uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Emil Sadok

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Web Services for IEEE 1451.1 Sensor Network Controllers

TITRE DE LA THÈSE / TITLE OF THESIS

E. Petriu

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

R. Liscano

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

A. El Saddik

B. Esfandiari

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

WEB SERVICES FOR IEEE 1451.1 SENSOR NETWORK CONTROLLERS

By

Emil F. Sadok

B.A.Sc. Computer Engineering

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

in partial fulfillment of

the requirements for the degree of

Master of Applied Science in Electrical Engineering

Under the auspices of the

Ottawa-Carleton Institute for Electrical and Computer Engineering

University of Ottawa

Ottawa, Ontario, Canada

April 2006

Your file Votre référence
ISBN: 978-0-494-18464-6
Our file Notre référence
ISBN: 978-0-494-18464-6

I hereby declare that I am the sole author of the paper.

I authorize the University of Ottawa to lend the paper to other institutions or individuals for the purpose of scholarly research.

Emil F. Sadok

I authorize the University of Ottawa to reproduce the paper by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Emil F. Sadok

To my Parents

Abstract

The following thesis entails the development and a feasibility study for the integration of a Service Oriented Architecture platform applied to the IEEE 1451 Sensor interface standard. The research work leverages a service oriented software-development approach via the XML Web Services Integration Framework (XWIF) process for exposing a sensor controller's application services to remote clients via web service based interfaces. Further, the J2EE framework hosts a web services based middleware. The J2EE middleware provides *WSDL* service interfaces for exposing operations required for sensor interaction, and a web-services communication mechanism via the *SOAP* XML based messaging protocol.

The thesis extends the IEEE 1451.1 instrumentation standard's based platform, by introducing a software development model for web-services based sensor network applications. The prototype architecture of a web enabled 1451.1 NCAP controller connectable via the web-services middleware is investigated. Further, the research contribution presents the feasibility for introducing a service-oriented approach to distributed sensor monitoring via the developed web-services based NCAP platform connected to a real-time smart multi-sensor. Moreover, this thesis will provide the academic and instrumentation scientific community a performance study for integrating emerging web-service technologies within the instrumentation domain.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere gratitude to my co-supervisors Dr. Ramiro Liscano and Dr. Emil Petriu, for their guidance, support, and patience throughout this research work.

Dr. Ramiro Liscano has provided both his time and support within my research area. As well, I would like to thank Dr. Emil Petriu for providing the opportunity to work as a Research Assistant and be part of the Sensing Modelling Research Laboratory (SMRLab) team.

I would also like to thank my Mother and Father for their love, encouragement, guidance and support throughout my graduate studies.

Further, this research project would not be possible without the assistance and thesis writing support of Mr. Rami Abielmona. I would also like to thank Mr. Ajith Kamath for his assistance as part of the Web Services Feasibility for Sensor Networks research team.

I would also like to acknowledge both NSERC research grants, CITO research grants for making this research work possible, and Mr. Jurij Zurba (AlertWorx) for providing sensory equipment used throughout this research.

Glossary

Instrumentation Related:

CCD	Cluster Capability Description
COM	Communications Port
CORBA	Common Object Request Broker Architecture
IEEE	Institute of Electrical and Electronics Engineers
IFS	Internal File System
NCAP	Network Capable Application Processor
OMG	Object Management Group
STD	Smart Transducer Description
STIM	Smart Transducer Interface Module
TII	Transducer Independent Interface
TIM	Transducer Interface Module
TBC	Transducer Bus Controller
TBIM	Transducer Bus Interface Module
TEDS	Transducer Electronic Datasheets
XDCR	Transducer

Internet Related:

HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
LAN	Local Area Network
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAN	Wide Area Network

Web-service(s) related:

B2B	Business to Business (Interface)
M2M	Machine to Machine (Interface)
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery, Integration
W3C	World Wide Web Consortium
WSDL	Web Service Definition Language
XML	Extensible Markup Language
XSD	XML Schema Definition

Project related:

JVM	Java Virtual Machine
JSP	Java Server Pages
J2EE	Java 2 Enterprise Edition
OO	Object Oriented
RPC	Remote Procedure Call
RMI	Remote Method Invocation
SJAS	Sun Java Application Server
UML	Unified Modelling Language

Chapter 1

Introduction

The following research project entails the integration of emerging Internet technologies as applied to the sensor-networking domain. Next-generation sensor networks encompass the realm of emerging Internet technologies as a middleware platform for sensor network controllers, commonly characterized as: networked-data acquisition controllers. Such data acquisition controller's communication technology is advancing, in the form of web-service enabled M2M (machine-to-machine) interfaces.

In order to monitor and control a sensor network application, the IEEE 1451.1 Standard's NCAP (Network Capable Application Processor) is employed in this research work. The purpose of this research work is to demonstrate the suitability for the development and architecture of a Web-Service enabled NCAP controller. Advantages for deploying such architecture include sensor domain platform independence via web-service communication technologies, machine-to-machine (M2M) interface integration and self-describing instrumentation platform capabilities. Further, web-services middleware provides a solution for disseminating sensor information over an Internet network and provides ease in the integration of web-service client development.

Problem Statement

This research work addresses a sensor network's controller:

1. Is there an eloquent solution for grouping and classifying the functional capabilities (services) and message interactions (communication protocol) for an IEEE 1451.1 NCAP within a loosely coupled environment?

A Web-Service model and implementation of this functional-interaction could essentially support any web based and/or wired/wireless human machine-interface (HMI) device based upon a 1451-NCAP's service description via a WSDL (service definition) and a SOAP XML based message communication protocol.

2. Based on a Web-Service(s) NCAP integration prototype, what is the feasibility and measured performance result for adopting the Web Services framework to a 1451.1 NCAP controller?

The measured performance results and feasibility will be presented in this research work. The notion of web-services applied to the 1451.1 NCAP provide the ability for a remote client to invoke application services hosted by an NCAP over the Internet, or any underlying network infrastructure. Further, the deployment of a web-service based 1451.1 NCAP allows for extensibility, interoperability, platform independence, and increased rapid application software development time within the instrumentation sector.

Research Contribution and Objectives

This thesis will promote research within the instrumentation community, through the development of a next generation NCAP network controller. By applying the concepts of the XML-Web Services Integration Framework (XWIF) model from [Erl04], a SOA oriented NCAP controller is presented. The SOA based NCAP is developed from a traditional IEEE 1451.1 legacy application-specific NCAP model. The XWIF development process is required to extend the 1451.1 NCAP model's Transducer Block and Function Block into a service-oriented environment. Further, the XWIF process model defines the inherent service-interfaces for remote client and server interaction for software applications, which do not contain web-service implementations. Based on the service oriented NCAP model, a feasibility and performance study is applied to the Web-services enabled 1451.1 NCAP Controller.

Performance metrics examine the traditional 1451.1 based application specific NCAP controller's network communications in comparison to a 1451 based Web-services (SOAP-based) network communication via an experimental prototype. Emphasis is placed on the real-time performance metrics of the web-services enabled NCAP via a client-server interaction model, and server-server (M2M) interface interaction models.

An experimental prototype validation will cover a real-time smart multi-sensor (temperature, humidity, airflow, light, external I/O's, sound) connected via a serial-bus (DB-9/RS-485) to a Web service(s) NCAP controller hosted onboard a Sun Java Application Server (SJAS).

Research Motivations

The motivation for conducting this research work is summarized as:

- The development and experimental analysis of a sensor network controller based on a widely accepted IEEE Instrumentation Standard is proposed. The current state of sensor-network controller research, surveyed in (Chapter 3) provides middleware solutions based on simulated sensor network/fieldbus concepts or non-standardized sensor controller architectures.
- There is a lack of instrumentation research studies, which examine the feasibility or provide network performance results (metrics) for adopting a web-service middleware to the IEEE 1451.1 NCAP.
- The following research work provides a fundamental base and grounds for development within a new emerging technological area of Web-Service(s) integrated with 1451.1 NCAP controllers.

Publications resulting from this Research

The following publications resulted from this research work:

1. Sadok, E.F.; Liscano, R.; Petriu, E. "Web-Services for Next Generation Sensor Network Controllers" (Poster Title), Web-Services applied to 1451.1 Sensor NCAP (Abstract Title), Ontario Centres of Excellence (OCE) Discovery 2006 Conference/Showcase – Toronto, Ontario Canada February 7, 2006.

2. Sadok, E.F.; Liscano, R. "A Web Services Framework for 1451 Sensor Networks", Proceedings of the 2005 IEEE Instrumentation and Measurement Technology Conference, 2005. (IMTC 2005) Ottawa, ON Canada May 17-19, 2005. (IEEE CNF)

3. Liscano, R.; Sadok, E. "A Web Service Based Wireless Sensor Network for the Management of Disaster for First Responder Emergency Scenario", IEEE Ottawa Section-Workshop on Cyber Infrastructure and Emergency Preparedness Aspects, Ottawa, ON, Canada April 21-22, 2005. (IEEE Workshop)

4. Liscano, R.; Sadok, E.F.; Petriu E.M. "Mobile wireless RSA overlay network as Critical Infrastructure for National Security" Proceedings of the 2005 IEEE International Workshop on Measurement Systems for Homeland Security, Contraband Detection and Personal Safety Workshop, 2005. (IMS 2005) Orlando, FL USA March 29-30, 2005. (IEEE CNF)

5. Liscano, R.; Sadok, E. "Sensor Networks as Critical Infrastructure for First Response Emergency Teams", IEEE TC 30 on Security and Contraband Detection of the IEEE Instrumentation and Measurement Society-Workshop on Critical Infrastructure Protection and Emergency Preparedness, Ottawa, ON, Canada, December 10-11, 2004. (IEEE Workshop)

Organization of Thesis

The following concepts will be presented throughout this thesis, Chapter 2 contains an overview of the background technologies namely, the IEEE 1451 sensor interface standard family, along with Web-Service technologies and Service Oriented Architecture platforms. Throughout Chapter 3, a state-of-art analysis of current research works within the domain of sensor controller and interface technologies is presented. Namely, middleware approaches to networked sensor controllers, along with web-service(s) and web-enabled research applications in the field of sensor interfaces are examined, as well as self-describing instrumentation platforms. Further, Chapter 4 provides a research justification to the aforementioned research problem. In Chapter 5, an analysis of the traditional 1451.1's NCAP architecture and proposed web-services NCAP prototype are examined. Additionally, the web-services based NCAP prototype software architecture for both server and client are presented. Performance results based on the communication methodology of the Web-services NCAP proof of concept model are compared to the traditional 1451.1 NCAP model in Chapter 6. Further, a summary of contributions and concluding remarks are discussed in Chapter 7.

Chapter 2

Instrumentation-Interface Platforms and Web Service(s) Technology

The following chapter provides the reader with an overview of the background concepts presented throughout this research work. The section will entail an introduction of the sensor interface standard IEEE 1451, along with the domain of web-service technologies, and web-service enabling middleware.

2.1 Instrumentation Review of IEEE 1451 Standards

Throughout the 21st century, a proliferation of sensor-based applications exists within our society. Many of these applications are transparent to end-users: asset tracking, habitat and monitoring, national security monitoring, plant and maintenance sensing applications. In light of such sensing, monitoring and tracking applications, a need for standardization exists within the instrumentation and measurement scientific community. Currently, there exists a wide array of interfacing technologies such as *Canbus, Profibus, DeviceNet, Fieldbus and Ethernet* for sensor based systems. However, such a plethora of interfacing techniques decreases instrumentation interoperability. The Institute of Electrical and Electronic Engineers (IEEE) addresses this issue of transducer (sensor and actuator) interfaces, by defining an "IEEE 1451-standard based transducer interface." The IEEE 1451 standard [IEEE1451] entails a common transducer interface for acquiring measurement data, which is network independent. The standard addresses the definition of a transducer's functional and non-functional characteristics within an embedded TED (Transducer electronic data sheet) structure located in a non-volatile memory onboard the transducer. Further, the standard provides distributed network control and measurement based on a client-server and publish-subscribe model, which is protocol independent. As illustrated in Figure 1, the IEEE 1451 Family of Standards incorporates a system-wide solution for interfacing between multiple transducer types (Analog and Digital) and a wide array of interface protocols (Wired and Wireless) for communicating transducer related data between its controller and the NCAP.

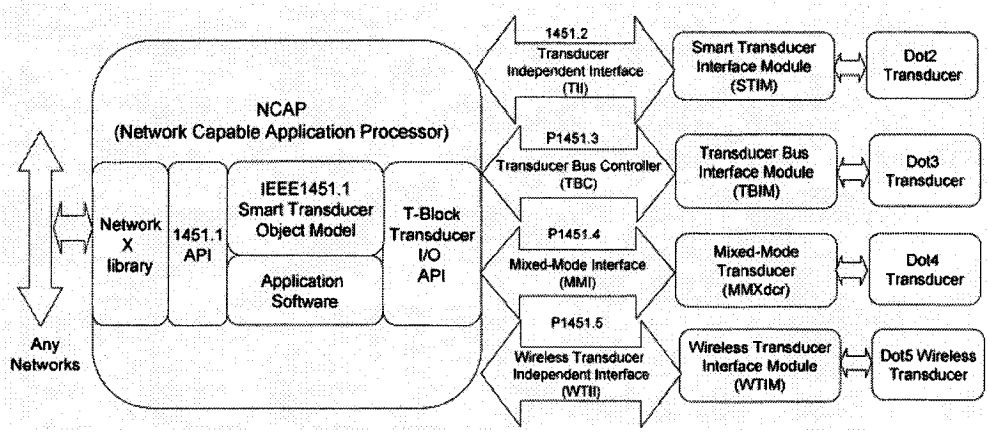


Figure 1: IEEE 1451 Family of Standards [LS04]

IEEE 1451.1 Model – NCAP Software Information Model (Standardized 1999)

The IEEE 1451.1 standard outlines the software-guidelines for development of a network capable application processor, which is defined as a platform independent and network-neutral object oriented model, thus promoting interoperability. The model addresses the concept of an external network-neutral communication scheme based on client-server and publish-subscribe modalities. The object-oriented model defines a distributed object software model characterizing a Data model for representation of 1451 data types, and an Object Model characterizing 1451: Components, Services, Transducer, Function, and NCAP blocks. Each of these components are based on objects, which can describe the functionality of an NCAP and acquired sensory data. Moreover, the functional (operational) objects of this model include transducer blocks, which provide a software-interface to the underlying 1451.x transducer interfaces, and encapsulate the details of the 1451.x hardware. From the transducer block, sensory data can be acquired, and categorized using the 1451 component objects. Furthermore, the application specific functionality for the 1451.1 OO model resides in a Function Block. A developer provides the application control functionality and application specific algorithms within this block. The Service blocks provide the external network-independent communication details facilitated by client, publisher, and subscriber ports. In addition, the NCAP block provides the services for the entire OO model, entailing ownership of blocks, and object instantiations. In a sense, the NCAP block acts as a housekeeper for the OO model and facilitates network communications.

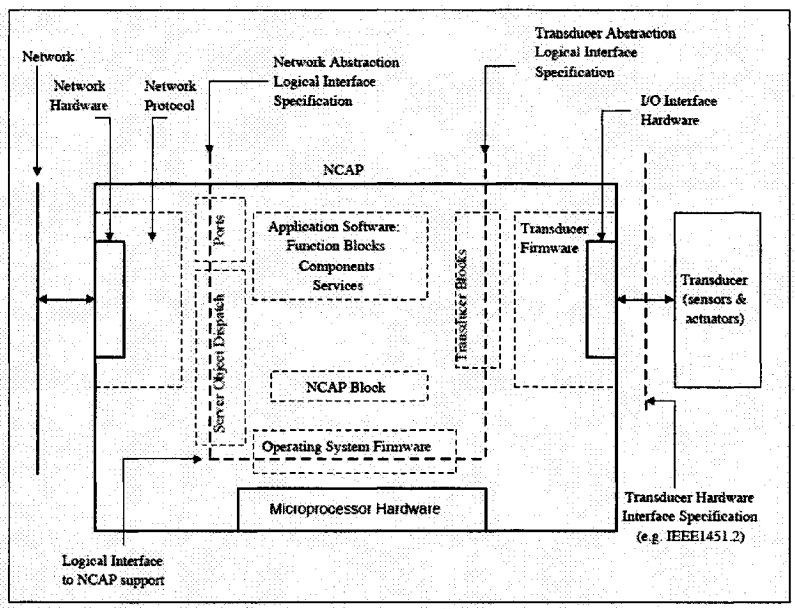


Figure 2: 1451.1 Network Capable Application Processor Model [IEEE1451.1]

It should be noted that a software developer within the procedure, for defining an Application specific NCAP, extends the network neutral, platform independent 1451.1 OO model. The functional capabilities of this model reside within a distributed object model, software representation, meaning for network communications, software objects are passed between client and server NCAPs. The NCAP illustrated in Figure 2, provides a distributed systems instrumentation platform for the control and dissemination of sensor network data over an external network.

IEEE 1451.2 STIM Model – Hardware Interface with TEDS Information (Standardized 1997)

The IEEE 1451.2 standard outlines the hardware sensor interface between an NCAP and a Smart-Transducer Interface Module. The 1451.2 STIM comprises of onboard transducers, signal conditioning and embedded Transducer Electronic Data Sheets (TEDS). Moreover, the standard proposes a digital Transducer independent interface (TII) for interconnecting an NCAP to the STIM. The TII interface is characterized by a standardized set of memory-mapped commands. These commands are comprised of a functional address (operation) and a channel address (transducer identification) representation. The commands allow for triggering sensors, reading/writing TEDS data, and acquiring sensory data for further processing onboard an NCAP. The interfaces supported by this standard include via RS-232, RS-485, Serial Peripheral Interface interconnections, and future enhancements to the standard include USB interconnections.

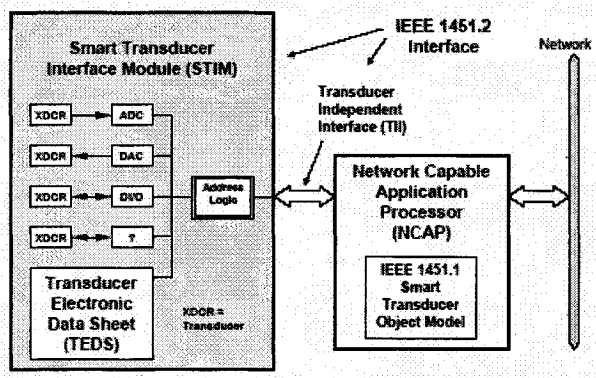


Figure 3: 1451.2 STIM Interface Model [IEEE1451.2]

As illustrated in Figure 3, the IEEE 1451.2 standardized interface is encapsulated within the TII interconnection. An STIM is commonly characterized as a microcontroller, hosting onboard or external transducers. The standard also provides a basis for the digital representation of TEDS data.

IEEE 1451.3 TBIM Model – Multidrop standard with TEDS information (Standardized 2003)

The IEEE 1451.3 standard provides a multi-drop bus sensor interface standard based upon the Home phone line networking alliance (HPNA) protocol. The HPNA protocol provides a two-wire bus, in which arrays of sensors share the transport medium for communication. The standard proposes the use of transducer bus interface modules (TBIM) along with a transducer bus controller (TBC) which provides data synchronizations and polling control capability to the sensor connectable multidrop bus. The transducer bus controllers reside within an NCAP's 1451.3 Transducer software block. Further, the standard outlines an XML and digital representation, for TEDS files which are housed inside the TBIM.

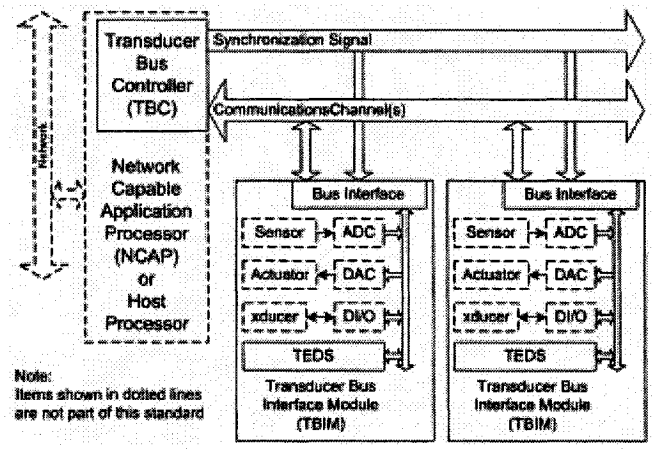


Figure 4: 1451.3 TBIM Interface Multidrop Model [IEEE1451.3]

IEEE 1451.4 Mixed Mode Transducer Model – MMI interface (Standardized 2004)

The IEEE 1451.4 standard addresses the need for interconnection of analog and legacy based sensors. The standard provides a mixed-mode interface (MMI) based on Class 1 internal circuitry defined for legacy transducers with analog interfaces, meaning analog sensor data and digital TEDS data is communicated over a shared line. The Class 1's internal circuitry differentiates between analog sensor data represented as a positive voltage transmitted over the shared line, to the digital TEDS data represented as a negative voltage transmitted over the shared line. (A negative voltage constitutes the representation of a logical 1 and a zero voltage represents a logical 0.) Moreover, the standard provides an additional MMI interface based on Class 2 internal circuitry defined for transducers which contain analog and digital interfaces, meaning the sensor data and TEDS data are communicated independently over separate lines.

The standard also introduces the concept of a minimal TEDS representation specifically for resource-constrained transducers. Moreover, the development of IEEE standardized templates for a variety of transducers is addressed. With these templates, onboard sensor TEDS data can be mapped and translated to the associated IEEE template, whereby the template provides a descriptive device capability and sensor data representation. The onboard TEDS are comprised of a bit mapping, meaning representational-descriptive device capability data is governed by the TEDS template. Further, in the case of legacy sensors, where no onboard TEDS exist, the concept of a Virtual TEDS is proposed. This means an NCAP or 1451.4 DAQ (Data acquisition device) can acquire the sensor's TEDS data located from a manufacturers Virtual TEDS database-via an Internet URI.

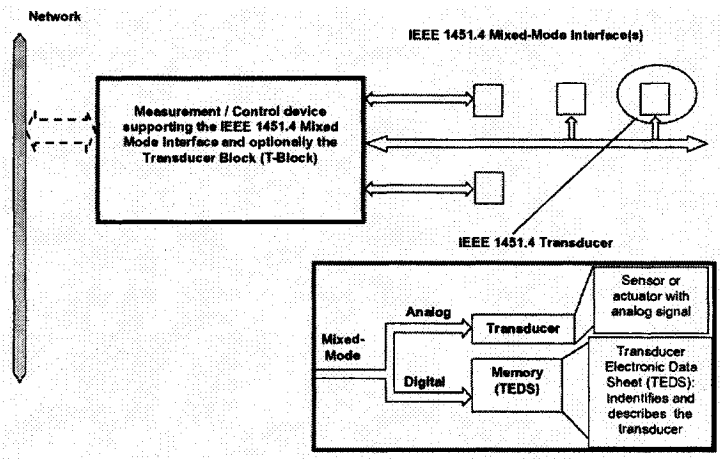


Figure 5: 1451.4 MMI Interface Model [IEEE 1451.4]

In addition, the standard outlines a 1451.4 Transducer Object-Model (Transducer-interface software capabilities) which may be part of an NCAP's 1451.4 Transducer Block, or solely located in a separate instrumentation DAQ device. Essentially, the Transducer Object model provides a minimal implementation of the functionality provided by the 1451.1 NCAP.

IEEE P1451.5 Wireless Transducer Interface Model (Proposed Standard 2005)

The IEEE 1451.5 working group addresses the need for integrating wireless-sensor interfaces interconnecting sensors to an NCAP. A Wireless Transducer interface module (WTIM) contains a radio-transceiver and facilitates the sensor interconnection to the NCAP. The wireless protocols act as a cable-replacement and are based on common IEEE 802.11.x Wi-Fi and IEEE 802.15.x PAN protocols. Typical interfaces are currently proposed for Bluetooth, Zigbee and the IEEE 802.11 protocols. Furthermore, the 1451.5 interface encapsulates the wireless channel's communication characteristics based on MAC (medium-access control) and PHY (physical) layers within a PHY physical TEDS. The role of PHY TEDS are being adopted into the emerging 1451.0 Common Communication based standard interface.

IEEE P1451.0 Common Functions and Communication Model (Proposed Standard 2005)

The IEEE 1451.0 working group addresses the concept of a universal common function and communication model encompassing the family of 1451 sensor interface standards. The 1451.0 layer proposes a module-communication interface (MCI), interconnecting an NCAP to a 1451.x TIM Transducer interface module. The MCI will provide a universal transducer interface layer within the 1451.1 NCAP, enabling the operations for low-level read/write functions to a TIM, discovery and registration, address group management, callback management, and IEEE 1588 time-synchronization. Moreover, a Common TEDS is proposed which encapsulates the message exchange protocols and command set for interconnecting TIMs-transducers.

IEEE P1451.6 CanOpen Bus Model (Proposed Standard)

The IEEE 1451.6 working group addresses the integration and development of an interface between an NCAP to the intrinsically safe CanOpen based transducer network protocol. The proposal addresses a mapping and integration of the 1451 based TEDS to the CanOpen based protocol.

2.2 Web Service(s) Review – Service Oriented Architecture

Web Service(s) Model (Service, Requestor, Directory) with SOA

A service-oriented architecture (SOA) is comprised of a web-service platform, which provides available "service" details to a client via a registry. A service is defined as a function or method with a clear endpoint interface in the form of a URI. Three roles are common to a SOA platform and are classified as: service requestor (Client), service registry (Directory) and a service provider (Host/Server). The basis of a SOA environment is to allow remote clients to query a registry server and obtain the underlying communication interface details, in order to connect and invoke a desired service, which is offered by a service provider, as illustrated in Figure 6. The World Wide Web Consortium [W3C] describes web-services as: "...a software application identified by a Uniform Resource Identifier (URI), whose interfaces and bindings are capable of being defined, described, and discovered as XML artefacts. A web service supports direct interaction with other software agents using XML-based messages exchanged via Internet-based protocols." The main driving force behind web-services lies in their interoperability, platform independence, extensibility, and programming language independence.

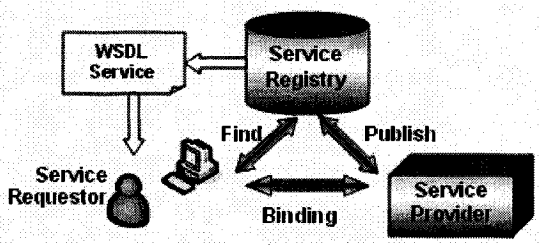


Figure 6: Service Oriented Architecture Model

Three main driving W3C XML based specifications are incorporated in the web-services model, namely a UDDI registry [UDDI], WSDL service description [WSDL], and the SOAP messaging protocol [SOAP]. A UDDI registry facilitates service discovery by providing service access details in the form of WSDL files. Clients can query the registry based on particular services offered and obtain the required connection details, in order to access the WSDL service based specification. A WSDL representation of a service provides a user with an abstract service interface, (namely details as to the available operations offered by a service) and a concrete service interface comprised of the protocol binding information which is accessed via a client. As well, a service interface implementation component is characterized via a URI, which points to the service location in order for a client to

connect to an available service. The underlying communication protocol in which WSDL interface (messages) operations are transported is facilitated via the SOAP protocol. SOAP provides a message-oriented platform, with an encoding mechanism in place for defining remote procedure calls (RPC) for client-server interactions, as well as, document-oriented calls for exchange of XML based documents. The SOAP protocol is classified as a transport independent protocol; however, it is commonly bound and transported over HTTP, SMTP, and FTP protocols. Extensions to the SOAP protocol such as the WS-Eventing specification [WSEventing] and WS-Notifications [WSNotifications] allow for defining publish-subscribe message interactions. The concept of web-services applied to the service-oriented paradigm is summarized by [ACKM04] as:

"Web service work on the assumption that the functionality made available by a company will be exposed as a service. In middleware terms, a service is a procedure, method, or object with a stable, published interface that can be invoked by clients. The invocation, and this is very important, is made by a program. Thus, requesting and executing a service involves a program calling another program."

Furthermore, a service oriented architecture platform facilitates the concept of machine-machine interfaces, whereby web-services are remotely accessible and integrated into varying software platforms. A closer examination of the web-service technologies incorporated within this research work is based on the WSDL service specification, along with the SOAP messaging protocol.

WSDL: Web Service Description Language – Web Service Descriptions

The Web Service Description language introduced by the W3C, provides the ability to define concrete service descriptions outlining the input-output relationship between operations exchanged amongst a server and a client. The WSDL service definition itself is a form of contract, meaning based on the explicit details of the WSDL document, interaction can occur between a recipient and a service adhering to the convention of the service definition. Moreover, the WSDL concept is analogous to the classical Interface Description Language (IDL) representation, which is used for defining services in distributed system domains such as CORBA. Similar to an IDL, the WSDL provides the ability to characterize the operations and services in a web-service domain. The advantage for employing WSDL is its ability to define the underlying service binding communication mechanism details, which is not described in an IDL file.

In addition, a web-service middleware such as: J2EE (SUN) or .NET (Microsoft), implement an automated procedure for creating a WSDL document, and specify mapping capabilities between an

XML based WSDL representation to the underlying programming language. For example, an XML based WSDL file is mapped to the underlying Java (web-service) implementation via J2EE's JAX-RPC software architecture.

Moreover, a WSDL V1.2 service description document is comprised of the following entities:

Definitions: provides the root of the WSDL document, housing service definitions and a definition for the XML namespaces used within the document. A namespace provides the ability to reference an external schema library within a WSDL document.

Example: <definitions name = WebReadSensor"
targetNamespace="urn:WebReadSensor/wsdl"> .....
..... xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"> </definitions>

Types: provide an XML representation of the data-types and objects characterized by an XML Schema,
Example: xsd:int, xsd:float, xsd:string

Messages: provide an explicit definition of the input-output parameters in a name-type format for characterizing the input and output parameters of a method/function call.

Example: <message name="WebReadSensorSEI_ReadSensor"/>
<part name="sensorID" type="xsd:int"/>
<message name="WebReadSensorSEI_ReadSensorResponse">
<part name="result" type="xsd:string"/></message>

The example code above defines an input sensor-id parameter, with an output result parameter in the form of a String (representing sensor data) is returned to a client.

Operations: define the method or operation name, and encapsulate the input-output defined message inside the body of an operation. There exist four operation interactions in a WSDL document. The operations may be defined as: One-way (Client invokes a service by sending a message), request-response (Client invokes a service, and the Server returns a response message), solicit-response (Server makes a request to a Client and the Client returns a response message), and Notification (Server sends a response to a Client without a request).

The two way-message capabilities for request-response and solicit-response are typical RPC calls, which provide synchronous interaction. The one way-message capabilities for one-way and notification provide an asynchronous interaction.

PortTypes: defines a grouping for the available service operations. (The PortType descriptor will be revised in the forthcoming WSDL V2.0 specification to an "Interface" descriptor.)

Example:
```
<portType name="WebReadSensorSEI">
 <operation name="ReadSensor">
        <input message="tns:WebReadSensorSEI_ReadSensor"/>
        <output message="tns:WebReadSensorSEI_ReadSensorResponse"/>
 </operation></portType> <!--Request-Response Interaction-->
```

The aforementioned WSDL document's components of: Type, Messages, Operations and Port-type define the abstract segment of the document, which provides a capability for reuse. From this information a software developer can reuse the service definitions and extend the abstract component for a number of different services with diverse bindings for communication. Next, the concrete implementation segment of the WSDL document specification provides the means for explicitly defining the communication/transport mechanism for exchanging messages, as defined by the operations of the Abstract Port-type definition.

Binding: defines the concrete protocol binding and data encoding for a Port-type's messages and operations. Typically, SOAP is utilized as a messaging protocol, which is transported over HTTP, FTP or SMTP. However, the protocol binding is protocol independent. In the case of a SOAP message binding, there exist two types of message-interactions. Namely, an RPC call whereby the SOAP message contains the input-output parameters for a method call, and conversely a Document-oriented call in which case the SOAP message contains input-output parameters defining an exchanged XML document, conforming to an agreed upon schema between both client-server.

An example depicts a SOAP RPC binding based on literal encoding schema:

```
<binding name="WebReadSensorSEIBinding" type="tns:WebReadSensorSEI">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
<operation name="ReadSensor">
  <soap:operation soapAction=""/> [1]
  <input>
   <soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></input>
  <output>
   <soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></output>
</operation></binding>
```

Port: defines an endpoint, interface binding accessible in the form of a URI, which maps to the protocol binding definitions. The URI is commonly the address of a SOAP processor for the WSDL service.

[1] This optional parameter "*soapAction*" can be used to specify the URI that identifies the intent of the message, and also acts as a Header in an HTTP POST request.
Based on the W3C SOAP 1.2 Specification [http://www.w3.org/TR/2003/REC-soap12-part2-20030624/#ietf-action]

Service: provides the ability to encapsulate and aggregate a number of Ports.

An example depicts the Service name and the associated Port hosting the web-service:

```
<service name="WebReadSensor">
<port name="WebReadSensorSEIPort" binding="tns:WebReadSensorSEIBinding">
<soap:address location="http://137.122.91.31:8080/WebNCAP/WebReadSensor"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"/></port></service></definitions>
```

Extensions to WSDL

The W3C provides complementary standards, namely the WSCI and BPEL models, which address the needs of the WSDL specification. The WSCI model provides a choreography interface for managing multiple web-service interactions. The choreography interface is modelled by a pattern, which characterizes the relative order of operations defined by multiple web-services. Currently, a WSDL document defines the available operations for a web-service, however the order in which operations are invoked is not captured. Thereby, for the case of integrating multiple operations from multiple web-services, a choreography interface is required to facilitate web-service collaborations. In terms of process flow and web-service execution, the Business process execution language (BPEL) addresses this requirement. The BPEL provides web-service orchestration, meaning the ability to compose and manage web-service message interactions. Moreover, internal business processes (code required to implement a web-service) are defined via process flow models and internal execution models. Additionally, BPEL provides the process-workflow specification for web-services and their respective operations defined by WSDL.

SOAP: Simple Object Access Protocol – Web Service Communication and Binding Support

The W3Cs based specification provides a lightweight protocol for exchanging messages in a distributed environment, via the SOAP protocol. The protocol provides a means for encapsulating XML based messages and documents within a SOAP Envelope structure. The SOAP concept is analogous to the form of mailing a letter, whereby an envelope provides a destination address, and encapsulates the contents of a message inside.

Moreover, SOAP messages are processed by SOAP nodes which contain a SOAP processor. The processor is required for handling the (transmission/reception) marshalling and demarshalling of SOAP messages and provides binding support to an underlying network communication protocol. The implementation for SOAP processors is commonly platform specific, however the messaging format is

standardized, which provides an interoperable form of communication between entities. The SOAP message structure contains header and body blocks. The header is optional, however this area provides the ability to add properties such as context, transaction, and authentication to the SOAP message. The body-block is required, since it contains the SOAP payload (message-data). There are three defined payloads for the SOAP message body, namely an RPC call in Figure 7, a Document-oriented payload in Figure 8, and a Fault-message. The RPC call message structure embeds the actual procedural call within the SOAP message, along with the input variable parameters. The response message contains an embedded result from the procedure. This format provides a programmatic style of interaction between two entities. Additionally, the document-oriented payload embeds an XML message within the payload of the SOAP body, this interaction results in the exchange of XML documents between two entities. A Fault message payload contains SOAP exception-handling details commonly raised when a request or response message cannot be handled by a SOAP Processor. Further, a SOAP attachment can provide the ability to embed images or files as part of the SOAP message.

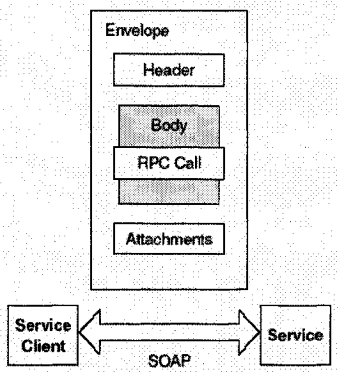


Figure 7: SOAP RPC Envelope [MTSM03]

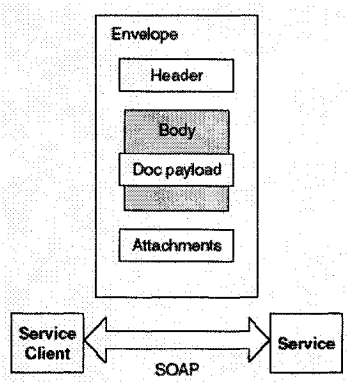


Figure 8: SOAP Document Envelope [MTSM03]

In addition to the SOAP payload, a SOAP encoding scheme is required for defining the data-types passed between two entities in RPC and document-oriented interactions. A "*literal*" SOAP encoding scheme adheres to the explicit representation of data-types as outlined within a schema or WSDL document. This means, a literal encoding must be resident within both client and server SOAP processors in order to process SOAP messages. Moreover, a SOAP "*encoding*" scheme adheres to the SOAP processor's internal encoding mechanism associated within a SOAP processor. Currently, the Web-Service Interoperability Organization (WS-I) mandates the use of a literal encoding, thus promoting interoperability amongst varying SOAP processors. Furthermore, the SOAP protocol is transport independent and facilitates an XML based message style of communication for web-services.

Extensions to SOAP (WS-Eventing and WS-Notification)

There also exist standards which extend the capabilities of the SOAP messaging standard, meaning publish-subscribe capabilities can be developed using the SOAP messaging protocol. A publish-subscribe model is realized via WS-Notifications or by the emerging web-services WS-eventing specification. The eventing-specification enables a publisher (event-source) to publish "*events*" over a specified time-duration to a subscriber's (event-sink). Further, subscribers can register to events which are explicitly defined within the WSDL service interfaces as an event-source. The capabilities provided by the WS-eventing of subscription messages, renew messages, and notifications are contained within the SOAP message structure. The WS-Notification provides similar registering capabilities for a subscriber to interact with a notification publisher. The notification publisher issues notification messages which are based on topics, and are sent to a registered notification consumer. The consumer acts as a web-service as part of the subscribers architecture.

2.3 Web-Service(s) in J2EE Framework

The concepts of Web-service technologies were introduced, namely WSDL and SOAP, however in order to take advantage of web-services a suitable middleware platform is required. A middleware platform implements web-service specifications and provides integration with existing software applications. The middleware can reside within an application server environment. One particular form of middleware for hosting web-services, is the J2EE platform illustrated in Figure 9. A J2EE platform hosts and provides web-service accessibility in the form of Servlets and Enterprise Java Beans which reside in web and EJB containers respectively. The middleware provides a messaging mechanism, by way of the JAX-RPC specification, required for WSDL service generation and SOAP processing.

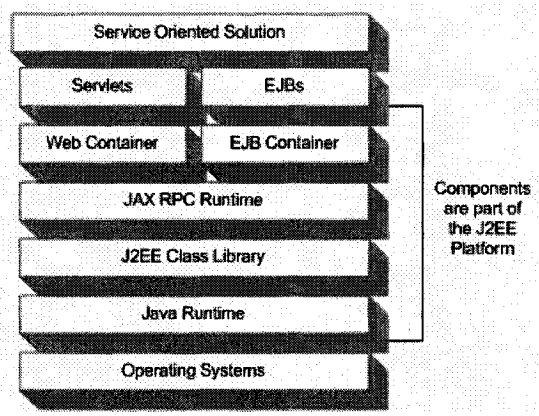


Figure 9: J2EE Platform for SOA (Adopted from [Erl05])

The J2EE platform is comprised of application programming interface's (API's) which offer different forms of functionality. Namely, JAX-RPC provides the underlying web-service SOAP RPC enabled protocol for service providers and service consumers, SAAJ provides a SOAP messaging capability with attachments, and additional J2EE middleware API's available in [J2EE05] provide the ability for developing, deploying and managing web applications onboard an application server.

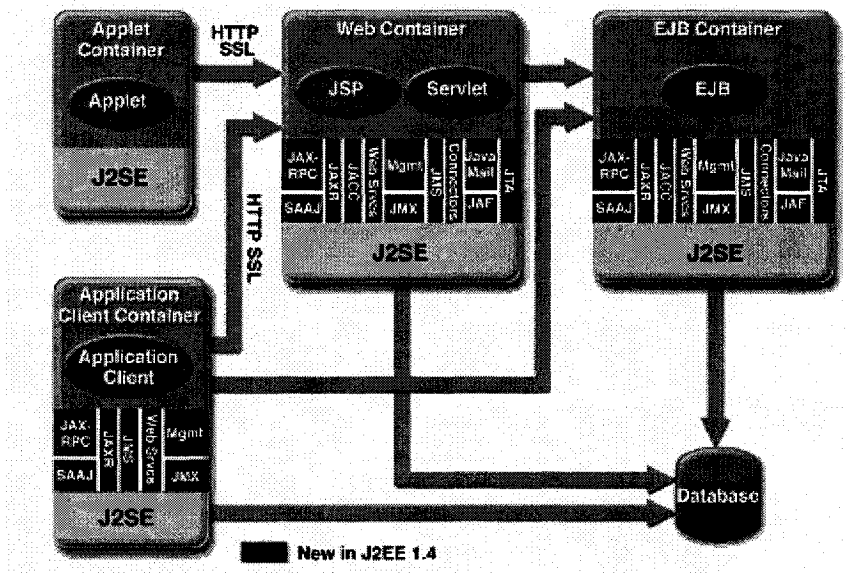


Figure 10: J2EE API's and a J2EE Middleware for SJAS [J2EE05]

From the J2EE architecture illustration in Figure 10, the server environment is responsible for handling and managing requests for client-server interactions, and can be categorized by the following layers: client tier, web tier, business tier, and Enterprise Information Systems (EIS) tier. The client tier is characterized as the client-interface facilitated by a browser connectable to dynamic HTML pages, or a client-side application (for example, a Java-applet, or application client container). The web-tier provides a client-side interface in the form of Java servlet technologies and JSP's, which are hosted onboard, the application server. (The servlet technologies process HTTP requests and responses, and host a dynamic HTML web page and integrate underlying web-services within the web page presentation format.) The business tier provides enterprise Java beans or servlets the ability to manage and encapsulate the web-applications functionality and logic. Finally, the EIS tier provides an interface to a database, or legacy applications using the application server's software-connector technologies. A software-connector may consist of JDBC interconnection to a database, or a Java RMI for connection to a legacy application.

Based on the J2EE middleware architecture, a web-service interaction pattern requires both a service provider and a service requestor. A service provider provides a service-endpoint interface (J2EE port URI for location of a web-service) which facilitates the interconnection between a web application or web-service hosted in the form of a servlet or JSP page to the underlying business component logic. The business component logic is characterized as a service-endpoint implementation, where JAX-RPC provides the required web-service mappings and bindings to a Java application at the service provider. The service endpoint interface is characterized from a WSDL document's service and port definitions, along with the accompanying port-type operations.

From a service requestor role, a WSDL document is acquired and by utilizing JAX-RPC, a service client definition can be generated and defined. There exist currently three specifications based on client side service generations. Namely: static stubs, dynamic proxy and dynamic invocations. The generated static stub includes a JAX-RPC generated copy of the WSDL service constructs such as service, ports and portType definitions for the service requestor. This stub generation provides an implementation specific skeleton copy (defines a proxy) of the remote method definitions which reside on a service providers application server. Further, the stubs are generated onboard the client at design time via the aid of the JAX-RPC tool. From a client side perspective, invocation of operations defined via the stubs resemble a local method call onboard the client, however the invocation is sent over a network to a server, in turn a result is returned to the requesting client for a remote method invocation. Further, the dynamic proxy resembles the generated stubs concept, however stubs are not generated until a method invocation is made at runtime. The current J2EE 1.4 middleware resembles this form of functionality within web-service client side generation. The dynamic invocation interface defines interaction between a service requestor and a WSDL service definition at runtime, and does not rely on the generated stub concept.

Moreover, the generated static stub (at design time) concept relies on an implementation specific view of a WSDL service definition, this means if a WSDL service's interfaces change over time, the static-stub generated client will become obsolete and interaction may not be possible. However, the dynamic generation methods provide client-side generation at runtime, meaning if a WSDL service interface changes, the client interface will be updated. Furthermore, the J2EE middleware provides the necessary message processing capabilities for service requestors and service providers, and an implementation of web-service standards and concepts.

Chapter 3

The instrumentation and measurement domain is widely progressing and rapidly expanding within the areas of: network-capable, automated measurement, and controller devices. Recent application areas include sensor network node-controllers, to standardized instrumentation middleware solutions provided by IEEE 1451, to the integration of web-enabled and services oriented approaches. Characteristics of an instrumentation application are integrated into the following areas: being networked capable, easily accessible, remotely configurable and self-describing. These capabilities are defined in next-generation sensor controllers incorporating both sensor-middleware/interfaces to the processing capability, and high-level data dissemination protocol capabilities over Internet networks.

Numerous large-scale and enterprise oriented research projects [JDDAC] and [SensorNet] provide emerging instrumentation-measurement platforms based on Web-service and Service Oriented Architectures. These developments range from a collection of task-oriented components with capabilities for collecting sensor data, to processing and data classification and network distribution.

The following chapter will introduce current research initiatives within the sensor-network controller domain, aimed specifically at integration solutions between common interface-technologies such as the IEEE 1451 sensor interface standard and the OMG Smart-Transducer Standard. Further, a review of current web-service(s) enabled instrumentation-middleware research is presented, along with web-enabled instrumentation middleware research. Concluding this chapter, a review of instrumentation platform and standardized sensor interface self-describing device capabilities are addressed for web-based and web-service domains.

3.1 Instrumentation Middleware Solutions

The instrumentation middleware is comprised of standards based procedures for interfacing transducers/sensors to a sensor controller. Common standards associated within the instrumentation domain include the IEEE 1451 Standard [IEE1451] for sensor interfacing, along with OMG's Smart Transducer Interface model specification [OMGST03].

IEEE 1451 – Sensor Interface Standard - Object Oriented 1451.1 NCAP Model

The current design specifications for a Network Capable Application Processor (NCAP) define a common object model (or information model) as outlined via the IEEE 1451.1 standard committee [IEEE1451.1]. The common object model is characterized as an abstract object-oriented software architecture defining each of the following: application software, function blocks, components, and services. Furthermore, two interfaces exist for the control of transducers and network communications, as defined by the 1451.1 standard. Transducer control is defined via an input-output logical transducer block interface, which encapsulates a transducer's hardware implementation into a software model. Network communication is defined by an NCAP block and port interfaces which contain a set of communication interface methods (client-server, publish-subscribe) in turn encapsulating the network protocol details. Moreover, the information model (also referred to as a common-object model) of the IEEE 1451.1 NCAP is characterized via a UML Model for the 1451.1 standard within the works of [LS03]. The UML 1451.1 model illustrates the attributes, methods, and behavioural information of the 1451.1 standard, as well as provides software developers a structured framework for implementing a 1451 NCAP application processor.

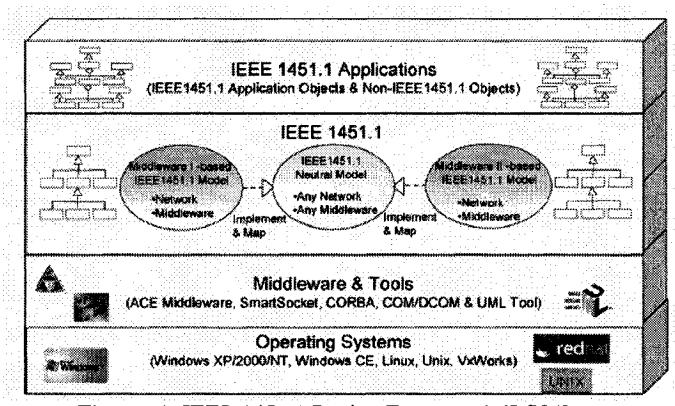


Figure 11: IEEE 1451.1 Design Framework [LS04]

By employing the development process of the 1451.1 UML model, a number of 1451-based NCAP projects are available: Lee and Song [LS04] and the work of, Lee and Schneeman [LS99]. These projects extend the common object model into a web enabled application specific NCAP. Furthermore, the works of Viegas and Pereira [VP05], and the work of, Sadok and Liscano [SL05] look at the applicability and provide a design for developing a web-service enabled NCAP sensor controller. Currently, research work encompassing web-service integration solutions applied onboard an NCAP application-specific software-model is still in its infancy.

OMG Smart Transducer Interface – OMG (2002)

The OMG Smart Transducer Interface specification provides a smart-sensor interface platform encompassing the TTP/A sensor fieldbus protocol, accessible via a CORBA based distributed system framework gateway. The OMG Group proposes a list of requirements for developing a smart-transducer interface which are outlined in [EP03]:

1. Real-time characteristics and functionalities for the smart transducer network.
2. Online diagnostic service capability.
3. Support for start-up and dynamic configuration.
4. Uniform naming and addressing scheme for all relevant data in the smart transducer system.
5. Generic interface that enables the smart transducer system to interact with other systems via a CORBA gateway.
6. Support of communication interfaces (UART) available on current low-cost microcontrollers.

According to Elmenreich and Pitzek [EP03]: The interface to the smart transducer employs the concept of an interface file system (IFS) that maps all relevant transducer data to a common address scheme. The IFS allows different application specific views of a system; namely, a real-time service interface, a diagnostic and management interface, and a configuration and planning interface. The real-time services provide timely real-time services to the smart transducer. The diagnostic and management interface provides transducer channel access. The configuration and planning interface provide access to node configuration.

Further [EP03] states: "The interface concept encompasses a communication model that allows accessing the IFS data via a uniform addressing scheme from a CORBA gateway object and provides real-time, time-triggered communication among the smart transducers." The proposed Smart Transducer Interface specification was adopted by the OMG in 2002.

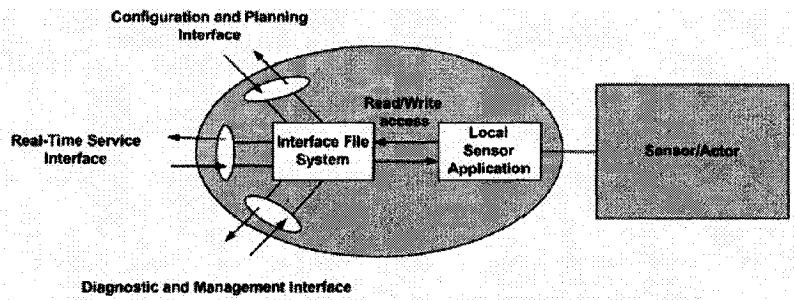


Figure 12: Smart Transducer Interface for OMG Specification [VM04]

As part of the OMG smart transducer interface, a web-service interface based on the three application-specific views of the OMG system are proposed by Venzke and Pitzek in [VPt03], and developed for accessing fieldbus systems by Merino [Merino04]. Also, Pitzek and Elmenreich [EP03] proposes and defines the use of incorporating a Smart-transducer XML based description framework for TTP/A fieldbus systems, which encapsulate the IFS interface.

A comparison between the IEEE 1451 and OMG standards yields similar device description and capability. The 1451-based smart transducer system encapsulates device functionality, onboard the transducer, in a TEDS file. The OMG based smart transducer interface contains an onboard local-software application hosting a local IFS file system. Such a system allows for grouping the sensor capabilities, readings, and characteristics, which are stored in [EP03]'s STD, smart transducer description XML based files. Henceforth, the IFS file system can be accessed and retrieved via the CORBA based gateway server. The server resembles the functionality provided via an NCAP in the 1451 approach. Both approaches are based on distributed object model systems. Note, however that the IEEE 1451.1 approach provides a platform independent software object model and the 1451 based standards coincide with the smart-transducer interface requirements proposed by the OMG group.

3.2 Web-Service(s) enabled Instrumentation Platforms

Web Services for Fieldbus Systems

The works of Merino [Merino04] introduce the concept for developing a web-services interface for accessing fieldbus systems comprised of the OMG Smart sensor interface. The Web service interface presented throughout Merino's research work focuses on a client side user-interface for remote accessibility hosted via a web-service SOAP processor gateway to the OMG Smart transducer interface. The generated web-service interfaces are defined for the application specific interface functionalities of: real-time service, diagnostic and measurement service, configuration and planning service. The three categories provide different forms of accessibility to a smart-transducer node. Moreover, client-side WSDL interfaces are designed with a high-level abstraction (i.e. login, read, execute) in order to access a fieldbus gateway. The gateway provides accessibility to the underlying smart-sensor interface nodes, which can be grouped in cluster formations. The cluster formations consist of sensor-connectable microcontrollers accessible via a TTP/A Fieldbus. This is a similar concept to the interconnection between an NCAP (Gateway node) connected to multiple STIM's as per the IEEE 1451.2 standard.

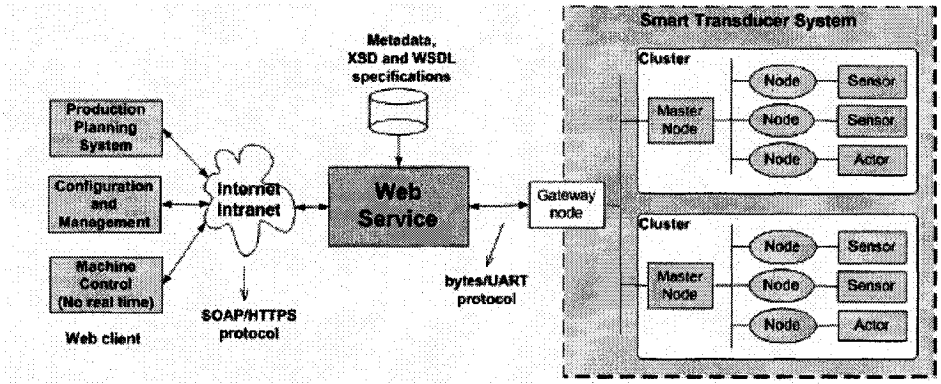


Figure 13: Web-Service Interface Access for OMG Standard [VPt03]

Merino proposes the development of a web-services interface SOAP-based gateway external to the Smart-transducer system, which extends the proposed access to the fieldbus system via a CORBA gateway node's interface. This design requires a web-service SOAP processor (handling SOAP messages) and a service processor for handling interface/operation requests to the Real-time service (Machine Control), Diagnostic-Measurement, and Configuration-Planning Interfaces. This means that requests are directed to a local cluster-capability descriptions (CCD) file containing the smart-transducer system's IFS and STD's onboard the service processor. Further access to the smart-transducer network would require a mapping between the web-service request and the low-layer UART-bytes protocol in order to access the Smart-Transducer system.

A request to the Smart Transducer system is handled via the CORBA gateway node, which facilitates requests for master-controller nodes of the sensor fieldbus.

An analysis regarding the aforementioned research work yields a simulated-web-service interface experiment. Three application web-service interfaces were developed for access to a Smart Transducer fieldbus system. However, there were no studies conducted with a physical Smart-transducer system. The work is based on a simulation for verifying the functionality of the SOAP based processor and service processor; namely, in the areas of: handling interface request calls, providing session management capabilities, and user management capabilities. The study provided a simulation for accessing an internal IFS file structure located onboard the web-service processor.

Further, the application logic for accessing the system is contained in the web-application, mainly a client-side user-interface as well as hosted in the fieldbus system at the gateway node. The work provides a structured Web-service based WSDL representation, along with a SOAP processor messaging schema and operation requirements for accessing fieldbus' remotely.

1451.1 NCAP for a Commercial .NET Framework

The works of Viegas in [VP05] provide a web-services enabled IEEE 1451.1 NCAP implementation via a commercial middleware framework. The developed NCAP application is based on a two-fold architecture namely, an "NCAP engine" encapsulates and implements the details of the underlying 1451.1 Application object model, and an NCAP web-service hosts the web-service interfaces for accessing the NCAP's services. The solution for hosting web-service is based on the Microsoft .NET middleware platform. The web-services tier is hosted via an ASP .NET enabled Internet Information (IIS) web server, which facilitates web-service requests from remote clients.

In addition, an Internal Process Communication (IPC) interface exists between the web-service layer and the application specific NCAP via the .NET remoting (distributed-system) protocol. Further, an interoperable IVI based sensor interface is realized within this architecture connected to the NCAP engine. Viegas research work provides an example for developing an NCAP application via the .NET web-services environment, which utilizes the SOAP/XML based communication protocol for NCAP network connectivity.

The developed web-service interfaces in this research work provide a remote-connection for accessing an IVI based instrumentation layer. The Interchangeable Virtual Instruments (IVI) foundation provides a sensor/instrumentation interface and a software-oriented class based instrumentation representation for connectable sensors and instruments.

Within the design of this .NET web-services enabled NCAP, it is not apparent if the application specific control functionality (i.e. measurement filtering) resides inside a Function block for the given NCAP application. It appears the IVI instrumentation interfaces; provide the system's application specific functionality. Moreover, the results of this work illustrate a web service interface example scenario, whereby an Internet/HTTP application layer transport is utilized for remote accessibility to the IVI instrument's sensor data via a web-browser. This work is based on client-side web-based interfaces,

hosted via ASP. NET and middleware interoperability issues for this system design platform are a challenge, in terms of cross-platform (operating system) interoperability. This is in terms of migrating this design to another middleware platform.

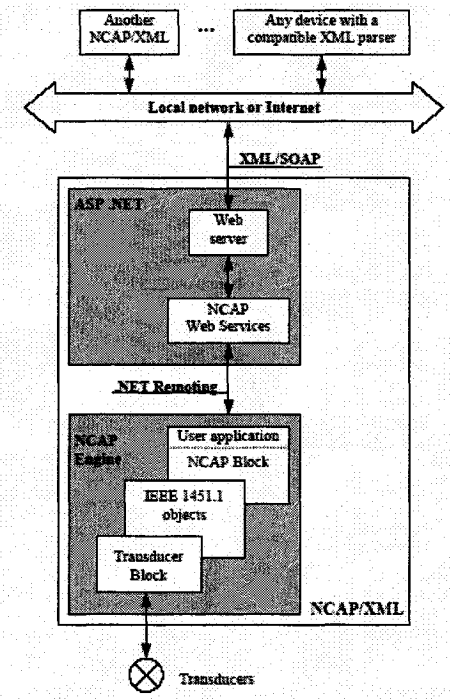


Figure 14: A Web-Service .NET 1451.1 NCAP [VP05]

Furthermore, at the NCAP engine layer, the 1451.1 communication services are modelled utilizing the .NET Remoting protocol, meaning NCAPs which host the .NET Remoting protocol may facilitate inter-NCAP communication services. The .NET Remoting technology is similar to the concept of web-service remote-procedure calls, however this technology is platform and implementation specific, thereby causing a loss in interoperability. (This means both client and server must contain the .NET Remoting protocol implementation to facilitate communications.) As well, the NCAP engine is separate from the server, relying on an internal .NET protocol to access the web-services, this can pose a performance issue. Note, an IPC is realized via the .NET remoting protocol, since the application is not resident onboard a server.

Moreover, a WSDL service definition or interface should be available to an end client within this architecture. This pertains to the available NCAP services for client and remote M2M configurations. In order to provide interoperable services and facilitate the concept of web-services, a service specification

characterizing the NCAP's functionality is required. The service specification also defines the explicit SOAP message encoding, which is an integral part for client-server communications. In terms of platform-interoperability, the defined message encoding style is a critical component, which must adhere to the WS-I SOAP messaging communication requirements for interoperable communication.

SOA for Wireless Sensor Networks

The works of Delicato [DPPC03] define a Service-oriented architecture (SOA) middleware platform for wireless sensor network applications. This middleware resides between the OSI models application layer and networking layer. Delicato's middleware specification provides a solution for sensor sinks to communicate with sensor nodes based upon data-centric underlying network protocols. For example, directed-diffusion routing is proposed as one form of a network layer. The essence of this work relates to the proposed usage of the SOAP communication protocol, whereby data-centric specific SOAP handlers are defined in order to accommodate tasks of: message-handling capabilities and specific routing capability/interest mappings.

Delicato's work is an abstraction of SOAP based wireless sensor network middleware, which is based on a sink-source node representation. The proposed communication methodologies are based on inter-SOAP communication via a Document-oriented messaging approach.

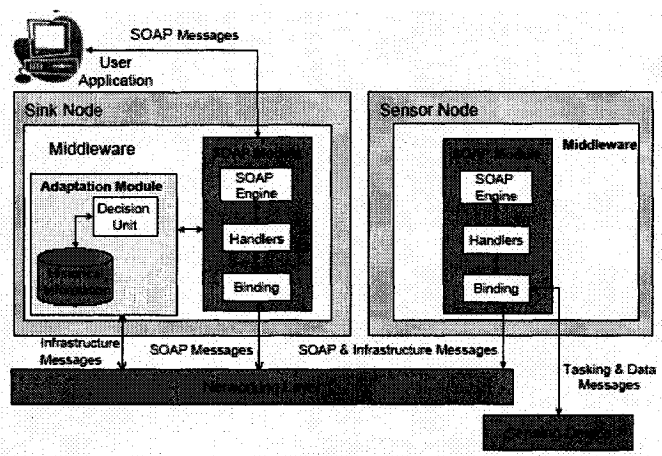


Figure 15: SOA/Web-Services Middleware for WSN [DPRP04]

The approach provides for self-describing device capabilities, by exchanging XML based files between each SOAP processor, in a Document-oriented message pattern. However, with this added description,

there is a cost in the size of the messages passed between each node. The functional capabilities are defined within a web-service WSDL document outlining the SOAP node and SOAP sink capabilities, primarily based on sensor metadata and underlying communication capabilities. The advantage of employing this concept is the ability for interconnecting wireless sensor networks in a machine-readable and node-descriptive manner.

3.3 SOA enabled Web-service(s) 1451 Solutions

Java Enabled 1451.1 JDDAC-Netbeams Project

The development of a 1451 web server-based solution is presented throughout the JDDAC research initiative. The research initiative [JDDAC] is classified as a "Java Net" developer-community project. The JDDAC initiative comprises of a number of integrated instrumentation-modules namely, a JMDI-Java measurement data interface: provides measurement communication and processing capabilities, JTI-Java transducer interface: providing measurement data acquisition and sensor meta-data representation. An application scenario, which employs these components, is based upon the "Netbeams" web-based application portal [Netbeams]. The portal provides sensory data information, via Numeric Really Simple Syndication (NRSS) XML-based messages. The sensor nodes within the architecture are inter-connected by a peer-peer JXTA sensor-network configuration. Furthermore, the JDDAC project provides a unique web-enabled development solution for configuring an NCAP application hosted by an Agilent Server via the JMDI Java measurement data interface.

The JMDI server is based upon the 1451.1 Object Oriented NCAP Model, and provides a client/developer the ability to configure application-specific measurement functionalities based upon a data-flow interface API. The data flow interface entails a unique application of the NCAP model, whereby groups of Function-Blocks (which characterize application specific functionality), are extended in order to implement internal-publish/subscribe interfaces. This solution provides the ability for each Function-block to perform its respective task of processing sensory data in a data flow graph-fashion. This concept is a realization of the component-model based 1451.1 OO model. By applying the specifications of the 1451.1 model, a data flow interface is available.

Moreover, the JDDAC project provides the means of defining a web-enabled online re-configurable NCAP model hosted by the Agilent Server. Developers implement XML based configuration files

depicting application specific Function Blocks, and Transducer Blocks for system-component development.

In addition, a unique characteristic of the JDDAC project includes a 1451.1 mapping between the 1451.1 Data Model to an XML based representation for communication. The mapping employs the traditional 1451.1 data model Argument Array (data-structure), utilized as a container for passing data within the 1451.1 communication services. The developed mapping includes translating the 1451.1 data model's Argument array into a Java based vector representation. In turn, a translation to an XML schema based complex-type, named an "arg-array" data type structure, is defined from the Java vector. This representation allows for easier mappings between Java and an XML based schema. From this XML based schema, the NRSS syndicated message structure is defined, which allows for a machine-readable and human-readable representation of the sensor-meta data. The concept of an RSS communication pattern is employed in NRSS, which is based upon a NRSS server pull-model, meaning clients request NRSS data feeds from the server.

Furthermore, the JTI interface encompasses a Java-based implementation of the 1451.x sensor-interface standards for connection to underlying sensors and instruments. This work defines a Java mapping for Transducer blocks located inside an NCAP; in turn, mapping to the lower-layer 1451.x sensor interface bit-stream representation. However, the JTI project is still in development.

Sensornet GOV – Based on OGC Sensor Web Enablement

The following enterprise-network solution for national security capabilities is based on a SOA enterprise-environment. The [SensorNet] project provides a service-based platform for connecting sensor-based nodes (acting as data collectors) that are hosted via a web-server platform to end users and remote sensors nodes. The sensor-based nodes encapsulate the functionalities of 1451.2 STIMs and its defining sensor interface, via a software-wrapper connected to a web-server. As well, functionality for legacy sensors, which do not contain TEDS, are emulated via software TEDS housed inside the software wrapper. The required sensor data is further processed onboard the sensor node, and categorized via a web-feature interface service termed WFS. The web-feature service provides the ability to group sensors and their respective data in request and response XML based messages over the Internet HTTP transport protocol.

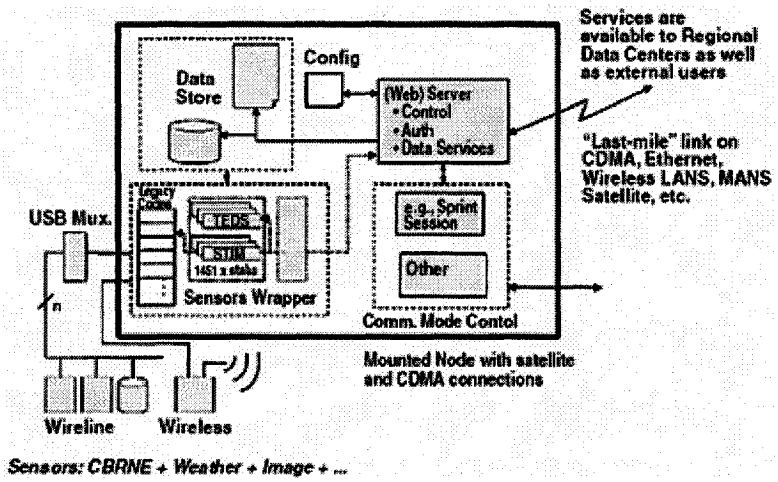


Figure 16: SensorNET Sensor Node [GSS05]

Finally, the sensor nodes are defined as resource-rich processors. The sensor node emulates the behaviour of an NCAP's functionality, providing the capabilities for filtering, averaging, and data validation, along with network communication capabilities. The sensor nodes also provide explicit service details, publication of sensor data, and alerts via an XML based Common Alerting protocol. Moreover, the aforementioned project provides a web-services enabled 1451 based sensor interface technology oriented towards the application of monitoring critical infrastructures and national security.

3.4 Web enabled 1451 Solutions

Similar 1451 enabled instrumentation research work, in terms of web-based smart sensor network controllers, are addressed within the Distributed Measurement Control (DMC) architecture of Lee and Schneeman [LS99] and the 1451.1 object oriented application specific NCAP model framework by Lee and Song [LS04]. Both models are quite complementary in terms of the NCAP design however the architecture for each varies. The application specific extended NCAP model [LS04] is based on an ACE middleware (tightly coupled) environment and communication services are facilitated via TCP/IP based network communications, in turn an Ethernet based sensor-bus is employed in [LS99] providing adequate performance measures for real-time data capabilities. Further, a Java web based interface is introduced in Ferrari's research [FFMST02], for remote monitoring and control of 1451 sensor networks as developed onboard an STIM.

The aforementioned research-projects provide web-enabled solutions for access to sensor data. However, the web based interfaces for interacting and communicating with transducers is not exposed

to the end user in a generic service interface format. Application specific functionality resides inside an NCAP, or in the case of a Java-web based interface, the application functionality resides within a Web-Smart Transducer Interface Module (STIM). Further [FFMST02] provides a performance metric for evaluating the critical response times over an HTTP transport, for accessing a sensor based TED and triggering a sensor channel for read-write operatives, via the web-enabled STIM.

The works of [FFMST02] expose methods to an end user which are based on low-level IEEE 1451.2 TII (Transducer-independent interface) syntax. The work of [LS99] abide by the 1451.1 based standard, however re-configurability at the application layer via a human-machine interface (HMI) does not exist, or requires further application programming support. For instance, a new sensor introduced within a sensor network, which contains added functionality, requires a service-interface to expose the proper methods for operation and interaction with an end-user. With the introduction of Command TEDS (outlined in the IEEE 1451.3 specification [IEEE1451.3] along with the recently introduced IEEE 1451.4 standard's [IEEE1451.4] transducer device-description schema files), one could provide added functionality to an end-user, when such XML based-TEDS and schema files (XSD's) are exposed via a web-service interface or via a Web-service application interaction tool.

One method, in terms of web-based management and control of a 1451 sensor network is proposed by Scherer in [STKV03]. This work provides the development and integration of an SNMP network-based, scaleable Smart transducer network applied to the 1451 standard. Whereby, 1451 transducers are managed and controlled via agents defined by the Simple Network Management (SNMP) protocol. The protocol defines a management information base (MIB), which is applied to a 1451 sensor network for providing system information of transducers (based on TEDS data) and an STIM. However, the application core exists onboard the transducer and is not resident within an NCAP module, thus resulting in a centralized architecture. The 1451.1 standard entails a distributed architecture, in terms of the following work a web-based NCAP service interface provides a distributed architecture adhering to the specifics of the standard. Further, a web-based services approach provides the ability to manage sensors based on their service descriptions.

Generally, the web enabled 1451 solutions are either STIM based, (meaning onboard internet-accessibility to the physical sensor itself), or defined within an NCAP controller. Further, the web-enabled NCAPs in this sense are tightly coupled, and often require the use of a Java applet for implementing client-interfaces.

3.5 Device Descriptions and Platform Capabilities

The works of [Horan05] addresses the need for providing explicit descriptions in terms of the functional capabilities of transducers, within a wireless transducer network. The following requirements and needs for introducing capability descriptions to sensors/transducer are addressed:

- The functionality of a device should be described in terms of its syntax and its semantics.
- The device and its description should be closely linked.
- The description should be machine-interpretable.
- An a priori agreement on standards is insufficient.

From the aforementioned literary resources, both IEEE 1451 and the OMG STI specification provide a transducer device description capability. The IEEE 1451 device descriptions are internally contained within the transducers, as the OMG smart interface specification separates the device description to an onboard local file-system device capability description, for a smart-transducer controller node.

Onboard Device Description IEEE 1451

The Transducer Electronic Data sheet (TEDS) representation provides sensor self-description and interface connection details for a transducer. Namely, the Transducer electronic data sheet, contains a machine-readable internal representation of a transducers characteristics, which coincides with traditional paper data-sheets outlining the sensor's intrinsic-extrinsic operating specifications.

The type of TEDS varies based on the implemented 1451.x sensor-interface protocol. Since each 1451 interface protocol is designed for varying transducer types and interconnection-bus types, there exists a differentiation in the type of TEDS used. However, a common/universal physical communication interface based TED will be shared amongst the varying 1451.x interface technologies: namely, the Common TED/Physical TED, which is in the process of development by the 1451.0 Working Group.

The following TEDS are associated with the 1451.2 digital representation for digital bus interfaces, and 1451.3 XML based representation for interconnected sensors, encompassing digital transducers with onboard memories, along with a Virtual TED representation for legacy-based transducers.

- BasicTED/MetaTED - define the static characteristics of transducer manufacturer, serial id, etc.
- TransducerChannelTED - define the transducer channels communication capabilities for digital interface
- CalibrationTED - define the mapping between raw transducer data to an interpretable corrected format.
- FrequencyResponseTED - define explicit characteristics of a transducer's signal-processing capability.
- VirtualTEDs - adopted for legacy based-sensors/transducer without an onboard memory EEPROM.

Moreover, in terms of the 1451.4 sensor interface standard, a mixed mode interface provides analog/digital data output. Further, the 1451.4 standard addresses the need for self-describing device capabilities associated with physically resource-constrained sensors. The TEDS representation is based on a bit mapping, located inside a small onboard memory of 640 bytes. The process for reading the contents of these TEDS involves the raw TEDS data (bit mapping) which is correlated with a corresponding IEEE Template located inside a 1451.4 Transducer software-block associated with an NCAP or a 1451.4 based DAQ instrument. The IEEE standardized templates provides a device description for a group of transducer-types (RTDs, Microphone, Pressure, etc.), and provide the means for identifying measurement data (via the associated Measurement units, Calibration information, etc.) of an onboard transducer.

Standardized Instrumentation Platform Capabilities

In addition, from an instrumentation platform device-capabilities oriented view, the IEEE 1451.4 standard extends the software representation model for a 1451.4 Transducer-Block based on a XML defined schema. The XSD schema provides the functional capabilities for a Transducer Object model. The notion of the 1451.4 standard is based on migrating NCAP component functionality to an intelligent data acquisition device. Note, the software object model representation includes: a defined transducer interface model, TEDS service, along with the MMI interface representation: for on-the-wire communication. As well, an XML device description based schema is available.

In terms of the OMG based sensor-interface, a separation exists between the location of the transducer description and the sensor-interface system descriptions from the onboard transducer. This standard does not propose an object oriented component model such as the NCAP network controller architecture, however software-application specific interfaces are defined for accessibility to a sensor-controller's internal File-system. It is this IFS which resembles the object model, providing details for sensor-metadata representations, transducer interfaces, and node properties. The file system provides a structured representation for a smart-transducer system, in the form of STDs. The Smart-Transducer Descriptions (STD) are based on capabilities defining the: Processor, Node, Protocol, Node service, which encapsulate the functional details of the Smart transducer node in a smart-transducer system.

The Processor and Node types are similar to that of the 1451 Meta-TEDS /Basic TEDS whereby vendor specific information, addressing, and serial numbers are characterized for micro-controller

based smart-transducer systems. Further, the Protocol encapsulates the details of the sensor node interconnection fieldbus (TTP/A), similar to that of 1451's Channel TEDS. As well, the Node service provides details of the sensor's metadata and local data types, as is found in Channel TEDS, and Calibration TEDS. Both standards provide coinciding device capability details however it should be clear that 1451 directly associates the sensor's capabilities within an onboard TEDS fashion, whereas the OMG sensor interface directly associates the sensor's smart-transducer Interface controllers capabilities onboard a node.

System-Wide Instrumentation Platform Solutions

In terms of enterprise-system solutions, such as the SensorNET project, device capabilities and system capabilities are an important characteristic. Currently, an expansion to the SensorNET project entails the integration process of adopting 1451 TEDS (static characteristics specific to transducer hardware) with [SensorML] (sensor platform characteristics based on functional models) as part of the Open Geo Spatial Consortium's Sensor Web Enablement [SWE05]) initiative. Additionally, the integration of [TransducerML] (Capturing/Streaming sensor data) is in consideration, as per the SWE initiative. These solutions can provide the means for providing self-describing: platforms, sensor-data, and transducers facilitating ease in device configuration.

Communication Solutions based on Descriptive Metadata

From the solutions presented within the service-oriented architecture domains, namely [DPPC03], the device description capabilities are integrated within SOAP based Document-formatted messages, meaning device description capabilities are passed between nodes. The web-services based communication protocol of SOAP provides for system interoperable and self-describing messaging capabilities. There is a similar self-describing messaging technique employed in [SensorNet] based on the Web Feature Services (WFS). The WFS service defines description of sensor data, and allows for device query invocations based on sensor-metadata (time, location). This means the sensory-data and platform node information is captured in a contextual format. Further, the [JDDAC] project provides an XML based messaging description of sensor metadata, via NRSS. This form of messaging retains and encapsulates the overall 1451.1 network communication message structure in an XML based document. Typical metadata associated with sensory data is categorized as: time-stamped, name-value data-pair tuples, as presented within the NRSS messaging solution for [JDDAC].

It should be noted that there is a need for self-describing and platform-description capability, embedded within the application layer messaging techniques for instrumentation platforms. Advantages for remote data retrieval at the application layer preserve the contextual information characterizing a system (sensor network controller) and its physical data (sensor data). Traditionally, the remote messaging structure is based on a lower-level data-transport syntax, and lower-level representation of sensor data. This is apparent in fieldbus interfaces and 1451 sensor interface communication. This means an end user or software developer require an intimate knowledge of the underlying supported sensor interface technologies. It is this shift in providing interoperable XML based communications which provide machine-readable, human-readable formats, driving the web-services communication paradigm into new application areas.

3.6 Summary of Instrumentation Middleware Solutions Presented

The following table outlines the characteristics of large-scale research projects encompassing the IEEE 1451 standard. The projects presented in Table 1, are summarized by the following descriptors: type of platform/middleware required for hosting a 1451 solution, the type of web-service accessibility supported by the middleware, the required sensor-interfaces, along with the available device descriptions for the instrumentation middleware platform and its inherent transducers.

Summary of Instrumentation Middleware – SOA (Large-Scale) based on 1451

Project	Platform Middleware	Web Service Accessibility	Sensor Interfaces	Device Descriptions
SensorNet GOV (Part of the OGC SWE Sensor-web enablement)	Web-Server + 1451.2 STIM Software Wrappers	WFS Document Oriented Message XML/HTTP CAP (Common alerting protocol)	Based on 1451.x sensor interfaces Sensor Collection Service (SCS) Nodes	Web Feature Service WFS (Query/Feeds) TEDS and SensorML
JDDAC/Netbeams 1451.1 JMDI Java.net project	J2EE Measurement Web Server Agilent 1451.1 NCAP	Web enabled Client-side NRSS sensor feeds	JTI encapsulate the IEEE 1451.2 Sensors P2P JXTA transport layer	JMDI XML Based Configurations NRSS sensor metadata

Table 1: SOA Large-scale Instrumentation Middleware Solution Summary

Additionally, the smaller scale research projects presented within this literary review follows. Comparisons between related research works are quickly assessed via reference to Table 1 and Table 2 outlining instrumentation middleware platforms.

Further, Table 2 outlines the characteristics of instrumentation middleware solutions, based on the type of middleware-platform, web-enabled or web-service accessibility, the sensor-interface utilized along with the instrumentation platforms device-description capabilities.

Summary of Instrumentation Middleware Platforms (Smaller-Scale)

Topic	Middleware Platform	Web Service Accessibility	Sensor Interfaces	Device Descriptions
Lee-Song: IEEE 1451.1 Object Oriented Application Framework Model	1451.1 OO Tightly-coupled distributed environment ACE Based Middleware	Web enabled TCP/IP based connectivity	Based on 1451.x sensor interfaces	1451 TEDS UML Object Model description NCAP Software
Schneeman–Lee: 1451 Distributed Measurement Control	Tightly coupled distributed environment	Web enabled Client-side Java Applet TCP/IP gateway	1451.2 Interface to sensor Ethernet Controlled	Based on 1451 TEDS and HMI User Interface, Applet
Venzke-Pitzek: OMG Smart Transducer Interface Descriptions Merino: Web service interface and architecture for Fieldbus system	OMG Smart Transducer Tightly-coupled CORBA gateway Client-Server environment Propose Web-Service SOAP Gateway	WSDL Client-interfaces: Real-time service interface, Diagnostic-maintenance, Configuration-Planning SOAP Processor	IFS Inter-file system mapping inside STD Connected to TTP/A Fieldbus network (time-triggered protocol/A)	Smart Transducer Descriptions XML Schema based IFS Inter file system sensor representation CCD Cluster capability descriptions
Viegas: 1451.1 using Commercial framework	Commercial .NET 1451.1 NCAP implementation ASP.NET based IIS Server	ASP.NET hosted solutions for accessing NCAP ASMX interface, SOAP, XML/HTTP	IVI Foundation Comm. API for NCAP Transducer Block	IVI Foundation Class files for virtual instrument
Delicato: SOA Environment for WSNs	Apache Axis SOA framework WSDL,SOAP Interface for a WSN	Web-service between Source nodes-Sink nodes WSDL, SOAP, TCP/IP	None design requires SOAP binding to Data-dissemination protocol	Document oriented SOAP Sensor Capabilities XML Schema
Sadok: 1451.1 WebNCAP Server WebNCAP Client Servlet/JSP	J2EE 1451.1 based implementation using JAX-RPC Sun Java (SJAS) Application Server	J2EE-SJAS hosted solution for NCAP F-Block & T-Block access via Servlet WSDL, SOAP	Java Comm. API library for Sensor Serial Interface for NCAP Transducer Block	WSDL NCAP description WX-Dux-2 onboard Sensor Description

Table 2: Small-scale Instrumentation Middleware Solution Summary

Chapter 4

Research Problem Revisited

A current problem in the instrumentation domain arises with the interconnection between transducers. Currently, there exist numerous proprietary protocols for inter-connecting transducers and connecting to network controller endpoints. It is quite common, in practice, to interconnect each device via a fieldbus technology. However, there exists a multitude of fieldbus technologies which vary in protocol communication, setup, and contain proprietary sensor interfaces.

Furthermore, the interconnection problem is prevalent in the event or scenario for interconnecting a variety of transducers based on heterogeneous types/communication protocol method. A solution is required for a loosely coupled system environment. (This is common amongst networked communication whereby bridges and gateways are employed in order to translate between differing communication protocols and protocol layers.)

The IEEE 1451 working groups have proposed a solution for interconnecting various transducers via a standardized interface for communication and interconnection of transducers to NCAP network controllers. This interconnection between an NCAP to transducers and inter-NCAP communications is based on a tightly coupled environment. This means dynamic interface generation and communication capabilities for inter-NCAP communications cannot be supported by the current IEEE 1451.1 standard, due to the tightly coupled environment. Further, the 1451.1 client-server development must mirror each other's interfaces. Developers must know, prior to interface development, the size of 1451 sensor-data message payloads, 1451 TEDS data etc. Consequently, issues arise in the case of an NCAP's service/device discovery, establishing, and maintaining communication between heterogeneous NCAP network controllers.

The purpose of this thesis is to demonstrate the suitability for developing and integrating a web-services middleware framework, to a 1451 transducer/sensor-network controller, as defined via the IEEE 1451.1 standard.

A web-service(s) NCAP prototype can provide the following advantages:

1. Communication based in a loosely coupled environment.
2. Self-describing service and device capabilities to facilitate service discovery.
3. Dynamic interface generation to facilitate inter-NCAP communication capabilities, based on a web-service(s) middleware platform, for NCAP Software development.

Research Justification - Synergy between 1451.1 and Web-Service(s)

Both IEEE 1451 and W3C's Web-Services are Emerging technologies respectively within their fields. The integration between two widely adopted standards, for sensor interface technology and Internet related technologies provides a two-fold solution for creating interoperable, service-oriented sensor-network controllers.

By integrating both of these technologies, the end result yields an interoperable, standardized, emerging technology platform. The 1451.1 model provides an interoperable object model for design of network controller application processors. The standardized 1451.1 neutral model allows engineers and software developers to build application specific NCAPs, whereby the neutral model is extended in order to fit an application's need. The interoperability evident within the 1451 standard is characterized via its sensor interface standards. Transducers can communicate across an interoperable bus governed via the 1451 sensor interface standards. Moreover, by integrating the web-services based Simple Object Access Protocol (SOAP) communication layer to the 1451.1 standard's network communication domain, one can achieve an interoperable communication mechanism, whereby SOAP's XML based communication mechanism is transport independent, and can be bound to any underlying transport layer.

Additionally, both technologies are standardized, the IEEE 1451 contains four current published sensor standards as of 2005: 1451.1 Network Capable Application Processor Information Model, 1451.2 Digital Smart Sensor Interface Standard, 1451.3 Multi-drop Sensor bus standard, and the 1451.4 Mixed-mode Sensor interface. Comparably, the W3C web-services genre is built on standardized service-oriented layers, namely W3C'S WSDL 1.2 Web Service Description Language, SOAP 1.2 Simple Object Access Protocol, along with a service registry UDDI Universal Description Device and Integration.

Further, the 1451 standard and W3C's web-services are referred to as emerging technologies, within their respective domains. Specifically, in the field of instrumentation systems and sensor networks: the adoption of web-services as applied to instrumentation provides unique alternatives for data dissemination, application integration, and sensor network management.

Chapter 5

5.1 Analysis of IEEE 1451.1 and a Web Service(s) NCAP

The following section will provide a technical analysis of an IEEE 1451.1 Application-specific NCAP, in terms of the software model/environment model, messaging capabilities which encompass network communications, and provides details for the integration of web-services. Moreover, the complete integration for an application specific NCAP model is realized throughout this Chapter by applying a web-service interface and software-wrapper to an existing application specific NCAP software model.

Analysis of the 1451.1 Software Object Model

The feasibility of integrating two technologies: IEEE 1451 and Web-services is presented. The main concept lies in the services offered via both technologies, and transforming the NCAP software application into service functionalities via SOA principles. The NCAP software model is designed in a tightly coupled manner, in which the server applications and client applications are uniformly integrated. Within an ad-hoc environment, it is very challenging to establish communication as a client to an NCAP server, based on the traditional concept of IEEE 1451.1 NCAP architecture.

However, it should be noted that the traditional architecture of an NCAP's 1451.1 Software object-model is directed at tightly coupled environments such as the industrial factory-floor setup. Such a domain, suggests that controllers are designed, and implemented for a specific, redundant application. In this case, the infrastructure does not change, and remains static for purposes of monitoring and control. From a software development perspective, this means that both client and server software functionality is written concurrently. A client side application will mirror the server side application in terms of data-message parameters, data-message size, and extended functionality method/operation calls.

Both client and server, in this fashion, must agree to the semantics outlining available method calls. This is the case with an NCAP's client-server communication, where a client will call a remote procedure (Get TED, specific via 1451.1 IO Read method, translated into an operation OpCode: 6162) on a remote server. This style of representation requires that the Client knows in advance, the software interfaces and available methods hosted by an NCAP server. It is also challenging for remote client interaction

when interfacing with extended or application specific methods as defined within the NCAP's Function block. In this case, non-standardized OpCodes define the extended application specific methods. This application environment can constitute a semi-homogenous architecture, whereby third-party client's lack appropriate interface details. Note, however the overall 1451.1 object-oriented architecture of an NCAP provides a heterogeneous system in theory, whereby software developers can extend and implement varying NCAPs over varying platforms and development environments. However, it is the integration process for interconnecting varying application specific NCAPs which poses a challenge to a developer.

In terms of remote client integration, NCAP clients cannot be easily generated "on-the-fly", since the 1451.1 software programming model is tightly coupled. From a communication services perspective, client-server message encapsulation/de-capsulation and marshalling/de-marshalling techniques are applied as governed via the 1451.1 standard. (These techniques are required for processing a sensor-parameter's data model object into an argument array message structure, which is sent over the network, along with TEDS data parameters and NCAP discovery announcements.) The standard introduces a uniform message encapsulation technique via the Argument Array structure.

This means that prior knowledge of the message format and message size (from the NCAP's server) must be known to a client-developer in order to process the data. Likewise, the publish-subscribe communication model also contains this similar challenge of data encapsulation. Hence, the remote client, must explicitly define the message size parameter within a subscriber interface, to facilitate publish-subscribe capabilities.

It should be emphasized that the 1451.1 Object model provides an abstract representation of the NCAP model, whereby Client-Server and Publish-Subscribe methodologies may conform to any underlying network protocol. This means that the 1451.1 on the wire message, requires a mapping and binding to a suitable network transport: TCP/IP, Ethernet, Bluetooth, WiFi etc. The challenge arises when integrating between the 1451.1 Network API's (Specific API to a transport layer) and the underlying 1451.1 data representation.

Moreover, in dynamic environments such as in critical response, and critical infrastructure monitoring, there is a requirement to provide on-the-fly connections to network controllers. The client-interactions are facilitated by a service-oriented approach based on a loosely coupled interaction over an Internet connection. A loosely coupled interaction means the use of one-way operations similar to that of web browsing, whereby sessions remain stateless, and clients invoke content data in the form of web pages via HTTP. In order to facilitate client interactions, a well-formed service interface is required. This is realized via a WSDL based service specification for a Web services NCAP system. A WSDL file contains the abstract service requirements for an NCAP system, along with a concrete implementation of the service communication bindings. With this WSDL XML document, (which is hosted by the developed Web service NCAP's SOA registry accessible in the form of a URI), remote clients can connect and generate client-side interfaces (with an IDE) based on the service specifications outlined in the WSDL document. The task of generating client-side interfaces via a WSDL document will shorten the development cycle and time of 1451.1 applications.

(A similar concept is presented in [LS03] in terms of the development of the 1451.1 Object Oriented Neutral model, which results in higher reuse of software and the availability of network-neutral 1451.1 models.) Note, the integration and development of an application-specific NCAP is no trivial task, due to the complexity of the 1451.1 object model.

"TED is to Sensor – As WSDL is to NCAP"

It should be noted that the current design of the IEEE 1451.1 NCAP Model, does not contain a clear service-description specification, in the form of explicitly published and advertised service details related to an NCAP. The role of the NCAP, as a network controller, can apply to a variety of environments and scenarios, and through a WSDL based service specification, a remote client can be generated in order to connect and interact with web-services based applications.

The notion for introducing the WSDL based service specification (in order to characterize the available operations and communication/connection-capabilities in an NCAP) stems from the initial TEDS concept, adopted in the IEEE 1451 standard. A sensor relies on a small fingerprint of self-describing capabilities data housed in a Transducer Electronic Datasheet onboard a sensor's EEPROM memory. Due to the memory constraints imposed by a sensor's onboard processing capabilities, a TED contains compact device description information, in a binary-format of 640 bytes from [IEEE1451.4]. Since the

NCAP controllers, are considered to be memory and resource rich devices, the XML based WSDL service specification is feasible for self-description capabilities for the NCAP. Further, by applying a WSDL based service representation, service discovery is possible for the NCAP within a service oriented architecture environment. Since the service, connection parameters (based on a URI) are well defined within the NCAP's WSDL service document.

5.2 Web-Service(s) for IEEE 1451 Standards and Application Specific NCAP

1451.1 Network Communication vs. W3C SOAP Communication

By applying the concepts of web-service technologies, namely the WSDL service based descriptions for an NCAP's functionality, and the underlying SOAP based communication, numerous enhancements can be added to the current 1451 Standards, in order to extend the functionality of the standard. Based on the 1451.1 NCAP model, development work was put into place by Agilent Technologies, for "Mapping 1451.1 to TCP/IP" of on the wire communications for IP. Note, the client-server communication interaction is based on pre-defined TCP ports, facilitated by socket connections. In a Java environment high-level API's, can be used for creating and establishing the socket connections between clients to the NCAP Server. Further, the publish-subscribe capabilities utilize the multicast IP protocol. Each of these options provides some drawbacks for a network-capable NCAP when compared to the web-services enhancement. The drawbacks of a traditional 1451.1 Application specific NCAP's network communications are discussed in detail below.

For Client-Server, when using predefined TCP ports in this manner, the NCAP system manages access and can delegate access only on the specific port. Advertisement of the port numbers to remote systems is not guaranteed, and for generating remote clients, such connection details are required in advance to interact with the NCAP system.

For Publish-Subscribe, when using a Multicast IP for sensor-data dissemination, network-topology issues arise. For instance, an NCAP, which publishes data on an internal multicast address, remains restricted to a local area network's topology, unless multicast routers are supported for external network communications. For clients located outside the subnet-network, connection integration capabilities are challenging due to network management and administrative policies (i.e. firewall) outlined for the subnet.

In order to circumvent these communication drawbacks, the integration of SOAP based processors for handling communication details can be attained for providing remote-network connectivity, between clients (HMI's) and other remote NCAPs. The advantage of employing the SOAP based communication protocol is in the binding to the HTTP application layer transport. The transport can traverse firewalls, and can provide remote access based on an Internet network topology. It should be noted, that the nature of the SOAP protocol/processor relies on an HTTP listener, which delegates the lower layer transport layer TCP ports for connection capability onboard the server.

In addition to the 1451 TCP/IP on the wire communications mapping, the 1451 standard's active work is being conducted within the wireless sensor networking interface standards. The communication interface consists of a wireless protocol link, driving the introduction of a physical layer based TED; by the 1451.0 working group. The physical layer provides a standardized and common communication protocol amongst standards.

1451.1 Distributed Architecture vs. Web Service(s) Architecture

The challenge in this work is to integrate a distributed system model into a web-service environment. Both models differ from a software-implementation perspective. The IEEE 1451.1 Object oriented model relies on object-based components, Object dispatch addresses (Server's network address) along with an object ID, objectTag for object-maintenance capabilities and object representation. Whereby, the distributed objects are exchanged between the client-server entities in 1451's RPC or publish-subscribe communication modalities. A web-services approach views the underlying application specific NCAP Application model as a legacy application. (Legacy, in the sense that remote capabilities are not easily defined to a remote user.) The web-service acts as a software-interface defined by a software wrapper for accessing local methods resident in the NCAP legacy application. The wrapper service, in turn, abstracts the details of the NCAP legacy application software's functionality.

Summary of a Software Wrapper Service from [Erl04]

Software-Wrapper Service	Typical Service Characteristics
Anticipated Usage Volume	Low to medium, the wrapper services generally provide generic interfaces to a subset of legacy application functionality.
Interface design characteristics	Legacy functions are represented by service interfaces, streamline for generic usage.
Message Types	RPC-centric or Document-centric
Deployment Requirements	Varies, depending on the connector technology required by legacy application

Table 3: Software Wrapper Service Characteristics

A web-service's implementation resides inside the software wrapper, and implements business-logic; code which is responsible for accessing internal NCAP application methods. (The NCAP application blocks are instantiated, and object calls are made to legacy-defined methods for remote accessibility via the business-logic functionality.) In another form, the business logic functionality can be viewed as an adapter-connected to the 1451 NCAP Application.

As a deployment requirement for utilizing a SOAP communication processor, a proxy service is required. A SOAP proxy service handles request and response messages between client-server entities. Generally, a SOAP proxy resides alongside the corresponding legacy application, within the native application's hosting environment. We would like to migrate the native application-hosting environment onto an application-server. (This means, an NCAP application can reside on its own, with the added support of a SOAP stack. However, in this research work we are moving the NCAP application directly onboard the Sun Java (SJAS) application server.) In this manner, the NCAP is now an application, which executes inside the Application Server environment.

1451.1 RPC and Publish-Subscribe of Application Specific NCAP

The 1451.1 communication services are based on the Distributed Object Models, whereby distributed objects are passed between systems, and executed in an RPC's server-side perform and client-side execute operations. The RPC procedure is fulfilled by marshalling the operand content of a required method or data into a 1451.1 Argument Array data structure. The Argument array is serialized into an on the wire format packet, and at the recipient the packet is deserialized and demarshalled. Common uses of RPC calls include calling a standard method on an NCAP, predefined by 1451.1 standardized OpCodes. The publish-subscribe capabilities operate in a similar fashion, however the data passed between recipients includes sensor-data readings, packaged in an Argument array. The argument array encapsulates a data-representation for the raw sensor-data.

Web service(s) NCAP Communication based on a SOAP RPC Messaging Layer

The SOAP protocol may be used for accessing a method defined in a legacy application via a JAX-RPC middleware interface. In order to integrate the SOAP services into the 1451.1 OO model, it is a matter of providing a JAX-RPC SOAP stack. The SOAP stack provides a proxy-service in order to dispatch and control SOAP message requests and responses, and interpret (via parsing) the contents of a SOAP message. As a system-integration process for migrating SOAP based communications to the 1451.1 OO

Model, there are two solutions available. Namely, the SOAP stack/processor can replace the existing Client Ports, and Subscriber-Publisher Ports of the 1451.1 Communication Services model. However, a data-type binding representation for the 1451.1 Data model format should be retained. This means, a SOAP message payload would conform to the Argument-Array data-structure for passing contents (Objects, sensor-data, method calls) between client-server entities. Alternatively, the second solution entails the addition of a SOAP stack/processor as part of the 1451.1 Communication services is feasible, which would offer an alternative method for (loosely coupled) accessibility to an NCAP. In this case, the 1451.1 standard's communication services (Client, publisher, subscriber ports) are retained.

5.3 "WebNCAP" Prototype Architecture

Application Specific NCAP Model

During the period of this research work, a developmental Application Specific IEEE 1451.1 NCAP model was released by the National Institutes of Standards and Technology (NIST). The IEEE 1451 working group, under the direction of NIST provides examples of 1451.1 Standard software implementations, via the [Open1451] group. The software models provide examples of the Network-Neutral 1451.1 Model (Java), with a UML based representation, along with a variety of NCAP example models.

NCAP Design Consideration – Application Process

The NCAP used throughout the development phase of this research project, entails a Java application specific NCAP for control of a Wx-Dux V-1.0 multi sensor. The application specific model was acquired from the Open1451 group, and permission was given for its use throughout this research work by K.B Lee and E.Y Song. The application specific model provides the base framework for an operational NCAP.

One of the first objectives in this research work is the design and migration of an Application specific Web-based NCAP. That is the NCAP's functionality would reside and be hosted by an application web-server. The concept of web-services, when applied to this Web-based NCAP, would allow for a quicker software-design, based on the WSDL interface, and SOAP based communication enhancements.

The Open 1451 NCAP application acquired is a standard stand-alone Java application. It provides a PC-NCAP (server) implementation for interfacing to a Wx-Dux V-1.0 multi-sensor, along with a Java application PDA-NCAP client program. This NCAP provides the ability to study and examine the software-development details, as governed by the IEEE 1451.1 Object Oriented Software model standard.

Furthermore, the available IEEE 1451.1 (2000) publication provides the description for a software developed OO model, which is network neutral. The design considerations for this research project require the stand-alone Java 1451.1 NCAP application to be migrated into a Web-based environment. This means, that the aforementioned open 1451 NCAP application would reside within a web-server. Currently the IEEE 1451.1 group is compiling an updated P1451.1 standard that proposes a Java implementation of the 1451 NCAP with web-server related functionality. The current 1451.1 published standard provides a programming language agnostic, IDL representation for operations described by the 1451.1 Object-oriented model.

It should also be noted, that the work of Lee and Song's NCAP Application specific model (from the Open 1451 project) extends the current 1451.1 standard by providing: network implementation details for a TCP/IP based network-transport layer binding. The binding provides on-the-wire data-marshalling techniques, along with a demonstrated client-server and publish-subscribe scenario. We speculate the details of the open1451 application specific Java model will govern the revised P1451.1 requirements standard.

Further, by migrating the open 1451 NCAP application to a web-server environment two development and testing scenarios are explored:

By retaining the previous Open 1451 NCAP application specific functionality, the web-server allows previous functionality of the NCAP to exist, meaning the design offers backwards-compatibility. (Note: Java based Access Control policies governed by the JDK version are modified in order to allow for network-access rights. For example, in order to establish a TCP Socket connection and open a TCP server port from a Java servlet.) It is feasible to host the previous functionality of the Open 1451 NCAP, inside the application-server, since all that is required is the underlying JVM for executing the application. Note, the Open 1451 NCAP resides in a web-container onboard the application server. As well, the application server manages the server policies and accessibility rights for the WebNCAP.

During the design phase of the Web-services NCAP, the following modifications were taken into account regarding the previous Open 1451 Java Application specific NCAP. An updated Wx-Dux V-2.0 multi-sensor was used within our model, meaning modifications to the NCAP application specific model were made, allowing for the previous NCAP model to operate with the sensor. The sensor provides a unique serial-based ASCII communication interface for issuing commands to the Wx-Dux V2.0 device, and sensory data is transmitted in a human-readable CSV format.

Further, modifications were made to the previous Open 1451 Java Application model that include:

- PC1451Transducer-Block interface details for receiving/acquiring data from the sensor unit.
- PC1451Transducer-Block interface details for marshalling the sensor data.
- PC Publisher Display-Blocks for Server based GUI-graph displays of sensor data.
- PDA Subscriber Display Package and PDA Subscriber Port (Client Interfaces).

Henceforth, the first design objective entails the migration of the original NCAP application (PC-NCAP) to a web application server and integrating a real-time sensor for testing purposes. This scenario allows for testing the traditional IEEE 1451.1 communication mechanism and validating that the NCAP can operate within an application server hosted environment.

The second design objective entails the development of a web-service extension to the Open 1451 Java application-specific NCAP model. This allows for a WSDL based service representation of the NCAP. Moreover, web-service interfaces are developed to extend and access application-specific functionality of the Open 1451 Java application; namely, access to the Transducer Block and the Function Block is created. Remote communication is facilitated by use of the SOAP protocol. A middleware platform based on J2EE is utilized, as part of the application server.

Further, the web-service design process is examined, and consists of two development options. One option is the creation of a web-service interface from the pre-existing NCAP application specific code, where a WSDL based interface is generated from a Java-application specific (business logic) interface implementation. The second option presents the notion of generating the Java application specific (business logic) interface implementation from a predefined WSDL document. In this manner, 1451.1 Function-blocks are created from a WSDL based representation, noting that (operational) application specific code is still required for the NCAP.

The NCAP application software model is resident in a web-container onboard the application server, with added web-service interface support. There are two deployment and interaction scenarios for the

web-services NCAP application model. It is envisioned in the first case that clients would connect via a URI and access a web-client servlet or JSP page in order to access the web-services offered by the local NCAP application server. (This is a deployment example for an internal web-service implementation of a client interface, which is local to the onboard server.)

In the second case, remote NCAP servers can connect via a URI to access web-services via client-side generated interfaces. (This is a deployment example for an external web-service implementation for remote server-server, or Business-to-Business (B2B), communication via SOAP over HTTP.) The latter case is the true defining functionality of web-services, whereby SOAP messages are exchanged between two server entities.

Next, the XWIF design process is presented for the development of a SOA based 1451.1 Java Application specific NCAP model.

XWIF for modelling service interfaces - SOA 10-Step Integration Process

The proposed solution encompasses the XML and Web Services integration framework (XWIF) methodology, as characterized within the works of Erl's Service Oriented Architecture text [Erl04]. By applying a web-services integration framework, to an IEEE 1451.1 NCAP software model, a web-services interface model is developed for a 1451.1 Application specific NCAP.

The first five steps of the XWIF model provide a definition for the application logic to be encapsulated via the SOA principles:

1. Choose a Service Model:

The service model required for integration of a web-service interface and a legacy NCAP application specific model is facilitated via the Software-Wrapper service model from [Erl04]. A software-wrapper service model encapsulates the details of an adapter connected to a legacy application. The adapter is required for interfacing with the legacy application, namely in terms of object instantiations, and method calls for the NCAP application. The adapter, and a software wrapper provide a Web-Service interface: Abstract service end-point interface methods are characterized by the wrapper, and an adapter characterizes the service implementation interface. The software wrapper is developed for the WebNCAP by utilizing the JAX-RPC J2EE API via Netbean's Integrated Development Environment.

2. Establish the scope of the service's business function:

The main service function facilitated by the service interface provides remote-accessibility to the legacy application specific NCAP's software functionality.

3. Identify known and potential service requestors:

Service requestors are characterized as: Client-side generated interfaces containing a presentation layer for client interaction onboard the server. For example: clients interacting with the web-service, may connect via a browser to a server-side dynamic HTML web page hosted via server-side servlets or JSP pages. Another form of service requestor is characterized as: server-side generated interfaces containing an interface for B2B scenarios, where two servers can communicate via M2M interfaces. Additionally, a web-service hosted locally on the server (Graphical-Visualization service) or remote software applications may constitute as a potential service requestors.

4. Identify required data bodies:

Data bodies are based on the SOAP RPC-literal formats, for encapsulating a programmatic interface within the SOAP message's payload. The SOAP RPC binding is used in order to invoke the NCAP legacy application's software methods for capturing sensor data. The data body consists of the sensor data categorized by an array of floats, or as a String. An abstract representation of the data bodies in the 1451.1 model, consists of the Argument Array data Structure.

Additionally, data bodies can also represent SOAP Document-literal formats, for transmission of TEDS and for reading and writing functionalities based on an XML TED representation. The TED data body in this format would consist of a TEDS Object, with defining parameters of the Basic TEDS features.

5. Explored application paths:

Two application path's can exist for the 1451.1 NCAP; namely, for interface accessibility. First, the addition of a SOAP processor to the web-service model acts as a direct interface to the legacy NCAP application offering a fine-grained interface. Secondly, abstract SOAP communication (coarse-grained interfaces) can be modelled to emulate the 1451.1 Communication services and conventions.

The next five steps of the XWIF model provide the design requirements for defining a service interface from the PC NCAP 1451 application logic.

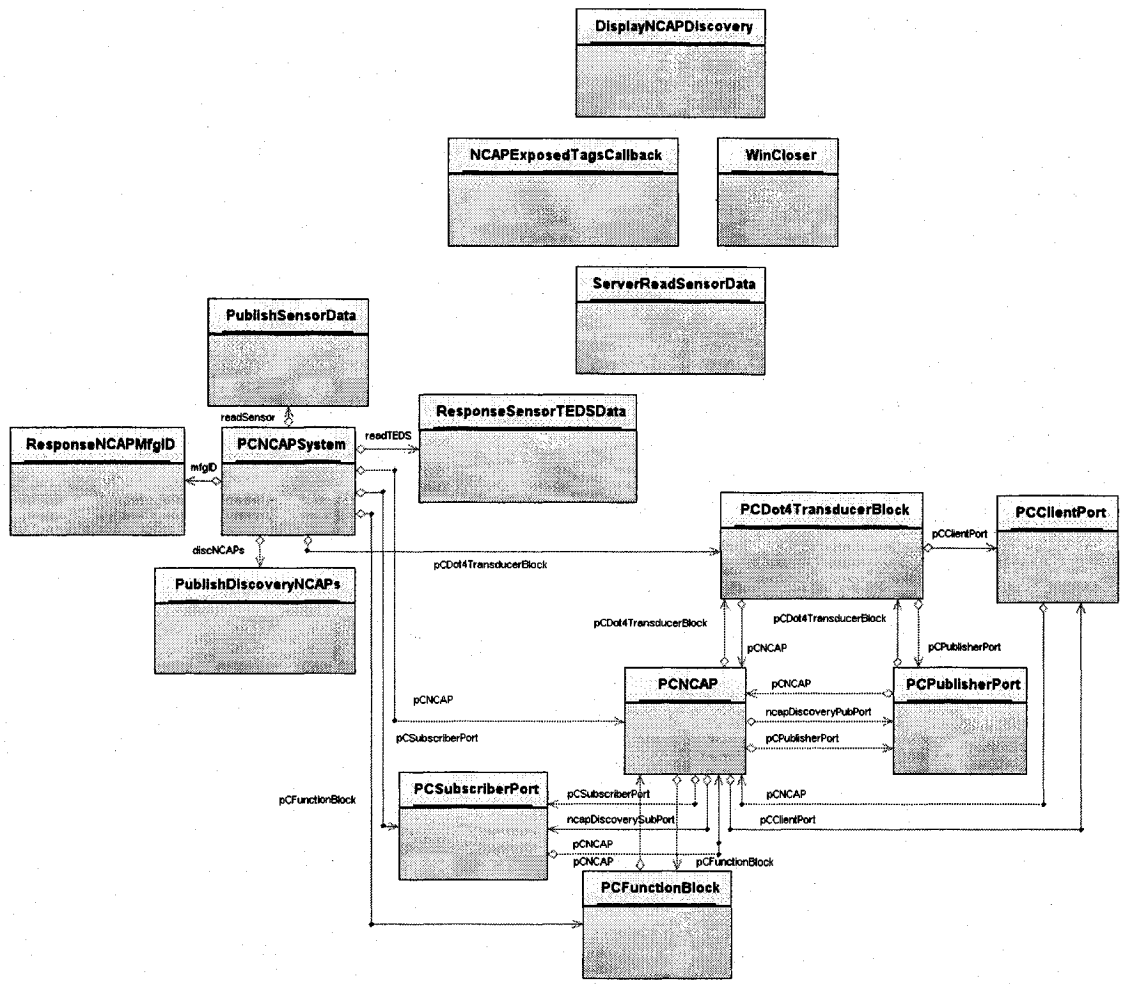


Figure 17: UML Based OO 1451.1 model for the 1451 PC NCAP Application

6. Define the encapsulation boundaries:

The service encapsulation boundaries are chosen by examining the 1451.1 application specific NCAP object oriented model, as illustrated in Figure 17. Specifically, the PC Dot 4 Transducer Block and PC Function Block will encapsulate the service details required for remote accessibility. The encapsulation boundary will provide the defining portType components for a WSDL Service representation, and will provide the service endpoint interface within the JAX-RPC J2EE architecture. A listing of the WSDL Service representation of the PC Dot 4 Transducer Block and PC Function Block is available in Appendix III.

7. Model the Service Interface:

The service interface details are modelled via the operations outlined in WSDL, namely, for the PC Transducer Block and PC Function Block. Remote accessibility is required within the PC NCAP Application's Transducer Block for providing a utility web-service which offers sensor Trigger capabilities. This is required for initiating communication between the PCNCAP and a sensor. The second service interface requires the accessibility to sensor data stored within the PC Transducer Block. Further, an interface is required for accessing a Web-TED located inside the PC Transducer block. The PC Function Block provides an interface for reading sensor-calibrated data, stored inside the PC Function Block.

8. Map out interaction scenarios:

The interaction scenarios for the defined service interfaces are based on the internal software hooks required for accessing sensory data, TEDS data, and triggering a sensor which are contained in the component classes of the 1451 NCAP model. Namely, an object instantiation is required for access to the PC Transducer block, along with an object instantiation of the PC Function block. Through this process, the methods defined within each object are made accessible to a remote service, within the web service implementation interface. (Illustrated in Figure 19) Further, for triggering the sensor (process required for initiating sensor data readouts from the sensor) a reliance on the PC Transducer Block, and the Server Read Sensor component class (Illustrated in Figure 18), are required. The Server Read Sensor component will provide the software-interface to the connected real-time sensor by way of the Java Communication API classes for serial-port communications. Further, the interaction scenarios XWIF process is required for mapping out the software component classes located inside the PC 1451 NCAP application, which aid in the web-service (business logic) implementation.

9. Design the Message structure:

The message payload structure is based on the SOAP RPC message, with a Literal format encoding for programmatic interfaces which encapsulates the operations and defined data payload. The SOAP RPC is one method for interfacing with legacy applications. Further a SOAP Document-Literal format can exist for the TEDS data messages. It should be noted that the message payload structure, associated with a WSDL files (service definitions) bindings component can be automatically generated via the JAX-RPC J2EE software.

10. Refine the Service Model:

The web-service interface refinement is applied to the Transducer and Function Block's, in terms of the overall SOAP based message format structure. For example, in order to provide self-describing data measurements, a document oriented SOAP message could be used. The message structure in this sense would consist of XML schema defined "complex-type" results. However, the complex type maps to an object representation of the sensor data in the Java service implementation. A String representation as proposed within the XWIF process model of Step 4, may be more suited for the sensor data, in terms of the SOAP processor serialization and deserialization (parsing of XML data) overhead. The overall large data-type and message structure can have a negative performance effect for web-service over-the-network communication due to the parsing of the XML data by SOAP processors at client-server entities.

By applying the concepts of the XWIF process model, a web-service interface and SOA design is possible for the PC 1451.1 NCAP Application. The overall architecture of a developed Web service(s) NCAP, termed as "WebNCAP" will follow.

5.4 "WebNCAP" Architecture Overview

The WebNCAP architecture is based on the PC 1451 Application specific NCAP model available from the [Open1451] group.

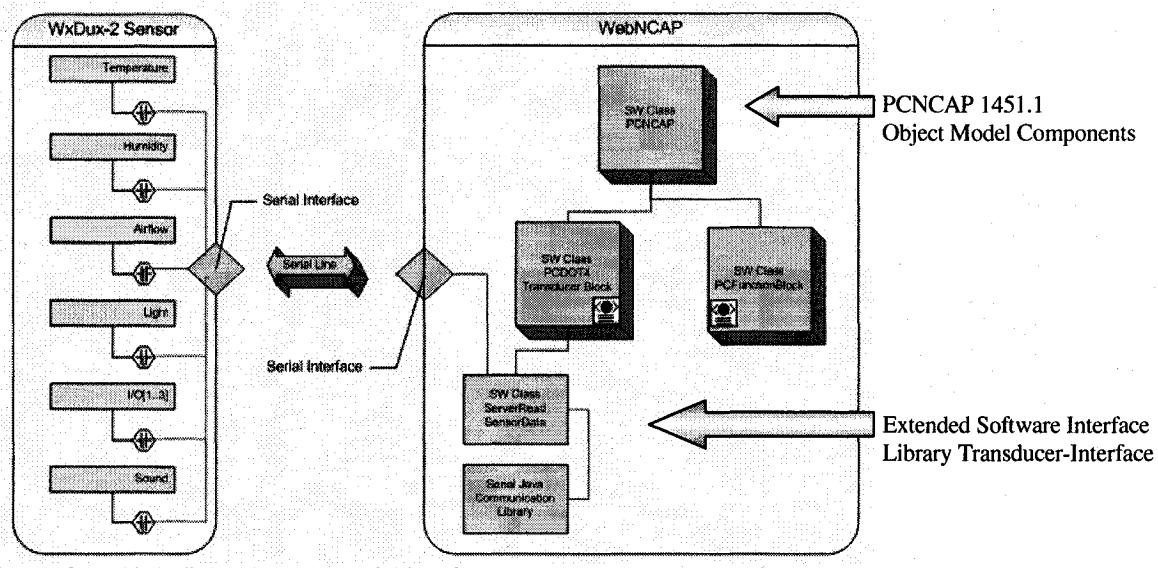


Figure 18: WebNCAP Architecture and Critical Components for System Interaction

In Figure 18, an overview of the WebNCAP's critical component objects are illustrated. Based on the previous XWIF design process, Step 8, the interaction pattern is based on this model. The encapsulation boundaries for the web-services offered are characterized via the service interfaces; namely, the PCDOT4Transducer Block and the PC Function Block. The sensor interface is realized via a Java serial Communications API, which provides a high-level software interface for communication to Serial Ports hosted by a personal computer. Furthermore, a Wx-Dux-2 multi-sensor is employed in this scenario to provide real-time physical-phenomena readings to the NCAP.

The Server Read Sensor Data component class abstracts the serial-port interface between the multi-sensor and the NCAP. Software operations, which are required for reading and writing parameters to the multi-sensor, are located within this component. Further, the Transducer Block relies on the Server read Sensor data component class, for interfacing with the sensor. Additionally, the first level of processing of sensor data occurs, directly inside the Transducer block. This is primarily where sensor data can be categorized via the 1451.1 parameter component classes providing physical description to the data within the 1451.1 application specific NCAP. Further, the second level of processing of the sensor data,

in terms of calibration, occurs at the PC Function Block. Based on the service interfaces defined in the XWIF process model, Step 7, remote-access to the functionality of the Function and Transducer blocks can be achieved. The corresponding UML Software object model diagrams (class/dependency diagram) for the PC Transducer Block and the PC Function Block web-service interface are available in Appendix IV.

"WebNCAP" Server Deployment & Client Interface

Based on the WebNCAP architecture and critical components aforementioned overview, a review of the WebNCAP server side deployment is required. The server sides "service interfaces" provide a service endpoint interface and service implementation for the WebNCAP as illustrated in Figure 19 below.

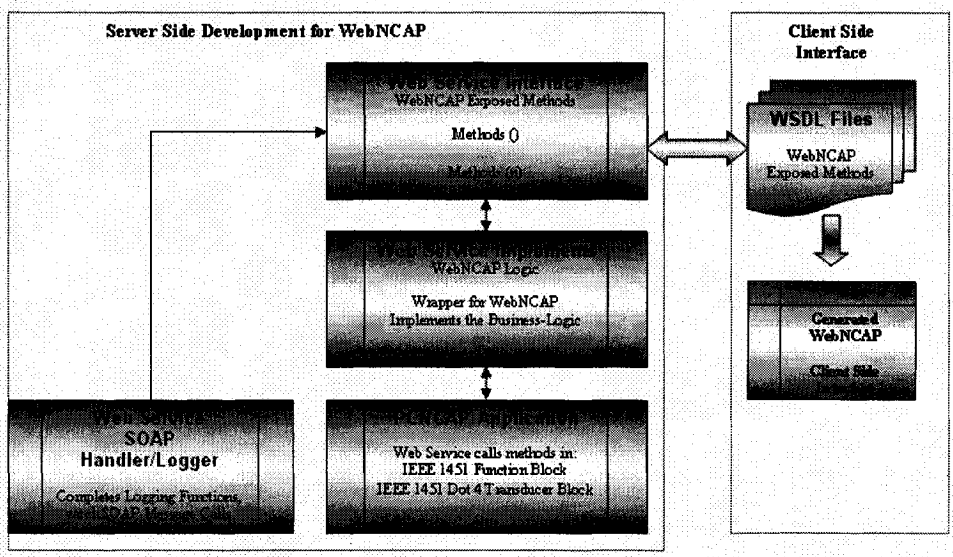


Figure 19: WebNCAP Server Deployment Components including Client Interface

The Server side deployment for a WebNCAP is characterized by the following functional-components: A Server Side Web Service Interface lists the Web NCAP abstract interface methods. The interface methods are characterized as the operations found in a WSDL file. These methods are the abstract representation of the services offered by a web-service and offer an entry point for remote accessibility. In order to access the abstract methods, a web Service implementation is enabled which contains the required business/integration logic, required for accessing/calling method inside a legacy PCNCAP application. In addition, a SOAP handler/logger is employed to provide logging capabilities for the

reception and transmission of SOAP messages, in the server. The SOAP logger is a vital component for the validation and performance results of the WebNCAP prototype.

Based on the server side deployment, a WSDL file characterizes the abstract web-service interface methods, and provides the underlying SOAP transport bindings, for facilitating network communication. It is noted, that the WSDL file is accessible for client-side interface generation as illustrated in Figure 19.

Furthermore, the WebNCAP server side application can be viewed as a SOA software wrapper service model. Whereby a software wrapper exposes the abstract interface methods (web-service interfaces) and encapsulates an adapter to interconnect to a legacy application. The adapter is modelled as the web-service implementation, which connects to the legacy application. The latter provides business logic which calls and instantiates the application side objects, of the 1451 NCAP model.

Web Service(s) Web Application Development to Deployment

A client's interface can be generated at run-time via the aid of a WSDL document, in order to generate client-side service operations, in the form of client-side stubs. The stubs are used as a software interface, at the client side, for invoking a desired service hosted via a remote server. This concept is based on the RPC methodology, under [JSR-109] utilizing the JAX-RPC middleware API provided by the J2EE framework.

The process for creating a WSDL based service specification to the client-side generation of service stubs is governed by the following:

By utilizing two widely adopted tools for service generation and service deployment, the web-service NCAP is compiled and deployed onto an application server. Essentially, the *wscompile* tool provides the service generation definitions for a WSDL based file, along with the required Java-XML based mappings utilized via an IDE. In addition, the *wscompile* tool packages the web service in a WAR web-archive file utilized by servlets. The WAR file is similar to a JAR file, providing necessary functional capabilities for running the web-based NCAP. (WAR file is an archive of the entire web-application developed within a given IDE which is deployable onboard an application server.)

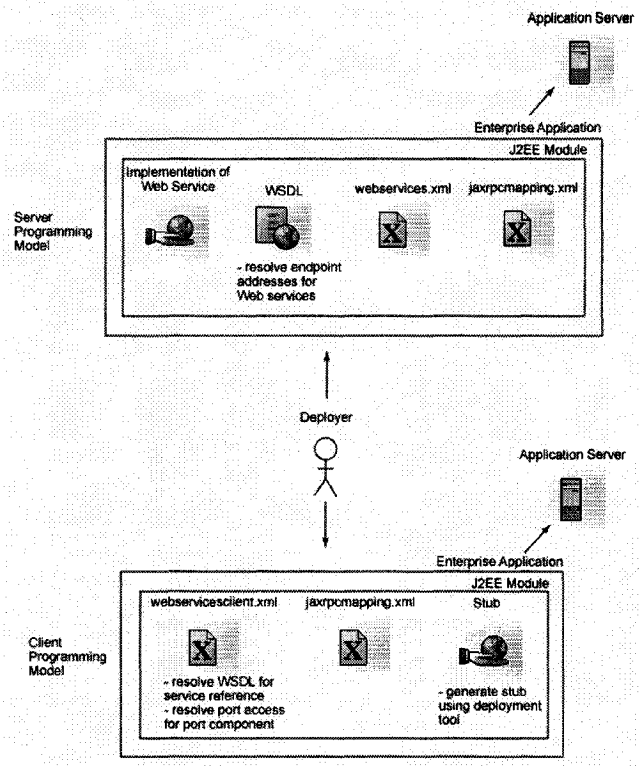


Figure 20: Client Side and Server Side Deployment Models [IBM03]

Additionally, in Figure 20, the client side and server side deployment scenarios are illustrated. From a client-side perspective, the J2EE *wscompile* tool provides the stub-generation for a service hosted via a remote server. The J2EE *wsdeploy* tool is used for deploying a web-service solution onboard an application server. (An application server creates an instance of the web-service project which is remotely accessible and enabled for client interactions.) Further, a WSDL mapping to Java is required at both client and server, which is facilitated by a generated XML based JAX-RPC configuration file. This file is generated by the *wscompile* java tool. The WSDL mapping to Java provides the generation of client-side stubs and remote methods.

Further, by utilizing the JSR-109 methodology within the WebNCAP design process, a J2EE generated stub is compiled for clients. This specification resembles a dynamic proxy client whereby, stubs are generated at runtime, meaning if a WSDL service definition changes over time, the end client can consistently connect to the web-service as compared to a static-stub based client side development option. It should be noted, that based on the WSDL service document, any form of client-side generation API's (such as JAX-RPC) may be used for varying IDEs and application server platforms.

System Integration Client IDE – Server Architecture IDE

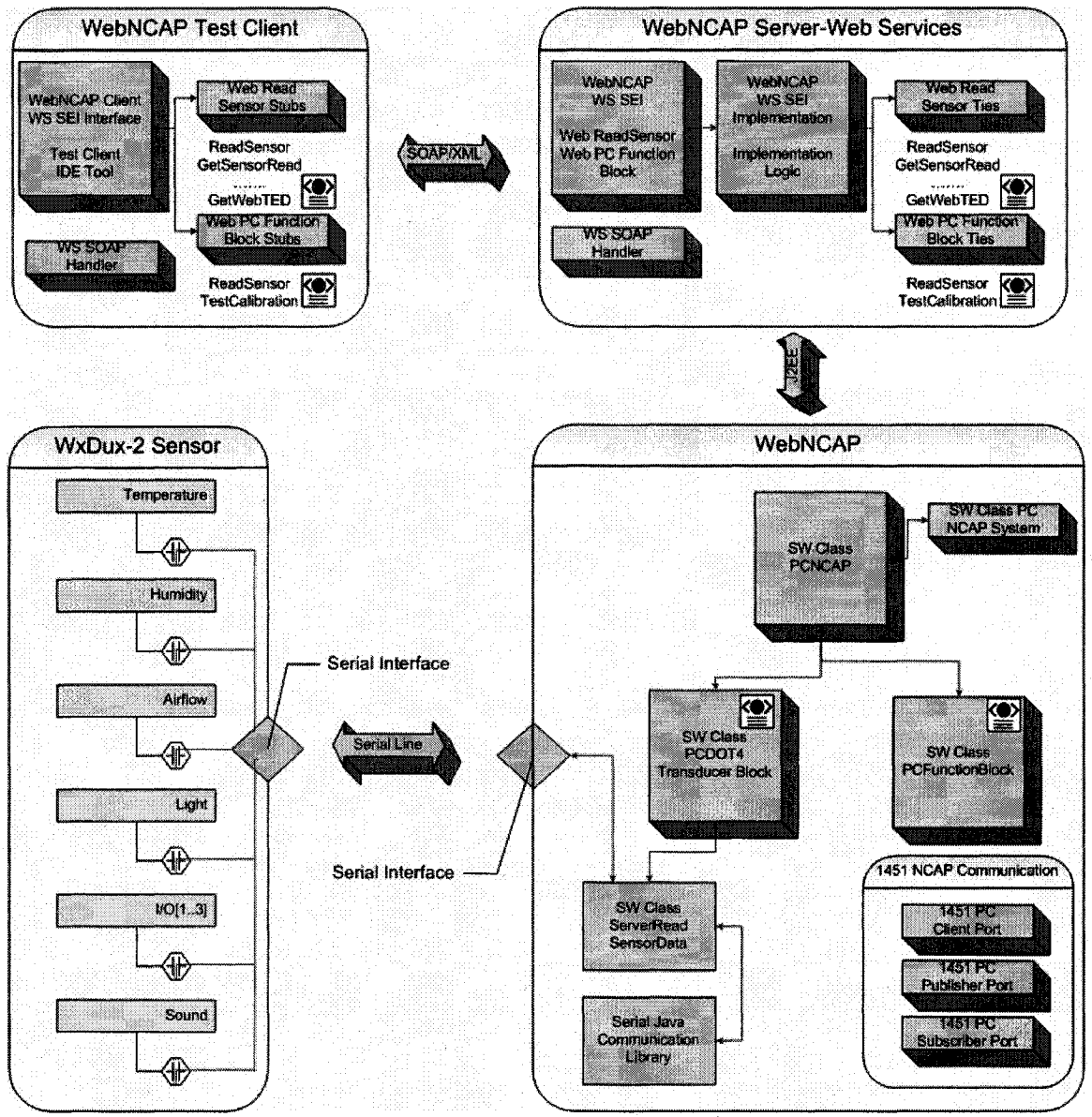


Figure 21: WebNCAP Prototype Architecture Environment

The following illustration depicts the Web-NCAP model which is comprised of the Web-NCAP application onboard the Sun Java Application Server V 8.0. The Netbeans 5.0 Developer IDE (integrated development environment) was utilized throughout the Web-service NCAP integration work. It should be noted that both the WebNCAP Server and WebNCAP Client are hosted via servlets, residing in a web-container onboard the application server for the illustration in Figure 21.

A Server-side Servlet hosts the web service endpoint interfaces (SEI) for the PC Dot4Transducer Block and the PCFunction Block via a WS_Servlet_WebReadSensor and WS_Servlet_WebPCFunction Block which call their respective Implementation classes. The implementation classes create an instantiation of the PC NCAP Application objects, required for initialization of the Objects inside the PC NCAP. Additionally, a Client interface is provided by the Netbeans IDE Test-Client tool for validating web-service functionality, further a Web NCAP Client is also realized which contains the generated server side service interfaces. The object model details of the developed servlets are available in the form of UML class/dependency diagrams in Appendix IV.

Furthermore, the WebNCAP application retains the communication services of the traditional 1451.1 NCAP, allowing for both traditional NCAP communications alongside with the web-service SOAP based messaging communication.

5.5 "WebNCAP" Communication - SOAP over the wire – HTTP Transport

The over the wire transport messages are communicated in the following manner. A client-side interface WebNCAPClient (servlet) hosting a dynamic HTML webpage provides a presentation layer format. A user can make requests by invoking a desired web-service operation. Upon an operation request, the servlet calls the web-service RPC inside its client-side servlet. At this level, the Web-tier is defined. The web-service call is made, the client's stubs are packaged, and the remote operation call is explicitly defined in a SOAP RPC message. In turn, the message is sent over the network using an HTTP POST to the server side web-service endpoint interface as defined by a URI. From the client's servlet, an HTTP Get Request is realized for requesting a web-service operation from the dynamic client HTML web page. (For example: Get WebNCAPClient Servlet, request parameters GetSensorRead).

At the server side's WebNCAP service interface (accessible via the URI), the SOAP request is unpackaged and execution of the RPC is made, for the desired operation. In turn, the operation response is packaged, and is sent back to the caller, client-recipient via the HTTP transport POST response. (For example POST for getsensorRead, web-TED) It should be noted that the HTTP GET requests are for the client-side dynamic HTML web page servlet interaction, and the POST request/response messages are for the server-server SOAP Message communication based on the web-service's service endpoint interface. The following SOAP over the wire and client dynamic HTML webpage servlet interaction is illustrated in Figure 22, and the corresponding client-side application screen-captures for web-service interaction are available in Appendix II.

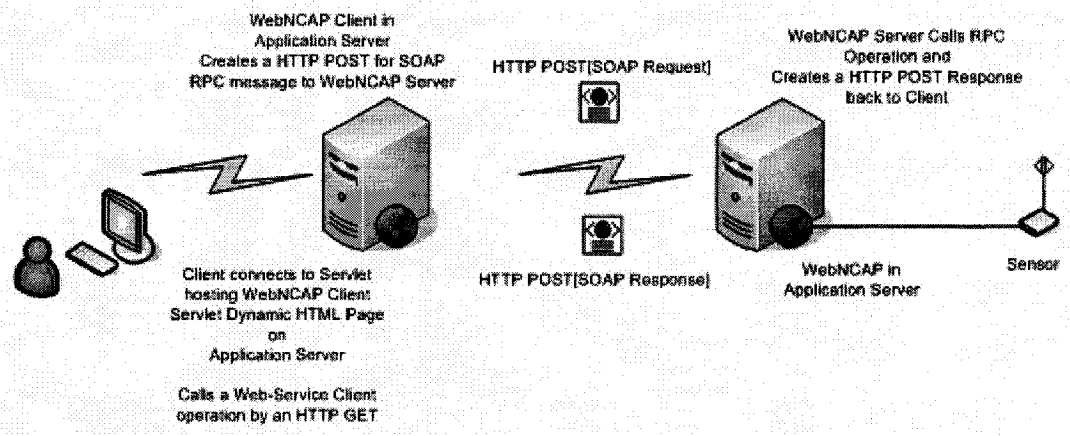


Figure 22: Client Servlet dynamic HTML Web Page Interaction and SOAP Message Interaction

The Web-service network communication is characterized as follows from a client-side servlet:

Network Communication at each layer of Network OSI Model for a Web NCAP Client Servlet Interaction:

SOAP	urn:webreadsensor\ ns: getSensorRead	
HTTP GET	WebReadSensorServlet Query String: ? getSensorRead	
TCP	8080	(Listener Port of the WebNCAP Client)
IP	137.122.91.93	(Address of the WebNCAP Client)

The POST message request-replies in the event of Server-Server SOAP message interaction are as follows:
Server A-A IP 137.122.91.31 <-> IP 137.122.91.31 {cmcslps8.site.uottawa.ca} 1-Server contains a *localhost* client
Server A-B IP 137.122.91.31 <-> IP 137.122.91.93 {cmcslps8.site.uottawa.ca to esadok.site.uottawa.ca} 2 servers

An Example below for a Client-Server interaction between the Servlet client-side interfaces to the interconnection of a remote WebNCAP Server SEI Service(s) Port is provided:

1. A request is made at the Servlet for sensor-data from the Wx-Dux-2 via the WebNCAP Client Servlet. Hence, at the presentation layer, the client requested the sensor-data dynamic-web page servlet operation. At the servlet, the web-service RPC Client-call is made, JSR-109 provides a dynamic port binding and invocation of the service request. A SOAP Proxy server handles this request, translates the web-service request into a SOAP message, and the underlying SOAP message is encapsulated into an HTTP POST getSensorRead call from the client interface. This is sent out onto the Network to a server hosting the web-service. (An HTTP Get is for browser interaction with a Dynamic HTML servlet.)

Below the SOAP handler-logger displays the request-SOAP message:

```
<env: Envelope xmlns:env=http://schemas.xmlsoap.org/soap/envelope/
      xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns0="urn:WebReadSensor/wsdl"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<env:Body><ns0:getsensorRead/></env:Body></env:Envelope>
```

2. Web NCAP Server's HTTP Proxy-listener port 8080 receives this request, instantiates a TCP Port to serve the request, and the SOAP Proxy instructs the JAX-RPC underlying method invocation to be made inside the WebNCAP Server. The server processes this request, and a response is sent back out onto the Network. What is interesting, here, is that the internal Web-service call at the server issues a POST response with the SOAP response data. This is because the SOAP message contains an RPC-message call and payload data. An HTTP GET response does not have the ability to store payload.

3. The Client-Side Servlet receives the response at the SOAP client handler-logger, and displays the output of the data:

```
<env: Envelope xmlns:env=http://schemas.xmlsoap.org/soap/envelope/
        xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns0="urn:WebReadSensor/wsdl"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<env:Body><ns0:getsensorReadResponse><result>27.47,81.44,13,1,80,99,99,99,77</result></ns0:getsensorReadResponse
></env:Body></env:Envelope>  <!--The Temperature Value requires Calibration-->
```

SOAP Listener for "WebNCAP" Client

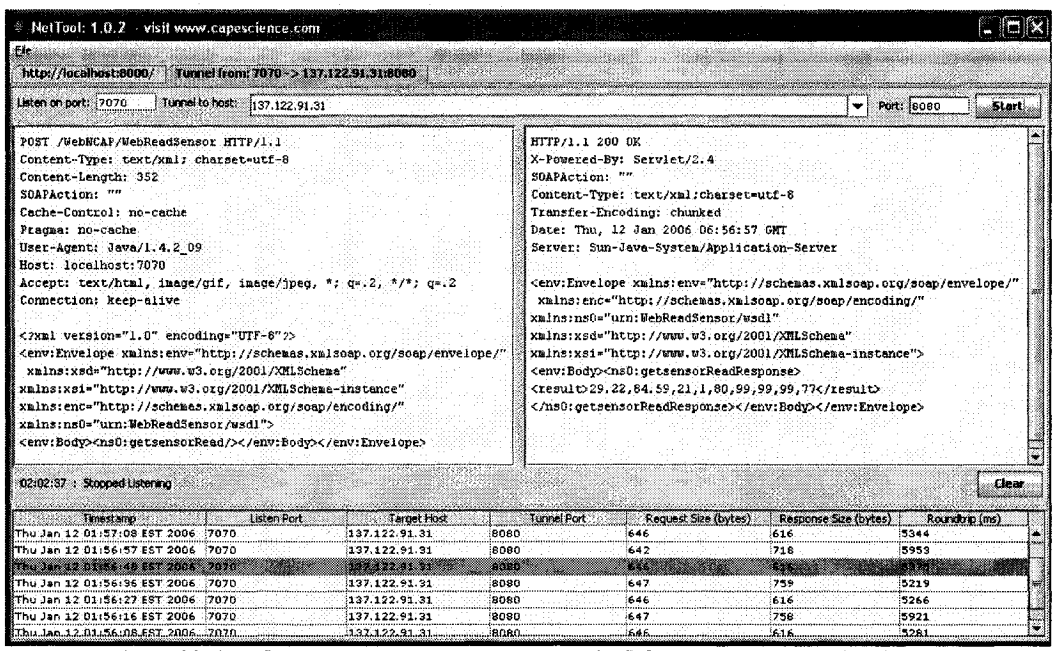


Figure 23: SOAP Request and Response Message for PC Transducer Block Service

Using a Network (TCP/IP Tunnel) tool from [NetTool] it is possible to capture SOAP traffic between a WebNCAP client and a remote WebNCAP server hosting the available Web-read sensor service. This web-service allows for communication to the PC Dot 4 Transducer Block and is illustrated along with the HTTP POST Request and Response header in Figure 23[2] as part of the WebNCAP prototype validation.

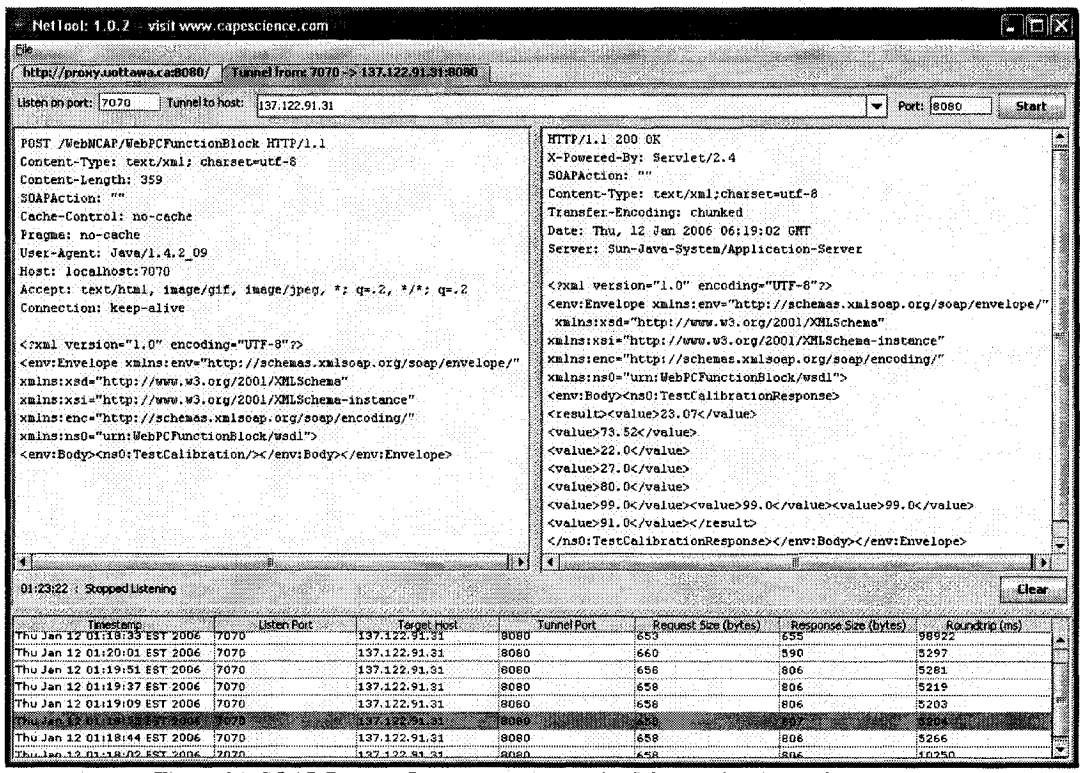


Figure 24: SOAP Request Response Message for PC Function Block Service

Further, the SOAP message interaction between a WebNCAP Client and a remote WebNCAP Server for a Test-Calibration web service request and response is illustrated in Figure 24[2]. This web service illustrates the concept of remote accessibility to a 1451.1 Function Block.

5.6 Revisit IEEE 1451.1 – A Web Service Extension for the 1451.1 Standard

Based on the development of a Web-services 1451.1 NCAP model, an extension is proposed to the IEEE standard. A formal definition regarding the IEEE 1451.1 NCAP Application Model (Component Model) is required for identifying suitable web-service system architectures. The Network-neutral platform provides an open UML implementation and representation of an NCAP's software model, which is extended by a developer to an application specific scenario. A discussion of the 1451.1 component model architecture will follow.

[2] NetTool Version 1.0.2 acts as a SOAP Message Listener, and provides network message characteristics. It is observed, that a large Roundtrip (ms) measurement occurs at the client. The WebNCAP's web-service SOAP communication relies on an HTTP Connection header of "Keep-Alive" messages. This HTTP header configuration, effects the NetTool's HTTP Client thread which does not correctly terminate, thus resulting in an inaccurate roundtrip (ms) network latency reading in Figure 23 and Figure 24.

Component Model Architectures Discussion

Based on the analysis of the 1451.1 application specific NCAP model, acquired from [Open1451], the application specific model resembles a homogeneous component model architecture. Whereby a homogeneous architecture can be viewed as a tightly coupled environment, which provides restrictions on communication interfaces, data-models are imposed on the component Model and the client-server relationship is mirrored as one-one entities. This style of component model architecture suggests a uniform platform and implementation exists on both client-server platforms. Further, a heterogeneous component model provides an environment whereby interoperable interactions occur over differing client-server platforms. In the latter environment, restrictions are not imposed on the client or server side platform implementations, and communication interfaces.

The challenge here is to provide a web-service interface for the IEEE 1451.1 application specific NCAP, which provides a heterogeneous platform, meaning the client and server platform implementations can differ. However, the communication between both client-server are based on a tightly coupled implementation from example application specific NCAP implementations. Further, the 1451.1 Model's analysis is based on the Open 1451 project group available at [Open1451]. The NCAP models analyzed entail the Gaithersburg 1.00 implementation {C++ and Java software development} and the PC1451NCAP Application {Java software development}.

Both NCAP software implementations provide a tight coupling between client-server NCAPs. This tight coupling results from the 1451.1 standard's defining operation messages (OP-Id's), commonly found within the OpID Class and communication structure. As well, the tight coupling, network communication, arises in the reliance on pre-defined communication TCP ports between an NCAP and Client. Furthermore, a challenge exists since the application specific and extended NCAP operations are not clearly published to a remote NCAP in a dynamic (easily accessible) manner.

This means in order to retain the current standard's application specific model, the following attributes from the 1451.1 standard must be captured and published to disparate client-entities in a WSDL file:

I: Operations - Application Specific Methods

The OP-IDs (operation-identification number) define the methods/operations in an NCAP, governed by the standards operation ID nomenclature. There exist specific operations for the Object Model Class

representations for instance: Communication Services, Component Model, Transducer-Blocks etc., however for application specific operations found in an Function Block, or NCAP, additional OP-Ids are declared solely by a developer. Further, the OP-ID is called during a client-server session from a Client-Port. In turn, the client-side OP-ID call via a Clients Execute operation will match with the remote systems OP-ID method implementation via a Server Perform operation, resulting in an RPC call. The Perform operations are commonly used inside an Function Block (depicted within a "switch-case" software structure) or NCAP Block and execute a requested operation on behalf of a client call. In turn, the result of an operation is sent back as a response to a client.

By introducing a WSDL convention, the software-method/operation call syntax can be retained as a descriptor within the 1451.1 NCAP. Whereby, The 1451.1 standard's process for operation identification is made via a lookup (OP-ID is matched with the operations syntactical representation). This means that a WSDL document would contain an operation name, along with the OP-ID representing the operation call.

II: Communication Services (Data Structure)

1. Argument Array Size (Data Structure for holding arguments for transmission) is explicitly declared
2. Each attribute is defined as an argument, and is encoded via name-value pairs.
3. An application specific method call is required for marshalling the Argument-Array structure (printArgumentArray in order to serialize the data into an Argument array structure)

The Argument Array Structure can be retained, within the form of a Document-based SOAP message. A Document-Style of representation provides the ability to exchange an XML based payload which contains an Argument Array structure (defined via a 1451 XML schema). The Argument array structure encapsulates the required operation and parameters, or simply parameterized data for client-server interactions. However, both client and server entities must agree to a predefined schema (schema can be defined via a WSDL file) in order to facilitate this style of SOAP message. For instance, in the case of a clients request to a server, the client must explicitly encode the proper data-types and format of the Argument-Array XML based message to the server. If the message does not adhere to the pre-defined schema, the server will raise an exception, resulting in an XML Fault message.

Chapter 6

Measurements and Results

The aforementioned research work is catered towards the concept of a web-service enabled platform for enabling self-describing instrumentation architecture. This service-oriented approach promotes future dynamic instrumentation environments over the current traditional distributed and tightly coupled instrumentation environment. Web-services are catered toward a loosely couple environment, much like the Internet, and are commonly found amongst heterogeneous networks.

By extending the 1451.1 NCAP system to a web-service environment, there is an extension in the vertical-integration of the underlying instrumentation system. This is commonly characterized as the OSI 7-Layer network model where, web-services will encompass primarily the Application, Presentation and Session, layers 7 through 5. With additional processing layers employed within the NCAP environment, an overhead in message processing can occur, based on the data size of a message.

The following message processing overhead case results where an "on-the-wire" message consists of a SOAP-XML based payload. A SOAP serializer is located at the server (WebNCAP server). It is required to map Java based-data types and operations, into an XML based-data type payload, and encapsulate this payload into a SOAP envelope, which transmits as a SOAP message (envelope+body) over an HTTP transport link. Furthermore, at the client a deserializer will retrieve the contents of the web-service call, and present, this data to the client.

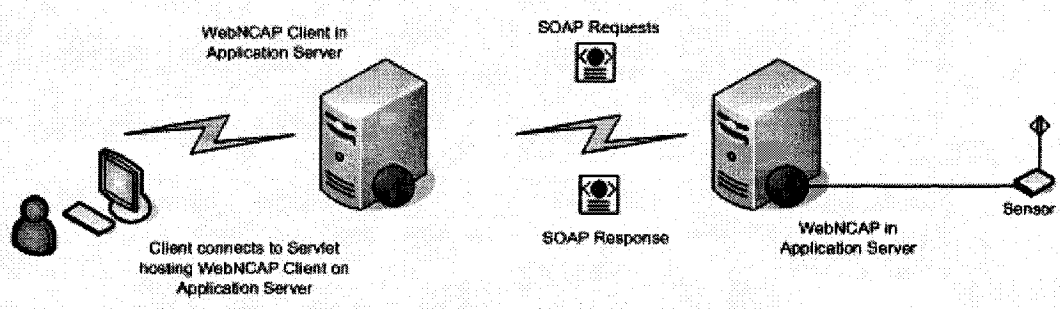


Figure 25: WebNCAP Client and WebNCAP Server Network Scenario

"WebNCAP" Performance Measurement Scenarios

Based on the design criteria and web-service interaction scenarios presented in Chapter 5, the performance measurement tests conducted, will encompass three scenarios for the WebNCAP. The first scenario entails a B2B topology, which validates the performance of the WebNCAP prototype server connected to a remote application server hosting a WebNCAP Client interface. The second scenario entails the performance of internal web-service communication, whereby the client side interface is located onboard the WebNCAP prototype server. The third scenario extends the second scenario whereby the WebNCAP Client interface connects to a campus Proxy-Server in order gain access to the WebNCAP Server. This validates the performance of a simulated application server located outside of a network subnet.

Performance Measurement based on SOAP Message Logging

For the basis of the web-service performance tests conducted within this research work, the aid of a SOAP handler-logger is developed. A SOAP message handler can provide logging and auditing capabilities at both client and server sides respectively. The client-side handler is utilized throughout the performance study, and results in logging a SOAP message request before a request is sent out over the network to a service endpoint. Additionally, logging the SOAP message response before it is received at the client provides the ability for measuring the round-trip performance of the SOAP communication methodology for the WebNCAP. Further, the performance analysis for testing of the IEEE 1451.1 communication is compared by following this same client-side logging convention.

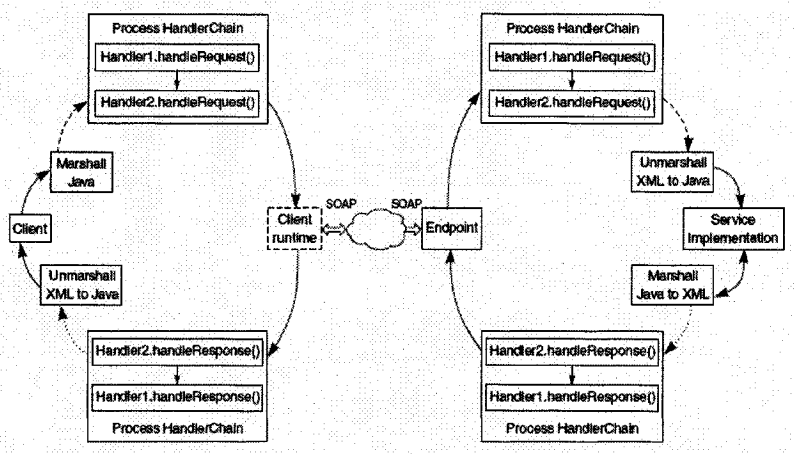


Figure 26: SOAP Message Handler-Logger Architecture [MTSM03]

"WebNCAP" Client and Server System Information

The prototype system for the WebNCAP Server consists of the following hardware setup:
WebNCAP Server and WebNCAP Client Application onboard a desktop IBM 2.80 GHz, Intel Pentium 4 CPU with 2.00 GB of RAM, hosted by the Windows XP Professional, Version 2002 Service Pack 2 Operating System. The software environment includes: (Java JVM/JDK 1.4.2_08 with J2EE 1.4, Netbeans IDE 5.0 Developer Build, and the Sun Java Application Server 8.1 2005Q1.)

A separate client desktop PC is required in order to test the web-service protocol communication performance between the WebNCAP server and a remote WebNCAP client, co-located within the same sub-network (100 Mbps Ethernet LAN).

The prototype WebNCAP client application consists of the following hardware setup:
WebNCAP Client onboard a desktop IBM 2.80 GHz, Intel Pentium 4 CPU with 256 MB of RAM, hosted by the Windows XP Professional, Version 2002 Service Pack 2 OS. The Software environment includes (JVM/JDK 1.4.2_09 with J2EE 1.4, Netbeans IDE 5.0 Developer Build, and the Sun Java Application Server 8.1 2005Q1.)

Performance Measurements Web Service(s) Information

The measurement results conducted throughout this research work provide the incurred network latency for varying "services" offered by the WebNCAP. Additionally, the services offered provide remote access to an NCAP's component blocks, namely the Transducer Block and Function Block.

The Transducer Block is comprised of the following web services:

A "*WebTED*" service for requesting and invoking legacy sensor (non 1451-compliant sensor) TEDS data from the onboard WebNCAP which contains a String array[] of 6 static transducer device characteristics. A "*sensor read*" service, which contains a String of 9 sensor data samples. As well as, a "*sensor Array*" read service which contains a Float Array[] of 9 sensor data samples.

The Function Block is comprised of a calibration service (Float Array[] of 9 sensor data samples), which corrects the temperature values acquired from the Wx-Dux V2.0 multi sensor.

Performance Measurements Web Service(s) System Interaction

A detailed system interaction diagram for the WebNCAP Client and WebNCAP Server (Java) servlets is illustrated in Figure 27. The interaction scenario depicts the required software components onboard a client and server in order to facilitate a web service based communication (SOAP) architecture. Note, that both Client and Server servlets can reside onboard a local machine, as will be the case throughout performance results scenario 2 and 3.

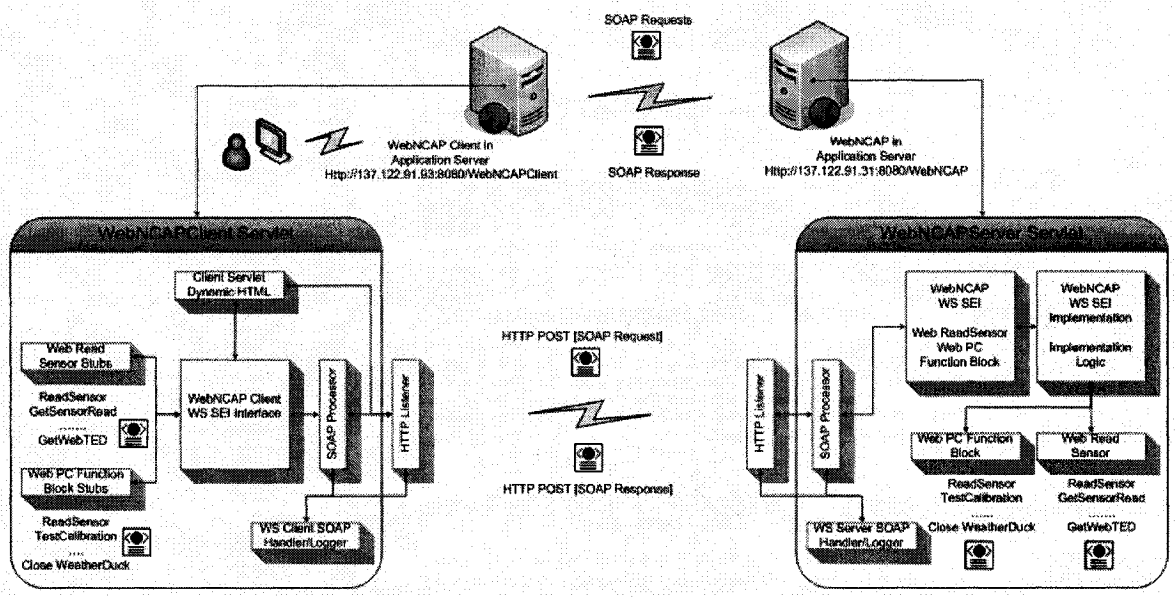


Figure 27: WebNCAP Client and WebNCAP Server System Interaction

A closer look at the required SOAP client side logger is illustrated in Figure 28. A client side SOAP logger is required for computing the web-service communication performance throughout this chapter. In addition, a server side SOAP logger provides the overhead in processing web service requests.

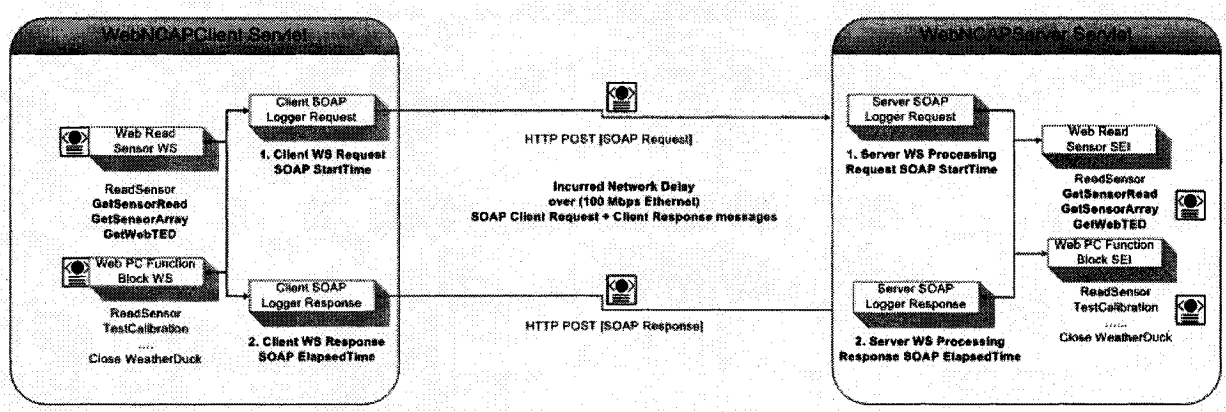


Figure 28: WebNCAP Client and WebNCAP Server SOAP Loggers

Performance Measurement Scenario I - B2B/M2M Scenario

The following measurement scenario is required for validating the B2B interaction between a WebNCAP client application server networked to a remote WebNCAP prototype server. The validation case provides an example of a typical web service interaction pattern. The measured values include the roundtrip for SOAP message request/response response time via a SOAP client logger.

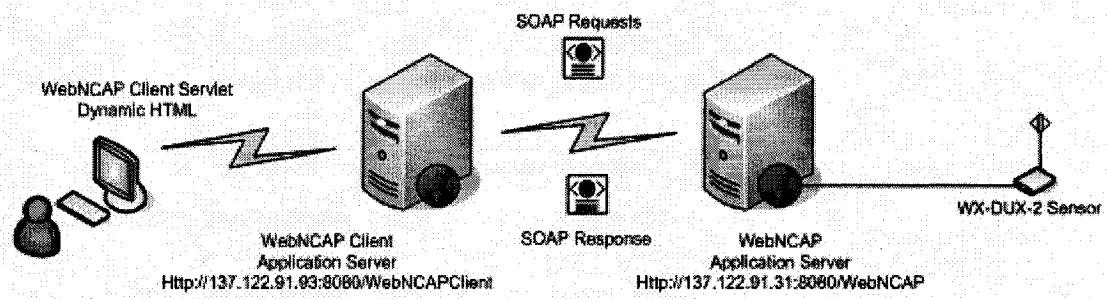


Figure 29: Performance Measurement I Network Topology

Test 1: SOAP Measured Response Time from WebNCAP Client accessing Transducer Block web service

SensorRead Interface (String 9 samples) (ms)	SensorArray Interface (float array[9]) (ms)	WebTed Interface (String array[6]) (ms)
32	593[3]	47
31	46	31
31	78	47
16	31	31
16	31	47
31	47	---
47	31	---
31	62	---
47	---	---

Table 4: Performance Measurement Scenario I-Test Case 1

Test 2: SOAP Measured Response Time from WebNCAP Client Transducer Block web service

SensorRead (ms)	SensorArray (ms)	WebTed (ms)
47	218	78
32	78	46
46	31	31
31	47	63
47	46	47
78	422[4]	32
47	47	---
94	47	---
31	125	---
31	---	---

Table 5: Performance Measurement Scenario I-Test Case 2

[3] Large Variation in response time (593 ms) is due to the Initial PCNCAP object instantiation and initial Servlet Initialization.

[4] Large Variation in response time (422 ms) is due to the Wx-Dux-2 sensor error, delay in the sensor auto-reporting cycle.

Performance Measurement Scenario II - Local Web Service

The following measurement scenario is required for validating the performance of internal web-service communication, whereby the client side interface is located onboard the WebNCAP prototype server. A client connects to the WebNCAP prototype server hosting a client side servlet interface in the form of a dynamic HTML web page. The web service calls are made internally between the server's *localhost* client-servlet and the service endpoint interface accessible via the HTTP listener on port 8080.

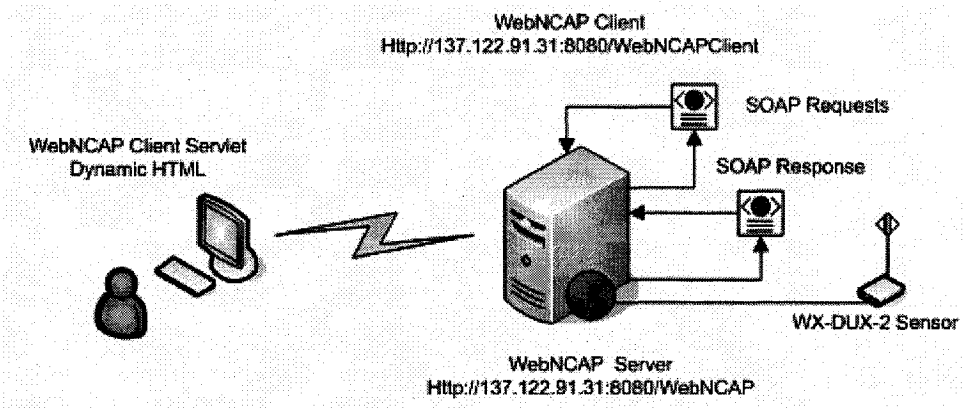


Figure 30: Performance Measurement II Network Topology

Test 1: SOAP Measured Response Time from WebNCAP Client accessing Transducer Block web service

SensorRead Interface (String 9 samples) (ms)	SensorArray Interface (float array[9]) (ms)	WebTed Interface (String array[6]) (ms)
47	47	47
15	31	31
15	32	32
31	31	32
47	31	31
31	31	32
32	63	---
15	32	---
31	31	---

Table 6: Performance Measurement Scenario II-Test Case 1

Test 2: SOAP Measured Response Time from WebNCAP Client accessing Transducer Block web service

SensorRead (ms)	SensorArray (ms)	WebTed (ms)
47	62	32
15	46	47
63	47	31
31	31	31
16	47	32
46	31	31
16	47	---
16	32	---
32	32	---
15	47	---

Table 7: Performance Measurement Scenario II-Test Case 2

Performance Measurement Scenario III - Local Web Service with Proxy Server

The following test provides a simulation for the SOAP request/response performance measurements for a server located external to the campus network. An application server requiring access to the WebNCAP prototype would require authentication via the campus proxy in order to interact with the web service offered. The simulation for this interaction is based on a Client-Servlet interface hosted onboard the WebNCAP Server, which connects to the University of Ottawa campus Proxy server. The proxy server redirects the SOAP requests/responses to the WebNCAP Server.

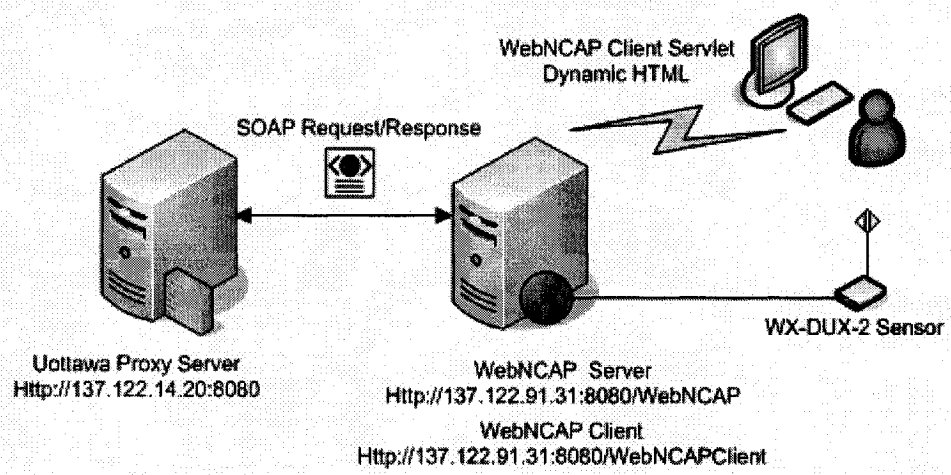


Figure 31: Performance Measurement III Network Topology

Test 1: SOAP Measured Response Time from WebNCAP Client accessing Transducer Block web service

SensorRead Interface (String 9 samples) (ms)	SensorArray Interface (float array[9]) (ms)	WebTed Interface (String array[6]) (ms)
234	187	625[5]
203	219	219
235	188	250
234	250	187
235	234	219
219	219	---
250	156	---
203	187	---
250	281	---

Table 8: Performance Measurement Scenario III-Test Case 1

Based on the Proxy server intermediary within this measurement scenario, it is apparent that the performance of the web services decreases, in terms of SOAP message round trip time. In comparison to the previous scenarios, the application server and client are located within the same internal network.

[5] Large Variation in response time (625 ms) is due to the Initial PCNCAP object instantiation and Servlet initialization.

Test 2: SOAP Measured Response Time from WebNCAP Client accessing Transducer Block web service

SensorRead (ms)	SensorArray (ms)	WebTed (ms)
203	234	266
234	204	172
250	250	250
235	250	203
234	172	171
250	234	---
203	266	---
218	204	---
234	250	---
235	266	---

Table 9: Performance Measurement Scenario III-Test Case 2

Performance Measurement Scenario Summary

In order to summarize the web-service communication performance amongst each scenario for similar web service "*types*", the following graphs (Figures 32-34) will depict the results solely from the Test #1 case. It is apparent that degradation occurs in the web service performance when traversing a network proxy-server. Further, by hosting a web service client interface onboard the WebNCAP prototype server, there is a decrease in the roundtrip SOAP message latency incurred, in comparison to a remote server-server (B2B/M2M) scenario.

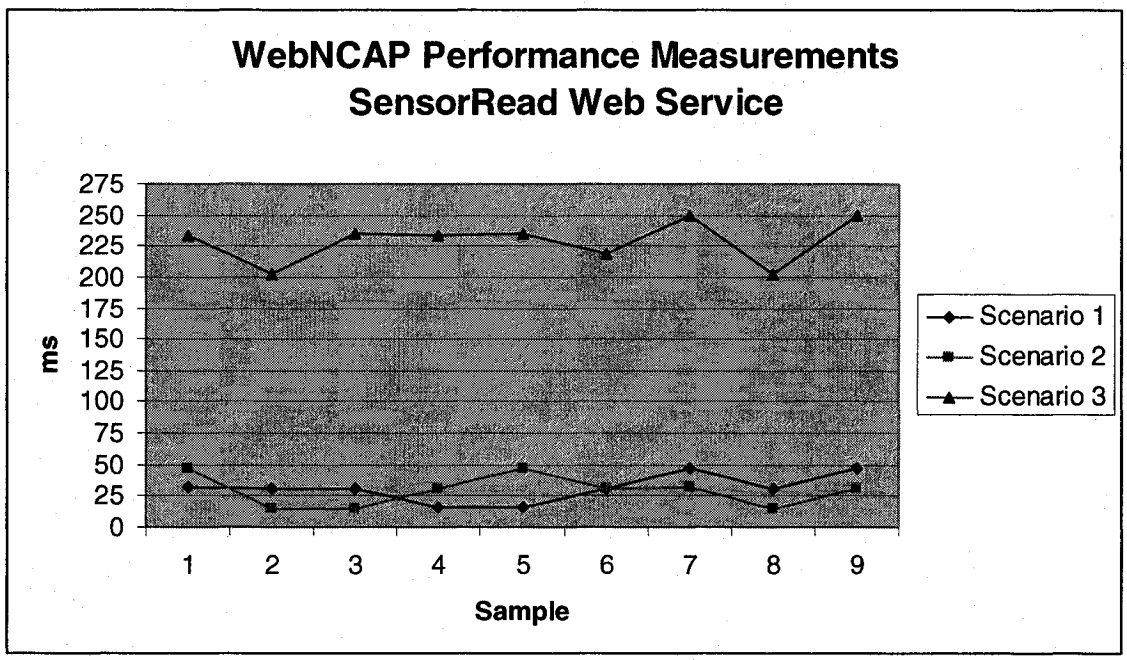


Figure 32: Performance Results for SensorRead Service Test 1

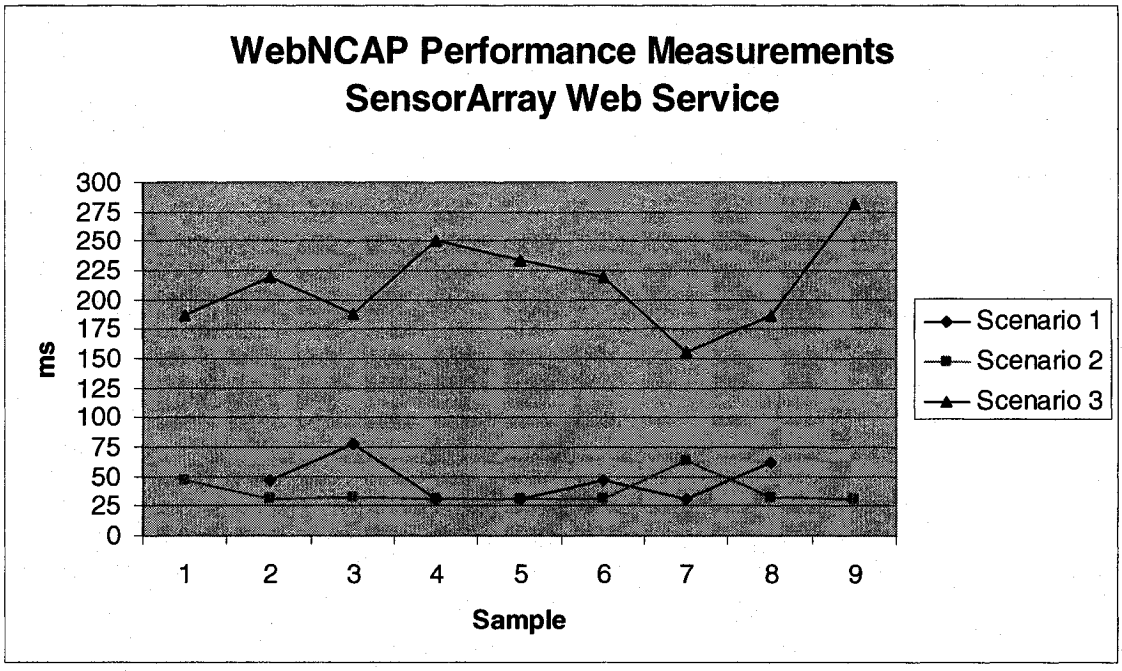


Figure 33: Performance Results for SensorArray Service Test 1

Note, in Figure 33: Sample 1 (593 ms) for Scenario 1 is omitted due to the large fluctuation in response time.

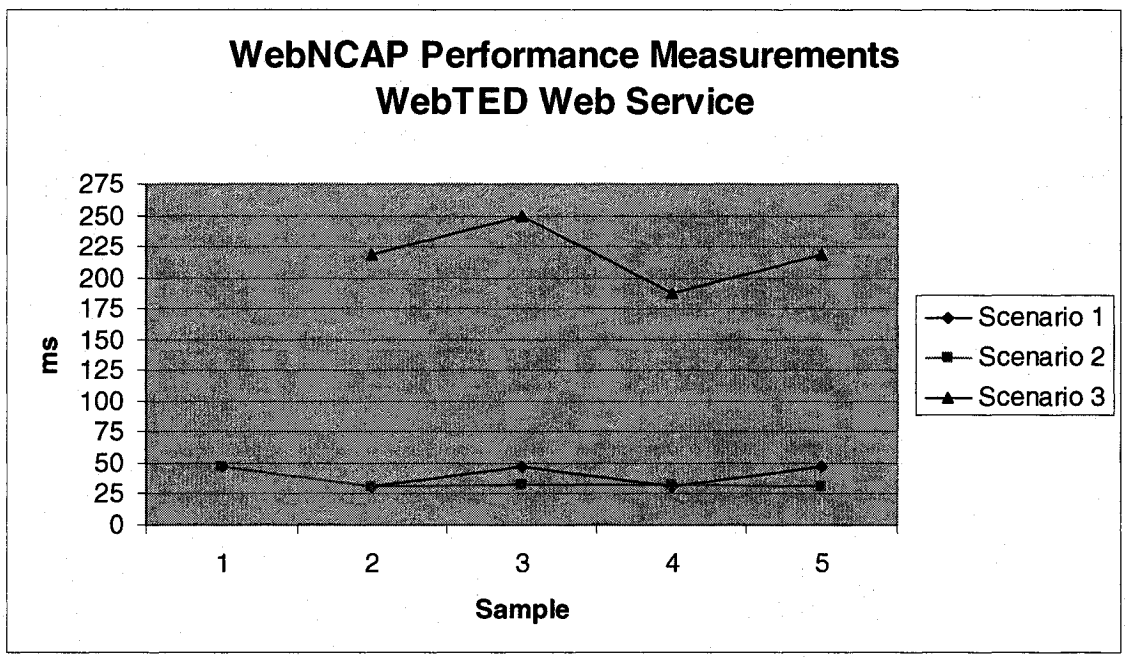


Figure 34: Performance Results for WebTED Service Test 1

Note, in Figure 34: Sample 1 (625 ms) for Scenario 3 is omitted due to the large fluctuation in response time.

In addition, a summary of the web-service performance amongst each scenario defined are illustrated in the following graphs (Figures 35-37) which depict the results solely from the Test #2 case.

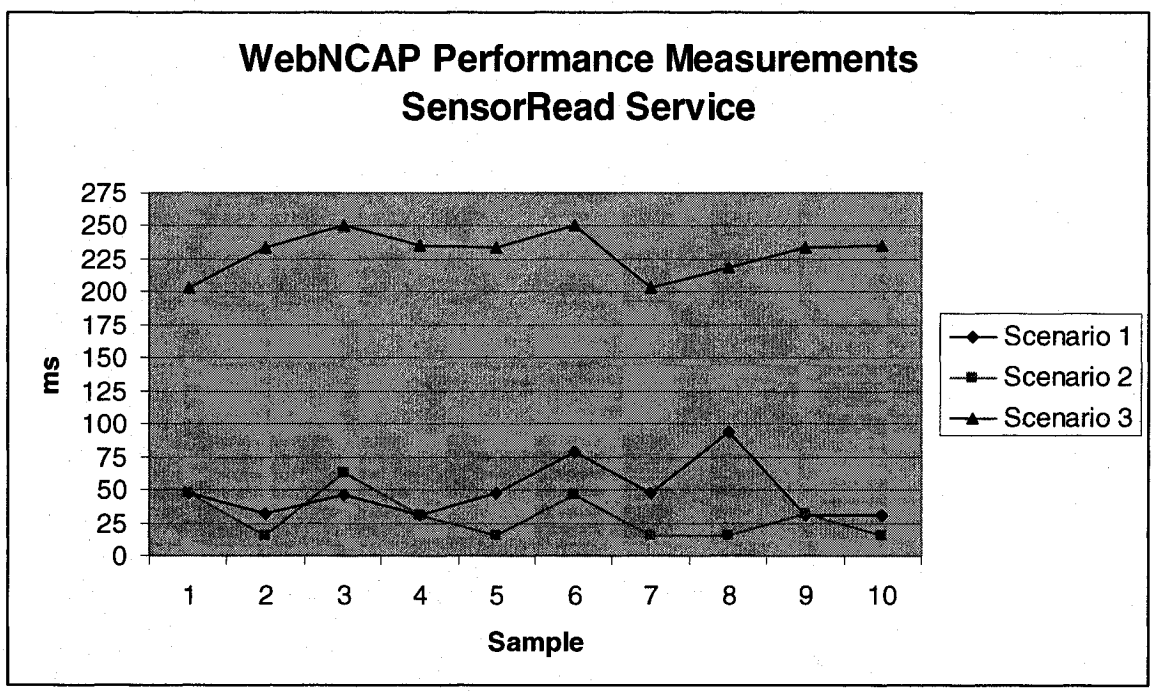


Figure 35: Performance Results for SensorRead Service Test 2

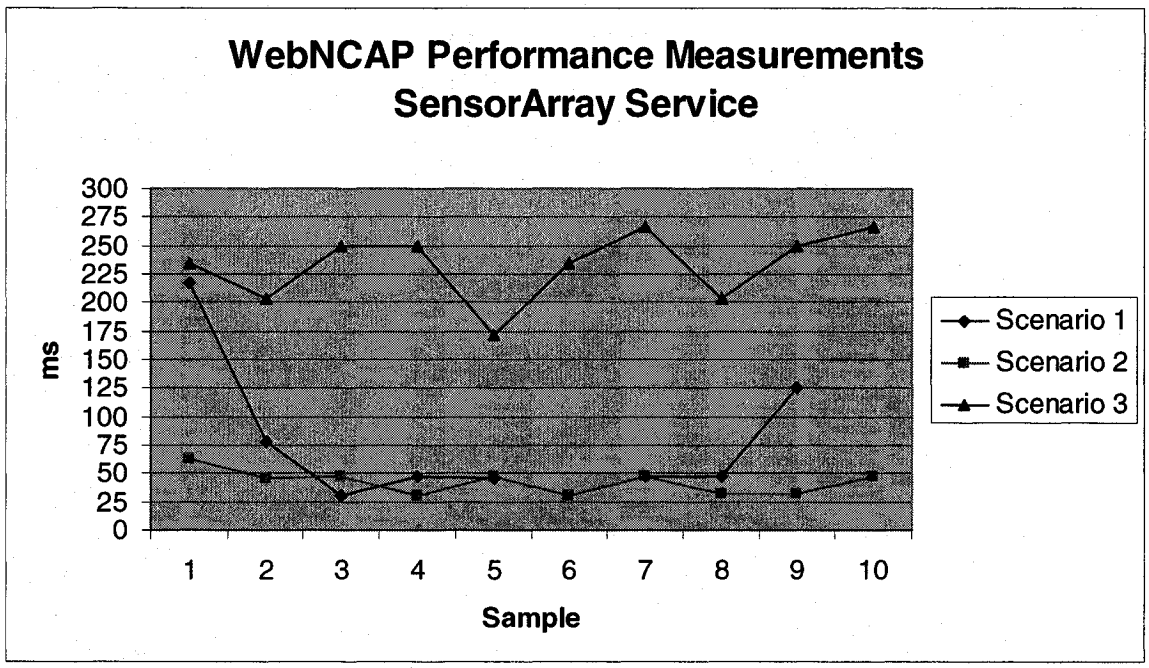


Figure 36: Performance Results for SensorArray Service Test 2

Note, in Figure 36: Sample 6 (422 ms) for Scenario 1 is omitted due to the large fluctuation in response time.

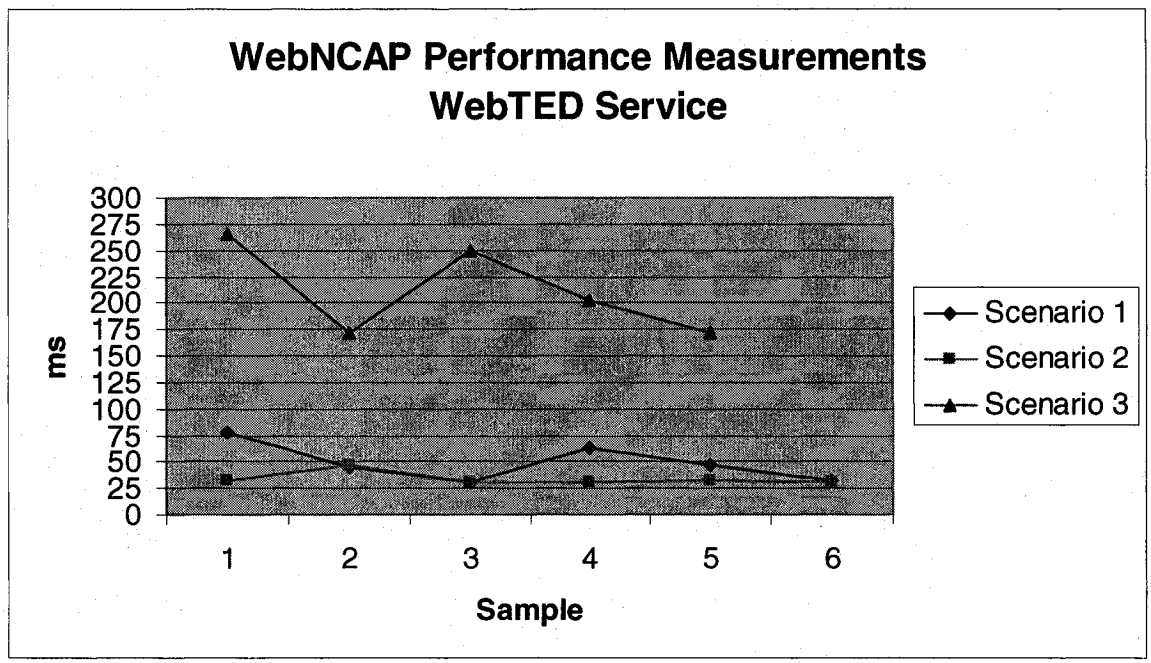


Figure 37: Performance Results for WebTED Service Test 2

Performance Measurement Scenario Statistics

The following (Tables 10-12) provide a summary of the measured response times for client-server SOAP based message interaction. The statistics are based on the acquired results, which are graphed in (Figures 32-37). The statistics presented include the calculated average (mean) for the result data set, the accompanying standard deviation along with a confidence interval[6]. The confidence interval is based on a confidence level of 95%, whereby a lower and upper bound sample mean is calculated for each given web service interaction scenario.

Table 10: SOAP Measured Response Statistics for *sensorRead* Web Service

***sensorRead* Web Service**	**B2B WS Scenario 1**	**Local WS Scenario 2**	**Proxy Server WS Scenario 3**
Test 1			
Mean (average)	31.33 ms	29.33 ms	229.22 ms
Standard Deviation	10.96 ms	12.53 ms	17.52 ms
Confidence Interval 95%	± 7.16 [24.17,38.50] ms	± 8.19 [21.15,37.52] ms	± 11.45 [217.78,240.67] ms
Test 2			
Mean (average)	48.40 ms	29.70 ms	229.60 ms
Standard Deviation	21.46 ms	17.24 ms	16.66 ms
Confidence Interval 95%	± 13.30 [35.10,61.70] ms	± 10.69 [19.01,40.39] ms	± 10.33 [219.27,239.93] ms

[6] Confidence Interval is computed via MS Office Excel 2003, with a 95% Confidence Level. The resulting confidence interval provides a lower and upper bound confidence level applied to the sample mean. For example, Table 10 provides a computed lower bound sample mean for Test 1, B2B WS Scenario 1 of 24.17 ms, and an upper bound sample mean of 38.50 ms.

Table 11: SOAP Measured Response Statistics for *sensorArray* Web Service

sensorArray Web Service	B2B WS Scenario 1	Local WS Scenario 2	Proxy Server WS Scenario 3
Test 1			
Mean (average)	46.57 ms	36.55 ms	213.44 ms
Standard Deviation	18.04 ms	11.20 ms	38.29 ms
Confidence Interval 95%	± 13.37 [33.20,59.94] ms	±7.32 [29.23,43.88] ms	± 25.02 [188.43,238.46] ms
Test 2			
Mean (average)	79.88 ms	42.2 ms	233.00 ms
Standard Deviation	63.13 ms	10.31 ms	30.64 ms
Confidence Interval 95%	± 43.74 [36.13,123.62] ms	± 6.39 [35.81,48.59] ms	± 18.99 [214.01,251.99] ms

Note, based on the web service *type* statistics offered (Table 10-12), the *sensorRead* service provides the lowest sample mean latency communication for remote sensor data retrieval.

Table 12: SOAP Measured Response Statistics for *webTED* Web Service

webTED Web Service	B2B WS Scenario 1	Local WS Scenario 2	Proxy Server WS Scenario 3
Test 1			
Mean (average)	40.60 ms	34.60 ms	218.75 ms
Standard Deviation	8.76 ms	6.95 ms	25.72 ms
Confidence Interval 95%	± 7.68 [32.92,48.28] ms	± 6.09 [28.51,40.69] ms	± 25.21 [193.54,243.96] ms
Test 2			
Mean (average)	49.50 ms	34.00 ms	212.40 ms
Standard Deviation	18.23 ms	6.39 ms	43.93 ms
Confidence Interval 95%	± 14.58 [34.91,64.09] ms	± 5.11 [28.89,39.11] ms	± 38.51 [173.89,250.91] ms

It should be noted, for the server side SOAP logger processing that a 16 ms latency is acquired for a local web-service (performance scenario 2) and a 32 ms latency is acquired for a B2B web-service scenario (performance scenario 1). This latency is a factor of the overhead in processing an initial web-service invocation. Further, any subsequent server side processing of SOAP messages resulted in repeatable 0 ms latency. This 0 ms latency is due to the server side SOAP logger's internal java timer, which is comprised of a precision time unit in milliseconds. Thus, for the server side processing of: web-service SOAP requests, method invocation, and generation of a SOAP response a negligible server side SOAP processing latency is attained.

Further, based on the calculated performance measurements in Tables 10-12, the calculated confidence intervals are quite large, thus resulting in large (lower/upper bound) sample mean estimates. An extended performance results scenario appears in Appendix V, providing a large sample data set distribution. Whereby a sample data set of 95 latency values for the *sensorRead* web-service, are recorded at the SOAP Client handler-logger. In addition a summary of the network latency statistics are presented for the extended performance scenario.

Performance Measurements IEEE 1451.1 Client-Server Communication

The IEEE 1451.1 NCAP provides two communication methodologies via a Client-Server RPC model and a Publish-Subscribe model. An analysis of the 1451.1 Client-server RPC based interaction is examined in order to compare the communication performance to the aforementioned web service results. The communication performance results of the IEEE 1451.1 are based on a client-server communication scenario utilizing the PCNCAP 1451 application and a client PDANCAP 1451 application. As stated in Chapter 5, a requirement of the development component entails migrating the original NCAP application (PC-NCAP) onboard a web application server and integrating a real-time sensor for testing purposes. This scenario allows for testing the traditional IEEE 1451.1 communication mechanism and validating that the NCAP can operate within an application server hosted environment.

Further, the performance measurement for 1451.1 roundtrip network communication is based on a SOAP message handler convention whereby a 1451.1 client request is logged before a 1451 Message packet is sent onto the network. A 1451.1 client response is logged before a client receives the un-marshalled 1451 response message. The TCP/IP protocols facilitate the underlying network communication for the NCAP server and client.

The performance measurements conducted are based on the following network communication topology illustrated in Figure 38, for the PDA NCAP Client and PCNCAP Server. It should also be noted, that the PDANCAP Client application and PCNCAP application are hosted on their respective SJAS application servers.

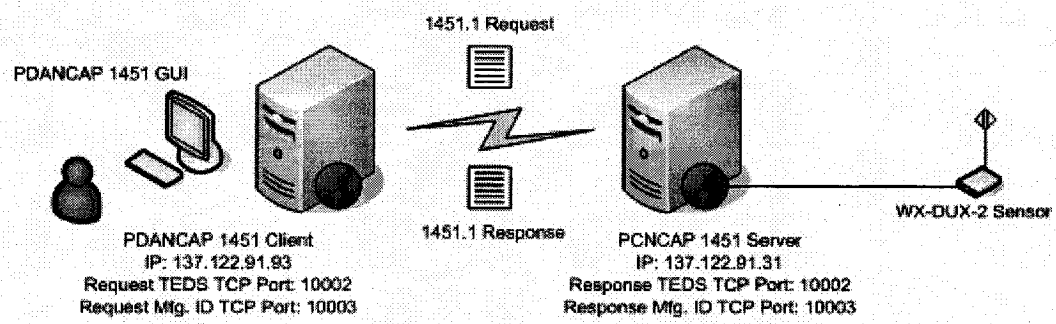


Figure 38: Performance Measurement 1451.1 NCAP Network Topology

The recorded performance results for executing a request TEDS procedure yields a 109 ms response time. For a request of a manufacturer ID (Mfg ID) operation, a 125 ms response time is attained. The response times were calculated based on the server output log provided by the onboard SJAS

application server, hosting the PDANCAP application. However, by closer examination of the 1451.1 client port's software run-thread method, a thread delay of 100 ms is imposed on the network client port. This extra 100 ms latency is an artifact of the application specific PDANCAP software design implementation. Thus, the resulting communication performance measure(s) are 9 and 25 ms for the 1451.1 NCAP server to client response time. Further, this study demonstrates that traditional 1451.1 NCAP application communications can co-exist with the developed web-service communication interfaces, onboard an application server.

Overall Summary of Performance Results

Based on the experimental interaction scenarios, suitable web service candidates for an application area of critical infrastructure and event monitoring are performance scenario 1, and 2. These scenarios provide an internal network communication approach with low latency observed at the service-requestor, a client machine co-located within the same subnet. The performance scenario 3 provided the greatest network latency, for a client machine located outside of the sensor network subnet. Comparably, the IEEE 1451 communication TCP/IP over Ethernet (100 Mbps), provided the lowest latency observed at the client. Moreover, additional tests of [LS99] provide quoted 5 ms network latency for IEEE 1451.1 communication within an Ethernet subnet.

The IEEE 1451 protocol can provide the real-time characteristics for an internal network monitoring application; however, a web-service approach is more feasible for dynamic interactions, involving remote client interface generation for third-party applications, heterogeneous NCAPs and in the event of critical infrastructure monitoring, on-site personnel and first responders can quickly obtain NCAP connection details.

It should also be noted, that the web service performance results are a function of the sensor's internal response rate, a sampling time measure in terms of samples per second. This rate will effect the web-service load on network requests, invocations, and transmission of data to client interfaces. A tradeoff appears in the response time of a sensor compared to the validity of sensor data. (By increasing the sampling rate of a sensor, validity of sensor data suffered in the case of the Wx-Dux-2 sensor.) Further, the size of the data samples collected from the sensor are also a factor, this sample data size can vary in relation to the sensor's sampling rate. In addition, the number of data samples, which are transmitted in SOAP/XML messages, affects the overall performance at both the Client (network-transport) and Server side processing (marshalling/un-marshalling) of SOAP messages.

Chapter 7

Conclusions

The presented 1451.1 web service enabled NCAP, termed "*WebNCAP*", enhances and provides the ability for transducer manufacturers and software developers to introduce "value-added" services within a 1451.1 environment. The addition of a web-services framework provides an interoperable mechanism for network communication, by employing the SOAP protocol as a message oriented network communication platform over an HTTP transport. Further, the web service based NCAP model facilitates inter-NCAP communications, via machine-machine interfaces (M2M) characterized by WSDL service descriptions.

The need for web service(s) based M2M platforms within a 1451.1 sensor network provides the ability of preserving a distributed architecture and enabling neighbouring NCAPs to publish their available application services. By adopting a new form of interface, via WSDL service descriptions, a machine-readable format is provided for initiating communication between NCAPs and clients in order to invoke and acquire services offered by one another. Furthermore, a WSDL service description provides a solution for grouping and classifying the functional capabilities and interaction for an NCAP within loosely coupled environments.

Summary of Conclusions and Contributions

Through this research contribution, it is shown by way of a critical analysis, prototype development and prototype validation, that a web services framework applied to the IEEE 1451.1 NCAP model can provide a more extensible and scaleable system. The traditional 1451.1 NCAP's tightly coupled architecture does not scale well in a dynamic environment, and generation of client side interfaces requires a complex software development process. Further, through the development of the WebNCAP prototype, a loosely coupled 1451.1 platform is established which promotes the nature of a Service Oriented Architecture (SOA), suitable for dynamic environments.

The research investigation conducted throughout this thesis provides a base-model and base-platform for development of Web Services enabled instrumentation network controllers. The design and concepts

presented can be extended in order to suit the needs for ad-hoc based critical infrastructure monitoring domains.

The prototype validation provides performance measurements for SOAP based message-oriented communication over an HTTP transport between a web service(s) WebNCAP server and a client web-service application server. The performance study analyzed, entailed a B2B (server-server) scenario, for the interconnection of a WebNCAP server to a client application server, co-located within the same subnet network. The performance measures for a 99.9% confidence interval yield a sample mean estimate of (36.03-45.15 ms) transmission latency for sensory data via a *sensorRead* web service. The second scenario consists of measuring the performance for internal web service communication, in the case of a client-side interface residing onboard the web service server. This test provided the fastest communication response for the *sensorRead* web service, with a 99.9% confidence interval sample mean estimate of (32.50-42.51 ms) for transmission latency. Further, the third scenario entails simulating a disparate application server client, networked via a campus proxy-server to the WebNCAP application server. The performance measures for a 99.9% confidence interval yield a sample mean estimate of (230.30-248.96 ms) latency for the *sensorRead* web service. In comparison to the application specific 1451.1 NIST NCAP, a repeatable latency of 9 and 25 ms is acquired, over direct TCP/IP connections, for client-server communications, co-located within the same subnet network. Further, for real-time control and monitoring purposes the IEEE 1451.1 NCAP is a suitable candidate in comparison to the web-service SOA approach.

In addition, based on the network latency results it is apparent that a tradeoff exists between the traditional IEEE 1451.1 tightly coupled environments, which promote faster communication vs. a loosely coupled service oriented architecture environments which promote interoperable client side generation, a message oriented communication scheme, and a self-describing instrumentation platform. A SOA solution applied to the IEEE 1451.1 NCAP provides the benefit of a more extensible platform, for control and monitoring of dynamic sensory network environments. Furthermore, it may be concluded that a Web service(s) NCAP application model could replace the traditional tightly coupled IEEE 1451.1 model.

Future Research

Future research work within this project includes the investigation of alternate web service based communication protocols, enhanced client-side application functionalities (web portal development) as well as examining the prototype WebNCAP's communication performance over a global Internet scale.

Alternate web service protocol development in the area of a suitable publish-subscribe communication mechanism based on a web services middleware framework is possible. The W3C's drive to promote emerging SOAP based communication standards, such as WS-Eventing and WS-Notifications provide a solution for integrating web services based specifications to the IEEE 1451.1 Model. In terms of user interaction, an enhanced client-side application is possible by web-portal development. The web-portal, hosted via a JSP based client-interface can provide web service integration capabilities with current location based mapping and visualization services such as "*google-maps*" functionality. Furthermore, in order to test the web service communication performance on a global (Internet) scale, future research collaborations between universities and public research organizations can provide a suitable network test-bed.

Bibliography

[1451.1] IEEE Standard for a Smart Transducer Interface for Sensors and Actuators-Network Capable Application Processor (NCAP) Information Model, This paper appears in: IEEE Std. 1451.1-1999 Publication Date: 2000

[1451.2] IEEE standard for a smart transducer interface for sensors and actuators - transducer to microprocessor communication protocols and Transducer Electronic Data Sheet (TEDS) formats, This paper appears in: IEEE Std. 1451.2-1997 Publication Date: 1998

[1451.3] IEEE Standard for a Smart Transducer Interface for Sensors and Actuators-Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems, This paper appears in: IEEE Std. 1451.3-2003 Publication Date: 2004

[1451.4] IEEE Standard for A Smart Transducer Interface for Sensors and Actuators - Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats,
This paper appears in: IEEE Std. 1451.4-2004 Publication Date: 2004

[ACKM04] G. Alonso, F. Casati, H. Kuno, V. Machiraju, "Web Services: Concepts Architectures and Applications" Springer, 2004.

[DPPC03] F.C. Delicato, P.F. Pires, L. Pinnez, L.F.R. da Costa, "A flexible web service based architecture for wireless sensor networks", Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on , (2003) pp. 730-735

[DPRP04] F. Delicato, P. Pires, J. Rezende, L. Pirmez, "Service Oriented Middleware for Wireless Sensor Networks" Technical Report NCE04/04. Federal University of Rio de Janeiro. http://www.nce.ufrj.br/labnet/research/sensornet/publications.htm

[EP03] W. Elmenreich, S. Pitzek, "Smart Transducers – Principles, Communications, and Configuration" Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems (INES), March 2003, Assuit, Egypt http://www.vmars.tuwien.ac.at/frame-papers.html

[Erl04] T. Erl, "Service Oriented Architecture – A Field Guide to Integrating XML and Web Services" Prentice Hall, 2004.

[Erl05] T. Erl, "Service Oriented Architecture – Concepts, Technology, and Design", Hardcover Prentice Hall, 2005.

[FFMST02] P. Ferrari, A. Flammini, D. Marioli, E. Sisinni and A.Taroni, "A low-cost smart sensor with Java interface" Sensors for Industry Conference, 2002. 2nd ISA/IEEE , 19-21 Nov. 2002 Pages: 161 – 167

[GSS05] B.L. Gorman, M. Shankar, C. M. Smith, "Advancing Sensor Web Interoperability", SensorNet Program, Oak Ridge National Laboratory, Article appears in Sensors - April 2005 Cover Story http://www.sensornet.gov/sen4_45_05e.pdf

[Horan05] B. Horan, "The Use of Capability Descriptions for Wireless Transducer Networks", SUN Technical Report TR-2005-131, February 2005 http://research.sun.com/techrep/2005/smli_tr-2005-131.pdf

[IBM03] J. Liu, Y. Lu, "Build interoperable Web Services with JSR-109" – IBM Document, August 2003, http://www-128.ibm.com/developerworks/webservices/library/ws-jsrart/

[IEEE P1451] – National Institute of Standards and Technology, IEEE 1451 http://ieee1451.nist.gov/

[J2EE05] "The J2EE 1.4 Tutorial for Netbeans IDE 4.1" SUN Tutorial, May 2005 http://www.netbeans.org/download/docs/41/j2ee-tutorial/index.html

[JDDAC] Java Distributed Data Acquisition and Control, https://jddac.dev.java.net/ (March 2006)

[JSR109] JSR 109 Specification – Implementing Enterprise Web Services – Nov. 2005 http://www.jcp.org/en/jsr/detail?id=109

[LS99] K.B. Lee, R.D. Schneeman, "Distributed measurement and control based on the IEEE 1451 smart transducer interface standards" Instrumentation and Measurement Technology Conference, 1999. IMTC/99. Proceedings of the 16th IEEE,Volume: 2 , 24-26 May 1999 Pages: 608 - 613 vol.2

[LS03] K. Lee, E. Y. Song, "UML model for the IEEE 1451.1 standard", Proc. of the 20th IEEE Instrumentation & Measurement Technology Conference (IMTC '03), Vol.2 , 20-22 May 2003, Pages:1587 – 1592.

[LS04] K. Lee, E. Y. Song, "Object-Oriented Application Framework for IEEE 1451.1 standard", Proc. of the 21st IEEE Instrumentation & Measurement Technology Conference (IMTC '04), Vol.2 , 18-20 May 2004, Pages:1182 – 1187.

[Merino04] Y. M. Vigil Merino, "Web Service Interface and Architecture for accessing Fieldbus Systems" Masters Thesis, Hamburg University of Technology, Hamburg, Germany, January 2004. http://www.ti5.tu-harburg.de/Publication/2004/

[MTSM03] J. McGovern, S. Tyagi, M. Stevens, S. Mathew, "Java: Web Services Architecture", Morgan Kaufmann Publisher, 2003.

[Netbeams] Netbeams Project, http://www.netbeams.org (March 2006)

[NetTool] Capescience Software - http://www.capescience.com/articles/using_nettool/ (Dec. 2005) Updated NetTool Project Link http://sourceforge.net/projects/nettool/ (March 2006)

[OMGST03] OMG Smart Transducer Interface Specification, OMG Group, 2003 http://www.omg.org/docs/formal/03-01-01.pdf

[Open1451] – Y. Song, PCNCAP (PJava1451Dot1) Reference Implementation from Open Implementation of the 1451 Standard Project via Sourceforge.net http://sourceforge.net/projects/open1451/ (revised January 2006)

[SensorNet] SensorNET GOV Project, http://www.sensornet.gov/ (March 2006)

[SensorML] SensorML Project http://vast.uah.edu/SensorML/ (March 2006)

[SL05] E.F. Sadok, R. Liscano, "A Web Services Framework for 1451 Sensor Networks", Proceedings of the 2005 IEEE Instrumentation and Measurement Technology Conference, 2005. (IMTC 2005) Ottawa, ON Canada May 17-19, 2005

[SOAP] SOAP 1.2 Specification – June 2003 - http://www.w3.org/TR/soap/

[STKV03] B. Scherer, C. Toth, T. Kovacshazy, B. Vargha, "SNMP-based approach to scalable smart transducer networks" Instrumentation and Measurement Technology Conference, 2003. IMTC '03. Proceedings of the 20th IEEE , Volume: 1 , 20-22 May 2003 Pages: 721 – 725

[SWE05] Sensor Web Enablement Project from the Open GeoSpatial Consortium (March 2006) http://www.opengeospatial.org/functional/?page=swe

[TransducerML] TransducerML Project http://www.transducerml.org/ (March 2006)

[UDDI] UDDI Specification – OASIS Group – http://www.uddi.org/specification.html

[VP05] V. Viegas, J. M. Dias Pereira, "Using a Commercial Framework to Implement and Enhance the IEEE 1451.1 Standard" IMTC 2005 – Instrumentation and Measurement Technology Conference Ottawa, Ontario, Canada, May 2005.

[VPt03] M. Venzke, S. Pitzek, "Accessing Fielbus Systems via Web Services" First Workshop on Intelligent Solutions in Embedded Systems (WISES 2003)

[W3C] World Wide Web consortium – http://www.w3c.org

[WSDL] WSDL 1.1 Specification – March 2001 - http://www.w3.org/TR/wsdl

[WSEventing] WS Eventing Specification – IBM 2004 – http://www-128.ibm.com/developerworks/webservices/library/specification/ws-eventing/

[WSNotifications] WS Notification Specification – IBM 2004 – http://www-128.ibm.com/developerworks/webservices/library/specification/ws-notification/

Appendix I

Please contact the author for the web service and project source codes by the following email: emil.sadok@gmail.com

The developed software utilized throughout this research project includes the following project source codes:

Server Application Code for the WebNCAP Server side Deployment, hosted by Sun Java Application Server 8.1:

WebNCAP WAR project archive – For the Sun Java Application Server deployment file
WebNCAP Project Files – For the SUN Netbeans IDE Version 5.0+ or higher

Client Code for the WebNCAP Client Application and Deployment, hosted by Sun Java Application Server 8.1:

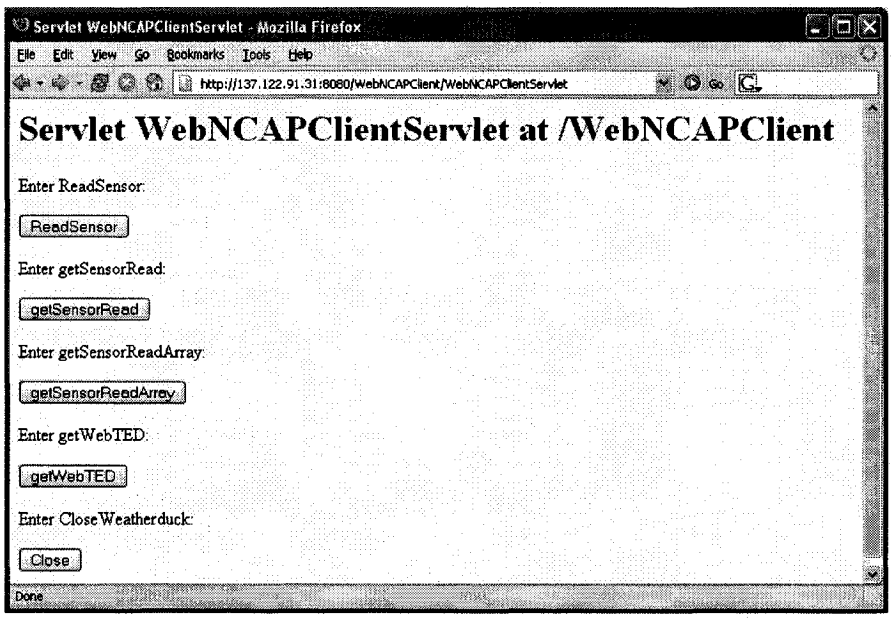
WebNCAP Client WAR project archive – For the Sun Java Application Server deployment file
WebNCAP Client Project Files – For the SUN Netbeans IDE Version 5.0+ or higher

PC NCAP Server and Client Application Code obtained from [Open1451]:

PCNCAP1451 IEEE 1451.1 NIST/Open 1451 modified by the author for Wx-Dux-2 interface
PDANCAP 1451 IEEE 1451.1 NIST/Open 1451 modified by the author for Wx-Dux-2 interface
Java Communications API/Library – For the WebNCAP and PC NCAP Application

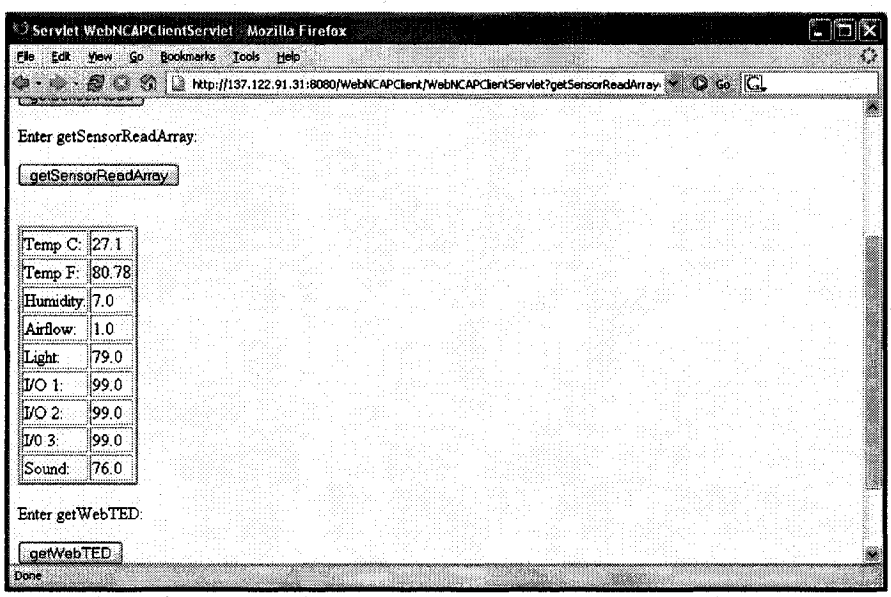
Appendix II

Screen Captures of Dynamic HTML Web Client Application Interface, which drives the prototype WebNCAP server, are illustrated.



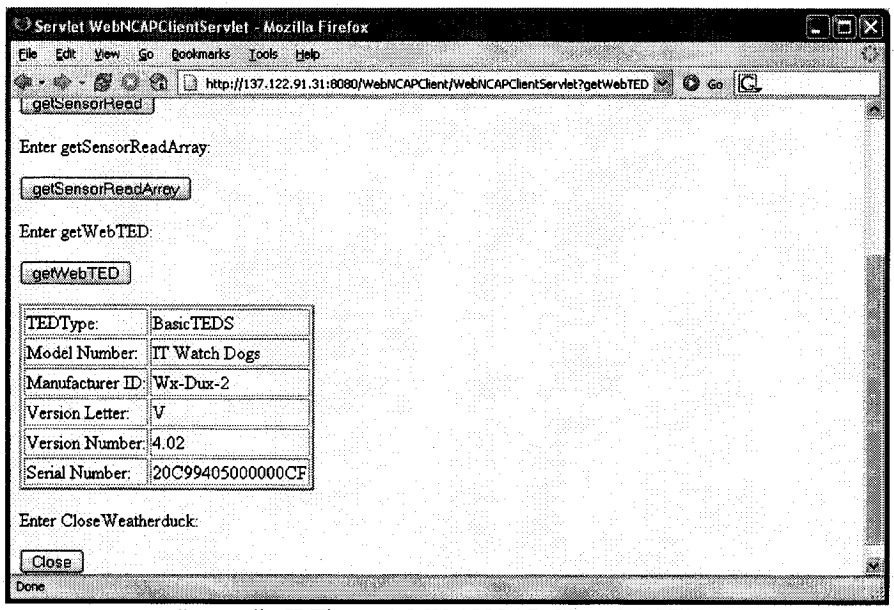
Appendix II-Figure 1.0 WebNCAP Client Prototype Servlet

In Figure 1.0 is the Client Side Interface, hosted onboard the WebNCAP Server, used for Performance Testing Scenario 2 (Chap. 6), whereby the Client Servlet is located on the WebNCAP Server.



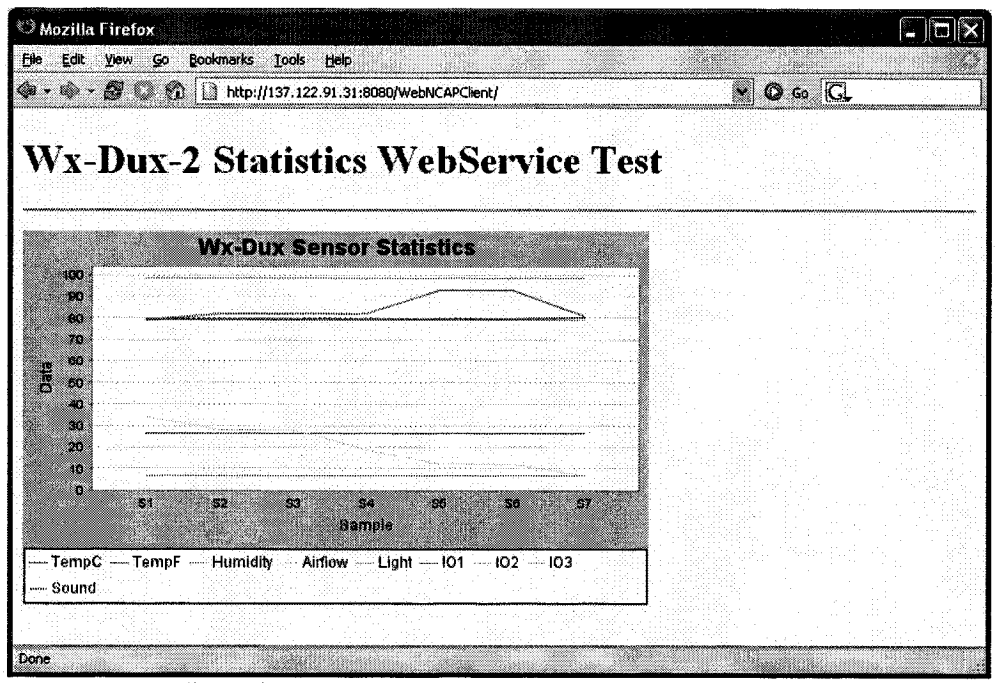
Appendix II-Figure 1.1 WebNCAP Client sensor Array Web Service

In Figure 1.1, a Sensor Read Array Service's results are illustrated via the client servlet.



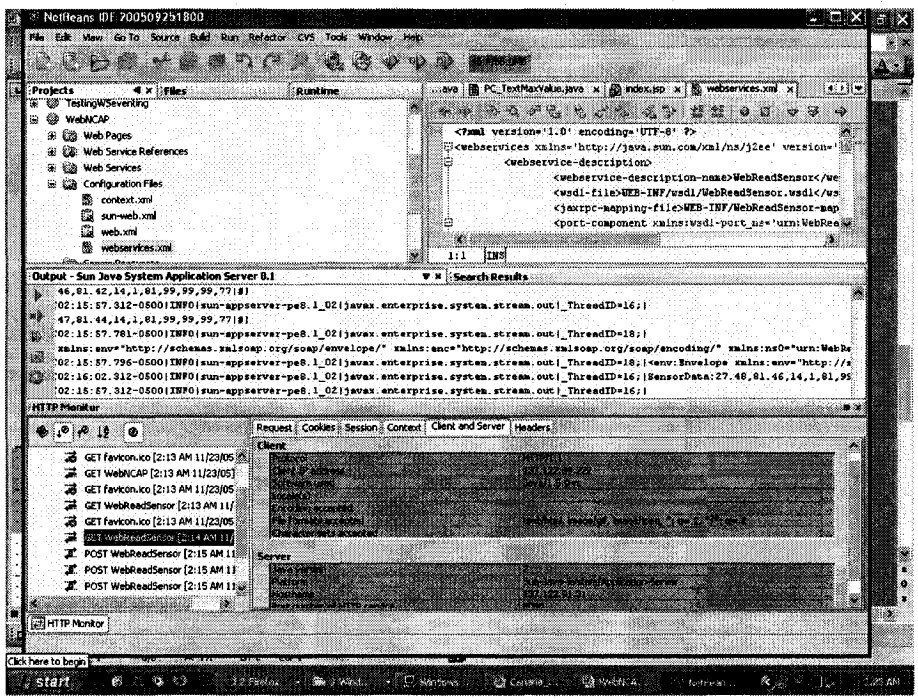
Appendix II-Figure 1.2 WebNCAP Client WebTED

In Figure 1.2, a Web TED Service's results are illustrated via the client servlet.



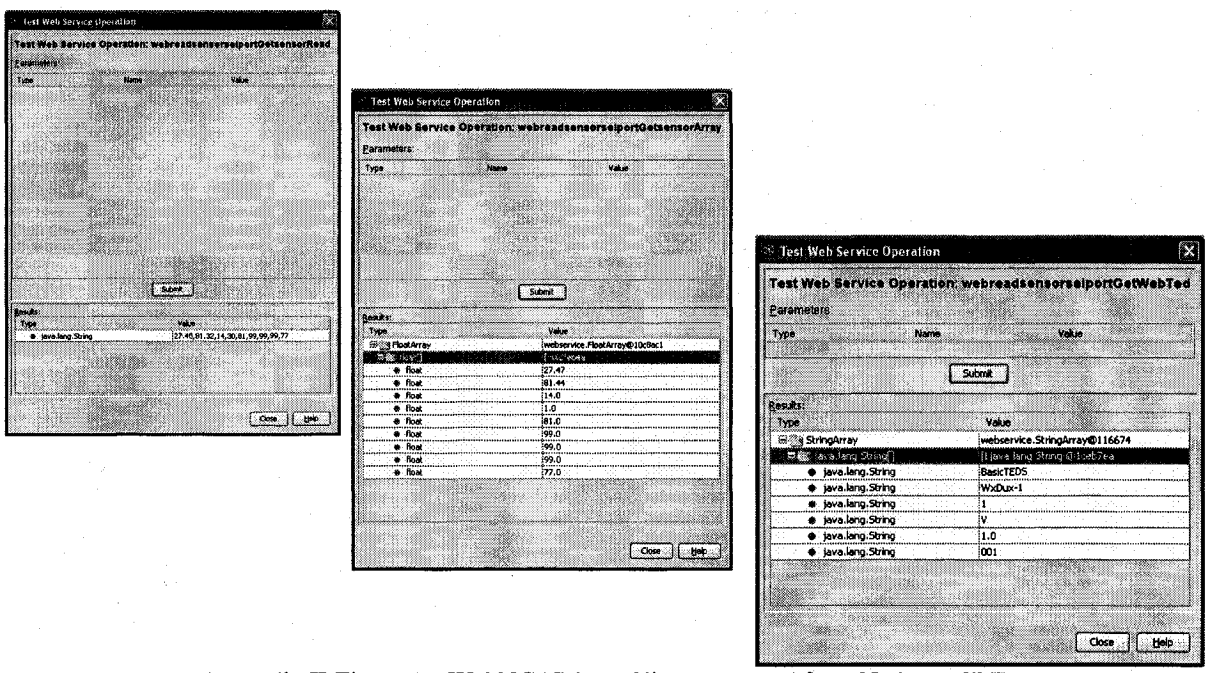
Appendix II-Figure 1.3 WebNCAP Client Prototype JSP Graphing Service

In Figure 1.3, a WebNCAP Client JSP page provides a prototype graphing/visualization web-service, which updates a graph result, based on the Wx-Dux-2 sensor's readings. The graph results are acquired by calling the internal Sensor Read web-service onboard the WebNCAP Server.



Appendix II-Figure 1.4 WebNCAP Server Netbeans IDE Environment w/ HTTP Monitor

In Figure 1.4 the Netbeans IDE tool environment is illustrated, an HTTP Monitor provides connection details for the SJAS application server, hosting the WebNCAP Client and Server.



Appendix II-Figure 1.5 WebNCAP Test Client generated from Netbeans IDE

In Figure 1.5, the Netbeans IDE Test Client tool environment is illustrated, for the available web-services offered for the Transducer Block, used in the Performance measurement Chapter 6.

Appendix III

WebNCAP Server - WSDL service definition for application specific 1451.1 Transducer Block


```
<?xml version="1.0" encoding="UTF-8"?><definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="urn:WebReadSensor/wsdl" xmlns:ns2="urn:WebReadSensor/types/arrays/java/lang"
xmlns:ns3="urn:WebReadSensor/types/arrays/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="WebReadSensor"
targetNamespace="urn:WebReadSensor/wsdl">
 <types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tns="urn:WebReadSensor/types/arrays/java/lang"
xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="urn:WebReadSensor/types/arrays/java/lang">
   <import namespace="urn:WebReadSensor/types/arrays/"/>
   <complexType name="StringArray">
    <sequence>
     <element name="value" type="string" nillable="true" minOccurs="0"
maxOccurs="unbounded"/></sequence></complexType></schema>
  <schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tns="urn:WebReadSensor/types/arrays/" xmlns:soap11-
enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="urn:WebReadSensor/types/arrays/">
   <import namespace="urn:WebReadSensor/types/arrays/java/lang"/>
   <complexType name="floatArray">
    <sequence>
     <element name="value" type="float" minOccurs="0"
maxOccurs="unbounded"/></sequence></complexType></schema></types>
 <message name="WebReadSensorSEI_CloseWeatherduck"/>
 <message name="WebReadSensorSEI_CloseWeatherduckResponse"/>
 <message name="WebReadSensorSEI_ReadSensor"/>
 <message name="WebReadSensorSEI_ReadSensorResponse">
  <part name="result" type="xsd:string"/></message>
 <message name="WebReadSensorSEI_getWebTed"/>
 <message name="WebReadSensorSEI_getWebTedResponse">
  <part name="result" type="ns2:StringArray"/></message>
 <message name="WebReadSensorSEI_getsensorArray"/>
 <message name="WebReadSensorSEI_getsensorArrayResponse">
  <part name="result" type="ns3:floatArray"/></message>
 <message name="WebReadSensorSEI_getsensorRead"/>
 <message name="WebReadSensorSEI_getsensorReadResponse">
  <part name="result" type="xsd:string"/></message>
 <portType name="WebReadSensorSEI">
  <operation name="CloseWeatherduck">
   <input message="tns:WebReadSensorSEI_CloseWeatherduck"/>
   <output message="tns:WebReadSensorSEI_CloseWeatherduckResponse"/></operation>
  <operation name="ReadSensor">
   <input message="tns:WebReadSensorSEI_ReadSensor"/>
   <output message="tns:WebReadSensorSEI_ReadSensorResponse"/></operation>
  <operation name="getWebTed">
   <input message="tns:WebReadSensorSEI_getWebTed"/>
   <output message="tns:WebReadSensorSEI_getWebTedResponse"/></operation>
  <operation name="getsensorArray">
   <input message="tns:WebReadSensorSEI_getsensorArray"/>
   <output message="tns:WebReadSensorSEI_getsensorArrayResponse"/></operation>
  <operation name="getsensorRead">
   <input message="tns:WebReadSensorSEI_getsensorRead"/>
   <output message="tns:WebReadSensorSEI_getsensorReadResponse"/></operation></portType>
 <binding name="WebReadSensorSEIBinding" type="tns:WebReadSensorSEI">
```

<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
<operation name="CloseWeatherduck">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></input>
<output>
<soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></output></operation>
<operation name="ReadSensor">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></input>
<output>
<soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></output></operation>
<operation name="getWebTed">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></input>
<output>
<soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></output></operation>
<operation name="getsensorArray">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></input>
<output>
<soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></output></operation>
<operation name="getsensorRead">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></input>
<output>
<soap:body use="literal" namespace="urn:WebReadSensor/wsdl"/></output></operation></binding>
<service name="WebReadSensor">
<port name="WebReadSensorSEIPort" binding="tns:WebReadSensorSEIBinding">
<soap:address location="http://137.122.91.31:8080/WebNCAP/WebReadSensor"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"/></port></service></definitions>

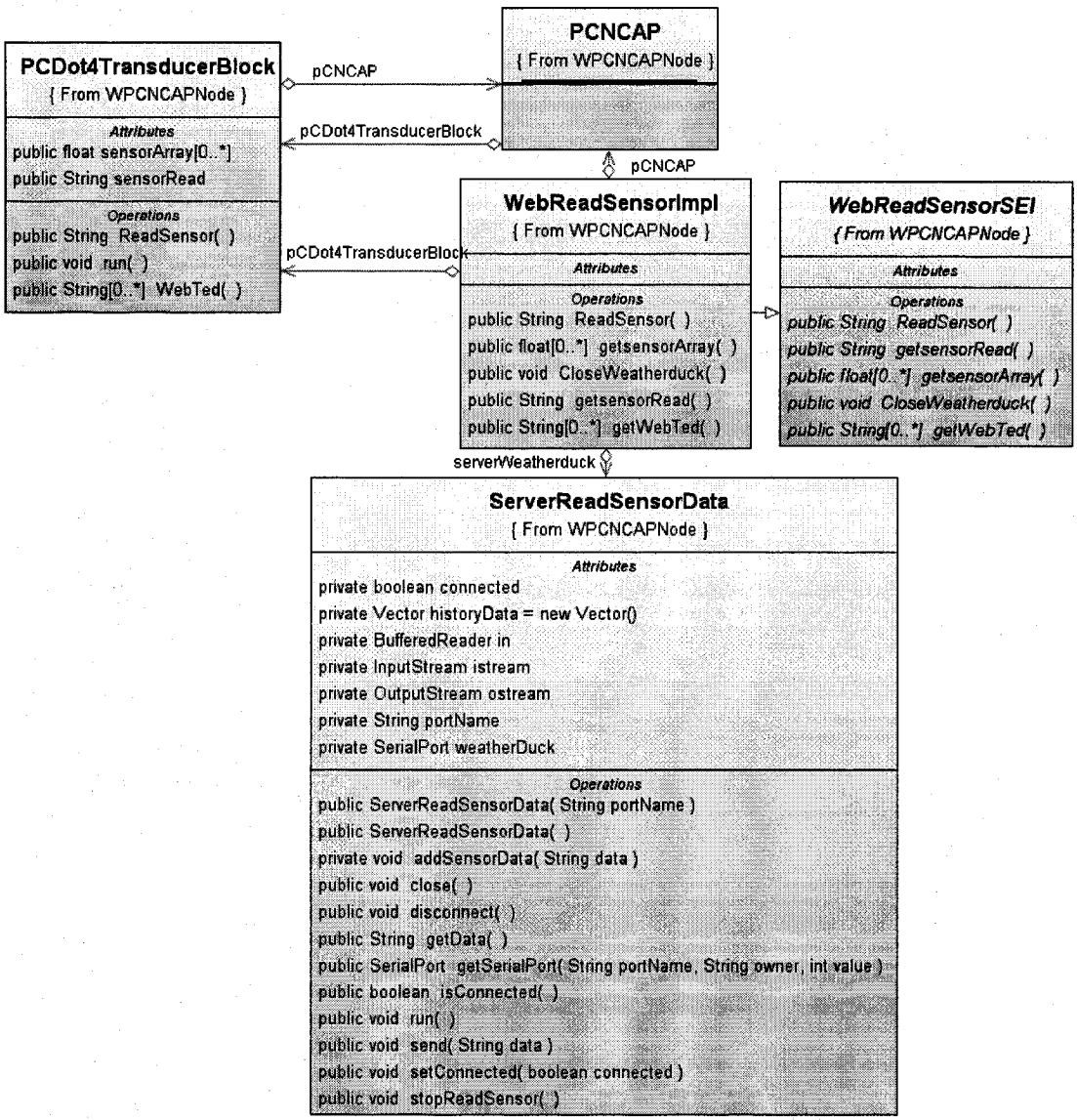
WebNCAP Server - WSDL service definition for application specific 1451.1 Function Block

```
<?xml version="1.0" encoding="UTF-8"?><definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="urn:WebPCFunctionBlock/wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns2="urn:WebPCFunctionBlock/types/arrays/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
name="WebPCFunctionBlock" targetNamespace="urn:WebPCFunctionBlock/wsdl">
 <types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tns="urn:WebPCFunctionBlock/types/arrays/"
xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="urn:WebPCFunctionBlock/types/arrays/">
   <complexType name="floatArray">
    <sequence>
     <element name="value" type="float" minOccurs="0"
maxOccurs="unbounded"/></sequence></complexType></schema></types>
 <message name="WebPCFunctionBlockSEI_DisableReadSensor"/>
 <message name="WebPCFunctionBlockSEI_DisableReadSensorResponse"/>
 <message name="WebPCFunctionBlockSEI_ReadSensor"/>
 <message name="WebPCFunctionBlockSEI_ReadSensorResponse">
  <part name="result" type="xsd:string"/></message>
 <message name="WebPCFunctionBlockSEI_StartPCNCAP"/>
 <message name="WebPCFunctionBlockSEI_StartPCNCAPResponse"/>
 <message name="WebPCFunctionBlockSEI_TestCalibration"/>
 <message name="WebPCFunctionBlockSEI_TestCalibrationResponse">
  <part name="result" type="ns2:floatArray"/></message>
 <message name="WebPCFunctionBlockSEI_disableReady"/>
 <message name="WebPCFunctionBlockSEI_disableReadyResponse"/>
 <portType name="WebPCFunctionBlockSEI">
  <operation name="DisableReadSensor">
   <input message="tns:WebPCFunctionBlockSEI_DisableReadSensor"/>
   <output message="tns:WebPCFunctionBlockSEI_DisableReadSensorResponse"/></operation>
  <operation name="ReadSensor">
   <input message="tns:WebPCFunctionBlockSEI_ReadSensor"/>
   <output message="tns:WebPCFunctionBlockSEI_ReadSensorResponse"/></operation>
  <operation name="StartPCNCAP">
   <input message="tns:WebPCFunctionBlockSEI_StartPCNCAP"/>
   <output message="tns:WebPCFunctionBlockSEI_StartPCNCAPResponse"/></operation>
  <operation name="TestCalibration">
   <input message="tns:WebPCFunctionBlockSEI_TestCalibration"/>
   <output message="tns:WebPCFunctionBlockSEI_TestCalibrationResponse"/></operation>
  <operation name="disableReady">
   <input message="tns:WebPCFunctionBlockSEI_disableReady"/>
   <output message="tns:WebPCFunctionBlockSEI_disableReadyResponse"/></operation></portType>
 <binding name="WebPCFunctionBlockSEIBinding" type="tns:WebPCFunctionBlockSEI">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
  <operation name="DisableReadSensor">
   <soap:operation soapAction=""/>
   <input>
    <soap:body use="literal" namespace="urn:WebPCFunctionBlock/wsdl"/></input>
   <output>
    <soap:body use="literal" namespace="urn:WebPCFunctionBlock/wsdl"/></output></operation>
  <operation name="ReadSensor">
   <soap:operation soapAction=""/>
   <input>
    <soap:body use="literal" namespace="urn:WebPCFunctionBlock/wsdl"/></input>
   <output>
    <soap:body use="literal" namespace="urn:WebPCFunctionBlock/wsdl"/></output></operation>
  <operation name="StartPCNCAP">
   <soap:operation soapAction=""/>
   <input>
    <soap:body use="literal" namespace="urn:WebPCFunctionBlock/wsdl"/></input>
```

```
    <output>
      <soap:body use="literal" namespace="urn:WebPCFunctionBlock/wsdl"/></output></operation>
   <operation name="TestCalibration">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal" namespace="urn:WebPCFunctionBlock/wsdl"/></input>
    <output>
      <soap:body use="literal" namespace="urn:WebPCFunctionBlock/wsdl"/></output></operation>
   <operation name="disableReady">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal" namespace="urn:WebPCFunctionBlock/wsdl"/></input>
    <output>
      <soap:body use="literal" namespace="urn:WebPCFunctionBlock/wsdl"/></output></operation></binding>
 <service name="WebPCFunctionBlock">
   <port name="WebPCFunctionBlockSEIPort" binding="tns:WebPCFunctionBlockSEIBinding">
    <soap:address location="http://137.122.91.31:8080/WebNCAP/WebPCFunctionBlock"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"/></port></service></definitions>
```

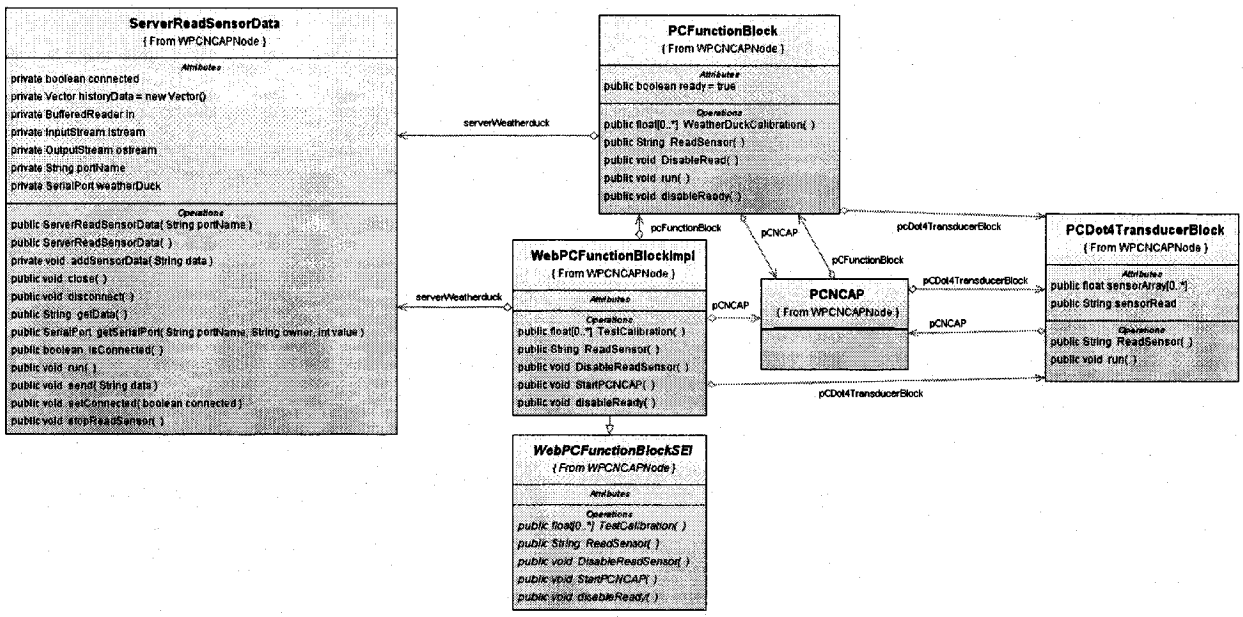
Appendix IV

This section provides the UML diagrams for the WebNCAP Server's web-services, namely a 1451 Transducer Block web-service for acquiring sensor data, along with a 1451 Function Block for onboard sensor data processing. The Web Read Sensor web service provides remote accessibility to the Transducer Block of the PC NCAP Application. The associated UML dependence diagram/class diagram is illustrated below in Figure 1.0:



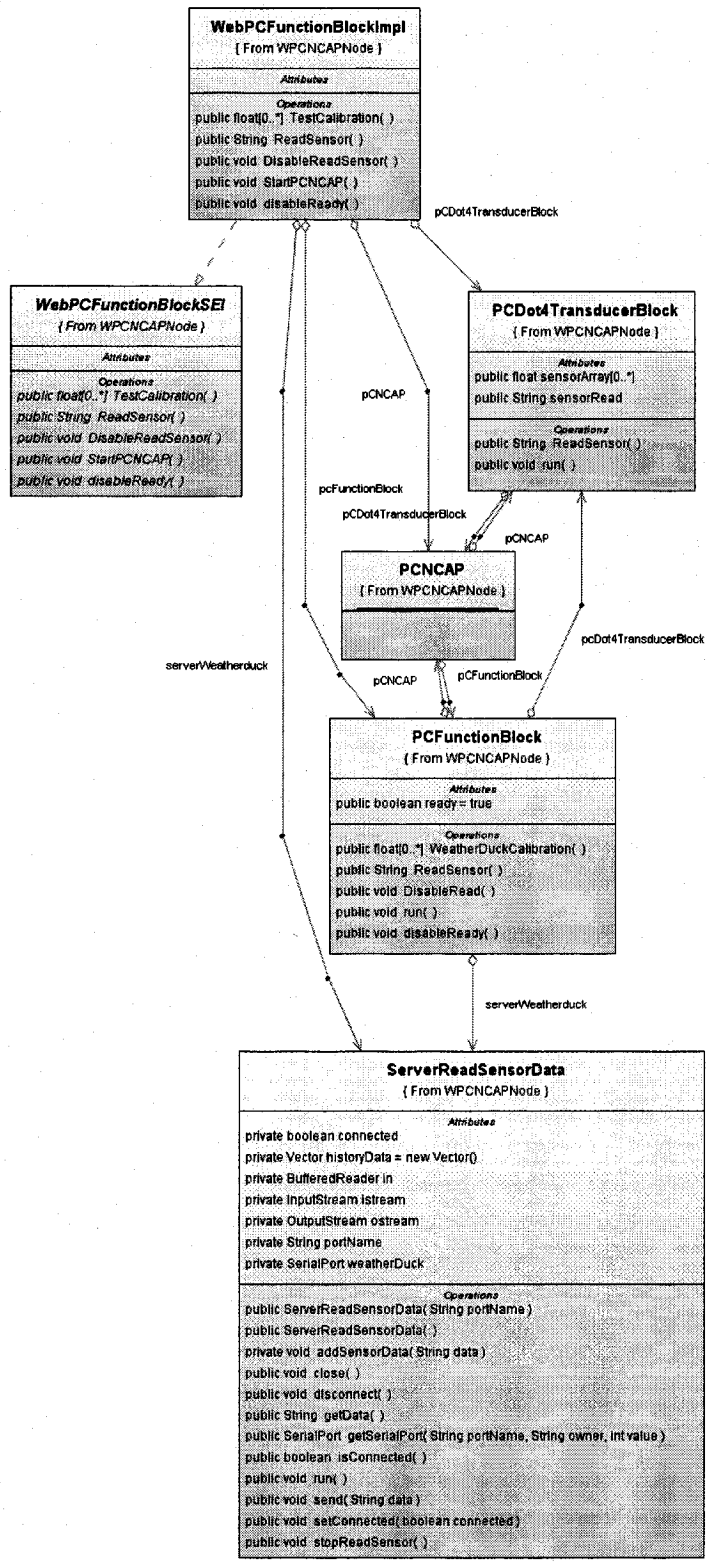
Appendix IV-Figure 1.0 WebRead Sensor Service for 1451.1 PC Dot 4 Transducer Block

The Function Block web service provides remote accessibility to the Transducer Block along with sensory data processing inside the PC NCAP Application. The associated UML dependence diagram/class diagram is illustrated in Figure 1.1 and Figure 1.2:



Appendix IV-Figure 1.1 Web PC Function Block Service for 1451.1 PC Function Block

An additional Function Block, web-service UML class/dependency diagram follows.



Appendix IV-Figure 1.2 Web PC Function Block Service for 1451.1 PC Function Block

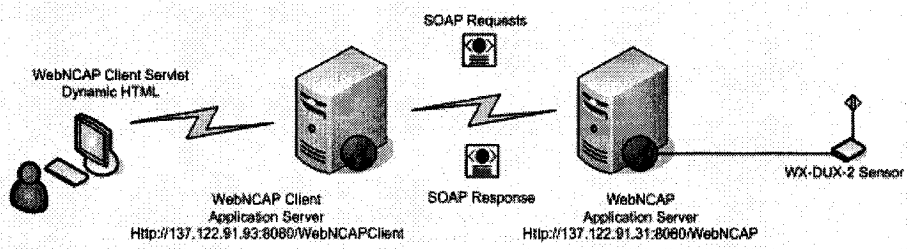
Appendix V

Extended Performance Measurement results are listed for a *sensorRead* web service (WS) for a 1451.1 Transducer Block utilizing the Web Read Sensor Service definition.

Extended Performance Measurement Scenario 1 - B2B/M2M WS

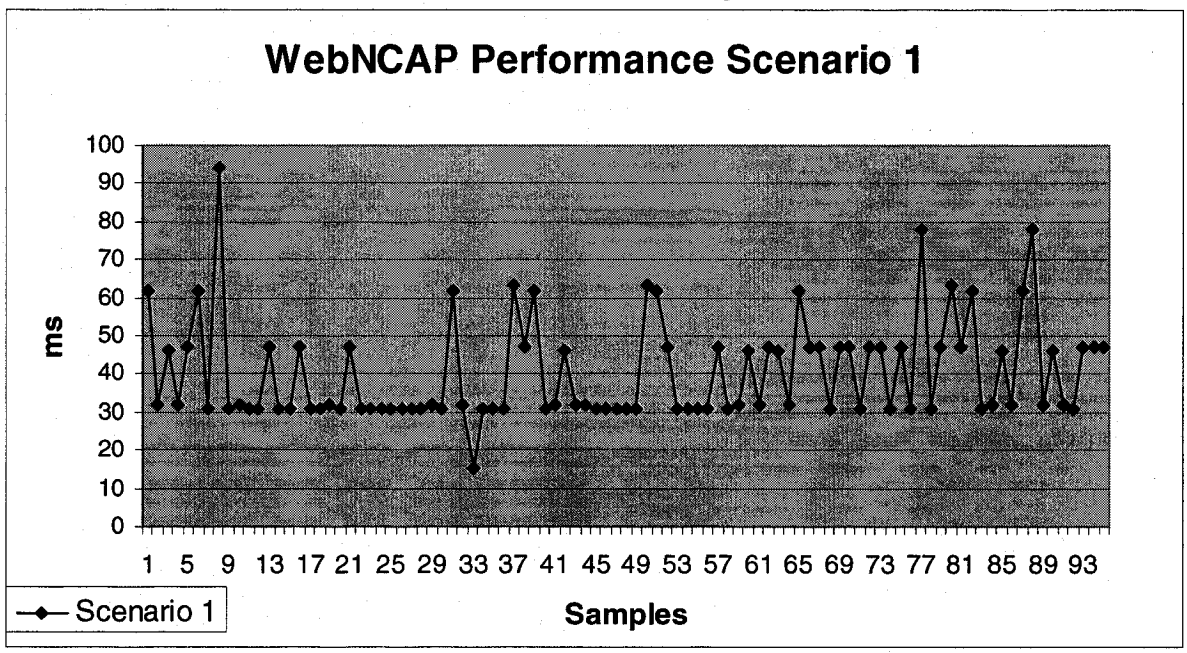
The first scenario provides the ability for testing the traditional web-service architecture B2B server interaction pattern, between two application servers. Two application servers in this case represent two remote NCAPs, which are co-located within the same subnet.

The significance of this test provides the ability to acquire/disseminate measurement data within an internal subnet (network) communication on-site. (This resembles the concept of network instruments promoting a M2M interface concept.)



Appendix V-Figure 1.0: Web Service Performance Scenario 1 for B2B/M2M Interface

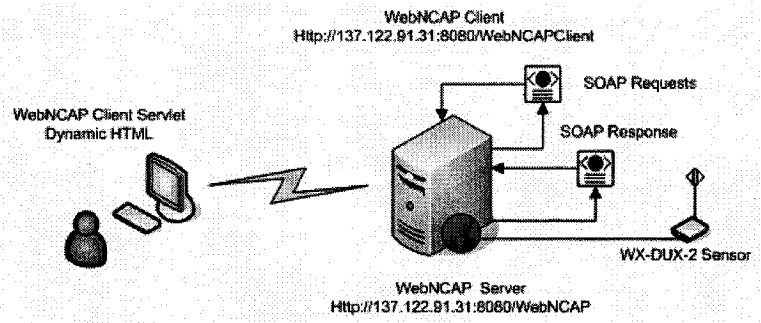
Web Services performance scenario test 1, based on the *sensorRead* operation of the PC Dot 4 Transducer Block.



Appendix V-Figure 1.1: B2B/M2M Performance Scenario 1 Test

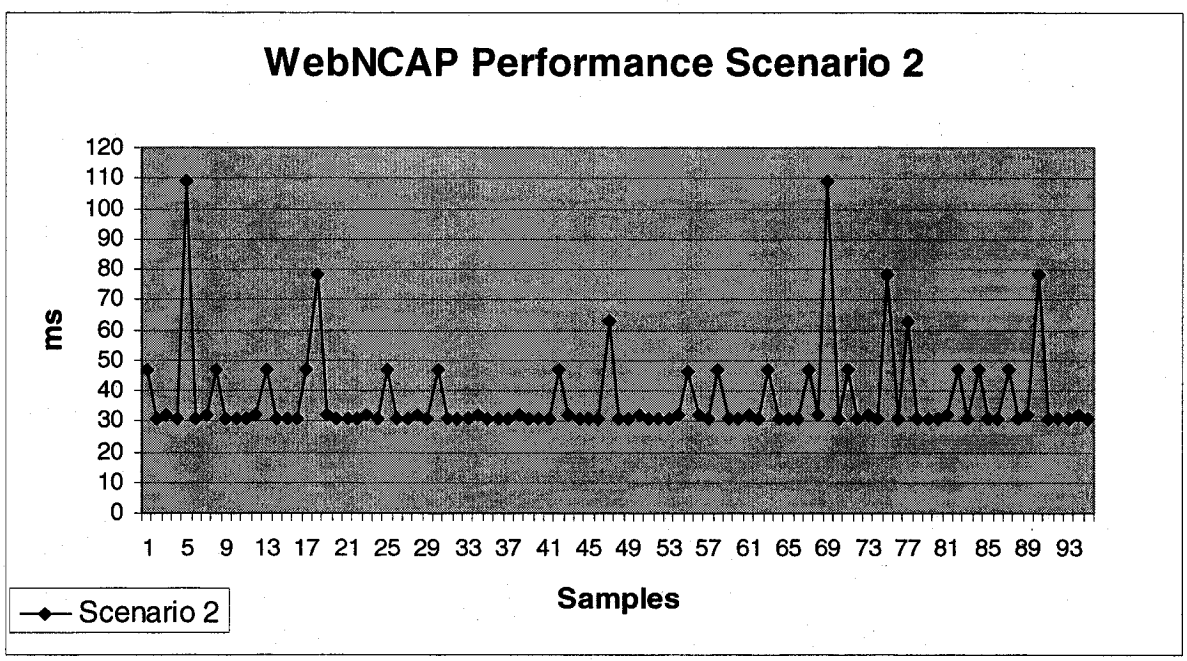
Extended Performance Measurement Scenario 2 - Local/Onboard WS

The second scenario provides the ability for testing an onboard local web-service architecture, whereby an NCAP hosts internal web-service calls for invoking measurements. (This resembles a web-server concept.) The significance of this test provides the ability for a first-responder to acquire and communicate with an onsite NCAP. The NCAP acts as a web-server, hosting a dynamic HTML web page for web-service interaction.



Appendix V-Figure 1.2: Internal Web Service Performance Scenario 1 Testing

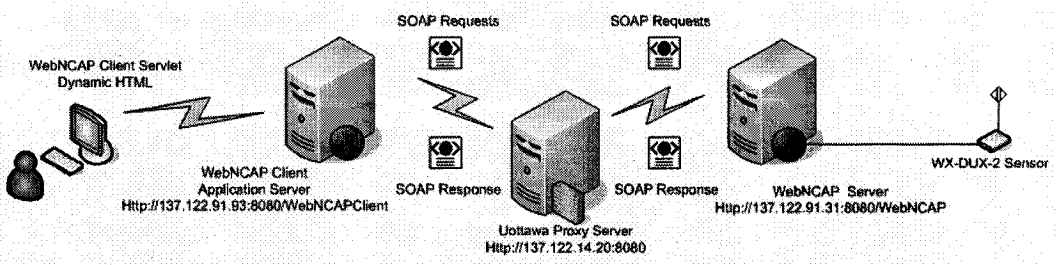
Web Services performance scenario test 2, based on the *sensorRead* operation of the PC Dot 4 Transducer Block.



Appendix V-Figure 1.3: Internal Web Service Performance Scenario 2 Test

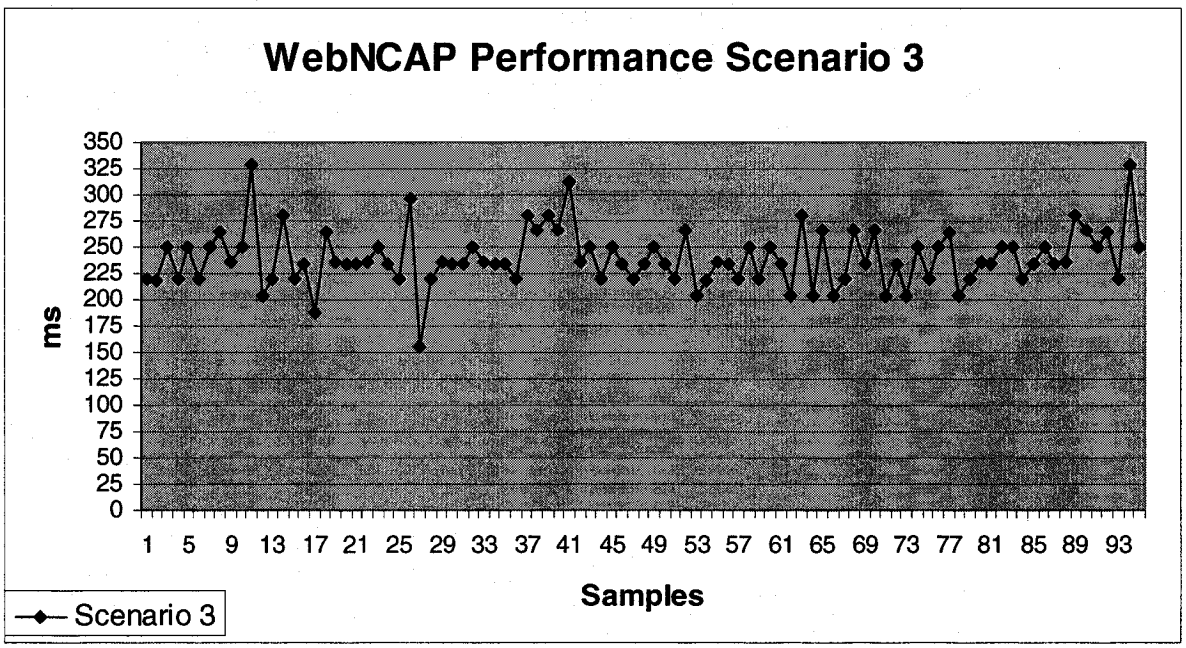
Extended Performance Measurement Scenario 3 - B2B via Proxy Server (VPN) WS

The third scenario provides the ability for testing the traditional web-service architecture B2B server interaction pattern via a proxy server, between two application servers. A proxy server is required for authentication and entry into a subnet for communication between monitoring NCAPs. This scenario provides a simulation for web-service performance results (latency measured at the client side) for web-service SOAP messages traversing a proxy server.



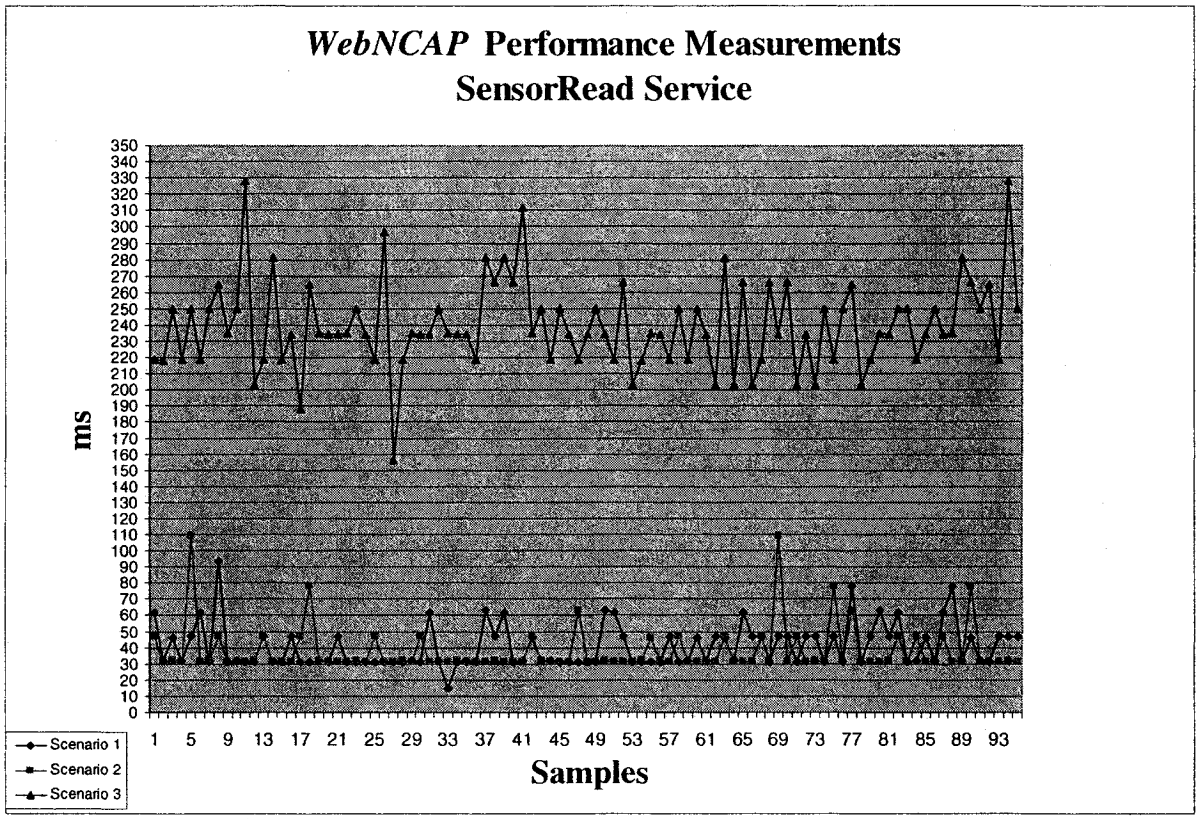
Appendix V-Figure 1.4: B2B/M2M via a Proxy Server Web Service Performance Scenario 1 Testing

Web Services performance scenario test 3, based on the *sensorRead* operation of the PC Dot 4 Transducer Block.



Appendix V-Figure 1.5: B2B/M2M via Proxy Server Web Service Performance Scenario 3 Test

Extended Performance Results – Overlay of Tested Scenarios



Appendix V-Figure 1.6: Overlay Results of Extended Web Service Performance Scenario 1, Scenario 2, and Scenario 3 Testing

An overlay summary for the resulting client-side SOAP communication latency is presented in Figure 1.6 above. Moreover, the sample mean, standard deviation and confidence intervals are acquired from the performance scenario resulting data sets in Figure 1.6. Three confidence levels are examined, 95%, 99%, and 99.9%, which are applied to the sample mean, providing lower and upper bounds estimates.

Performance Measurement Analysis

SOAP Measured Response Time from WebNCAP Client accessing *sensorRead* web service

***sensorRead* Web Service**	**B2B WS Scenario 1**	**Local WS Scenario 2**	**Proxy Server B2B Scenario 3**
Mean (average)	40.59 ms	37.50 ms	239.63 ms
Standard Deviation	13.51 ms	14.82 ms	27.64 ms
Confidence Interval 95%	± 2.72 [37.87,43.31] ms	± 2.98 [34.52,40.48] ms	± 5.56 [234.07,245.19] ms
Confidence Interval 99%	± 3.57 [37.02,44.16] ms	± 3.92 [33.58,41.42] ms	± 7.30 [232.33,246.93] ms
Confidence Interval 99.9%	± 4.56 [36.03,45.15] ms	± 5.00 [32.50,42.51] ms	± 9.33 [230.30,248.96] ms

Note: The extended results scenario provides a larger sample size distribution (95 samples), thus resulting in a lower confidence interval in comparison to the initial web service performance tests conducted (Chap. 6). The confidence interval provides the ability to offer a statistical range (lower/upper bound) for the acquired sample means (average) for each WebNCAP performance scenario's client-side SOAP message latency.