

**A VHDL Software Model for Networking Smart Transducers  
through Bluetooth Technology**

By

Deepika Devarajan, B.E.

The Ohio State University, 2001

Professor Steven B. Bibyk, Adviser

Wireless smart transducers have rapidly become popular for several control applications. The interfacing of smart transducers to a wireless medium is an attractive solution for reconfigurable networks but this has not been explored extensively for small-span networks using existing wireless technology. Wireless sensors today are equipped with RF capability at the physical layer but do not lend themselves to cooperative networking. Smart transducer interfaces have recently been standardized to enable the interoperability of smart transducers with a variety of instruments, microprocessor based systems, fieldbus and control networks and using this interface with an existing wireless networking technology provides a quick, cost-effective, ubiquitous and simple solution for reconfigurable, cooperative wireless smart transducer networks.

In order to enable the smart transducer interface to operate with a wireless networking technology we propose a “network infrastructure” in this thesis and investigate the definition and design of a software model for such an infrastructure. The network communication models standardized by the IEEE 1451 and the different wireless alternatives that may be adopted for autonomous smart transducer networks are examined and evaluated. The Bluetooth wireless technology is chosen for the wireless interface and the OBEX Session protocol is used for network communication between the smart transducers in a client-server network architecture. The software model for the network communication is described via behavioral VHDL constructs and simulation outputs corresponding to the network operation are obtained.

## TABLE OF CONTENTS

<b>ABSTRACT.....</b>	<b>ii</b>
<b>DEDICATION.....</b>	<b>iv</b>
<b>ACKNOWLEDGEMENTS.....</b>	<b>v</b>
<b>VITA.....</b>	<b>vii</b>
<b>LIST OF FIGURES.....</b>	<b>x</b>
<b>LIST OF TABLES.....</b>	<b>xii</b>
<b>LIST OF ABBREVIATIONS.....</b>	<b>xiii</b>
 <b>CHAPTERS:</b>	
<b>1. INTRODUCTION .....</b>	<b>1</b>
1.1 EMBEDDED SOLUTIONS FOR WIRELESS SMART TRANSDUCER NETWORKS...	1
1.2 CONTRIBUTIONS OF THIS THESIS.....	4
1.3 THESIS OUTLINE.....	5
<b>2. REVIEW AND BACKGROUND.....</b>	<b>7</b>
2.1 APPLICATIONS OF WIRELESS SMART TRANSDUCER NETWORKS.....	8
2.2 SMART TRANSDUCERS.....	9
2.3 SMART TRANSDUCER NETWORKS.....	10
2.4 IEEE 1451 SMART TRANSDUCER INTERFACE FOR SENSORS AND ACTUATORS.....	12
2.5 THE IEEE 1451 NETWORK COMMUNICATION SPECIFICATION.....	23
2.6 WIRELESS SOLUTIONS FOR SMART TRANSDUCER NETWORKS.....	28
2.7 THE BLUETOOTH WIRELESS SPECIFICATION.....	34
2.8 OBEX WITH BLUETOOTH.....	46
<b>3. DESIGN OF IEEE 1451.1 COMMUNICATION ON BLUETOOTH.....</b>	<b>58</b>
3.1 WIRELESS SMART TRANSDUCER NETWORK ARCHITECTURE .....	58
3.2 THE IEEE 1451.1 NETWORK INTERFACE.....	60
3.3 THE OBEX SESSION PROTOCOL INTERFACE.....	64
<b>4. VHDL IMPLEMENTATION OF THE NETWORK INFRASTRUCTURE.....</b>	<b>66</b>

4.1 NATURE OF THE VHDL IMPLEMENTATION.....	66
4.2 ENTITY AND ARCHITECTURE DESCRIPTIONS.....	67
4.3 THE TEST BENCH DESIGN.....	72
<b>5. CONCLUSIONS AND FUTURE WORK.....</b>	<b>76</b>
5.1 CONCLUSIONS.....	76
5.2 FUTURE WORK.....	77
<b>LIST OF REFERENCES.....</b>	<b>79</b>

## List of Figures

<b>Figure</b>		<b>Page</b>
2.1	THE IEEE 1451 CORE PROTOCOLS [JOH00]	14
2.2	NETWORKED SMART TRANSDUCER MODEL [STD99]	16
2.3	THE IEEE 1451.1 CARD CAGE OBJECT MODEL [FRA00]	17
2.4	THE IEEE 1451.2 TRANSDUCER-MICROPROCESSOR INTERFACE [LEE00]	20
2.5	THE IEEE P1451.3 MULTIDROP CONFIGURATION [LEE00]	21
2.6	THE IEEE P1451.4 MIXED MODE TRANSDUCER CONFIGURATION [LEE00]	23
2.7	THE IEEE 1451.1 CLIENT-SERVER NETWORK COMMUNICATION MODEL [STD99]	25
2.8	THE CLIENT-SERVER COMMUNICATION SEQUENCE DIAGRAM	27
2.9	THE BLUETOOTH CONNECTION STATE DIAGRAM [HAA00]	38
2.10	BLUETOOTH PICONETS AND SCATTERNET	38
2.11	THE BLUETOOTH PROTOCOL STACK [MET98]	40
2.12	OBEX HEADER FORMAT	49
2.13	THE OBEX CLIENT-SERVER COMMUNICATION	50
2.14	THE CLIENT-SERVER DISTINCTION IN OBEX	53
2.15	THE RELATIONSHIPS BETWEEN THE PROFILES	55
3.1	WIRELESS SMART TRANSDUCER NETWORK ARCHITECTURE	59

3.2	OBEX SESSION FOR A SMART CLIENT TRANSDUCER	64
3.3	PACKETS SENT DURING OBEX COMMUNICATION	65
4.1	THE CLIENT OBEX INTERFACE AND ITS COMPONENTS	67
4.2	THE CLIENT OBEX INTERFACE ENTITY	68
4.3	THE OBEX OBJECT ENTITY	70
4.4	THE OBEX SESSION ENTITY	71
4.5	FIRST SIMULATION OUTPUT	73
4.6	SECOND SIMULATION OUTPUT	74
4.7	THIRD SIMULATION OUTPUT	74
4.8	FOURTH SIMULATION OUTPUT	75
4.9	FIFTH SIMULATION OUTPUT	75

## List of Tables

<b>Table</b>		<b>Page</b>
2.1	COMMON OBEX OPERATIONS AND RESPONSES	50
2.2	COMMON OBEX HEADER IDS	51

## List of Abbreviations

ACL	Asynchronous Connection-Less
CAN	Controller Area Network
CEBUS	Consumer Electronics Bus
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CVSD	Continuous Variable Slope Delta
DECT	Digital Enhanced Cordless Telecommunications
DSP	Digital Signal Processor
DSSS	Direct Sequence Spread Spectrum
ETSI TS	European Telecommunications Standards Institute's Technical Standard
FHSS	Frequency Hopping Spread Spectrum
FP	File transfer Profile
GAP	Generic Access Profile
GOEP	General Object Exchange Profile
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IrDA	Infrared Data Association
IrMC	Infrared Mobile Communications
IrOBEX	Infrared OBject EXchange
JVM	Java Virtual Machine

L2CAP	Logical Link Control and Adaptation Protocol
LIAC	Limited Inquiry Access Code
LMP	Link Manager Protocol
LOS	Line Of Sight
MAC	Media Access Control
MCU	Micro-Controller Unit
MEMS	MicroElectroMechanical Systems
NCAP	Network Capable Application Processor
NIST	National Institute of Standards and Technology
OBEX	OBject EXchange
OPP	Object Push Profile
OSI	Open Systems Interconnection
PCM	Pulse Coded Modulation
PIM	Personal Information Management
PPP	Point-to-Point Protocol
QoS	Quality of Service
RPC	Remote Procedure Calls
RSSI	Received Signal Strength Indicator
RTL	Register Transfer Language
SCO	Synchronous Connection Oriented
SDP	Service Discovery Protocol
SIG	Special Interest Group

SoC	System-on-a-Chip
SP	Synchronization Profile
SPI	Serial Peripheral Interface
STIM	Smart Transducer Interface Module
SWAP	Shared Wireless Access Protocol
TBIM	Transducer Bus Interface Module
TCP	Transmission Control Protocol
TCS	Telephony Control protocol Specification
TEDS	Transducer Electronic Data Sheet
TII	Transducer Independent Interface
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UUID	Universally Unique Identifiers
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WLAN	Wireless Local Area Network
WUG	Wireless User Group

# CHAPTER 1

## INTRODUCTION

### *1.1 Embedded solutions for Wireless Smart Transducer Networks*

The integration of micro-machining and microelectronics has led to a revolution in the control systems area through the development of MicroElectroMechanical Systems (MEMS). MEMS allows the integration of micromechanical structures, sensing elements, and signal conditioning blocks in a single sensor module. Recent advances in IC manufacturing processes allow the incorporation of computational capability such as a Digital Signal Processor (DSP) or Micro-Controller Unit (MCU) with an integrated sensor module on a single chip. These highly integrated monolithic circuits are targeted at both extremely specialized as well as general sensor applications and are part of the System-on-a-Chip (SoC) revolution. SoC sensors that are mass-produced using MEMS technology are inexpensive, expendable, shock-resistant, and lend themselves to easy installation and remote control and are finding interesting applications such as “Camera-on-a-Chip”, monolithic hearing aids, and vehicle dynamics sensor.

Sensor networking is a key issue for monitoring and control applications. The embedding of increasing amounts of intelligence in transducers (sensors/actuators) has led to the development of smart “information” transducers. The diversity of the transducer market has resulted in a profusion of several low-cost solutions for smart transducer networking

that are targeted towards specific applications. Some of the popular fieldbus and control network solutions used today are the Controller Area Network (CAN) in the automotive industry, LonTalk™ for factory automation in industrial control networks, and BACNet for office and building automation [FRANK00].

Recent industry efforts towards interoperability in a wide range of applications have attempted to focus on the definition, functionality, and communication protocol standards for smart transducers. The Institute of Electrical and Electronics Engineers (IEEE) and the National Institute of Standards and Technology (NIST) have defined the IEEE 1451 set of standards for a Smart Transducer Interface for Sensors and Actuators in an effort to overcome the incompatibility issues that arise while interfacing smart transducers to instruments, microprocessor-based systems, fieldbus and control networks [LEE\_K00]. The objective of these standards is to define an architecture that allows transducers to connect into any live distributed control network in a true “plug-and-play” fashion, such that automatic system identification and configuration is facilitated.

The use of wireless communications for networking smart transducers is an attractive option and facilitates a reconfigurable, mobile, wireless connectivity model. Wireless sensors have rapidly become popular for several applications wherein sensors are equipped with a RF communication capability [WSEN01]. However, in current implementations the RF communication is similar to a bit-pipe and doesn't lend itself to networking or “plug-and-play” operation. Recent research efforts in wireless sensor networks have focused on designing low-power, energy-efficient protocols for

communication between sensor nodes, where typically thousands of sensor nodes participate in the communication [SINA00, AGR00]. Another approach for wireless networking of smart transducers is to embed an entire communications protocol stack that includes a wireless networking technology but this may be overkill for low-power and low-cost sensors.

In this thesis, we argue that for small-span networks, the use of an existing wireless technology is a quicker, cheaper, and simpler solution for networking transducers rather than the design and development of a new low-power protocol stack. The main requirements of such a transducer network would be prolonged low-power operation, ability to form self-organizing networks based on proximate connectivity, and the use of a “light” communication protocol that may be easily embedded in the smart transducer nodes. An example application for a small-span wireless smart transducer network is within an Unmanned Aerial Vehicle (UAV) that is remotely monitored by a high speed RF link. The UAV may possess one or more “piconets” for monitoring environmental condition (ex. Temperature), engine condition (ex. Motor speed), aircraft condition (ex. Fuel tank), and the actual application of the UAV (ex. Law enforcement) [BIB00, WINS00, NASA00].

An analysis of the existing short-range wireless networking technologies, viz., Infrared Data Association (IrDA), HomeRF, IEEE 802.11x, and Bluetooth™<sup>1</sup> suggests Bluetooth as the best alternative for ubiquitous, wireless smart transducer networking on account of

---

<sup>1</sup> Bluetooth is a trademark owned by its proprietor and used by companies under license.

its low-power operation, easy embeddability into smart transducer nodes, and its capability of proximity networking.

The design and development of a *network infrastructure* for interfacing IEEE 1451 Smart Transducers to a Bluetooth network may be carried out in either software or hardware. For embedded systems, it is advantageous to include more complexity in hardware, with semiconductor and VLSI technology paving a way for SoC implementations [SOC99]. In addition, smart transducers typically possess limited or no software capability and may not be powered by an operating system, or a Java Virtual Machine (JVM). We adopt a software approach to develop a hardware design of the *network infrastructure* that allows the easy transition to embedded hardware implementation.

### ***1.2 Contributions of this thesis***

This thesis defines a software model of the *network infrastructure* for communication of IEEE 1451 smart transducer nodes on a Bluetooth network. The thesis introduces smart transducer networks and the IEEE 1451 specification for interfacing smart transducers. The thesis investigates interfacing IEEE 1451 based smart transducers with existing wireless networking technologies as a solution for wireless smart transducer networks. It examines the various alternatives for networking smart transducers on existing wireless technologies and suggests the use of Bluetooth for the establishment of autonomous smart transducer piconets. The network communication models specified in the IEEE 1451 standard and the Bluetooth protocol stack are studied and a client-server network

model using the OBject EXchange (OBEX) Session protocol is designed. The client-side functionality is modeled in VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) using behavioral constructs and a proof-of-concept implementation of the *network infrastructure* is presented.

### ***1.3 Thesis Outline***

Chapter 2 introduces smart transducers, networking smart transducers and the recently standardized IEEE 1451. It also provides a brief overview and comparison of existing wireless technologies that may be used in wireless smart transducer networks and examines the Bluetooth wireless specification which has been selected as the best alternative, and the OBEX protocol in detail.

Chapter 3 addresses the issues for developing the *network infrastructure* by examining the network communication models outlined in the IEEE 1451 standard and the interaction with the OBEX protocol. It also outlines the design of an IEEE 1451 Smart Transducer interface for a client-server model using the OBEX Session protocol.

Chapter 4 explains the behavioral modeling of the client-side interface in VHDL of a proof-of-concept implementation and presents some simulation outputs that describe the network communication.

Chapter 5 summarizes and concludes the thesis with recommendations for future research in this area.

## **CHAPTER 2**

### **REVIEW AND BACKGROUND**

There are many applications of interest that need wireless smart transducer networks. In this thesis, we propose the use of existing wireless networking technologies to network smart transducers since such a solution is cost effective, scaleable, and simple to implement. In order to develop such a solution, we require an interface between any given smart transducer and a wireless networking technology of choice. There is a variety of wireless networking technologies available in the market. The question then arises as to how to select a wireless networking technology and once it is selected, how to interface a smart transducer to it. We address this in this chapter by comparing standard wireless networking technologies, investigating the available interfaces for smart transducers, and elaborating on the wireless technology of choice. Smart transducers are specified by the IEEE 1451 standard and this chapter provides an overview of smart transducers and smart transducer networks. We briefly describe the IEEE 1451 specification and the associated network communication models [1451STD99]. We then focus on wireless networking of smart transducers and a comparison of existing wireless technologies is made. We provide an overview of the best wireless alternative, Bluetooth and the specific protocol of interest for smart transducer networking in the Bluetooth stack, OBEX.

## *2.1 Applications of Wireless Smart Transducer Networks*

Consider a remotely monitored autonomous airborne vehicle, commonly known as an Unmanned Aerial Vehicle (UAV). UAVs may be used for a variety of applications, typically to monitor surrounding conditions or activity while airborne, perform some on-board processing, and transmit the information to a Ground Station. The military may use UAVs for monitoring enemy activity, the environmentalists for atmospheric pollution control, and the law enforcers for surveillance operations. The applications for UAVs are limitless since they provide an inexpensive, remote autonomous monitoring solution [BIB00].

UAVs inherently require several smart sensor modules that can coherently interoperate. These smart sensors may be used to perform the main application of the UAV (ex. gas sensor for pollution control) and to monitor the environmental conditions for the flight of the UAV (ex. wind velocity sensors). The smart sensors may also monitor the engine condition of the UAV (ex. motor speed sensor) as well as the health of the vehicle (ex. pressure within the UAV) [NASA00].

The multitude of smart sensors within the UAV are required to be networked intelligently such that only the relevant information collected is transmitted to the Ground Station. An onboard “central server” node can collect the information that has been obtained by the smart “client” sensors, process it and transmit it to the Ground Station on a high-speed

wireless link. The central server node may also be designed to receive instructions from the Ground Station (ex. decrease current acceleration) and transmit this information to the appropriate actuator.

The networking of the smart transducers within a UAV on a wireless medium enables autonomous, reconfigurable, ubiquitous piconets to be formed without the onus of cumbersome wiring. Smart transducers may be interfaced to the piconets in a “plug and play” manner allowing scalability and the use of UAVs for different applications. The distributed network architecture described in the above example is applicable for several monitoring and control scenarios such as factory or building automation, on-board networks in automobiles, unattended ground operations by the military, and in harsh terrain or climate conditions.

## ***2.2 Smart Transducers***

The phrase “smart transducer” has been interpreted to mean a variety of intelligent sensing devices over the past twenty years as sensor products, with highly sophisticated features are being brought to market [SMART00].

Previously, transducers capable of operating with digital data qualified as smart transducers. A more recent requirement of smart transducers is their ability to operate on the sensed signal in a logical fashion to increase the value of information that the transducer processes [RNJ00]. These “information transducers” embed local intelligence

with the transducer element typically with built-in processors. Smart transducers have migrated to special purpose devices that may support advanced features such as local control, self-diagnostics, adaptive calibration, automatic compensation, etc. Smart transducers are now capable of self-identification when interfaced to a network using electronic data sheets thus shifting towards a “plug-and-play” environment. They are also characterized by various degrees of communication capability depending on the networking requirements of the sensor system. The ease of deployment of smart transducers and integration into any networked application environment, coupled with the advanced sophisticated features that they offer for a small price has popularized their use in the industry.

### ***2.3 Smart Transducer Networks***

Smart transducers may be networked together to form a distributed system with each transducer node performing similar or different sensing operations. For example, fiber optic sensors may be networked to test the structural integrity of smart structures or the remote monitoring of a UAV may be possible through a localized sensor network within the vehicle.

The diversity of the transducer market has resulted in a profusion of several low-cost solutions for networked smart transducers resulting in various field bus and control networks targeted towards specialized applications. We can see several examples of such

specialized applications in the form of transducer specific networks in diverse industries such as the automotive industry, in factory automation, and office building automation.

The Automotive Industry has adopted several control networking protocols that are suitable for the large volume productions and multiplexed wiring. The SAEJ1850 established by the Society of Automotive Engineers is the industry standard for multiplexing and data communications in US Automobiles and defines the application, data link, and physical layers of the Open Systems Interconnection (OSI) Networking Model. The Controller Area Network (CAN) protocol popular in trucks and for On-Board Diagnostics functions is preferred for high-speed data communications and supports distributed real-time control. Other protocols supported in the automotive segment are the A-Bus developed by Volkswagen AG and the Vehicle Area Network developed by Renault Auto [FRANK00].

Industrial networks use distributed multiplex systems for factory automation and have manifested through several different protocols. CAN has been adapted for industrial networks in systems such as DeviceNet™ for low-cost, low bandwidth communications. The LonTalk™ protocol adopted by the semiconductor equipment manufacturing industry defines all seven layers of the OSI model and is different on account of its peer-to-peer architecture. Profibus, HART, Topaz, ARCNet™ are examples of other protocols that are popular for factory automation applications. The wide variety of existing networking protocols has prompted industry to migrate towards an open standard that can offer “plug-and-play” capability, such as the Fieldbus. Fieldbus refers to a non-

proprietary digital two-way communication standard that defines application, data link, and physical layer services of the OSI model for the process automation industry. The slow standardization of the Fieldbus and the lack of semiconductor products in the market to implement the sensor nodes have retarded the initial excitement for this new standard [FRANK00].

Office and Building Automation has been mainly implemented with the BACnet protocol used for energy management in smart buildings and is being supported by protocols for meter reading, security system, and self-diagnostics that are not yet standardized. Home Automation is migrating from the X-10 protocol previously used for lamp and appliance controls towards the Smart House Application Language protocol. The Consumer Electronics Bus (CEBus) and LonTalk™ are other contenders for building automation [FRANK00].

Interfacing transducers to the numerous existing control networks with support for an extensive set of communication protocols requires significant effort and expense from transducer manufacturers [RNJ00]. Transducer manufacturers and system integrators are required to redesign the transducer interfaces for each type of multivendor network. This redesign process is expensive, time consuming and prevents interoperability of sensors due to proprietary or unique interfaces.

## ***2.4 IEEE 1451 Smart Transducer Interface for Sensors and Actuators***

Recent industry efforts towards interoperability in a wide range of applications have attempted to focus on the definition, functionality, and communication protocol standards for smart transducers. The IEEE and the NIST have defined the IEEE 1451 set of standards for a Smart Transducer Interface for Sensors and Actuators in an effort to overcome the incompatibility issues that arise while interfacing smart transducers to instruments, microprocessor-based systems, fieldbus and control networks.

The objective of these standards is to define an architecture that allows transducers to connect into any live distributed control network in a true “plug-and-play” fashion, such that automatic system identification and configuration is facilitated [LEE\_K00]. The IEEE Working Group for the standards has attempted to achieve this by specifying a network independent object model for interfacing smart transducers to any target network and descriptive datasheets embedded within smart transducers for self-identification of each device. As a result, the same transducers can be used on multiple control networks and the selection of a control network for measurement, and control application is no longer dependent on transducer compatibility constraints. This would also encourage transducer application developers to shift their focus to adding value and innovation to their applications rather than developing interfaces for different networks.

Smart transducers (either smart sensors or actuators) typically follow a signal chain composed of a raw transducer element, signal conditioning block, signal conversion

block, a microprocessor block, and a communication transceiver [RNJ00]. The core of the IEEE 1451 set of standards attempts to provide a layer of abstraction between the microprocessor and the signal conversion blocks, as well as the communication transceiver and the field network to provide network independence to the smart transducers. Thus, the standards provide a means to achieve transducers-to-network interchangeability as well as transducer-to-networks interoperability [LEE\_K00].

The IEEE 1451 family of standards is comprised of four independent standards, 1451.1, 1451.2, P1451.3 and P1451.4, the latter two still waiting approval from the Standards Committee. The standards may be used either separately or together as part of a complete IEEE 1451 solution [CAN99].

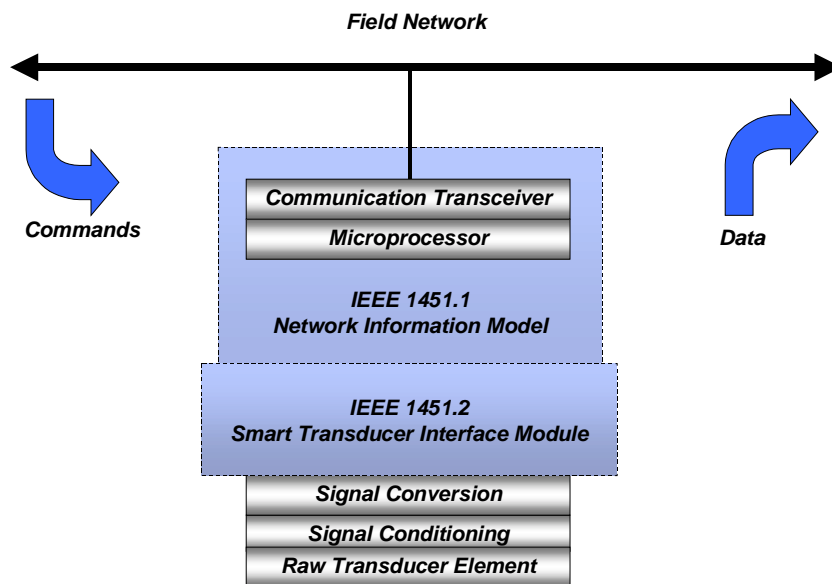


Fig. 2.1 The IEEE 1451 Core Protocols [RNJ00]

The specifications are defined in such a manner that each higher numbered specification is completely contained within the confines of a lower numbered specification. This provides tight encapsulation of the hardware details at each level, similar to the data hiding concepts in object-oriented programming or protocol layering definitions in networking.

#### ***2.4.1 The 1451.1-1999 Standard for a Smart Transducer Interface for Sensors and Actuators – Network Capable Application Processor (NCAP) Information Model***

The IEEE 1451.1 standard defines a network independent Common Object Model to enable the interfacing of smart transducers to any network [1451STD99]. Standardized access to a physical transducer is possible by a programmable interface that may be provided by transducer manufacturers or network vendors.

The 1451.1 specification provides the object-oriented definition of a Network Capable Application Processor (NCAP) that encapsulates the details of the transducer hardware implementation. It also provides the definition of application-level access to network resources and the framework for application access to transducer hardware.

The IEEE 1451.1 architecture defines three types of models – an object model for the software components of the IEEE 1451 system, a data model for the information communicated across the specified object interfaces, and a pair of network communication models.

The networked smart transducer model is described in the specification through two views, the physical and logical views.

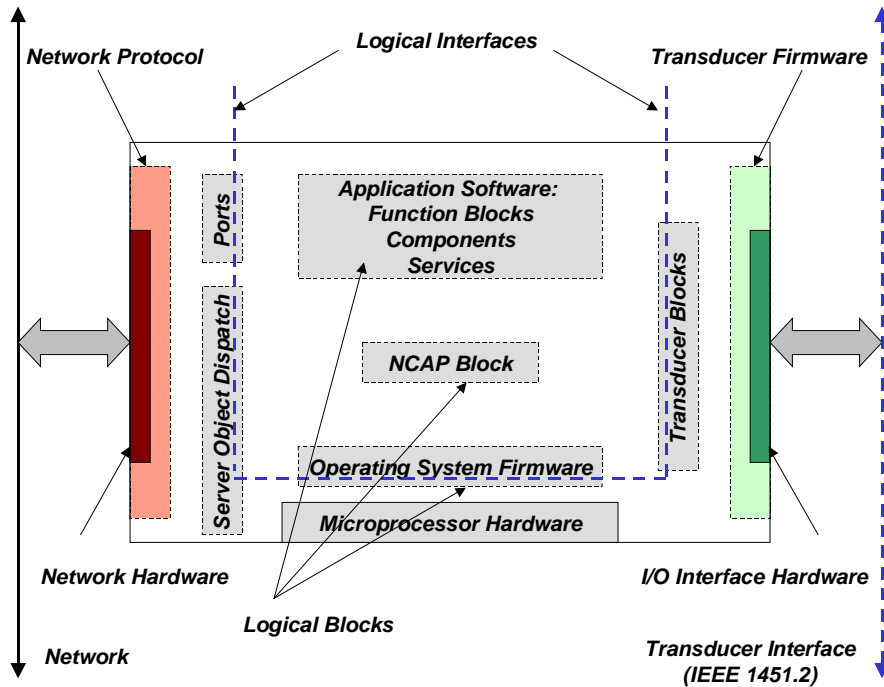


Fig. 2.2 Networked Smart Transducer Model [1451STD99]

The physical view of the system identifies the actual components present in the system, viz. the network hardware, I/O interface hardware, microprocessor hardware, transducers and the communication network. The logical view of the system can be a view of the logical components or the logical interfaces of the system. The application software on the NCAP, operating system firmware, network protocol, etc are logical components and may be grouped as application and support components. The logical interfaces defined by this view are the network abstraction interface, transducer abstraction interface, and the

logical interface to NCAP support between the operating system firmware and the microprocessor hardware [1451STD99].

The information model is represented as a set of object classes, attributes, methods and behavior that provide an abstract view of device characteristics in object-oriented terms [LEE\_K96]. The card-cage paradigm that is commonly used to describe plug-in I/O functionality in motherboards is used here to describe the different functionality that may be used in the NCAP model.

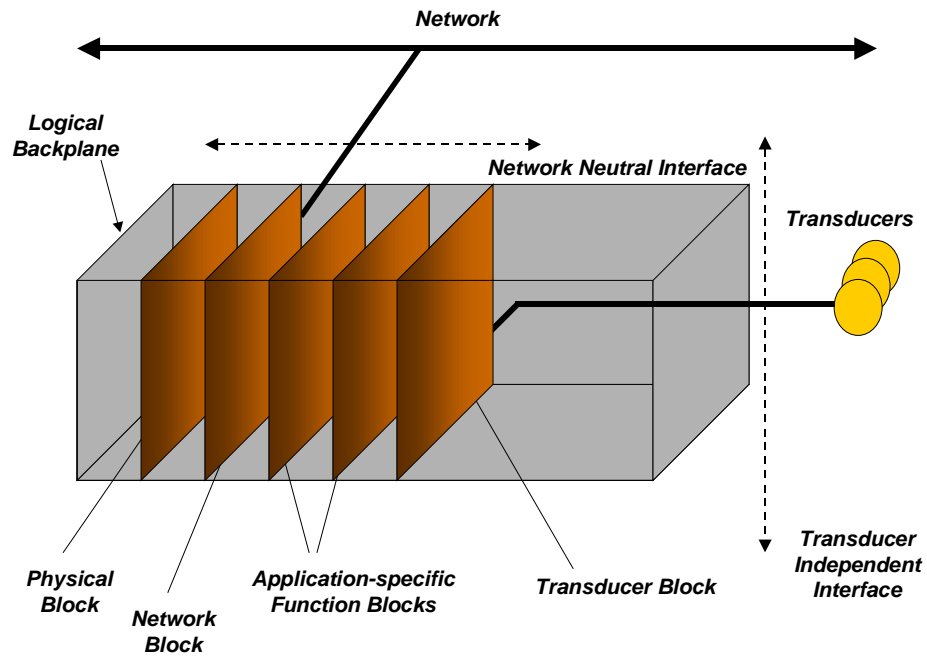


Fig. 2.3 The IEEE 1451.1 Card Cage Object Model [FRANK00]

The blocks that may be implemented in an IEEE 1451.1 design include a Physical Block, one or more Transducer, Function, and Network Blocks. The Physical Block provides the backplane for the card-cage structure and contains the logical hardware and software

resources in the model. The Transducer Blocks abstract the capabilities of each transducer that may be connected to an NCAP, and are primarily used in the configuration phase. The Function Blocks provide a skeletal area for application-specific data and methods that may be plugged in whenever necessary. The Network Blocks are used to encapsulate all network communication operations from the rest of the NCAP functionality through a network neutral interface. The Network Blocks are based on the principles of Remote Procedure Calls (RPC) commonly used in distributed systems [LEE\_K96]. The interfaces that are exposed by the IEEE 1451.1 Model are the interface to the transducer modules and that to the network.

#### ***2.4.2 The 1451.2-1997 Standard for a Smart Transducer Interface for Sensors and Actuators – Transducer to Microprocessor Communication Protocols and TEDS (Transducer Electronic Data Sheet) formats***

The IEEE 1451.2 standard was the first of the 1451 family of standards to be approved by the IEEE and provides a standardized digital interface and communication protocol that directly addresses the problem of interfacing multiple connection schemes with different buses and microprocessors [RNJ00, WOODS97, CAN99].

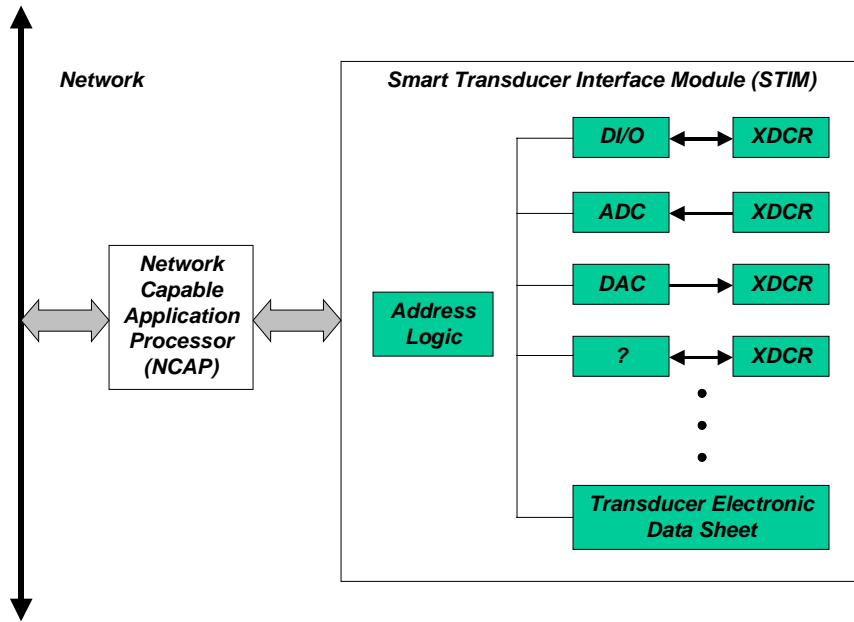
The specification defines the Smart Transducer Interface Module (STIM), Transducer Electronic Data Sheet (TEDS) and its data format, and the communication on the Transducer Independent Interface (TII) through a standard digital interface and communication protocols involved.

The STIM may consist of up to 255 transducers, along with signal conditioning and conversion circuitry, the TEDS and the TII. The STIM can connect to six different types of transducers and uses some addressing logic to identify the transducers. The transducers in the STIM should be capable of describing themselves when interfaced to a network using key parameters.

The TEDS provides a standard description of the type, attributes, operation and calibration of the transducers and is usually stored in non-volatile memory. It allows transducer manufacturers to burn specific fields about their products such as date of manufacture, calibration specifics, version number, etc. It is retrieved during automatic system configuration or on request, to be transmitted to the NCAP. The TEDS thereby eliminates the need for manual entry of transducer parameters, allows easy maintenance, upgrades and replacements while allowing “plug-and-play” operation through reliable and accurate electronic data sheets. Though the amount of data held within the TEDS will vary depending on the STIM implementation, some mandatory data would have to be provided by the transducer manufacturer.

The TII provides a 10-wire digital interface for communication with the NCAP and is modeled for synchronous serial communication based on the Serial Peripheral Interface (SPI) protocol. It allows the host or the NCAP to obtain sensor readings or actuator actions as well as request TEDS data.

The 1451.2 standard also describes a correction engine that can either be included within the STIM or at a remote location to model the output data in the desired format, if necessary.



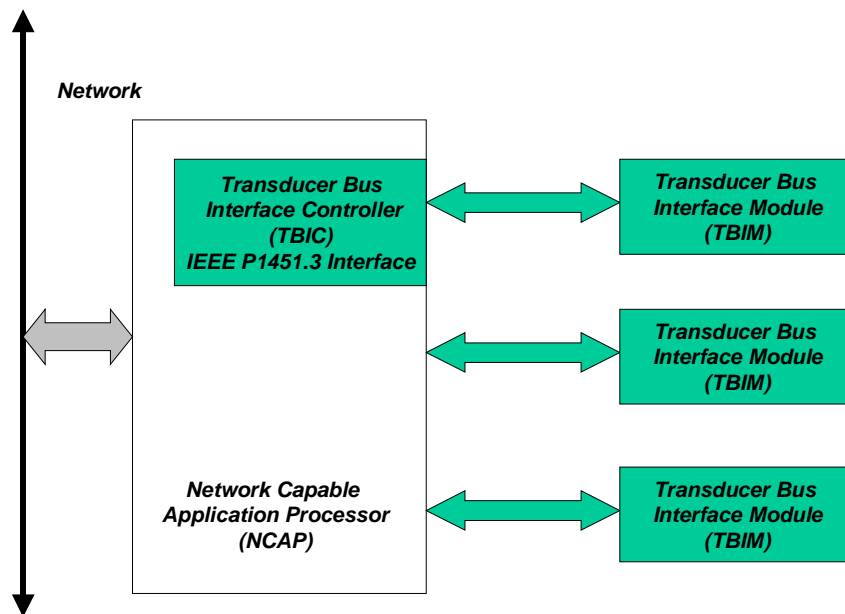
*Fig. 2.4 The IEEE 1451.2 Transducer-Microprocessor Interface [LEE\_K00]*

The communication between the NCAP and STIM on the TII is controlled by the NCAP. The NCAP may initiate a measurement or action by triggering the STIM, and the STIM responds with an acknowledgement once the requested function is completed. The STIM may interrupt the NCAP only when there is an exception, failure or fatal error at its end.

### ***2.4.3 The P1451.3 Standard for a Smart Transducer Interface for Sensors and Actuators – Multidrop Distributed System for Interfacing Smart Transducers***

The P1451.3 standard proposes to define a specification for a standard physical interface for connecting multiple physically separated transducers in a multidrop configuration [LEE\_K00, FRANK00].

Systems supporting a distributed topology, especially in harsh environments or in the presence of a large array of transducers, require extensive synchronization. The communications for such systems sometimes may require high bandwidth and nanosecond time correlation requirements.



*Fig. 2.5 The IEEE P1451.3 Multidrop Configuration [LEE\_K00]*

The 1451.3 standard proposes to use a single transmission line to supply power to transducers and provide communications between the bus controller and the Transducer Bus Interface Module (TBIM). The Bus Controller for the TBIMs as well as the network interface to support several buses will be contained within the NCAP. The interface will support TEDS and the TBIM will be capable of supporting several transducers. Some of the important issues to be resolved for this project are the protocols for addressing the transducer nodes, dealing with short power dropouts, etc. This minibus standard is designed to be small and inexpensive for easy integration to transducers.

#### ***2.4.4 The P1451.4 Standard for a Smart Transducer Interface for Sensors and Actuators – Mixed-Mode Transducer Interface***

The P1451.4 standard proposes to define the mixed mode interface specification for analog transducers with analog and digital operating modes. It is primarily intended for enabling legacy applications with analog sensors and wiring to communicate with a digital 1451 network with least cost and complexity [LEE\_K00, FRANK00].

The standard proposes that the communication of the digital TEDS data be shared with the analog signal from the transducer using a minimum set of wires. This may be achieved by allowing the TEDS data to be transmitted initially through digital mode operation followed by transmission of analog data during normal operation. The simultaneous transmission of analog and digital data may be accomplished through a multi-wire module also.

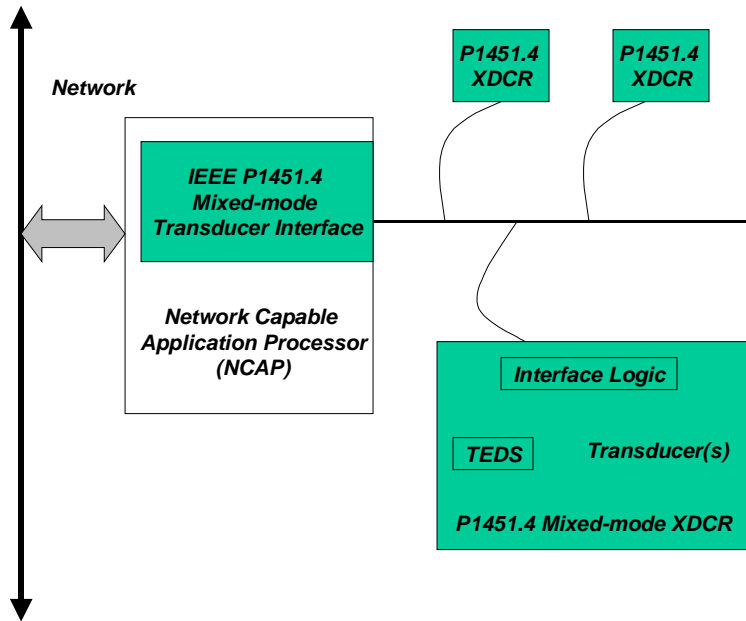


Fig. 2.6 The IEEE P1451.4 Mixed Mode Transducer Configuration [LEE\_K00]

To summarize, the family of IEEE 1451 standard interfaces attempts to allow network independent interfacing of transducers and offer the benefits of self-identification of transducers, self-configuration, electronic documentation, easy transducer upgrade and maintenance, increased data and system reliability and automatic or remote transducer calibration [CONWAY99].

### 2.5 The IEEE 1451.1 Network Communication Specification

The IEEE 1451.1 standard specifies an architecture that is applicable to distributed systems consisting of one or more NCAPs, communicating over a network [1451STD99]. The NCAPs interact with the physical world via attached transducers that may be

designed according to the IEEE 1451.2 standard. The architecture provides a network abstraction layer that specifies the software interfaces between application functions on an NCAP and a communication network in a manner independent of any specific network.

The two network communication models that have been specified for communication between objects in the IEEE 1451.1 system are a tightly coupled, point-to-point, client-server model for one-to-one communication and a loosely coupled, publish-subscribe model for one-to-many or many-to-many communications [1451STD99]. The network communication models define the syntax and semantics of the software interfaces between application objects and the communication network and are independent of any specific network or protocol. The translation of the IEEE 1451.1 communication operations for a specific network, specifically the marshaling and demarshaling routines for the translation of IEEE 1451.1 data to the underlying network format is performed by the *network infrastructure* and is expected to be provided by network software suppliers.

### ***2.5.1 The IEEE 1451.1 Client-Server Network Communication Model***

Client-server communication is implemented by instantiating a pair of communication port objects, viz. the Client Port Object and a Server Object [1451STD99]. The software architecture outlined by the IEEE 1451.1 specification instantiates these objects from abstract classes defined in the standard. Several other classes need to be instantiated to enable the network communication but the core functionality of client server

communication is embedded within these two classes. The Port objects are associated with a network-specific identifier called the Object Dispatch Address and this is usually referred to as the ServerDispatchAddress when it is used for a Server Object.

The two application-level operations that support client-server communication are the *Execute* operation performed by the client-side Client Port Object and the *Perform* operation performed by the server-side Server Object. The *Execute* and the *Perform* operation together provide the remote-object-operation-invocation messaging style used by RPC in distributed systems.

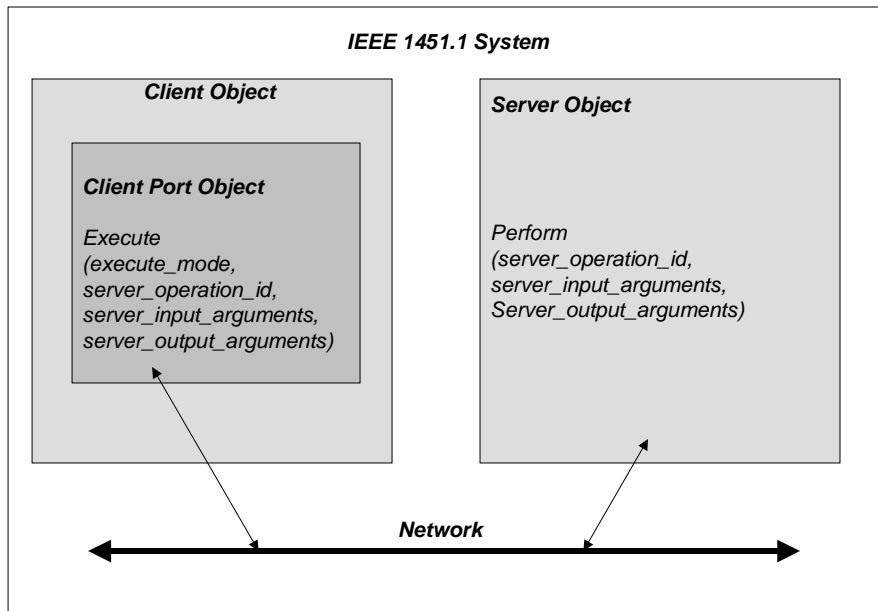


Fig. 2.7 The IEEE 1451.1 Client-Server Network Communication Model [1451STD99]

The operations that are defined within a class may be classified as local and network visible operations. Local operations of an object A, can be invoked only by objects within

the process space belonging to A while network visible operations may be invoked by the *network infrastructure* also.

The steps followed in such a client-server communication may be briefly listed as follows: [1451STD99]

1. The Client Port Object identifies the network address of the Server Object with which it wants to establish a client-server communication and associates this with its `ServerDispatchAddress` attribute. (This is usually performed in the system-commissioning phase for static networks, and dynamically for adhoc networks.)
2. The Client Object is provided with a reference to the Client Port Object during system initialization.
3. The Client Object invokes the `Execute` operation on the Client Port Object indicating the operation that it would remotely invoke on the Server Object through the `server_operation_id`. It provides the input arguments for the operation to be performed (`server_input_arguments`) and references to the output arguments that it hopes to receive from the Server Object (`server_output_arguments`).
4. The *network infrastructure* translates the `Execute` operation and the associated arguments to the underlying network protocol format. At the server side, the *network infrastructure* identifies the request as a `Perform` operation.
5. The `Perform` operation determines the operation that has to be carried out corresponding to the `server_operation_id` value and invokes it on the Server Object.

6. The results of the operation corresponding to the `server_operation_id` are bound to the `server_output_arguments` of the Perform operation.
7. The *network infrastructure* encapsulates the Perform operation result values into the underlying network format and sends it to the Client Port Object. The *network infrastructure* on the client-side decapsulates the data from the Server Object.
8. The data extracted from the Server Object is used to return the Execute operation to the Client Object with the `server_output_arguments` bound.

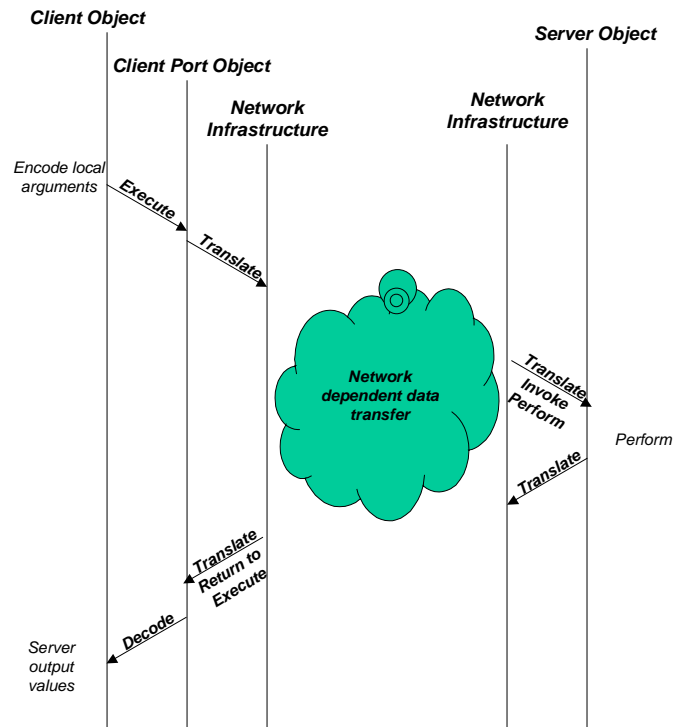


Fig. 2.8 The Client-Server Communication Sequence Diagram

### ***2.5.2 The IEEE 1451.1 Publish-Subscribe Network Communication Model***

The publish-subscribe communication model provides a loosely coupled mechanism for network communications between objects where the sending object, the Publisher object does not need to be aware of the receiving objects, the Subscriber objects [1451STD99, FRANK00]. This model is supported by the two operations, Publish on the Publisher Port Objects and the AddSubscriber with an associated call back operation on the Subscriber Port Objects. This model is specified for peer-to-peer communication and uses a combination of domains, keys, topics, and qualifiers to aid Subscriber Port Objects in determining publications of their interest.

Wireless smart transducer piconets are autonomous networks that typically communicate information to a remote controller through either wired or wireless media. Hence, it is recommended that a client-server model be adopted so that a central server node may possess the remote communication capability as well as greater processing capabilities to coordinate the activities of the client nodes in the piconet. The details of the Publish-Subscribe communication are not presented here since they are not relevant to our target network architecture.

### ***2.6 Wireless Solutions for Smart Transducer Networks***

Short-range wireless communication between transducer nodes in a network may be implemented through popular existing technologies like Bluetooth, HomeRF, IEEE

802.11x, IrDA, etc. [BRAY00, MULLER00, MILLER00]. IrDA is a short-range point-to-point protocol that operates in the Infrared spectrum using optical communication centered around 850 nm. Bluetooth and HomeRF are radio specifications for short-range point-to-multipoint voice and data transfer. IEEE 802.11x defines a larger set of protocols for both RF and IR solutions on a Wireless Local Area Network (WLAN). The major constraint for transducer nodes is power management and recent research efforts in the area of wireless transducer networks have focused on power-conserving collective communication methods for distributed environments. In what follows, the different options for communication between transducer nodes viz., IrDA, IEEE 802.11x, HomeRF and Bluetooth are discussed and the choice of Bluetooth as the best alternative for wireless smart transducer networks is explained.

IrDA specifies a cable replacement technology for point-to-point communication between adhoc data access points, with a maximum distance of 3 feet. This technology requires Line Of Sight (LOS) communication within a 30° narrow angle cone at rates between 9600 bps and 16Mbps. IrDA transceivers operate at a transmit power of 100mW and are identified by 32-bit physical addresses. Authentication and encryption protocols are not described at the link layer for IrDA but the short range, LOS requirement, and narrow angle of the infrared beam inherently provide security. The main applications for IrDA are simple data transfer and synchronization between static devices at a short range. An example application is the exchange of business cards between two handheld computers that are equipped with IrDA ports. The IrDA protocols are popular since they offer no interference with other electronics, inexpensive high bandwidth communication

and are supported by a wide range of hardware and software platforms due to their relatively early acceptance. Since several usage models suggested for Bluetooth overlap with those developed for IrDA, interoperability with the higher-level IrDA protocols has been a key issue for the Bluetooth developers.

Among the RF technologies, the IEEE 802.11x standard has been proposed as a wireless extension to the IEEE 802 LAN protocols, HomeRF targets standalone and wireless extensions to home networking of consumer electronics, and Bluetooth has been developed to target low-power, low-cost applications for portable devices. All three RF technologies are popular for medium to short-range wireless communications and the application usually determines if the technologies are directly competing with each other or are complementary to one another.

The IEEE 802.11x WLAN standard targets network communications between computers and peripherals that are migrating towards wireless technologies. The standard defines three physical layers – Frequency Hopping Spread Spectrum (FHSS) at a rate of 1 or 2 Mbps and Direct Sequence Spread Spectrum (DSSS) at 1, 2, 5.5, and 11 Mbps, both in the 2.4 GHz ISM bands and Baseband Diffuse IR at 1 or 2 Mbps. Although DSSS offers a higher data rate, this comes at the cost of either higher transmit power requirements or smaller coverage and a lower resistance to interference. The transmit power required for an IEEE 802.11x transceiver may be up to 1 W and this would cover a range of 300 feet. The 802.11x Medium Access Control (MAC) layer provides a basic access mechanism with clear channel assessment, channel synchronization, and Carrier Sense Multiple

Access with Collision Avoidance (CSMA/CA) scheme. A 48-bit standard MAC addressing scheme is used that is compatible with other 802 MAC standards and a challenge-response authentication algorithm as well as a standard 40-bit encryption (with optional 128-bit encryption) is defined at the link layer.

HomeRF is an open industry specification for supporting wireless digital communication between PCs and consumer electronic devices anywhere in and around the home. It may be viewed as a relaxed version of IEEE 802.11x and the Digital Enhanced Cordless Telecommunications (DECT) standard for cordless telephony, for combining data and voice distribution in the home environment. HomeRF is based on the Shared Wireless Access Protocol (SWAP) that supports CSMA/CA for high-speed packet data service and TDMA service for isochronous interactive voice communication. It also operates in the 2.4GHz ISM Band using FHSS but hops at a lower frequency of 50 hops/second and is optimized for wireless communication at 1 or 2 Mbps depending on the modulation scheme. HomeRF devices can communicate over a distance of 150 feet and require a transmit power of 100mW. These devices use a 48-bit addressing scheme and the Blowfish encryption algorithm at the link level. In addition, the SWAP protocol is implemented on a PC card rather than embedded as a SoC. The FCC has recently relaxed restrictions in the 2.4 GHz ISM bands that may allow HomeRF devices to communicate at up to 10 Mbps.

Bluetooth is an open standard that has been developed recently to provide low-power, low-cost, wireless communications for portable devices. It uses FHSS at a hop rate of

1600 hops/sec in the 2.4GHz ISM band to support a data rate of 1 Mbps, over a distance of 30 feet. The transmission power required by Bluetooth is 1mW and is considerably lesser than the other technologies. Bluetooth devices also use a 48-bit addressing scheme and a 128-bit key for authentication and a variable length key, between 8 and 128 bits for encryption. Through the design of multiple power-saving modes of operation and a single chip implementation of RF and logic components, Bluetooth is able to achieve significant power saving compared to other technologies. Also, Bluetooth supports the concept of “pervasive computing” that enables devices to link up automatically when they are within the range of one another to form a Personal Area Network, without explicit user initiation. Recently, IEEE has adopted Bluetooth as its base 802.15 standard for personal area networks. Activities directed towards increasing the data rate to tens of Mbps are also in progress.

For a wireless smart transducer network, the key issues to be considered are the requirements of prolonged low-power operation of the transducer nodes without manual intervention for charging batteries and the ability to form a self-organizing network such that the individual nodes are capable of cooperative functions [POTTIE98]. For communication between transducers, it is preferable that a “light” communication protocol be implemented to minimize the communication overhead on the transducer nodes. An “embeddable communication solution” would be preferred since the transducer nodes would typically be SoC or single board components. Also, since the number of transducer nodes that form a smart transducer network is usually large, an inexpensive solution would obviously be preferred.

The LOS restriction and the limited range of operation imposed by IrDA are significant enough to eliminate its use in a networked transducer scenario though it lends itself to low-power, low-cost, high bandwidth and a global communication solution.

IEEE 802.11x is not a power conserving initiative, nor does it lend itself to embedded solutions. In addition, the IEEE 802.11 MAC protocol is simple for a LAN environment but complicated for a transducer network because of the mechanisms for collision avoidance and mobility management.

HomeRF provides a lighter protocol version of IEEE 802.11x for data but is more targeted at cordless telephony applications for home networking. This technology is neither power-sensitive nor embeddable. The HomeRF specification is still in the process of being standardized and is expected to be a more expensive solution than Bluetooth.

On the other hand, Bluetooth, with its four different power modes of operation, is an excellent solution for wirelessly connecting transducer nodes. Its design was specifically focused towards inexpensive solutions to enable the wireless operation of portable devices, such that the entire protocol stack may be realized on a single chip. Apart from the main advantages of a low-power, inexpensive and easily embeddable solution, Bluetooth promotes proximity networking which is a basic requirement for adhoc wireless transducer nodes. The hidden computing profile promoted by the Bluetooth SIG allows the usage of this technology in scenarios that allow automatic connection

establishment, configuration and synchronization of Bluetooth-enabled devices, with no human interference. This feature allows the easy incorporation of self-organizing capabilities in wireless transducer networks.

The main argument against supporting Bluetooth technology till date has been the inertia of the companies that are part of the Bluetooth Special Interest Group (SIG) to be able to market Bluetooth products at the originally estimated price of \$5 a piece and the increasing complexity in support that is being gradually added to the original specification. However, history has shown that wireless data communication devices are slow to catch on but show rapid growth once the consumer is familiar with the product as seen in the case of IEEE 802.11x and IrDA products. It is expected that the capabilities that Bluetooth possesses would propel its popularity in the market, similar to the success story of IEEE 802.11x, albeit with time.

### ***2.7 The Bluetooth Wireless Specification***

Bluetooth is an open technology specification governed by the Bluetooth SIG to enable short-range wireless voice and data communications [BRAY00, MULLER00, MILLER00, BTH\_SPEC01]. The objective behind this specification is to enable the replacement of networking models of small span that were typically connected by cable, with a short range wireless communication technology. The Bluetooth SIG was formed by the industrial leaders in mobile computing and telephony and presently boasts a membership of over 2100 companies.

In a nutshell, Bluetooth provides a global, low-power, low-cost, small-sized, embedded wireless solution that enables the communication of existing portable devices and networks. It promotes an unconscious connectivity model based on proximity networking, which allows automatic communication and configuration between Bluetooth-enabled devices that can be deployed worldwide.

### ***2.7.1 Bluetooth System Architecture***

Bluetooth wireless communication operates in the unlicensed 2.4 GHz ISM band with support for both voice and data communications at a maximum gross data rate of 1 Mbps, over a range of 10-100m.

The inherent problems of interference and fading over the unlicensed spectrum are overcome through a low-cost, simple FHSS radio solution. In most countries, the 2.400.0 – 2.483.5 GHz spectrum is divided into 79 channels of 1 MHz width over which the FHSS may be implemented. With a high hop rate of 1600 hops/sec, RF interference is drastically reduced with an added advantage of some degree of security through unique hopping patterns.

The synchronous voice channels are provided over Synchronous Connection Oriented (SCO) links using circuit switching with a slot reservation at finite intervals. The

asynchronous data channels on Asynchronous Connection-Less (ACL) links are provided using packet switching with a polling access scheme.

The asymmetric data rate for a Bluetooth node is 721 Kbps (with 57 Kbps in the opposite direction) while symmetric data rate is around 432 Kbps over a single channel, with the designs permitting graceful degradation of both voice and data in busy radio environments.

The design emphasis for Bluetooth has been on high performance, inexpensive, low-power modules with a required and nominal range of 10m and 0dBm output power. The Bluetooth chipset may be augmented with an external power amplifier to extend the range to 100m, if the output power is increased to +20dBm.

The higher layer transport and application protocols in the Bluetooth architecture are borrowed from similar domains of wireless networked communication enabling the interoperability and reuse of existing standards. For example, the Bluetooth solution includes an IrDA interoperability layer that uses some of the protocols such as Infrared Object Exchange (IrOBEX) and Infrared Mobile Communications (IrMC), defined by the IrDA for exchanging and synchronizing data.

A Bluetooth network is established with peer devices that possess identical hardware and software interfaces, distinguished by a 48-bit address. However, when a wireless connection is initiated between two devices, the devices are temporarily assigned master

and slave roles with the initiating device functioning as master. The master device governs the FHSS communications between the devices by providing the hop sequence and phase based on its address and native clock. A maximum of seven active devices may participate in the wireless communication with the master, forming a network called a piconet. The piconet is hence characterized by a frequency-hopping channel defined by the master that serves to synchronize the slaves. The master-slave relationship is maintained only for the duration of the existence of the piconet and is restricted to the baseband level, and the devices function as peers in the higher protocol layers.

The low-power requirements of Bluetooth devices is possible through the existence of two baseband states and four baseband modes that have different energy requirements. A Bluetooth device may be in a connected or standby state depending on whether it is powered. The connected state of the device allows it to function in one of four modes, active, sniff, hold, or parked depending on its level of responsiveness and power consumption. An active slave provides the fastest response time and consumes the most power since it listens to all packets that it receives from the master device. A slave in sniff mode becomes active at certain regular intervals thereby providing a response time and consuming power depending on the length of the sniff interval. In hold mode, the master and slave decide on a hold interval during which period the slave ceases to be responsive to the packets from the master, and may be less responsive than a slave in sniff mode. However, power savings in the hold mode would typically depend on the hold time duration and whether the slave communicates on other links during the hold period.

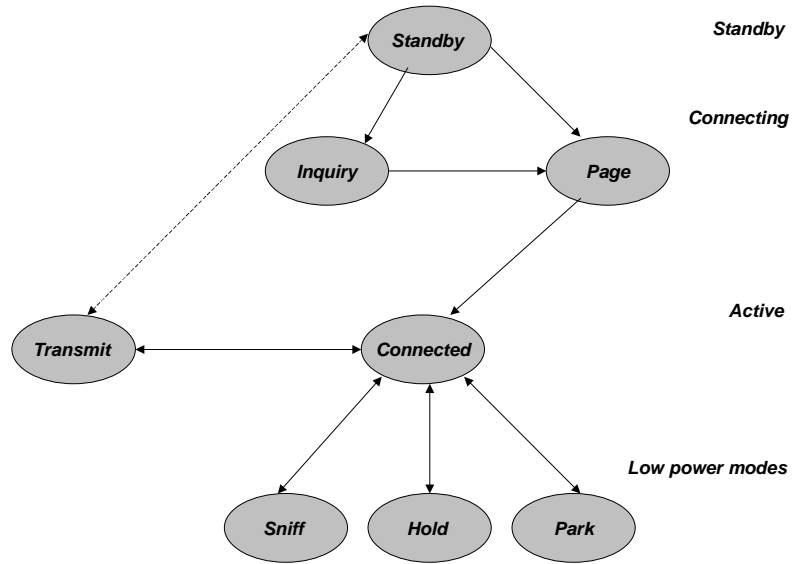


Fig. 2.9 The Bluetooth Connection State Diagram [HAART00]

Adaptive transmission power with a Received Signal Strength Indicator (RSSI) is another feature that allows power saving and minimizes radio interference. This allows a master to estimate the required transmission power setting for each slave depending on the slave's proximity.

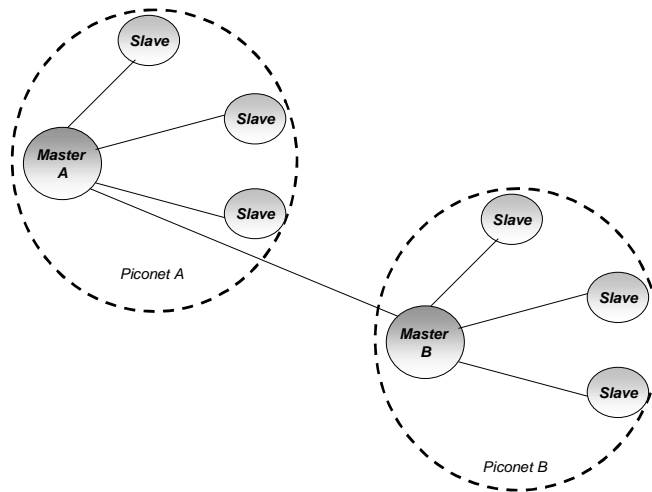


Fig. 2.10 Bluetooth Piconets and Scatternet

Spatial and temporal overlaps of piconets is possible since each piconet is characterized by the hopping sequence specified by the piconet's master and the same device cannot be the master of more than one piconet (however, the master may participate as a slave in other piconets). Hence, a scalable network of Bluetooth devices may be established through overlapping piconets and these are referred to as scatternets.

The link level protocol supports both authentication and privacy, allowing users to develop a domain of trust between their personal devices. Connections may require a one-way, two-way or no authentication, based on the challenge response algorithm. The privacy of the connection may be supported by a stream cipher that is suitable for silicon implementation with key lengths up to 128 bits. The authentication and privacy support defined in the specification are appropriate for the short range and applications may implement software security mechanisms if more security is required.

### ***2.7.2 Bluetooth Protocol Specification Overview***

The Bluetooth Open Specification developed by the Bluetooth SIG defines the hardware, software, and interoperability requirements for short-range wireless connectivity and ad hoc networking. The specification is divided into two volumes, the first dealing with the core protocols presented in a bottom-up fashion and the second dealing with the profiles and usage models for the technology [BRAY00, MULLER00, MILLER00, BTH\_SPEC01].

The Bluetooth protocol stack may be classified into three groups, the Transport Protocols, the Middleware Protocols, and the Application Protocols. The Transport Protocol Group corresponds to the physical and link layer protocols of the OSI Model and is responsible for creating, configuring, and managing both physical and logical links between Bluetooth enabled devices. The Middleware Protocol Group provides the actual transport protocols and comprises both protocols developed by the SIG for Bluetooth specifically as well as third party and industry standard protocols. Legacy applications such as a web browser application as well as Bluetooth specific software applications are included in the Application Protocol Group.

The overview of the Bluetooth Protocol Stack presented below follows a bottom-up approach with a focus on the functional requirements of each layer of the stack.

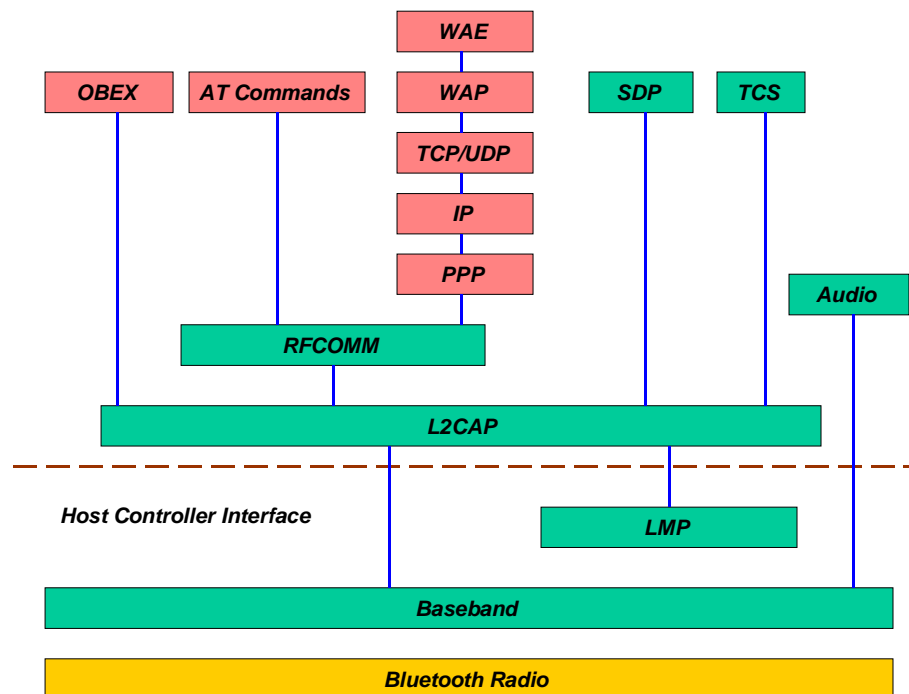


Fig. 2.11 The Bluetooth Protocol Stack [MET98]

The Bluetooth Radio layer details the design specifications for Bluetooth transceivers and does not introduce any new Bluetooth-specific protocols. It specifies a pair of logical interfaces for carrying data and control information but the interfaces have not been standardized to allow chip designers to determine the most cost effective way of power conservation. The main responsibilities of the Bluetooth Radio Layer are to generate and modulate the carrier signal as well as determine and maintain the power level for transmission.

The Baseband Layer defines the master and slave roles of the devices and performs the piconet and device connection control functions. The native clock in each Bluetooth device provides the synchronization when the device functions as a master and uses its unique address to provide the hop sequence to the piconet members. The baseband layer is also responsible for distinguishing between the link types depending on whether the connection is synchronous and also controls the media access through polling, packet processing and monitoring packet types. The most important function of the baseband layer is however maintaining low-power operation through the use of the connected and standby states with a requirement for devices in standby to pass through an inquiry and a page state before being included in the piconet.

The Link Manager Protocol (LMP) performs the main functions of security management, power management, and Quality of Service (QoS) management apart from some transmission scheduling duties. Though the security algorithms are defined in the baseband layer, the actual challenge response transaction for device authentication is

performed using link keys. Also, the link may be encrypted according to the one bit stream cipher implementation in the specification with a maximum key size of 128 bits. The Link Manager also supports power control by negotiating the mode of operation depending on the amount of participation required by the slave in the piconet and the proximity of the slave to the master device. QoS is an issue to be considered for voice traffic and the Link Manager manages appropriately allocates bandwidth for high priority periodic transmissions.

The Logical Link Control and Adaptation Protocol (L2CAP) layer in the Bluetooth protocol stack provides functional abstraction of the lower layer protocols to the host device. It concerns itself with only asynchronous packet transmissions and supports protocol multiplexing thereby enabling multiple protocols and applications to share the air interface. It also enables the segmentation of large packets used by higher layers to smaller packets for transmission and performs the corresponding packet reassembly function at the receiver. The L2CAP layer on peer devices also negotiates the acceptable grade of service for maintaining QoS.

Usually, the Radio, Baseband, and Link Manager Layers may be packaged together on a hardware module that can be attached to a host device and the higher layer protocols are implemented in software on the host. To enable the interoperability of the modules from different manufacturers, a common interface is required that allows the higher layer protocols to access the hardware components through a communication protocol. The standardized interface between a host controller on the hardware module and the host

together with the corresponding communication protocol between them is collectively referred to as the Host Controller Interface.

The Middleware Protocol Group includes two protocols developed for Bluetooth specifically, namely the RFCOMM and the Service Discovery Protocol (SDP) protocols. The other protocols that are supported by this group are the IrDA interoperability protocols, standard networking protocols like PPP, TCP/IP, WAP etc., and the Telephony Control protocol Specification (TCS).

RFCOMM is a serial port emulator to applications and facilitates the easy migration of applications modeled for cabled serial communications to the realm of wireless serial communications. Based on the European Telecommunications Standards Institute's Technical Standard (ETSI TS) 07.10, RFCOMM enables multiplexing of serial communications over a single serial link. The requirements for Bluetooth communication that are supported by RFCOMM include RS-232 signal compatibility, remote status and configuration with the aid of the SDP, and support for both an internal and external serial port. RFCOMM performs the important function of providing support to legacy serial applications and to a certain extent, networking applications due to the absence of a detailed networking profile by the SIG in the present Bluetooth specification.

The SDP defines a standard method for Bluetooth devices to discover and learn about the services offered by other devices in the vicinity. It enables the adhoc piconet the ability to discover and self configure thereby augmenting the end user value in dynamic networks.

Services are identified by Universally Unique Identifiers (UUIDs) that are assigned by service providers in a manner such that they are not duplicated. SDP maintains a database of service records that contain the information necessary for a client to access a service. SDP is used by first initiating an L2CAP channel between the client and the server to obtain SDP information from the server, and then setting up a separate connection to use the service. SDP provides the protocol to drive the discovery process but does not define the applications that may use SDP or an interface to these applications.

The TCS defines the control signaling required to establish telephone connections, for both data and voice, across a Bluetooth link. TCS connections are established on L2CAP links and multiple calls could be handled through multiple instances of TCS, each with its own L2CAP identifier. Telephony control for Bluetooth may be categorized into four modules as Call Control, Group Management, Connectionless TCS, and Protocol Discrimination. The Call Control module manages the connection and disconnection signaling associated with telephony. The Group Management module follows the concept of a Wireless User Group (WUG) and refers to the ability of a group of devices to share telephony services, establish quick connections amongst members and manage group membership. The Connectionless TCS module refers to the ability of devices to exchange call-signaling information without actually placing a call or establishing a TCS call connection, typically audio control and company related information. The Protocol Discrimination module handles direct messaging between L2CAP and the other three modules in TCS. It is to be noted that TCS does not provide features such as call forwarding, three-way calling, etc.

Audio is not a separate layer of the protocol stack per se but rather a specific packet format that may be transmitted directly over the Baseband layer on SCO channels. The two types of encoding schemes specified for Bluetooth audio are Pulse Coded Modulation (PCM) for general audio and a more error-tolerant Continuous Variable Slope Delta (CVSD) Modulation for voice conversations.

The protocols that are defined on RFCOMM are existing protocols that have been adopted by Bluetooth to promote interoperability and reuse of applications. The Point-to-Point Protocol (PPP) is a link layer protocol that defines the transmission of IP datagrams on serial point-to-point links and is popular for dialup Internet connections. Internet Protocol (IP) is a network layer protocol that delivers datagrams between different networks through routers. Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are transport layer protocols commonly used with IP for connection-oriented and connectionless communications, respectively. Wireless Application Protocol (WAP) is an open specification for transmitting and displaying Internet content on small wireless devices such as handheld computers and cellular phones. WAP applications are built within the Wireless Application Environment (WAE) that defines a web content delivery model with gateway functions. The AT Commands are modem control commands that provide an alternative method for establishing call control and are used by legacy applications that are yet to migrate to TCS. OBEX is a session layer protocol initially proposed by IrDA to support the exchange and synchronization of objects in a simple and spontaneous manner.

The client-server model outlined by the IEEE 1451 specification follows a session-level communication between clients and servers wherein client nodes “Execute” the “Perform” operations on the server. A detailed study of the Bluetooth protocol stack suggests the use of a light version of the OBEX protocol for establishing the client-server communication. The other option for session-level communication through the use of TCP/IP’s session protocol is not attractive because of the communication overhead due to algorithms such as the three-way handshake, congestion control, flow control, etc., that are implemented by this protocol. Also, the use of OBEX as a session protocol enables the adoption of a variety of transport protocols such as RFCOMM for serial communication, TCP/IP for internetwork communication, or even an infrared physical layer communication using IrDA. These arguments encourage the interfacing of the *network infrastructure* to the OBEX layer of the Bluetooth protocol stack and the following section discusses OBEX in more detail.

## ***2.8 OBEX with Bluetooth***

The Middleware Protocol Group uses the IrDA interoperability protocols at the session layer for enabling the synchronization and object exchange functionalities required by Bluetooth. The material for this section has been mainly obtained from [BRAY00, MULLER00, MILLER00, BTH\_SPEC01, OBX\_SPEC99].

The motivating factors for reusing the IrDA interoperability protocols in the Bluetooth protocol stack are the similarity in the applications that run on Bluetooth and IrDA as

well as the Bluetooth SIG's tendency to reuse existing protocols where appropriate. These protocols satisfy the requirements of the Generic Object Exchange Profile defined in the Bluetooth specification for transferring, exchanging, and synchronizing data.

The IrOBEX (used interchangeably with OBEX) provides a light version of object exchange services similar to the Hyper Text Transfer Protocol used on the World Wide Web through simple PUSH and PULL operations. The main advantages of OBEX are that it is a compact, flexible, and extensible session layer protocol with cross platform interoperability and easy mapping to the Internet data transfer protocol. Also, OBEX is not limited to quick transfer disconnect scenarios and allows connections to idle when transfers are extended over a period of time.

Though IrDA specifies both connection-oriented and connectionless sessions for OBEX, since only connection-oriented protocols are supported in the Bluetooth environment, the Bluetooth specification calls for use of only connection-oriented OBEX. OBEX transactions typically consist of a request PDU issued by a client followed by a response PDU issued by the server. The client and the server roles are interchangeable and the device initiating the transaction assumes the client role. The OBEX protocol functions using the main operations of CONNECT and DISCONNECT to initialize and terminate sessions and the GET and PUT operations to exchange data objects.

OBEX has been defined to operate over middleware protocols like RFCOMM and TCP/IP. It is to be noted that the specification discusses the use of OBEX over TCP/IP,

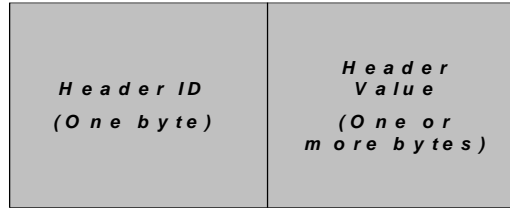
but it does not define how TCP/IP should operate natively over Bluetooth transports. As a result the interface for TCP/IP with the lower Bluetooth protocols is ambiguous and not standardized.

The OBEX model used by Bluetooth is identical to the IrOBEX model and its definition includes an *object model* to represent data objects that provides a standard format for transferring objects and a *session protocol* for transferring request and response messages between devices.

### ***2.8.1 OBEX Object Model***

The Object Model specifies that OBEX objects will be represented in the form of headers that can hold both data objects and information about the data. All OBEX headers are optional, the number and type of headers used will depend on the method in which the devices are configured, and the functionality they support.

OBEX headers comprise of two parts: a *Header ID* and a *Header Value*. The *Header ID* determines the nature of the information and the format of data that is contained in the header while the *Header Value* is the actual data contained within the header. The format of data is encoded in the two most significant bits of the Header ID while the type of the header information is encoded in the lower six bits as shown in Table 2.2.



*Fig. 2.12 OBEX Header Format*

### ***2.8.2 The OBEX Session Protocol***

The OBEX Session protocol describes the details of the communication between the client and server objects. As mentioned earlier, the OBEX Session Protocol can use either RFCOMM or TCP/IP as its transport layer. If OBEX packets are carried on RFCOMM frames, they can be directly encapsulated since the OBEX packets include a length field. The devices that run on RFCOMM should ensure that they possess both client as well as server capability and allocate separate channels for each OBEX server in the vicinity.

The OBEX client-server communication is based on a Request-Response messaging protocol wherein every Request Packet is required to be acknowledged by a Response Packet, called SUCCESS. Since OBEX Packets are limited in size to 64 Kbytes, Requests packets that contain messages greater than this size are split over several packets, with the most significant bit of the Opcode being used as a continuation flag. Each Request Packet, part of a large message is acknowledged by a CONTINUE Response packet except for the final Request packet that is acknowledged by a SUCCESS Response.

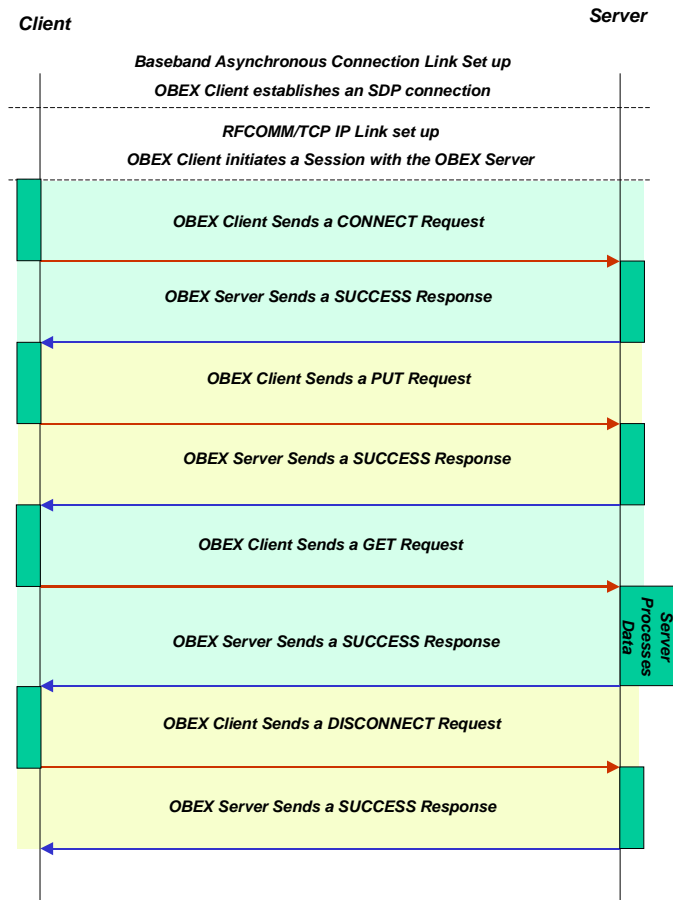


Fig. 2.13 The OBEX Client-Server Communication

<i>Opcode ID</i>	<i>Opcode Name</i>
0x80	Connect
0x81	Disconnect
0x02	Put
0x03	Get
0xFF	Abort
0XA0	Success
0x90	Continue

Table 2.1 Common OBEX Operations and Responses [OBX\_SPEC99]

<b>Header ID</b>	<b>Header Name</b>	<b>Description</b>
0XC0	Count	4 byte integer giving number of objects to be sent
0x01	Name	2 byte null-terminated Unicode text string denoting object name (a three byte header length denotes empty name header)
0x42	Type	Byte sequence, length prefixed with 2 byte unsigned integer, null terminated ASCII text denoting object type (default is binary)
0xC3	Length	4 byte value giving the length of the object, in bytes (use http content header for object sizes > 4 GB)
0x05	Description	2 byte null-terminated Unicode text string describing the object
0x46	Target	Byte sequence, length prefixed with 2 byte unsigned integer denoting the type of service the object is being sent to
0x47	HTTP	Byte sequence, length prefixed with 2 byte unsigned integer denoting the HTTP version 1.x header with the terminating CLRF
0x48	Body	Byte sequence, length prefixed with 2 byte unsigned integer for the actual object – the object may be sent as several chunks
0x49	End of Body	Byte sequence, length prefixed with 2 byte unsigned integer denoting the last part or final chunk of the body
0xCB	Connection ID	4 byte value used when multiplexing connections on OBEX to identify which connection is being used, should be the first header

*Table 2.2 Common OBEX Header IDs [OBX\_SPEC99]*

Since both Connection IDs and Targets identify the destination of headers and provide the same kind of information, only one of them is used at a time. Application parameters, authentication challenges and responses use an extra layer of structure to carry information for secure communication. Since each header has its own Header ID, the order in which the headers are sent is not important but the Connection ID, if used should

be sent as the first header so that the recipient can identify which OBEX connection the data must be demultiplexed to and sent.

An OBEX Session may be set up only if the OBEX Client possesses either the RFCOMM Channel ID to be used or the server's TCP port number. The OBEX Client may send SDP requests to the server if it requires the channel number value indicated in the OBEX Server's service record. The OBEX Session is initiated with a CONNECT request from the client that specifies the parameters for the session. The connection is established only if the OBEX Server accepts the connection with a SUCCESS Response. Objects may be transferred using PUT and GET packets that contain the data to be transferred within the BODY header. Large objects would be segmented over a set of PUT/GET requests, the final request distinguished by a special packet. OBEX Clients may use TARGET headers if the data packets have to be directed to a particular service or a SETPATH operation if the object being transferred has to be placed at a particular directory.

### ***2.8.3 Bluetooth Profiles for OBEX usage***

The Bluetooth profiles for OBEX describe an abstract profile called the General Object Exchange Profile (GOEP) and three specific profiles called the Object Push Profile (OPP), File transfer Profile (FP) and the Synchronization Profile (SP). We will review the GOEP and the OPP that we propose to use for the *network infrastructure*.

### 2.8.3.1 The Generic Object Exchange Profile

The GOEP provides generic interoperability for the application profiles using OBEX capabilities and defines the interoperability requirements for the lower link and physical layer protocols. It is an abstract profile that defines the support offered by Bluetooth devices for the object exchange usage models and is adapted for the Synchronization Profile, File Transfer Profile, and the Object Push Profile.

The GOEP makes a very clear distinction between client and server roles that may be assumed by a pair of devices. For each operation of pushing or pulling an object, the device that initiates the transfer assumes the client role while the device that offers the object exchange service assumes the server role and this distinction holds, even if there is an exchange of objects. It is to be noted that the client and server roles are applicable only at the OBEX level and are not concerned with the master and slave roles that a Bluetooth device may assume in a piconet.

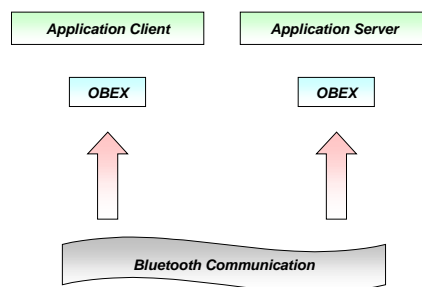


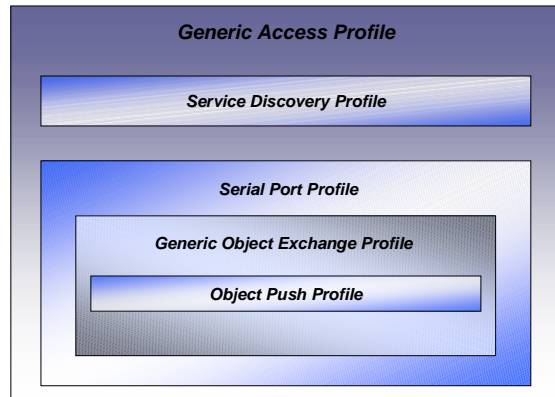
Fig. 2.14 The client-server distinction in OBEX

The GOEP requires the use of the Generic Access Profile (GAP) that defines the general procedures for discovering the identities, names, and basic capabilities of other Bluetooth

devices that are ready to accept connections and service requests. GAP categorizes the devices into discoverability, connectability, and pairing modes. A discoverable device executes inquiry scans regularly and responds to inquiries. Such a device may respond to all inquiries when in a general discoverable mode, inquiries that specify a Limited Inquiry Access Code (LIAC) in the limited discoverable mode, or to no inquiries when in a nondiscoverable mode. A connectable device is responsive and executes page scans regularly while a nonconnectable device does not respond to pages. It is to be noted that the discoverability and connectability modes might be set independent of each other. A pairable device allows itself to be authenticated by another device through a challenge-response transaction at the link layer and pairing is a key issue for the GOEP.

Object exchange between two Bluetooth devices is possible only if the devices form a trust relationship. To enable this, a bonding procedure is followed between the devices that is initiated by pairing, resulting in a common link key that may be used for subsequent authentication. General bonding requires further higher layer procedures to be executed while dedicated bonding requires only the pairing procedure at the link layer.

An OBEX initialization procedure may follow the bonding procedure before the first use of the server by the client and may require some security procedures such as encryption or application-level authentication depending on the application profile.



*Fig. 2.15 The relationships between the profiles*

The SP was the motivating factor behind using the OBEX protocols in the Bluetooth stack. This profile enables the device-to-device synchronization of Personal Information Management (PIM) data between two Bluetooth-enabled devices. This profile uses IrDA's IrMC client and server definitions to push and pull objects with OBEX. The FP is used to pull and push two specific objects, files and folders, bi-directionally and supports the creation, browsing, and deletion of these objects remotely. The OPP differs from the FP in that it enables the one-way transfer of objects through simple PUT and GET operations defined in OBEX.

### ***2.8.3.2 The Object Push Profile***

The Object Push Profile (OPP) falls under the abstract GOEP profile description and is the simplest of the object exchange profiles. It defines the requirements for one-way object transfers for both specific objects like vCard (business card objects), vNote,

vMessage, and vCalendar as well as generic object formats that are supported by the communicating devices.

The key difference between OPP and FP is that the former supports a half-duplex profile while the latter allows full-duplex transfer of objects. Also, unlike the FP, the OPP requires lesser user interaction and supports a usage model that allows unsolicited transfer of data similar to an email service. This form of pushing data into another Bluetooth device without a three-way handshake sequence promotes the hidden computing model that Bluetooth attempts to target.

The server and client roles defined by the GOEP are adapted as the PUSH SERVER (the device that provides an object exchange service) and PUSH CLIENT (device that pushes and pulls objects to and from the PUSH SERVER) in the OPP.

The OPP defines three functions with respect to a business card exchange model and these are classified as the Object Push, Business Card Pull, and Business Card Exchange. The Object Push operation is mandatory and refers to any object while the latter two are optional and specific to a business card object. Each of these operations needs to be initiated by the application using the functionality and is preceded by an inquiry operation to discover the objects in the neighborhood. The typical sequence of steps followed in an Object Push operation is as follows. To enable the PUSH SERVER to receive an object from the PUSH CLIENT the former must be in general or limited discoverable mode. This allows the PUSH CLIENT to select the PUSH SERVER it

would communicate with, and send the object with any authentication information, such as the Bluetooth PIN, if necessary. If the selected PUSH SERVER is incapable of providing the Object Push Service, a suitable error message is to be provided to the PUSH CLIENT. The PUSH SERVER may be designed to accept objects selectively also. The Business Card Pull function may be used to extract a business card object from the PUSH SERVER and the Business Card Exchange function is a combination of the Business Card Push and Business Card Pull operations.

The Object Push function supports any object format that is recognized by both the PUSH SERVER and the PUSH CLIENT and the formats supported by the PUSH SERVER should be identified in the Server Discovery Database. The PUSH SERVER should indicate to the PUSH CLIENT if the object format it received is not supported. The Bluetooth specification requires that the PUSH SERVER be able to receive multiple objects within an OBEX connection though the PUSH CLIENT can send one or more objects within a single OBEX connection. The objects may either be sent together using a single PUT operation for all objects or through separate PUT operations for each object.

## **Chapter 3**

### **DESIGN OF IEEE 1451.1 COMMUNICATION ON BLUETOOTH**

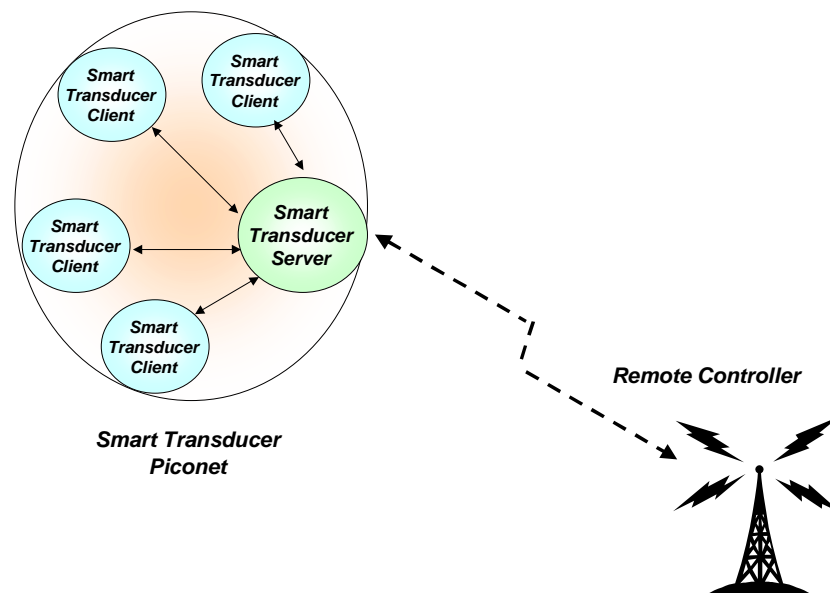
In the previous chapter, we discussed wireless networking technologies, and in particular Bluetooth as the most suitable option for networking wireless smart transducers. We also focused on the OBEX protocol as the candidate for session management between a smart transducer and a central node since OBEX is part of the upper layers of the Bluetooth protocol stack and forms a light networking protocol. The smart transducer itself presents a standard interface to any networking technology and the IEEE 1451.1 standard specifies this. In this chapter, we consider issues in interfacing the IEEE 1451.1 specified interface to OBEX and develop a design for this interface.

#### ***3.1 Wireless Smart Transducer Network Architecture***

The nature of communication between the IEEE 1451.1 Smart Transducer nodes on Bluetooth can be better understood by perceiving it within the framework of a network architecture. An example wireless smart transducer network architecture that can be adapted for a variety of applications is described.

Smart transducer nodes may be distributed in a system that performs either collective or disjoint monitoring and control functions. The “smartness” of the transducers may be

modeled to varying degrees and a smart transducer in general can possess a sensing element, some signal conditioning and converter blocks, a microcontroller or a digital signal processor block, memory, and a networked communication interface [SMART00]. Transducers can be capable of primitive sensing and signal conditioning functions as well as complex SoC functions such as a gas analysis system, mass spectrometer, “camera-on-a-chip”, etc [FRANK00]. Transducers may be either homogeneous or heterogeneous and are networked wirelessly in a client-server model. Smart “client” transducers are connected to a central server node that may possess some smart sensing functionality itself.



*Fig. 3.1 Wireless Smart Transducer Network Architecture*

The server node possesses additional intelligence, processing, and communication capability such as long-distance wired or wireless access, connectivity to the Internet, web based control, etc. The sensor technology and the wireless technology are designed

for low-power operation of the client and server nodes. The nature of the underlying wireless technology determines network characteristics such as the number of smart transducers in the network, range of communication, data rate, etc. For Bluetooth this translates to 8 active nodes and 255 parked nodes in a piconet, communicating over a maximum distance of 10 m between the client and server nodes, at a data rate of 1 Mbps. The transducers may be deployed manually, air-dropped or shell launched, depending on the application. Note that these transducers are self-configuring and thus our focus is on smart transducers that can form autonomous, self-organizing, reconfigurable, and ubiquitous networks operating on low power.

### ***3.2 The IEEE 1451.1 Network Interface***

The objective of interfacing IEEE 1451 smart transducer nodes on a Bluetooth network can be achieved by the design of a *network infrastructure* that is capable of translating the IEEE 1451 datums to the OBEX format required for Bluetooth communication. It is necessary to understand the nature of the client-server communication exposed by the smart transducer node to the *network infrastructure* as well as the Bluetooth data format that is to be transmitted wirelessly.

The smart transducer client node invokes an Execute operation on its Client Port when data has to be either transmitted or received from the server node [1451STD99]. The Execute operation possesses the arguments required to remotely invoke a Perform

operation on the server node. The Interface Definition Language (IDL) specification of the Execute operation is as follows:

```
IDL: ClientServerReturnCode Execute
      (IN UInteger8 execute_mode,
       IN UInteger16 server_operation_id,
       IN ArgumentArray server_input_arguments,
       OUT ArgumentArray server_output_arguments);
```

Where

*UInteger8* represents an 8-bit unsigned integer

*UInteger16* represents a 16-bit unsigned integer

*ArgumentArray* is an array of type *Argument*, which may be defined according to a specified enumeration

The *execute\_mode* argument specifies the nature of the client-side behavior during the invocation of an Execute operation. The predefined values that this argument can take are {EM\_RETURN\_VALUE (0), EM\_NO\_RETURN\_VALUE (1)}, and the rest of the 253 values have been reserved. By invoking the Execute operation with an EM\_RETURN\_VALUE, the client is blocked until a server response is obtained while the invocation with an EM\_NO\_RETURN\_VALUE allows the client to “send-and-forget” in a non-blocking mode. The typical application of the non-blocking mode is for data acquisition systems where the server interaction with the client is limited to accepting the data logged by the client. The sophisticated wireless smart transducer network architecture targeted operates on an adhoc Bluetooth master-slave piconet wherein application-level client server communication requires reliable connection-oriented data delivery, and consequently a blocking mode of operation.

The *server\_operation\_id* refers to the operation that needs to be invoked on the server and the *server\_input\_arguments* refer to the list of arguments that are being provided by the

client to invoke the specified operation on the server. The `server_output_arguments` gives a reference to the output arguments from the server operation.

The `ClientServerReturnCode` is of an underlying type of `UInteger32` (32-bit unsigned integer) and is expressed as a sequence of four fields, `portCode`, `performCode`, `operationMinorCode`, and `operationMajorCode`. The higher order 8 bits of the `ClientServerReturnCode` form the `portCode` and represent the return code of the client-side Port Object's `Execute` operation. The next 8 bits of the `ClientServerReturnCode` specify the `performCode` that represents the return code from the server-side Object's `Perform` operation. The next 8 bits form the `operationMinorCode` that specify the operation-specific return code of the operation invoked on the server-side Object. The lowest 8 bits of the `ClientServerReturnCode` form the `operationMajorCode` that represents system-level return status information.

The *network infrastructure* accepts the arguments of the `Execute` operation, marshals the arguments in the network specific OBEX format, and sends the message to the lower protocol layer for delivery across the network. The server-side *network infrastructure* on receiving a message for its server ID, identifies the message, demarshals the data to extract the operation ID and the encoded input arguments, and invokes the `Perform` operation using this data. The `Perform` operation invoked on the server Object is defined in IDL as follows [1451STD99]:

```
IDL: ClientServerReturnCode Perform
      (IN UInteger16 server_operation_id,
       IN ArgumentArray server_input_arguments,
       OUT ArgumentArray server_output_arguments);
```

The Perform operation returns to the *network infrastructure* on the server-side with appropriate values for the ClientServerReturnCode, and server\_output\_arguments and the *network infrastructure* marshals and sends these to the lower layer protocol for delivery across the network. For a blocking mode of operation, until the client-side operation times out, the *network infrastructure* waits to demarshal the data in the network message.

The IEEE 1451 Specification also describes a method for maintaining Block Cookies to store the identity of the server in each client node. The SDP layer of the Bluetooth protocol stack maintains a list of the services that are available and the location of these services within a piconet and the service records may be accessed to identify the target server directly. Consequently, no provision needs to be made for caching the server's Block Cookie in each client node.

The *network infrastructure* interfaces with the OBEX layer of the Bluetooth protocol stack and uses the SDP layer to identify the available services and their locations amongst the devices participating in the piconet. The *network infrastructure* sets up an OBEX session when the client needs to transmit or receive data to or from the server. For an adhoc piconet, servers may change with time or several servers may exist within the piconet and the SDP provides the means to identify the target server for any client communication. Consequently, the client object may possess the capability to identify and execute an operation on more than one server by specifying to the *network infrastructure* the target server ID.

### 3.3 The OBEX Session Protocol Interface

The OBEX Session protocol defines the sequence and format of data to be transmitted as byte arrays by the lower layer protocols. For this implementation, a five-state session has been designed for the client side communication.

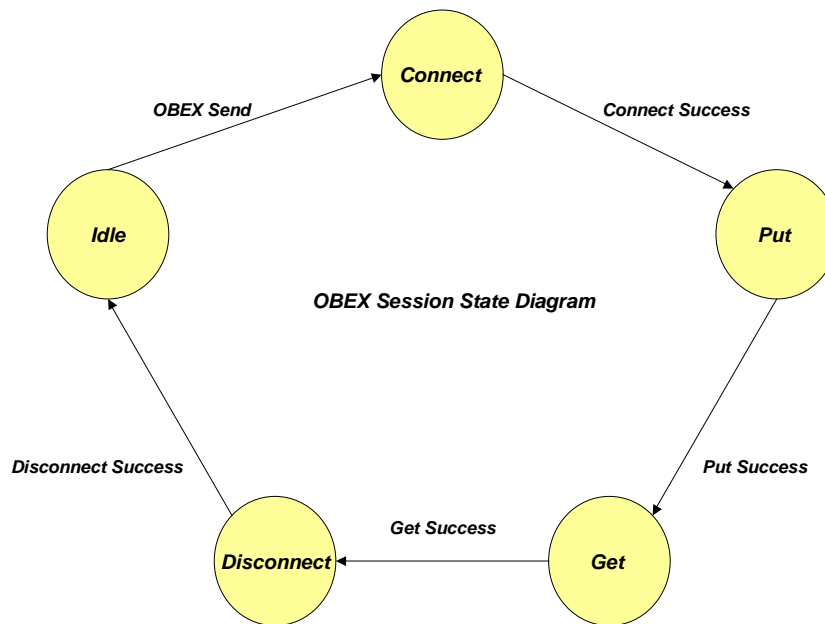


Fig. 3.2 OBEX Session for a Smart Client Transducer

A session is initiated when the Client Object provides the data to be transmitted to the server causing a state transition from the IDLE to the CONNECT state. A CONNECT packet with the target server ID is transmitted and a CONNECT SUCCESS response from the server causes a state transition to the PUT state. The client arguments for the server are encapsulated such that they may be transmitted along with the BODY Header in a PUT packet and a PUT SUCCESS response causes a state transition to the GET state. The client requests the server output arguments through a GET packet and the

server responds with the GET SUCCESS packet that encapsulates the server output. A state transition to the DISCONNECT state occurs on reception of the GET SUCCESS packet and a DISCONNECT packet is sent to the server. The DISCONNECT SUCCESS transitions the state to the IDLE state and the OBEX Session is terminated, and the Execute operation is unblocked. The state machine and the format of the packets sent in the OBEX Session are shown.

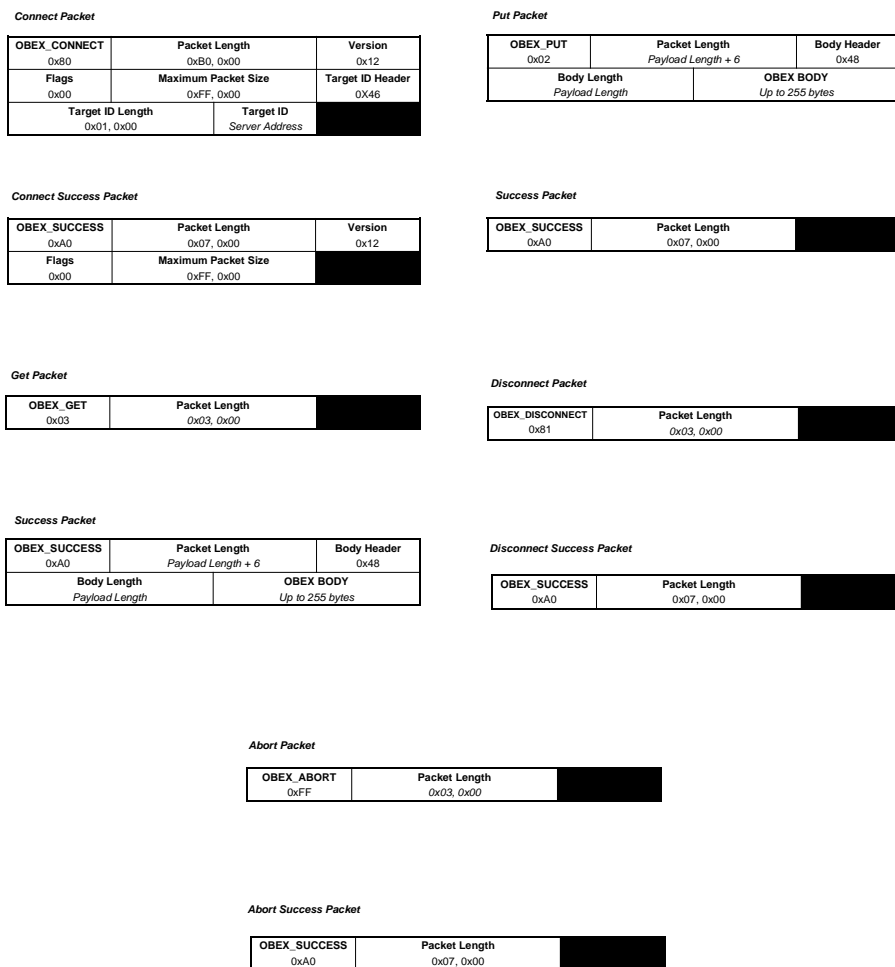


Fig. 3.3 Packets sent during OBEX Communication

## CHAPTER 4

# VHDL IMPLEMENTATION OF THE NETWORK INFRASTRUCTURE

As discussed in the previous chapter, we design a *network infrastructure* that is capable of translating the IEEE 1451 datums to the OBEX format required for Bluetooth communication to achieve our objective of networking smart transducers using Bluetooth technology. This *network infrastructure* is best implemented in hardware for a low-power solution. It is beyond the scope of this thesis to provide a complete implementation of such an interface that takes into account the complete specifications of IEEE 1451.1 and OBEX. However, based on the design of the previous chapter, it is possible to use VHDL to implement a proof-of-concept software model of this interface. In this chapter, we use the Mentor Graphics QuickHDL simulation tools to design, simulate, and test such a proof-of-concept VHDL implementation.

### ***4.1 Nature of the VHDL Implementation***

We present a proof-of-concept VHDL implementation of the proposed *network infrastructure* for the client-side below. Behavioral modeling has been adopted here to describe the functionality of the *network infrastructure* and provide a reference model for future designs. In order to synthesize this design the constructs would have to be changed

to Register Transfer Language (RTL) models. Also, this implementation represents only the client-side behavior and provides the server-side responses from the test bench simulation environment. The VHDL code for this software model is attached as an appendix to this thesis.

#### 4.2 Entity and architecture descriptions

The *network infrastructure* translates the IEEE 1451 Client smart transducer node requests to Bluetooth specific OBEX format and vice versa. The *IEEE 1451 Client-OBEX Interface* encapsulates the client-side arguments into an OBEX compliant format and sets up an OBEX session, sends the client arguments, receives the server response, and disconnects the session. The functionality of the *Client OBEX Interface* is divided into two parts, the first for the translation of the IEEE 1451 datums to OBEX compliant format and vice versa, and the second for the establishment of the OBEX session.

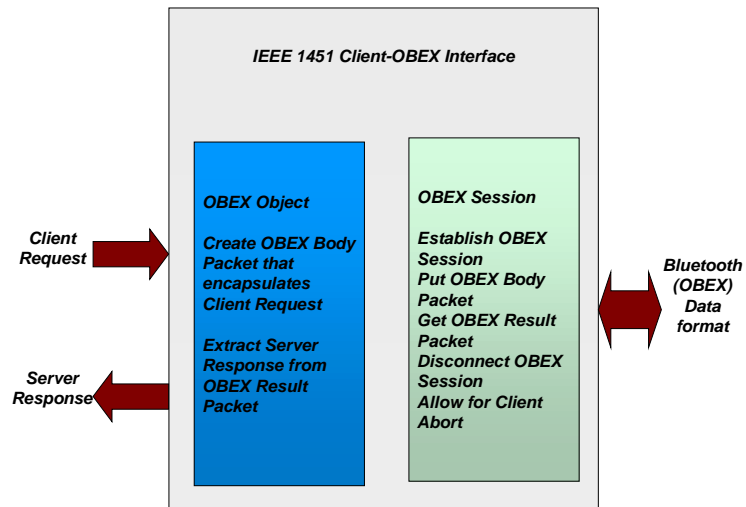


Fig. 4.1 The Client OBEX Interface and its components

The *Client OBEX Interface* instantiates the *OBEX Object* and *OBEX Session* components. On being triggered by a client request, the *Client OBEX Interface* provides the stimulus to the *OBEX Object* component to encapsulate the client arguments into a format that may be sent within an OBEX packet. It also provides the stimulus to the *OBEX Session* component to begin an OBEX session. The *OBEX Session* component indicates to the *Client OBEX Interface* when the server response is received in the GET SUCCESS packet. The *Client OBEX Interface* then triggers the *OBEX Object* component again to extract the server response and then provides the Client Port with the server output arguments and the return codes. The *Client OBEX Interface* also provides an internal clock for the *OBEX Session* component. The architecture associated with the *Client OBEX Interface* Entity is mixed and possesses both structural as well as behavioral constructs.

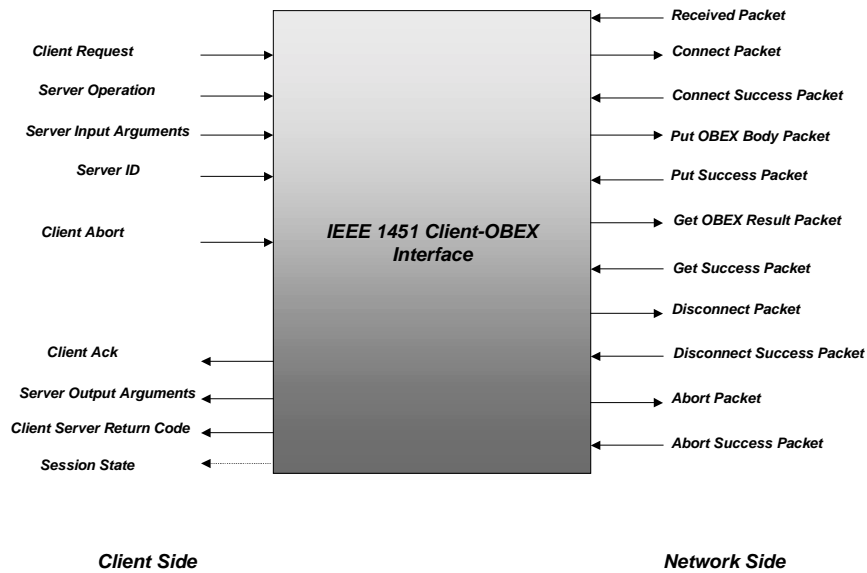


Fig. 4.2 The Client OBEX Interface Entity

The VHDL description of the Client OBEX Interface Entity is as follows:

```
ENTITY interface IS
PORT(
    client_request : IN BIT;
    client_abort   : IN BIT;
    serverid       : IN UInteger8;
    server_operation : IN INTEGER;
    server_input_args : IN integer_array;
    received_pkt   : IN BIT;
    connect_success_pkt : IN byte_array(0 TO 6);
    get_success_pkt : IN byte_array(0 TO payload_packet_length);
    success_pkt    : IN byte_array(0 TO 2);
    client_ack     : OUT BIT;
    server_output_args : OUT integer_array;
    portcode      : OUT UInteger8;
    performcode   : OUT UInteger8;
    majorcode     : OUT UInteger8;
    minorcode     : OUT UInteger8;
    connect_pkt   : OUT byte_array(0 TO 10);
    put_pkt       : OUT byte_array(0 TO payload_packet_length);
    get_pkt       : OUT byte_array(0 TO 2);
    disconnect_pkt : OUT byte_array(0 TO 2);
    abort_pkt     : OUT byte_array(0 TO 2);
    session_state : OUT state);
END interface;
```

The *OBEX Object* Entity when stimulated by the *Client OBEX Interface*, translates the client request arguments into a byte array that may be included within the BODY header of a PUT packet and extracts the server response from the GET SUCCESS packet. The Client Server Return Codes for the Execute operation are determined by the *OBEX Object* Entity. Error codes for timeout, abort and error in server response are examples of the information that is provided by the Client Server Return Codes. The architecture associated with the *OBEX Object* Entity is behavioral.

The VHDL description of the OBEX Object Entity is as follows:

```
ENTITY obex_object IS
PORT
    (encapsulate_object : IN BIT;
     server_operation   : IN INTEGER;
     server_input_args  : IN integer_array;
```

```

decapsulate_object : IN BIT;
obex_result_object : IN obex_result;
timeout_error : IN BIT;
obex_body_object : OUT obex_body;
server_output_args : OUT integer_array;
portcode : OUT UInteger8; -- error for client port operation
performcode : OUT UInteger8; -- error for server's perform
majorcode : OUT UInteger8; -- error for system level status
minorcode : OUT UInteger8); -- error for operation invoked on server
END obex_object;

```

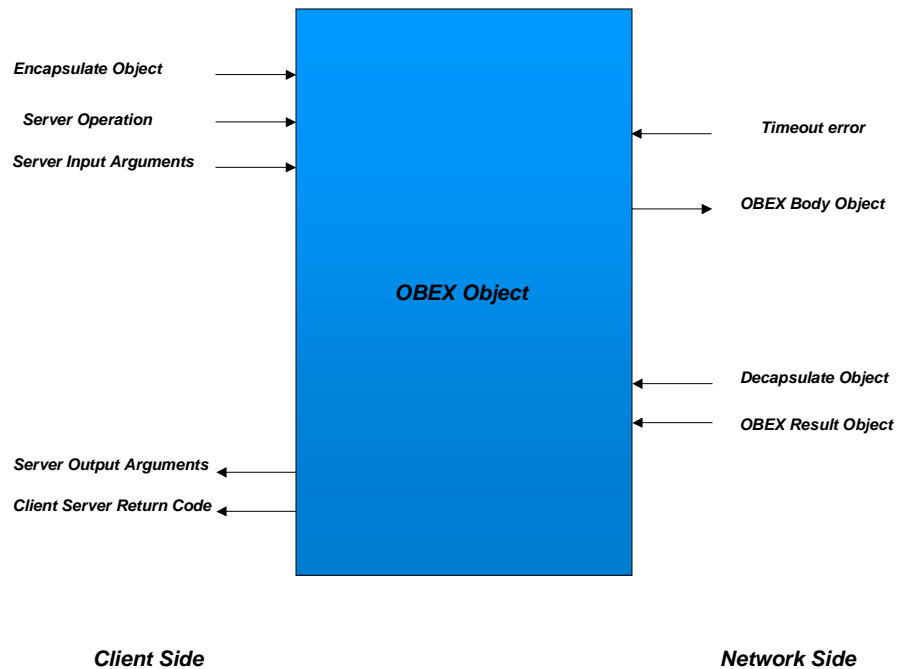


Fig. 4.3 The OBEX Object Entity

The *OBEX Session* Entity is modeled as a finite state machine with five distinct states. The *Client OBEX Interface* Entity stimulates this entity when a session needs to be established. The *OBEX Session* Entity cycles through the CONNECT, PUT, GET, DISCONNECT, IDLE states and outputs the associated packets as byte arrays for the lower layer protocol. The finite state machine may be reset by an Abort stimulus from the

Client Port, through the *Client OBEX Interface* Entity and progresses from one state to the next when it receives a SUCCESS response from the server. Also, if a server response is not received within a specified period, the OBEX session times out and aborts the session. This architecture is behavioral and extensively uses procedures for building the OBEX packets.

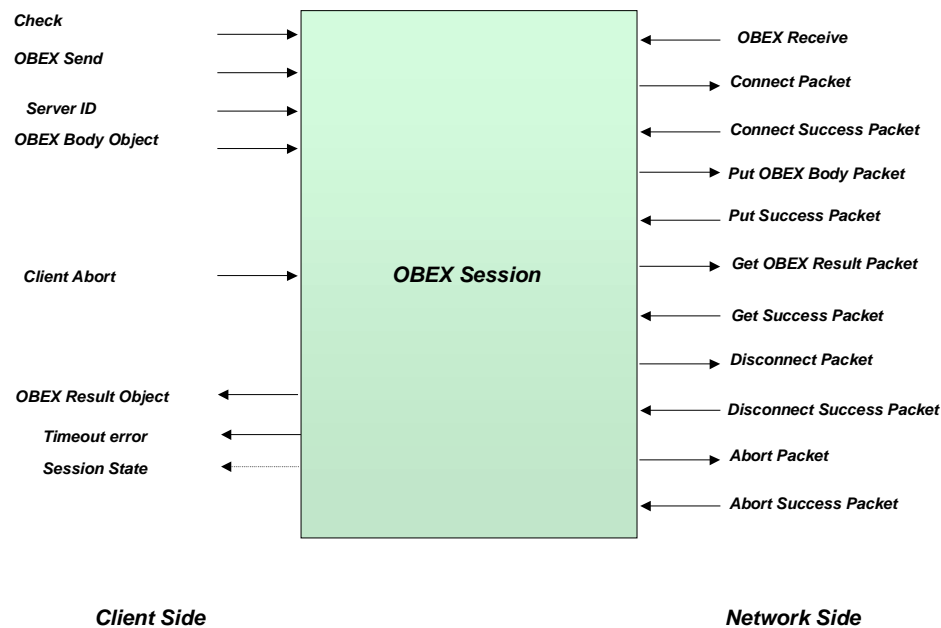


Fig. 4.4 The OBEX Session Entity

The VHDL description of the OBEX Session Entity is as follows:

```

ENTITY obex_session IS
  PORT(check : IN BIT;
        client_abort : IN BIT;
        server_id : IN UInteger8;
        obex_body_object : IN obex_body;
        obex_send : IN BIT;
        obex_receive : IN BIT;
        connect_success_pkt : IN byte_array(0 TO 6);
        get_success_pkt : IN byte_array(0 TO payload_packet_length);
        success_pkt : IN byte_array(0 TO 2);
        obex_result_object : OUT obex_result;
        session_state : OUT state;
  );
END ENTITY obex_session;

```

```

    connect_pkt : OUT byte_array(0 TO 10);
    put_pkt : OUT byte_array(0 TO payload_packet_length);
    get_pkt : OUT byte_array(0 TO 2);
    disconnect_pkt : OUT byte_array(0 TO 2);
    abort_pkt : OUT byte_array(0 TO 2);
    timeout_error : OUT BIT
  );
END obex_session;

```

The TYPE, CONSTANT, FUNCTION, PROCEDURE declarations and definitions for this implementation as well as the IEEE 1451 specific declarations are grouped together in a separate package.

### ***4.3 The Test Bench Design***

The test bench for the *Client OBEX Interface* design provides both the stimulus from the IEEE 1451 Smart Transducer client node as well as simulates the server side responses for OBEX communication and the Perform operation output. It verifies the establishment of a session for “Execute” operations and the transmission of the input arguments to the server correctly. It verifies the correct functioning of the abort operation in the middle of client communication and the proper assignment of the ClientServerReturnCode for the client. The graceful recovery of the OBEX Session when an expected server response is not received, based on a timeout value is also verified. The simulation outputs for some of the test bench cases follow with a brief description of the operation being formed.

The first simulation shows the expected correct functioning of the *Client OBEX Interface*. The Client Request signal is provided once the Server ID, Server Input Arguments and the Server Operation are available, by the test bench. The OBEX Session State transitions

from the listen\_state (IDLE) to the connect\_state and a CONNECT Packet is sent. The CONNECT SUCCESS server response stimulus from the test bench causes a state transition to the put\_state is acknowledged with a PUT Packet that holds the encapsulated server arguments. The server's SUCCESS response to the PUT Packet simulated by the test bench causes a state transition to the get\_state with a GET Packet being sent. The test bench responds with the server results encapsulated in a GET SUCCESS Packet and this is decoded by the OBEX Object into an OBEX Result Object. A DISCONNECT Packet is then sent to terminate the session in the disconnect\_state and the test bench simulates the server's SUCCESS response, causing the session to go back to the listen\_state.

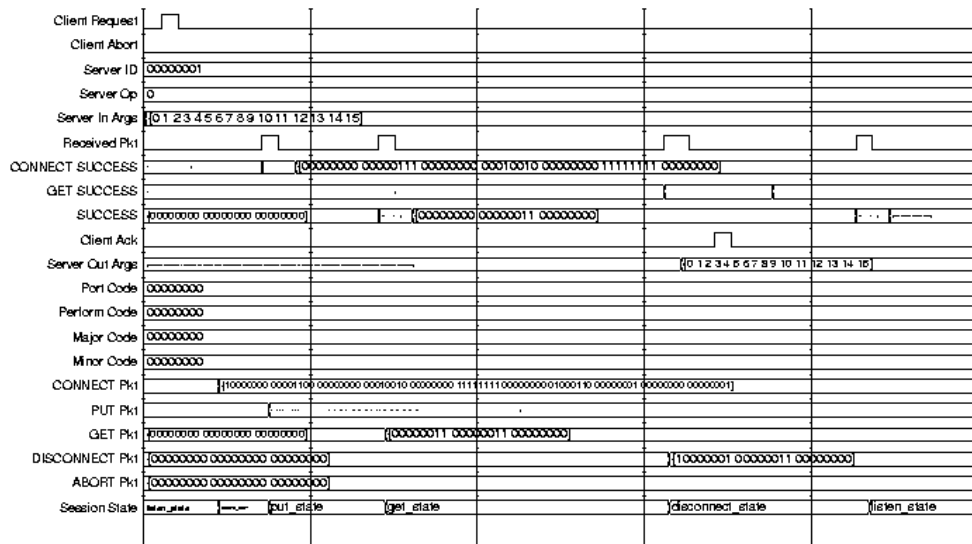


Fig. 4.5 First Simulation Output

The second simulation output shows a similar sequence with the variation of the server responding with an error in the Perform operation causing the appropriate error codes to be provided to the client.

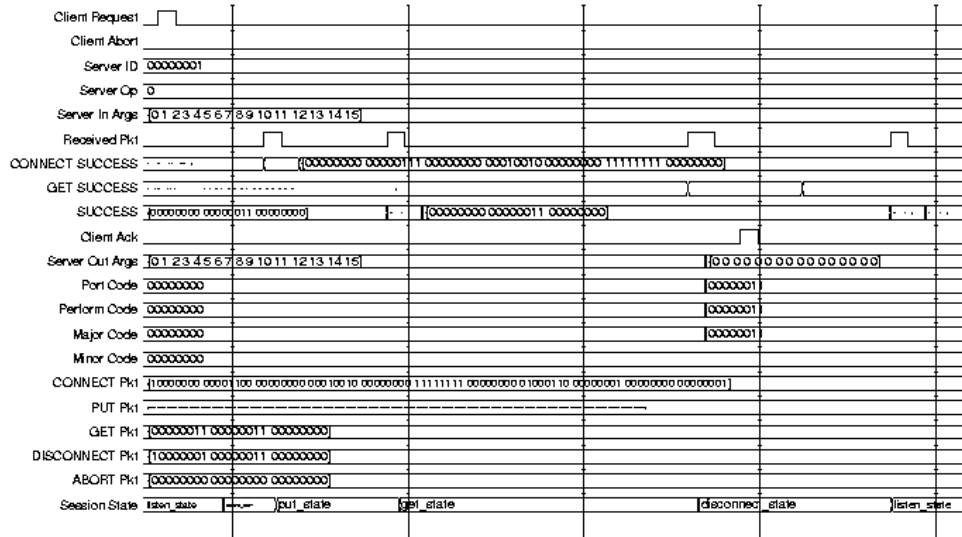


Fig. 4.6 Second simulation output

The third simulation output shows the operation of the OBEX session when a Client Abort signal is provided after the PUT SUCCESS packet is received from the server. The session gracefully exits and reverts to the listen\_state from the put\_state.

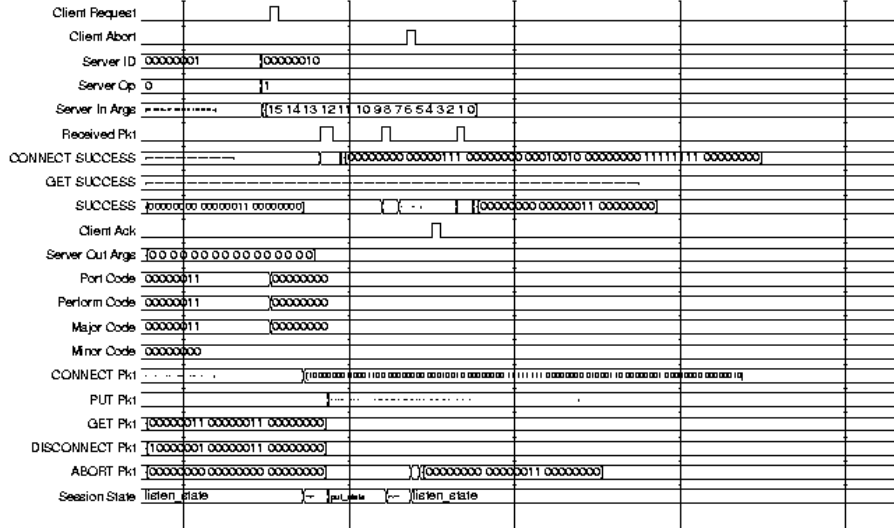


Fig. 4.7 Third simulation output

The fourth simulation output shows the session behavior when a timeout occurs. The client's CONNECT request to the server does not receive a response and the session times out from the connect\_state to the listen\_state.

The fifth simulation output also is an example of client-side session timeout when the response to a PUT packet is not received in the put\_state.

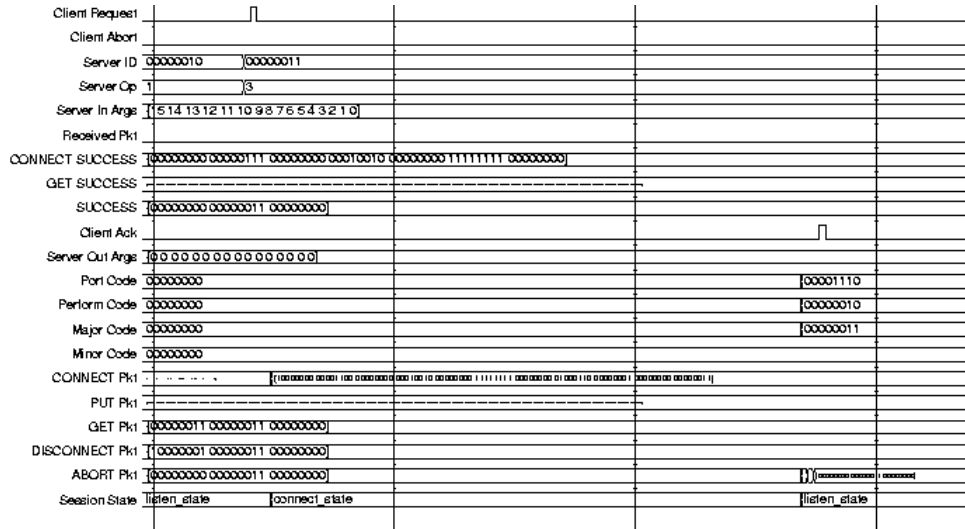


Fig. 4.8 Fourth Simulation Output

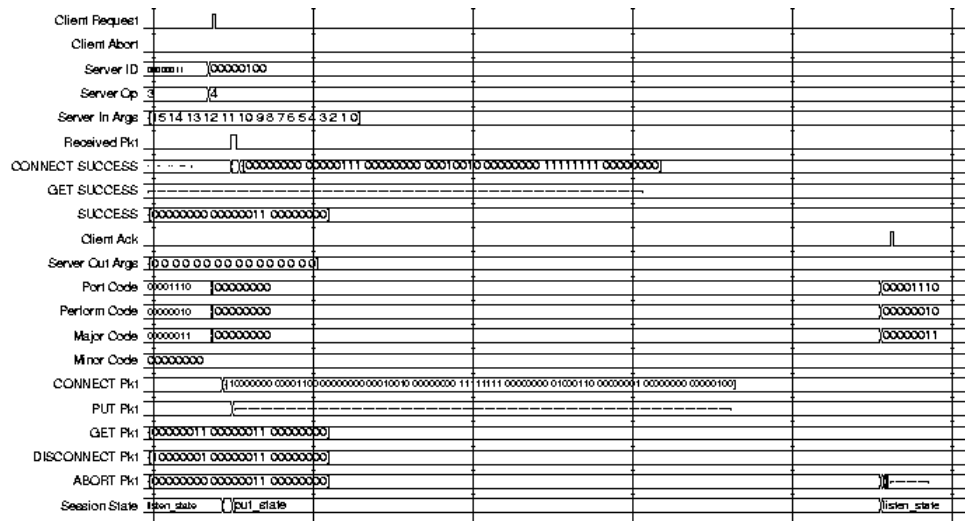


Fig. 4.9 Fifth Simulation Output

## **CHAPTER 5**

### **CONCLUSIONS AND FUTURE WORK**

#### ***5.1 Conclusions***

This thesis has proposed a software model for interfacing smart transducers on a Bluetooth network. This approach involved the definition and design of a *network infrastructure* for interfacing a smart transducer with a wireless networking technology. For this purpose, smart transducer networking and the recently standardized IEEE 1451 Specification for transducer interfaces were investigated. The use of an existing wireless technology to network smart transducers is suggested.

The characteristics of a variety of wireless networking alternatives were examined. In particular, the IrDA protocol, HomeRF, Bluetooth and the IEEE 802.11x standards were considered as potential candidates. Bluetooth was selected as the best alternative because of its features – low-cost, low-power consumption, simple protocols, and the ability to operate in reconfigurable ad hoc manner using service discovery.

The *network infrastructure* design involved the interfacing of the IEEE 1451.1 specification of a smart transducer interface with the OBEX session protocol that

operates on Bluetooth. A software model of the *network infrastructure* for client-server smart transducer communication on a Bluetooth network was designed and the client-side functionality was implemented using VHDL for a proof-of-concept solution. The design was incrementally implemented to verify the functionality of the software model and by simulating the client side requests and the server responses for the *network infrastructure*.

## ***5.2 Future Work***

As part of future work, this implementation may be extended to the server-side *network infrastructure* to provide a complete end-to-end Bluetooth solution for wireless smart transducer networks. Additional details and support can be overlaid on the existing implementation to enable specific functionality and processing for smart transducers if required, at the application level.

The behavioral VHDL implementation may be modified into more efficient and synthesizable code so that it can be directly used in an FPGA or ASIC that can be connected to Bluetooth hardware already available. Other software models based on JHDL [JHDL00] can also be examined for increased functionality.

This implementation considers using the OBEX session protocol for client-server communication. It is possible to either enhance this using TCP/IP or simplify it with a customized protocol for specific applications.

In the future, smart transducers are expected to possess the capability of hosting an operating system or a JVM that can support software applications and the VHDL design may be translated to a high-level description.

## LIST OF REFERENCES

- [1451STD99] IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Network Capable Application Processor (NCAP) Information Model, IEEE 1451.1- 1999 Standard, <http://standards.ieee.org/catalog/olis/im.html>, 1999
- [AGR00] Jon Agre and Loren Clare, An Integrated Architecture for Cooperative Sensing Networks, *IEEE Computer* pages 106 – 108, May 2000
- [BIB00] <http://eewww.eng.ohio-state.edu/~bibyk/ee582/win00/Aero582.pdf>
- [BRAY00] Jennifer Bray and Charles Sturman, *Bluetooth: Connect Without Cables*, Prentice Hall PTR/Sun Microsystems Press, December 2000
- [BTH\_SPEC01] *The Bluetooth Specification*, <http://www.bluetooth.com/developer/specification/specification.asp>, February 2001
- [CAN99] Tom Cantrell, Car 1451, Where Are You, *Circuit Cellar Ink*, <http://www.edtn.com/embapps/emba041.htm>, February 1999
- [CONWAY99] Paul Conway, et al., *IEEE 1451.2: An Interpretation and Example Implementation*, [http://www.ul.ie/~pei/PDF\\_files/fp17.pdf](http://www.ul.ie/~pei/PDF_files/fp17.pdf), 1999
- [DUNBAR00] Mike Dunbar, Where Wireless Communications and the Internet Meet, *Sensors Magazine*, <http://www.sensormag.com/articles/0900/89/main.shtml>
- [FRANK00] Randy Frank, *Understanding Smart Sensors*, Second Ed., Artech House, April 2000
- [HAART00] Jaap C. Haartsen, The Bluetooth Radio System, *IEEE Personal Communications*, pages 28-36, February 2000
- [INTEL00] Kris Fleming et al., *Architectural Overview of Intel's Bluetooth Software Stack*, [http://developer.intel.com/technology/itj/q22000/pdf/art\\_2.pdf](http://developer.intel.com/technology/itj/q22000/pdf/art_2.pdf), 2000

- [JHDL00] <http://www.jhdl.org>
- [LEE\_K00] Kang Lee, IEEE 1451: A standard in support of Smart Transducer Networking, *Proceedings of the 17th IEEE Instrumentation and Measurement Technology Conference*, pages 525 –528, 2000
- [LEE\_K96] Kang Lee, Rick Schneeman, A Standardized Approach for Transducer Interfacing: Implementing IEEE-P1451 Smart Transducer Interface Standards, *Proceedings of Sensor Conference*, October 22-24, 1996
- [MET98] Riku Mettala, *Bluetooth Protocol Architecture*, White Paper, <http://www.bluetooth.com/developer/whitepaper/whitepaper.asp>, 1999
- [MILLER00] Brent A. Miller, Chatschik Bisdikian, and Anders Edlund, *Bluetooth Revealed: The Insider's Guide to an Open Specification for Global Wireless Communications*, Prentice Hall, September 2000
- [MULLER00] Nathan J. Muller, *Bluetooth Demystified*, McGraw Hill, September 2000
- [NASA00] Ed Baroth, Integrated Vehicle Health Management at JPL, *Proceedings of the Space Transportation Day and Technology Workshops '00 organized by NASA*, <http://stday.msfc.nasa.gov/presentations/IVHM3.pdf>
- [OBX\_SPEC99] Counterpoint Systems Foundry, Inc and Microsoft Corporation, *IrDA Object Exchange Protocol*, <http://www.irda.org/standards/pubs/IrOBEX12.pdf>, March 1999
- [ORA00] Jyrki Oraskar, *Bluetooth versus WLAN 802.11x*, <http://www.hut.fi/~joraskur/BT2.pdf>
- [POTTIE98] Gregory J. Pottie, Wireless Sensor Networks, *Information Theory Workshop*, pages 139 – 140, June 1998
- [RNJ00] Robert N. Johnson and Stan P. Woods, Overview and Status Update for IEEE 1451.2: Transducer to Microprocessor Communications Protocols and Transducer Electronic Data Sheet (TEDS) Formats, *Proceedings of the Sensors Expo*, <http://www.telemonitor.com/doc/dot2.pdf>, May 2000

- [SINA00] Chavalit Srisathapornphat et al., Sensor Information Networking Architecture, *Proceedings of the International Workshops on Parallel Processing*, pages 23 – 30, August 2000
- [SMART00] Schavitz Sensors, *Smart Transducers for Industrial Applications*, <http://www.schavitz.com/news/smart.html>
- [SOC99] Henry Chang et al., *Surviving the SoC Revolution*, Kluwer Academic Publishers, July 1999
- [TRAVIS95] Bill Travis, Smart-Sensor Standard Will Ease Networking Woes, *EDN Magazine*, Cahners Publishing, <http://www.ednmag.com/ednmag/reg/1995/062295/13df1.htm>, June 22, 1995,
- [TRAVIS99] Bill Travis, Sensors Smarten Up, *EDN Magazine*, Cahners Publishing, <http://www.ednmag.com/reg/1999/030499/pdfs/05cs.pdf>, March 4, 1999
- [WARRIOR99] Jay Warrior, Smart Sensor Networks of the Future, *Sensors Magazine*, [http://www.sensorsmag.com/articles/0397/net\\_mar/main.shtml](http://www.sensorsmag.com/articles/0397/net_mar/main.shtml), March 1999
- [WINS00] Sandeep Vardhan et al., Wireless Information Network Sensors: Distributed In Situ for Mission and Flight Systems, *Proceedings of the Aerospace Conference*, pages 459 - 463, March 2000
- [WOODS97] Stan P. Woods, The IEEE-P1451.2 Smart Transducer Interface Module, *Proceedings of Sensors Expo*, [http://www.hprie.com/pdf-files/sen5\\_97.pdf](http://www.hprie.com/pdf-files/sen5_97.pdf), May 1997
- [WSEN01] Wayne W. Manges and Glenn O. Allgood, Wireless Sensors – Buyer Beware, *Sensors Magazine*, <http://www.sensorsmag.com/articles/0401/18/index.htm>