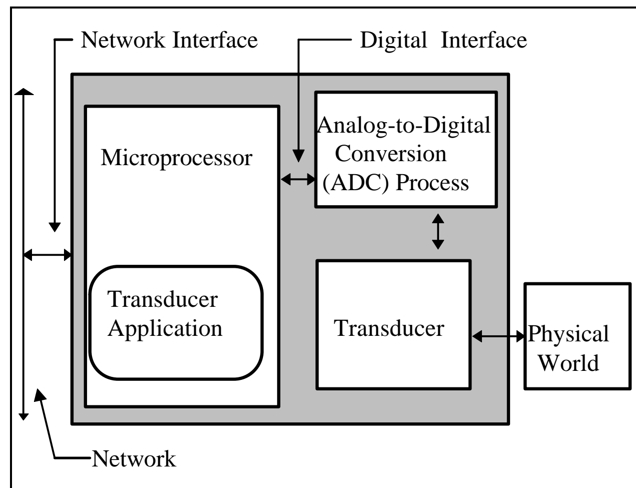

Figure 1: A Typical Network Transducer.

In its most general form, a transducer physically interfaces with an embedded microprocessor in order to support some form of hardware input and output channels from the microprocessor to the transducer, as shown in Figure 1. This interface provides a data path between the microprocessor and the transducer.

In addition, note that in Figure 1 the interface from the microprocessor to the network forms a communication pathway to a networking substrate. This leads to the definition of a networked transducer, which simply means the transducer has the ability to provide calibrated data to the network and the ability to respond to queries from the network.

Located onboard the microprocessor, shown in Figure 1, a network-capable application typically executes a variety of algorithms suited to the particular transducer's application domain, such as interacting with a pressure or temperature sensor. In order to use the network communication medium, an applicator must be aware of and interact with the specific control network technology. In addition, the application must also be capable of using the transducer interface connected to the control network microprocessor to send and receive data to and from the transducer. This in effect positions the application as a laminated structure uniting both a network interface on one side with a transducer interface on the other.

1.2 What is a Smart Transducer?

What additional features does a generic transducer device have to exhibit for it to be considered *smart*? A *smart* transducer needs to provide additional capabilities other then merely the correct representation of a sensed or controlled physical quantity. In relation to the standard's definition, as defined in the IEEE P1451 drafts, a transducer is smart if it contains additional functionality that simplifies the integration of the transducer into any networked application environment. Another smart feature is the ability of self-identification of the transducer to the system. This is accomplished by providing the capability to embed key information about the transducer and its performance in a standardized format in a small amount of nonvolatile memory associated with the transducer. At power up or query from the system the transducer can identify itself to the host processor. This feature enables the automation of diagnostic, configuration, and identification procedures across a multivendor environment. In fact, these and other capability-based standardized features provide the generic transducer with a smart moniker that connotes greater functionality, portability, and extensibility. Standardization of these features increases interoperability.

1.3 Transducer Interface Problems

One motivating reason for defining the interface standard is current problems transducer manufacturers face when integrating their devices into multivendor networks and heterogeneous hardware environs. Because the network and the transducer must expose their two interfaces directly to the application, any attempt to migrate the application to another platform is just cause for a complete redesign of the application's interface to the new environment. Transducer manufacturers and system integrators currently struggle with these issues while trying to manufacture and market sensors for cross-industry application domains and multivendor networks. The redesign process is time-consuming and expensive leading to transducer products that take a longer time to market with higher price tags. In addition, all prospects for interoperable, plug-and-play sensors and actuator devices are lost because of proprietary or unique interfaces. Transducer manufacturers must now expend a great deal of engineering effort to cover several control network vendor technologies instead of designing the device once for all networks that adhere to the standardized interfaces.

The interface between the microprocessor and transducer that presents many problems for transducer manufacturers when they want to interface their products with a multitude of microprocessor buses. A different hardware/software interface must be designed for each bus the vendor chooses to support.

1.4 A Standards-based Solution

If a standardized approach to interfacing both the application with the network and the microprocessor with the transducer device is available, then companies can leverage it to more effectively provide cross-industry support for their products while reducing the engineering and time to market issues that currently plague implementers. That is, through this standardized or common interface, the same transducers can be used on multiple control networks, and the selection of a control network for measurement, and control application is totally free of transducer compatibility constraints. Moreover, expanding and crossing into different markets increases competition while driving down prices. Transducer application designers can focus more on adding value to their applications without being concerned about developing interfaces for every possible network or microprocessor that their respective companies decide to target. Increasing value-added features will lead to more innovative applications for end-users. More importantly, the standardization process provides a level playing field for development. That is, smaller transducer manufacturers could now enter markets whereas before only companies commanding enough resources and capital can afford to develop products across multiple nonstandardized interfaces.

These issues have become the key motivation for forming cross-industry based working groups to define a networked smart transducer standard. In an attempt to provide a concrete representation of the standard, a reference implementation of the standard has been developed and will be the focus of the discussion in the next section.

2. IEEE P1451 Overview

Recognizing a need to remedy the transducer interfacing problems, the Committee on Sensor Technology of the Instrumentation and Measurement Society of the Institute of Electrical and Electronics Engineers (IEEE) has been working on defining a standard for a Networked Smart

Transducer. The proposed IEEE P1451 is a two-part standard that essentially combines a smart transducer device Information Model targeting software-based, network-independent, transducer application environments (P1451.1) with a standardized digital interface and communication protocols for accessing transducer data from the transducer via the microprocessor (P1451.2). Figure 2 shows the component layout of the proposed interface for a P1451 Networked Transducer.

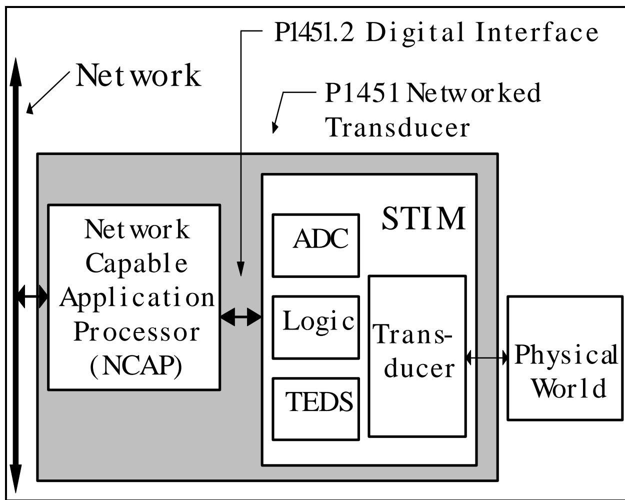

Figure 2: A P1451 Networked Transducer.

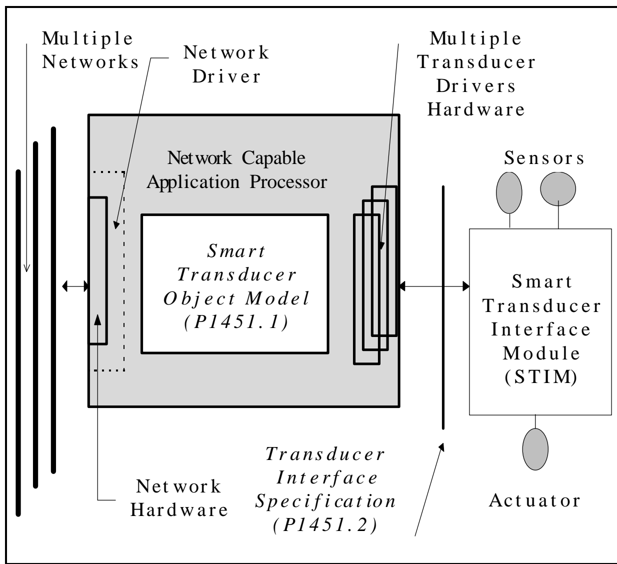

Figure 3: A Standardized P1451 Smart Transducer.

Figure 3 illustrates the complete P1451 smart transducer standards model comprising both the interface to the Information/Object Model along with the Transducer Interface specification. Notice in Figure 3 where the standardization process specifically address the transducer industry's two most problematic areas. These are the definitions of a standardized digital interface between the transducer and the microprocessor, as well as a standardization of the application elements that impact network communication.

The P1451.1 specification provides an abstract interface description that ultimately will be transcribed into a concrete application programming interface (API) when developers implement the model. The functional API interface of P1451.1 is used to demonstrate the reference implementation and likewise requires greater coverage in order to properly address the implementation. The hardware resources used in this demonstration are based completely on P1451.2; however, the specification implementation is not discussed in detail. These two areas of the standard will be further discussed in the next sections.

2.1 P1451.1 - Information Model

The proposed P1451.1 Draft Standard, *the Network Capable Application Processor (NCAP) Information Model*, centers around the object-oriented definition of an NCAP. The NCAP is the object-oriented embodiment of a transducer device. This includes the definition of all application-level access to network resources as well as the framework for application access to transducer hardware, as shown in Figure 3.

The complete definition and specification of the NCAP constitutes the Information Model and is the basis for the P1451.1 specification. The Information Model strives to lay out a framework that abstracts the characteristics of a networked Smart Transducer device using object-oriented design techniques. In the standard, the object-based aspect of the Information Model is referred to as the Smart Transducer Object Model and is shown in the center of the NCAP in Figure 3. The NCAP definition encompasses a set of object classes, attributes, methods, and behaviors that provide a concise description of a transducer and the network to which it may connect. By modeling the transducer device in object-oriented terms, an abstract view of device characteristics can be coalesced into a singular model. The model is sufficiently general to encompass a wide variety of networked transducer application services. Moreover, the Object Model tackles

the two specific problem areas by standardizing on the linkages between how applications interact with physical sensors and actuators in the system and how these same applications interact with the attached networking medium.

The Object Model provides standardized access for NCAP applications to the network by defining a network-neutral communication model for both intra- and inter-device interaction. Standardized access to the physical transducer is provided by a programmatic interface based on a device driver interface model. In effect, the NCAP application is laminated between two standardized device driver models — one a network driver and the other a transducer interface driver as shown in Figure 3. By adhering to both models consistently, applications may be reused and migrated to other networks without major reengineering effort. Network and transducer vendors simply provide these driver *stubs* to link with an NCAP to facilitate a portable and interoperable, plug-and-play transducer application environment. The specifics of how the standard addresses these issues will be discussed in the next section.

2.1.1 Using Object-Oriented Design

The Object Model uses an object-oriented design methodology for describing smart transducers. Therefore, the major objective of the P1451.1 working group was to define a class containment hierarchy that identified specific classes, methods, attributes, and behaviors that accurately define a networked smart transducer device object. Figure 4 illustrates the class containment used in the draft that makes up the Smart Transducer device.

The P1451.1 draft models the capabilities of a network capable transducer using the familiar rack or card-cage paradigm used to describe plug-in I/O cards in a personal computer (PC). That is, a PC consists of a backplane or bus where special I/O cards representing specific functionality can be plugged in. The cards are represented in the model by *blocks*. The blocks are essentially block classes and represent the highest form of functionality in the model. The standard uses the card-cage model to describe the transducer device. It uses two types of classes to construct these cards between most notably *Base* and *Block* classes.

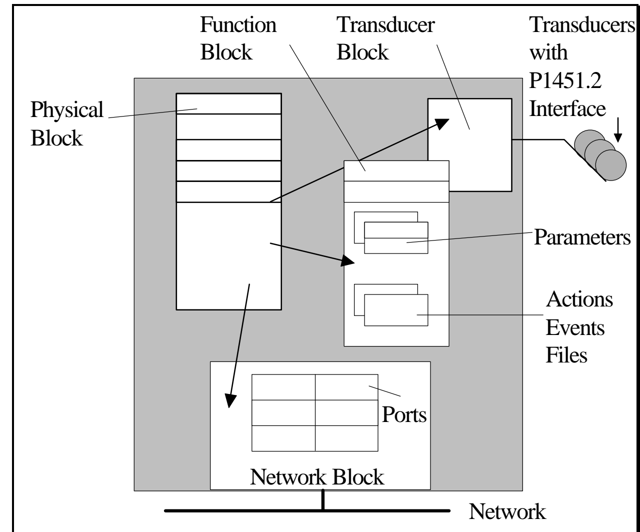

Figure 4: The P1451.1 Containment Hierarchy.

The classes defined by P1451.1 consists of four *Block* classes: one Physical Block, and one or more Transducer Blocks, Function Blocks, and Network Blocks. Notice how each Block class may include specific Base classes from the model. The Base classes include Parameters, Actions, Events, and Files, and provide component class. It is important to note that the *details* of the P1451.1 specification reflect Version 1.75 of the Draft Standard, and may change in later version. Each of these will be briefly discussed.

All classes in the model have an abstract or root class from which they are derived. This abstract class includes several attributes and methods that are common to all classes in the model. This provides a central class definition to be used for instantiation and deletion. In addition, methods for getting and setting attributes within each class are also provided.

2.1.2 Block Classes

Block classes form the major blocks of functionality that can be *plugged* into the card-cage to create various types of devices. One Physical Block is mandatory as it defines the card-cage and abstracts the hardware and software resources that are used by the device. All other blocks and component Base classes can be referenced from the Physical Block.

A *Physical Block* represents the card-cage and contains all the logical hardware and software resources in the model. These resources determine the basic characteristics of the device being assembled. Information contained in the Physical Block as attributes include the

manufacturer's identification, serial number, hardware and software revision information, and more importantly, data structures that provide a repository for other class components. As previously mentioned, the Physical Block is the logical container for all components in the device model; therefore, it must have access to and be able to locate all available resources instantiated by the device. The data structures provided by the Physical Block house pointers (Instance_ID) to these components thereby providing easy indirect access to them. In order for the Physical Block to resolve address queries from the network (i.e., a remote NCAP requests an attribute from the Physical Block), a hierarchical addressing scheme based on unique *Tags* (ASCII descriptions of the block or component name) that can be concatenated together to form fully qualified addresses is used to communicate with the device or device object across the network. The Physical Block is the centralized logical connector or backplane that the other Block classes *plug* into. Therefore, for the Physical Block to find other components in the system it must provide a *Locate* method.

The *Transducer Block* abstracts all the capabilities of each transducer that is physically connected to the NCAP I/O system. That is, during the device configuration phase, the TEDS information from the hardware device is read. This information describes what kind of sensors and actuators are connected to the system. This information is used by the physical block to create and configure the necessary type of transducer block.

The transducer block includes an I/O device driver style interface for communication with the hardware. The I/O interface includes methods for reading and writing to the transducer from the application-based function block using a standardized interface (i.e., *io_read* and *io_write*). The I/O device driver paradigm provides both plug-and-play capability and *hot-swap* feature for transducers. This means any application written to this interface should work interchangeably with multiple vendor transducers. In a similar fashion the transducer vendors provide an I/O driver to the network vendors with their product that supports this interface. The driver is integrated with the transducer's application environment to provide access to their hardware. This approach is identical to the *Ioctl* interface found in device drivers for mainstream operating systems such as MS-DOS and UNIX.

The *Function Block* provides a transducer device with a skeletal area in which to place application-specific code. The interface does not place any restrictions on how an application is developed. In addition to a *State* variable (which all block classes maintain), the Function Block contains several lists of *Parameters* that typically are used to access network-visible data or to make internal data available remotely. That is, any application-specific algorithms or data structures are contained within these blocks to separately allow for integration of application-specific functionality using a portable approach.

The *Network Block* is used to abstract all access to the network by the Block and Base classes using a network-neutral, object-based programming interface. The network model provides an application interaction mechanism based on the familiar remote procedure call (RPC) paradigm found in today's client-server distributed computing settings[3]. The RPC mechanism supports both a client-server and a publisher-subscriber paradigm for event and message generation. In support of the two types of application interaction, a communication model that centers around the notion of a *port* is defined in the specification. This means, if a block wishes to communicate with any other block in the device or across the network, it must first create a port that logically binds the block to the port name. Once enough information about the addressing of the port is known, the port can be bound to a network-specific block address. At this point the logical port address has been bound to the actual destination address by the underlying control network technology. Any transducer application's use of the port name is now resolved to the endpoint associated with the logical destination. This allows a late binding effect on application uses of the ports so that addresses are not hard-coded or dependent upon a specific architecture. The port capability is similar to the TCP/IP application-level socket programming interface where a socket is created and bound using the TCP/IP specific tuple: port number and Internet address in dotted notation. Once bound, the socket can be used for message and data transfer.

2.1.3 Base Classes

Base classes represent the basic building blocks used by the block classes. They are generally used within block classes to provide application functionality. The base classes include: *Actions*, *Events*, *Parameters*, and *Files*.

Actions provide a model for control interactions between the various block classes that define a system. Essentially, all actions are called using an *Invoke* method and may be either blocking or nonblocking in their communication of the action.

Events model the generation of asynchronous communication of signals in the system. That is, if an

application wishes to have a certain occurrence of something to happen at a given time in the system, then the designer simply creates an event with a certain time period. The underlying event generation and control mechanisms provided by the network will be used to support this capability.

The *Parameter* class represents network-visible variables in the model. Parameters have two methods associated with this class for reading and writing to these network accessible data storage locations. Parameters are typically found in the Function blocks to give access to network variables to executing applications.

Files provide a means for applications to up and download information to the device. The kinds of transfers of information are not specified nor are the structure of the data. Either stream or record-oriented data streams are used. A minimal file transfer state machine is defined in the specification.

This ends the brief discussion on the P1451.1 specification. The P1451.1 draft implemented for the demonstration results in a suite of software that represents the concrete reference implementation. Other parts of the demonstration require hardware resources and the implementation of the P1451.2 protocol specification. The hardware portion of the standard will be briefly discussed in the next section to provide the reader with some background information. The implementation as it relates to the demonstration will be discussed later in the demonstration architecture area.

2.2 **P1451.2 - Transducer to Microprocessor Interface**

The P1451.2 draft specification, *the Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*, provides a standardized digital interface and communication protocol that directly addresses the problem of interfacing multiple connection schemes with different buses and microprocessors. In addition, the standard includes the definition of a smart transducer interface module (STIM) and a transducer electronic data sheet (TEDS).

2.2.1 STIM

A STIM consists of a transducer, signal conditioning and conversion circuitry, a TEDS, and necessary logic circuit to implement the P1451.2 9-wire digital interface and protocols to communicate with a NCAP. A single transducer or up to 255 transducers can be supported by a STIM. Each transducer in a STIM is defined as a channel. A multichannel STIM is one that contains multiple transducers and thus form a multivariable STIM; for example, a temperature sensor, pressure sensor, and air flow sensor combined together to form a mass-flow sensor.

2.2.2 TEDS

A TEDS is a scaleable and extensible electronic data sheet that allows transducer manufacturers to *burn* in specific fields about their product such as manufacture date, version information, and calibration specifics, etc., into a small nonvolatile memory associated with the transducer hardware. The TEDS structure is divided into Meta-TEDS, Channel TEDS, Calibration TEDS, Application-specific TEDS, and Extension TEDS. Meta-TEDS contains the TEDS data field that is common to all transducers connected to the STIM. The field contains information such as overall description of the TEDS data structure, worst case STIM timing parameters, etc. Channel TEDS contains information for one specific transducer in a STIM. Each transducer has its associated Channel TEDS containing information such as physical units, uncertainty, upper/lower range limits, warm-up time, etc.

The Calibration TEDS contains valuable information such as calibration parameters, and calibration interval. The Application-specific TEDS are for application specific use by end-users, while Extension TEDS is reserved for implementing future and industry extension to P1451.2. The TEDS information provides a self-identification capability for transducers that is invaluable for maintenance, diagnostics, and determining mean-time-before-failure characteristics. This capability has generated a great deal of enthusiasm and is considered to be a potential boon to the sensor manufacturing industry.

2.2.3 Digital Interface

Communications between the STIM and NCAP are defined by a 9-wire physical specification and a set of protocol to access the TEDS information, read sensors, and set actuators. The data transfers are based on SPI-like (serial peripheral interface), bit-transfer protocol. The NCAP usually initiates a measurement or action by means of triggering the STIM, and the STIM responds with an acknowledgment once the requested function is completed. The STIM can interrupt the NCAP if an exception such as a hardware error, calibration failure, or self-test failure occurred.

3. Implementing the Object Model

The P1451.1 Object Model represents an abstract, object-oriented model for describing a network-capable transducer device, and is a good method for capturing the design requirements for such a device. However, in order to realize a reference implementation in software, the P1451.1 specification must first be transcribed from an abstract interface definition into an application programming interface based on a conventional programming language. Because the P1451.1 Object Model was defined using an object-oriented methodology, using C++ as the object-oriented language environment become a natural choice to map the model. The next section revisits each block class and briefly describes how they are implemented during migration from abstract interface definition to concrete reference implementation.

3.1 Class Implementation

The majority of the software development effort involving the reference implementation was spent on constructing and integrating the P1451.1 Object Model. This effort concentrated on implementing a C++ framework to support the *Block* and *Base* classes from the P1451.1 specification. The software necessary to realize an implementation of the P1451.1 classes will be discussed below in greater detail. The class implementation includes:

- The *Physical Block* provides a central hub for resources in the NCAP. In order to provide this capability certain implementation aspects have been derived. Namely, using the list attributes defined in the specification, a data structure can be built to house the pointers to the various components in the system. A *Locate* method is provided that parses the string addressing information to determine what pointer in the data structure needs to be de-referenced.

- The *Network Block* provides all the underlying network support needed by the transducer device. In order to support the networking aspects of the P1451.1 Draft, we needed to implement the network-neutral parts of the specification using a specific networking technology. Instead of using a vendor-specific control networking technology, we wanted to implement the network protocol using the ubiquitous TCP/IP protocol bundled with the Windows 95*** operating system. Specifically, the application-level implementation of the TCP/IP protocol suite from Microsoft — called WinSock Version 2.0. Therefore any P1451.1 API method or function call that requires the services of an underlying control network (i.e., *SendRequest*), would now use an equivalent application-level TCP/IP based function or macro to emulate those requests for services (i.e., *send*). TCP/IP was chosen because of its ubiquity, availability, and the developers familiarity with integrating it into the application environment. Before a block could communicate a request using the *SendRequest* API however, a port structure needed to be created and connected. The port capability was implemented using the socket API of TCP/IP. Clearly, in a real implementation of the standard however, ports represent a slightly lower-level of integration then do sockets in the parlance of TCP/IP. That is, whenever a block wants to communicate with another block, a new socket-like endpoint would not be created. Moreover, in this implementation, one socket is created using TCP/IP and every port structure created would simply use the singular socket connection to send its information. A socket for each port creation would be too much overhead on the operating system. Therefore the receiver of the message from the block in the TCP/IP implementation simply determines what port the message came from and redirects the message to the specific block. This method is more consistent with how current control network vendors would provide their implementations, i.e., using a pseudo interrupt-driven scheme for message arrival and delivery.

- The *Function Block* contains all vestiges of the user's custom transducer application environment. The function block with its defined attributes provides a skeletal envelope in which to package a user application. The function block uses the parameters it defined to communicate network variable information. In this implementation, the function block is rather generic in that it does not support a great deal of custom functionality. It merely sets up parameters to read the transducer information when called upon by a query process from across the network. In addition, the function block contains an event that can be initiated to simulate the event-based communication of reading sensor data from the NCAP. All these capabilities are set up in a rather sterile fashion so that the software tool developed for the demonstration can trigger or query the function block for the desired results.

- The *Transducer Block* provides the capability for the application to interact directly with the transducer interface using a device driver interface paradigm. As previously mentioned, in order for the transducer

device to communicate with the application in a standardized fashion, an I/O driver interface must be used. In the implementation, the driver interface has been setup to be a simplistic subset of the complete specification. This was a reasonable approach as all the demonstration required was the ability to read both the TEDS information and actual sensor readings from the transducer device. Therefore, the only method from the abstract interface definition given by the standard that was need by the implementation was by the *io_read* method. We have not yet implemented the capability to write actuator data because we did not utilize any physical actuators in the demonstration system. Likewise, we have not yet implemented the capability to write information to the TEDS fields.

3.2 A Library-based Implementation

The P1451.1 portion of the reference implementation was developed as a C++ dynamic link library (DLL). A dynamic link library contains executable images of function calls that an application will call and use. When an application calls a method or function contained in a dynamically linked library, the library that contains the image of the target function must be found by the Windows operating system and brought into main memory. Once this process has completed, the actual function executes and proceeds as if the called function were statically linked with the application's image. Clearly, this process occurs rapidly and provides an efficient means for managing reusable code and memory space within the operating system. The dynamic link library implementing all the P1451.1 NCAP functionality provides a convenient and centralized area for defining the functional interface definition of the specification. Applications that require NCAP-based services simply link with the DLL to access all the standardized methods defined in the P1451.1 specification.

The integration of the class software provides the reference implementation with the capability to support and interact with NCAP-based transducer applications. In order to utilize the software to retrieve and interact with actual sensors in the system, however, several hardware components representing the P1451.2 draft standard are needed. The next section provides a brief introduction to the hardware pieces used and how they were integrated in the reference implementation.

4. P1451.2 Hardware Resources

The hardware that was needed to demonstrate the P1451.2 digital interface between microprocessor and transducer included an actual pressure sensor input to the demonstration. In addition, the pressure sensor contains an on-board TEDS description to allow up and downloading of these fields. The hardware component of this demonstration illustrates how sensor/actuator manufacturers would use the P1451.2 standard to provide portable, plug-and-play, interoperable products for the process control industry as an example.

The hardware area as specified in P1451.2 has been encapsulated in this demonstration by using the parallel port in a PC connected to the pressure sensor through the 9-wire interface. The parallel port was used as it provides easy access to the software/hardware environment and it lends itself to easy integration into the PC environment - a major concern for us when developing this scenario.

5. Demonstration Architecture

In order to demonstrate the capabilities of the P1451 standard as proposed, it became necessary to pull all the software and hardware pieces together to form the reference implementation. The reference implementation provide the means to demonstrate the capabilities of the standard in an interesting venue.

This demonstration uses a software tool in a way that will exercise both aspects of the P1451 standard reference implementation. The tool uses the P1451.1 functional API to configure and query a remote NCAP over the network. As part of the configuration process, the hardware interface defined by the P1451.2 specification is implicitly used and exercised as well. A closer look at the system pieces that make up the demonstration are discussed next.

5.1 System Configuration

Figure 5 illustrates a high-level view of the architectural components needed to demonstrate the reference implementation. Two notebook computers form a small sub-network consisting of two nodes. The notebook computers, each simulating NCAP, connects to the network via an internal PCMCIA Ethernet interface card. Both notebooks run the Microsoft Windows 95 operating system and are configured to use the TCP/IP protocol stack (WinSock V2.0) that comes bundled with the system.

The physical network consists of an Ethernet-based, 10 Mbps, 10BaseT twisted-pair network. For compactness and demonstrative purposes, each node is connected directly to a pocket-sized 4-port 10BaseT Ethernet hub. This device provides the physical backbone network needed for the demonstration.

Essentially, the two notebooks form a client-server relationship over the network in order to demonstrate the standard. That is, one notebook (shown in Figure 5 as the Application PC) executes a *client* software application tool called *NCAPTool*.

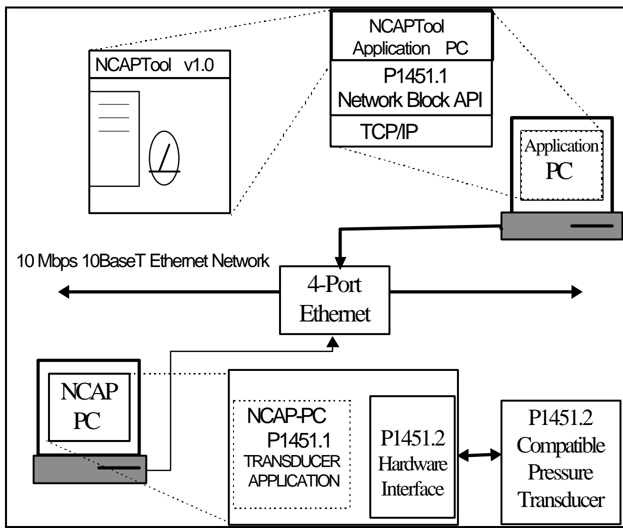

Figure 5: NIST Demonstration Architecture.

This software tool drives the demonstration by exercising the API associated with P1451.1 to query a server process executing on the remote NCAP-based notebook (shown in Figure 5 as the NCAP PC). The query process results in message-based method invocations sent from the client to the server API calls in this demonstration. All messages and invocations utilize the P1451.1 standard interface exclusively. All results from the server sent back to the client are packaged and received in a standardized form as well. Therefore, all interactions between the client and the server are carried out using the standardized interfaces defined by P1451.1, providing a complete standards-based environment for NCAP interactions.

5.1.1 Server Configuration

The server-based NCAP-PC is a fully functional PC-based version of a standard P1451 smart networked transducer device. Physically, the only difference between the PC implementation and an embedded application is that one has been targeted to the PC environment for demonstration purposes.

The software portion of the NCAP-PC implements the P1451.1 Information Model. The core software component found in both notebooks is comprised of the C++ DLL that implements the P1451.1 standard; however, the way in which the standard API and subsequent DLL interface are used varies between the Application-PC and NCAP-PC. For instance, using *NCAPTool*, the Application-PC only uses the standard P1451.1 network block interface API to configure and exercise the remote NCAP-PC for demonstrative purposes. Because the Application-PC does not have any associated transducer hardware, it does not require any P1451.2 standard capabilities. It merely uses the standardized functional interface provided by the P1451.1 based C++ DLL to access and manipulate the remote NCAP-PC capabilities. In fact, many test and diagnostic tools on the control networking market today use this style of interaction where the specific API of the vendor network is used as the PC-based entry point to access and manage remote node hardware and software via the network during configuration.

In theory, the C++ DLL developed for the NCAP-PC could be cross-compiled target to an embedded microprocessor environment "as is". The underlying TCP/IP networking code used to implement the network block would of course need to be rewritten to conform to the new target control network.

The hardware portion of the server-based NCAP-PC implements a fully compliant P1451.2 hardware interface that connects a pressure transducer to the demonstration system. Figure 6 illustrates a close-up view of the parallel port hardware connection that provides a means for physically connecting the P1451.2 9-wire interface to the parallel port.

Notice the use of the parallel port device on NCAP-PC. The parallel port provides a reasonable way to demonstrate a PC-based implementation of the 9-wire digital interface proposed in P1451.2. The P1451.2 compatible pressure sensor was built and provided by SSI Technologies. The sensor's 9-wire interface is terminated with a DB-9 connector. The DB-9 connector was then

mated with a 25- pin parallel port cable and plugged into the printer port of the notebook computer. A special device driver was developed that would translate the application request for hardware service into standard API calls using the *Transducer Block* of P1451.1.

This special parallel port driver was needed to supplant the original printer port device driver in order to provide the software linkage to all P1451.1 compliant transducer block API calls bound for the pressure sensor. The driver then translates any P1451.1 transducer block API *method* (i.e., a function call such as *io_read* that actually reads the sensor data) into its compatible 9-wire signaling protocol as defined in the P1451.2 digital interface specification. The firmware implementation of the digital interface would then interact with the pressure sensor hardware to request the data; ultimately sending the data back to the transducer block where the request originated.

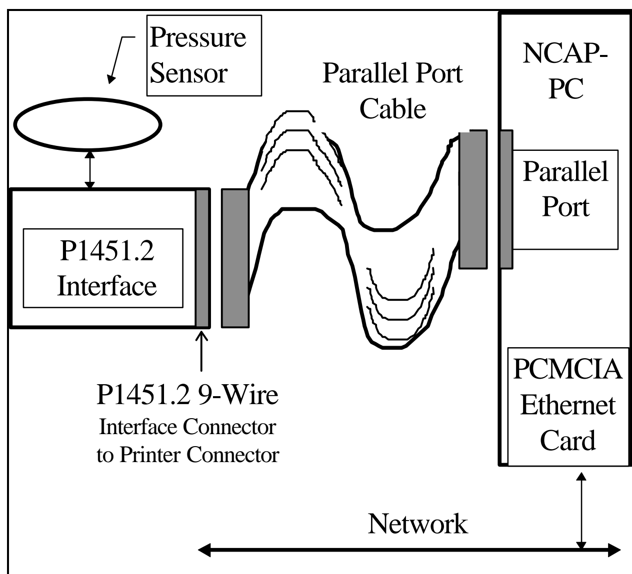

Figure 6: Parallel Port Interface Implementation.

This transducer interface was completely built conforming to the P1451.2 interface specification. A specially developed parallel-port device driver was used to create the PC-based platform from which the application of the standard could be tested. This provide both accessability and ease of use.

The SSI Technologies' pressure sensor is the key hardware component illustrated in this demonstration.

Having the hardware located on NCAP-PC, combined with the ability to access the data from it remotely, illustrates the powerful accessibility options from both standards. NCAP-PC implements the full P1451 standard providing intra- as well as inter-NCAP access to its object attributes, embedded TEDS information, and sensor data.

5.1.2 Client Configuration

The client configuration does not require all the services of P1451.1 to operate. However, we used the same DLL based on the C++ implementation of the P1451.1 specification that the server application uses. This was possible, because all that we needed from the DLL was the class definition for the network block. To drive the NCAP-PC demonstration, we developed a Windows 95-based configuration tool to interact with an NCAP across the network using the portable application and network-capable framework of P1451.1. The graphical user interface based diagnostic tool we developed, called *NCAPTool,* was compiled and linked with the P1451.1-based DLL to provide a visual front-end into the standardized NCAP methods or function calls.

The *NCAPTool* application links with the C++ DLL to supply a class instantiation of the network block. As part of the *NCAPTool* initialization, a network block class is instantiated or created. From this point on all queries made to the remote NCAP-PC using the tool would be facilitated through the network block of *NCAPTool*. When the server process executes, one of its functions is to wait or listen to incoming connections (using the parlance of TCP/IP). When a connection occurs (the client is trying to communicate) a duplex transport connect is then established between the client and server. Queries and other requests then use the channel that has just been established. All TCP/IP functions have been embedded into the network block to envelope the associated API. This facilitates network-neutral control network technology.

6. Demonstrating the Standard

A useful question one might ask at this point is, "what elements from the P1451 draft standard can be shown and how can they be demonstrated using this reference implementation?" To answer this question we need to focus on the capabilities of a Windows 95-based client *NCAPTool* application we developed for this purpose.

Using *NCAPTool*, a user can configure, diagnose, exercise, and control a remote device on the network by interactively issuing API-based methods defined by the P1451.1 Draft. The tool allows the operator to access specific object attributes, read sensor data from the device,

and retrieve the TEDS's manufacturing and calibration data from across a network protocol based on TCP/IP. Specifically, the tool demonstrates how the standard may be used in a diagnostic setting to illustrate several key ideas from both parts of the P1451 standard; including the ability to:

- provide standardized read and write access to the smart transducer device over a network,
- exercise the digital communication protocol and hardware interface to access the sensor data and TEDS information from the transducer device remotely,
- illustrate the logical block addressing scheme to access object attributes, and finally,
- demonstrate the network-neutral API for sending and receiving information over the network.

In general, the software tool accomplishes these tasks by allowing the operator of the NCAPTool to enter specific query instructions using text boxes, pull-down menus and individual windows. For instance, an operator may enter a request for a serial number or software revision number on the remote NCAP-PC. Alternately, an operator may issue read requests on the remote NCAP-PC to obtain the sensor data directly from the transducer. In addition, the TEDS information may be retrieved from the remote NCAP. Object addressing and block interactions are also demonstrated using individual windows with specific menus for requesting the information. A graphical window can be popped up which shows a dial gauge of the pressure sensor updated with the current pressure in real-time illustrating events and data transfer over the network directly from the remote NCAP-PC.

It becomes easy to envision how such a tool might be used remotely over the Internet to help a technician retrieve information about a particular sensor. That being said, we chose the diagnostic and monitoring venue because it is familiar to most network vendors, system integrators, sensor manufacturers, and users.

7. Summary

Currently transducer manufacturers, system integrators, network vendors, and users are faced an enormous problem of trying to support multivendor networks and bridge the smart transducer market across industries. In order to motivate the necessity of the IEEE P1451 standards effort, the relevant problems have been presented here in detail. The solutions to these problems have also been addressed by the two-part P1451 draft standard, *A Smart Transducer Interface for Sensors and Actuators*. To illustrate the salient features of these draft specifications in a demonstration setting, a reference implementation has been developed by NIST researchers. The reference implementation represents a concrete example of the Information Model from P1451.1 as well as the digital hardware interface implementation from P1451.2.

To exercise various features from the standard, a software tool called *NCAPTool* was developed to provide visual interaction so that a user can query the NCAP for specific standardized information. This information includes object attributes, TEDS information, and sensor data from the transducer.

In an effort to illustrate the integration of the P1451.1 and P1451.2 draft standard implementation, a P1451.2-compatible pressure sensor is interfaced to the P1451.2 reference implementation through a parallel port. The software tool combines all the key features of the IEEE P1451 Draft Standard to provide a demonstration of its most pragmatic elements.

8. References

[1] *IEEE P1451.1, Draft Standard for a Smart Transducer Interface for Sensors and Actuators — Network Capable Application Processor (NCAP) Information Model*. Institute of Electrical and Electronics Engineers, Inc., New York, to be submitted, 1996.

[2] *IEEE P1451.2, Draft Standard for a Smart Transducer Interface for Sensors and Actuators — Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*. Institute of Electrical and Electronics Engineers, Inc., New York, August 1996.

[3] Birrel, D. A., Nelson, B. J., *Implementing Remote Procedure Calls*, ACM Transactions on Computer Systems, Vol. 2, No. 1, February 1984, Pages 39-59.

9. Acknowledgment

During the development and integration of the reference implementation, several companies and individuals were very helpful in bringing this demonstration to fruition.

A special thanks to Stan Woods and Alec Dara-Abrams of Hewlett-Packard who have been instrumental in providing the special parallel port driver for use in the implementation.

SSI Technologies supplied much of the hardware both in emulation form and actual hardware implementations of the P1451.2 draft standard. The hardware provided by SSI was instrumental in developing the interfacing between the transducer block and the hardware I/O driver software.

We would especially like to thank the reviewers of this document for their advice and encouragement during this manuscript development process.

*** Certain commercial products are identified in this paper in order to adequately describe the proposed standard. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose or the only ones that could be used.