Université d'Ottawa · University of Ottawa

Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Igor SHAROVAR

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

M. A. Sc. (Electrical Engineering)

GRADE - DEGREE

Department of Electrical Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

Smart Sensor with Network Plug and Play Capabilities

V. Groza

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

D. Petriu

J. Zhao

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Smart Sensor with network Plug and Play capabilities

by

Igor Sharovar

School of Information Technology and Engineering

Submitted in partial fulfillment of the requirements
for the degree of Master of Applied Science

Faculty of Graduate and Postdoctoral Studies
The University of Ottawa
Ottawa, Ontario
May 2004

Your file Votre référence
ISBN: 0-494-01606-X
Our file Notre référence
ISBN: 0-494-01606-X

ABSTRACT

This thesis proposes enhancements to the network capabilities of the IEEE 1451.1 standard. The IEEE 1451.1 standard belongs to the popularly used, innovative group of specifications called Smart Sensor, . It defines a standardized way to design and develop transducers that can be easily connected to a sensor network. The IEEE 1451.1 standard defines a Common Object Model , which may be used to design the network capability of a Smart Sensor device, but does not define a standard way to connect Smart Sensor devices to different types of networks.

A variety of modern technologies is applied today, allowing ease with which network devices connect to networks. One such technology is Universal Plug and Play (UPnP), which defines the mechanism and protocol for the discovery of network devices, as well as their control and event functions.

The goal of this thesis is to expand the IEEE 1451.1 standard to make Smart Sensor devices capable of working as UPnP devices. Achieving this will enable the design of an UPnP Smart Sensor that will not only connect different types of transducers to a sensor network, but will also provide a standard and easy mechanism to connect these devices to different networks, including the Internet, while minimizing the effort required for their design and implementation.

Acknowledgements

I would like to gratefully acknowledge the enthusiastic supervision of Dr. Voicu Groza during this work, and the great research opportunity he provided me. Dr. Groza allowed me to explore the recent technology in transducer design and new network protocols and specifications that permit connection of pervasive devices in easy ways. His extensive experience and in-depth knowledge of sensor design and networking guided me during my work.

I would like also to thank to my wife and my children for their love and consistent support that definitely helped me to finish this thesis.

TABLE OF CONTENT

DETAILED TABLE OF CONTENT

List of Figures

List of Tables

Acronyms

API	Application Program Interface
ARP	Address Resolution Protocol
DHCP	Dynamic Host Configuration Protocol
GENA	General Event Notification Architecture
HTTP	Hypertext transport protocol
ICMP	Internet Control Message Protocol
IP	Internet protocol
NCAP	Network Capable Application Processor
OS	Operation System
RMI	Remote Method Invocation
SOAP	Simple Object Application Protocol
SSDP	Simple Service Discovery Protocol
STIM	Smart Transducer Interface Module
TCP	Transmission Control Protocol
TEDS	Transducer electronic data sheet
TTL	time to live
UC	Ubiquitous computing
UDN	Unique device name
UDP	User Datagram Protocol
URI	Universal unique identifiers
URL	Uniform Resource Locator
UPC	Universal Product Code
UPnP	Universal Plug and Play
USN	Unique Service Name
UUID	Universal unique identifier
XML	Extensible Mark-up Language

1 Introduction

1.1 Background

Our generation today lives in the second wave of computing technology, where personal computing is predominant. The first wave was the era of mainframes, where many people were to share one unit. The third wave, beginning at this time, is the era of ubiquitous computing (UC), which forces the computer to live in simultaneity with people, as a part of their lives. The main goal of ubiquitous computing is to design enhanced computers made available through our physical environment, while maintaining them as invisible to the user [1].

Ubiquitous computing will soon be as popular as personal computing is today. Some scientists predict this should happen between 2010 and 2020. The main feature of UC devices will be the sharing of information amongst them, mostly via the Internet. Today "thin client" devices that use the Internet and cost a few hundred dollars are prevalent. In UC we will see the creation of "thin server" devices with full Internet server capabilities costing only ten dollars or less. The next generation of Internet Protocol, such as IPv6 [2], will then resolve the addressing issue of IP that is now critical in IPv4.

The move from the PC to UC era has already begun. There are now two widespread technologies that are ideal candidates to become UC devices: they are embedded devices and the Internet. People today use many devices that contain microprocessors. These include: clocks, microwave ovens, TV remote controls etc. By connecting these devices to the Internet, we will be capable of connecting millions of information sources. We could, hence, have a hundred information delivery systems even in our homes. Examples include: Entertained Centres that load new movies from the Internet, clocks that correct time after a power failure, coffeemakers that can be operated using a remote control, and home security systems that can be monitored by owners outside their homes.

Creating such devices require standardization in two directions: standardizing how to connect a specific type of device to the standard network component, and a specification on how to connect, discover, and control UC devices.

An example of the first type of standardization is a group of standards, IEEE1451, called the Smart Sensor. These standards define the specification needed to design transducers capable of working with different types of network interfaces. A part of the standard group called IEEE 1451.1 defines the Common Object model that creates a network interface of a Smart Sensor device.

An example of the second type of standardization is the Universal Plug and Play (UPnP) specification that defines how to connect a device with network capabilities at home or to a public network. UPnP specifies discovery of devices in networks, and also their control and event functions.

1.2 Smart Sensor Standards

The Smart Sensor standards are a set of specifications used to design network capable transducers (sensors and actuators), from the interface to the transducer itself, using object-model representation of network behaviour, attributes and network communication.

The standards have been developed through participation of many companies that are involved in sensor measurement and control, as well as control network companies [39]. The goal of such standards is to connect analog transducers to existing digital networks without a limit with regards to the manufacturer of the transducer or the type of network, while allowing sensors and actuators to share common buses in order to reduce complexity in wiring and reduce costs [41].

Today there are four Smart Sensor standards defined: IEEE 1451.1, IEEE 1451.2, IEEE 1451.3 and IEEE 1451.4.

The first standard, IEEE1451.1, defines the Common Object model for the network and transducer interfaces. The core of the standard is the definition of Network Capable Application Processor (NCAP) that allows the design of a flexible, modular network interface, the measurement and control functions involved, and a transducer

interface. Any control network device may be connected to any transducer, or a group of transducers, with appropriately configured NCAP.

The IEEE1451.2 standard defines the format of the Transducer Electronic Data Sheet (TEDS), a memory device attached to the transducer, which stores transducer identification, calibration and correction data. TEDS is the part of the Smart Transducer Interface Module (STIM) that is also defined by the standard. STIM also specifies a protocol to communicate with NCAP microprocessors that allow any transducer or a group of transducers to send (or receive) information to (or from) the network.

IEEE1451.3 defines a specification to design transducers that can reside on a common set of wires.

IEEE1451.4 defines a Mixed Mode Interface for analog transducers with analog and digital operating modes.

1.3 Universal Plug and Play Network

Because of the ubiquity of the Internet and autonomous attempts to integrate different devices, several technologies and specifications have appeared, enabling ease of discovery and control of network devices. One such technology is UPnP. Although UPnP is a new technology, many companies support it, namely: Microsoft, Intel, HP, and Philips.

UPnP inherits its name from the Plug-and-Play technology that simplifies installation and configuration of devices into PCs. UPnP extends Plug-and-Play to the network environment. The purpose of UPnP is to automate installation and configuration of a (small) network as much as possible. This means that UPnP devices can join or leave the network without any effort from a network administrator. UPnP capable devices offer services to a user and inform the user about them. The control of the UPnP can be automated with some optional level of manual control.

UPnP specification defines two types of UPnP devices: a control point and an UPnP device.

The UPnP device plays a role as a server and the control point works as a client. The control point can be run as a separate process or run in a web browser.

UPnP relies on well-known standards and network protocols such as TCP/IP, HTTP, Dynamic Host Configuration Protocol (DHCP) and XML. This makes the implementation of UPnP devices easier compared to the usage of other network protocols, because of the wide use and availability of TCP/IP and HTTP. Since UPnP does not specify an Application Program Interface (API), it is independent of any particular operating system or hardware. Defining the API is left to device vendors.

In general, UPnP includes five major functions: assigning IP addresses, device discovery, control, and event and presentation functions.

A device must have an IP address to work in a UPnP network. The IP address could be hard coded, but the most flexible approach is to assign an IP address when the device enters the network. The IP address can be assigned either using a DHCP server or, if DHCP service is not available, using an Auto-IP algorithm.

The next important UPnP function is the device discovery mechanism, which is based on the Simple Service Discovery Protocol (SSDP). The discovery of the device can be performed either by a device's announcement or a control point discovery request.

The UPnP control function allows a control point to directly get information from a device. It based on Simple Object Application Protocol (SOAP).

The UPnP event function is used to notify a *client point* by changing a device's variables that reflect a current state of device. The event function includes an event subscribing mechanism and event notification.

The UPnP presentation function allows using a standard web browser to control a UPnP device. This function is optional and relies on vendor specific implementation, but is a useful tool to a user.

1.4 Motivation and Thesis Objectives

Despite the well-defined Common Object model of the network interface, the Smart Sensor standards do not specify a mechanism to work in a network environment, where a device could be easily and automatically attached, discovered, and started. The Smart Sensor only defines Plug and Play mechanism on the level of transducer – NCAP.

It means that different types of sensors and actuators could be easily attached to and used in one processor that is designed according to IEEE1451.1.

Adding to IEEE1451.1, network Plug and Play features allow the design of a Smart Sensor device that can not only automatically attach transducers to its own transducer – the NCAP interface - but can also be automatically attached to networks. It will create complete Plug and Play capabilities of Smart Sensor from transducer hardware to network interface that will definitely minimize design effort for its implementation.

1.5 Thesis Contributions

The goal of this thesis is to add network Plug and Play features to Smart Sensor. Despite the other technologies that provide Plug and Play in networks, we choose UPnP as a mechanism to add network Plug and Play features to Smart Sensor because of its well-defined, device oriented mechanism, and its support by many influential companies.

We propose to modify the Common Object Model of IEEE 1451.1 by adding new classes that support UPnP capabilities. Some classes are inherited from existing IEEE1451.1 classes and the others are added as new classes in the IEEE1451.1 model. Adding new classes provides both a detailed specification and implementation support.

We also designed a UPnP Smart Service Template, as a part of the UPnP specification requirements. The template describes Smart Sensor functionalities that can be visible through UPnP network.

IEEE1451.1 network supports two communication models: a client-server, and a publisher-subscriber. It gives the general specification of these models, but does not give a mechanism of real implementation. The new UPnP NCAP specification inherits and modifies these models in terms of UPnP architecture. The client-server model of the new UPnP NCAP specification supports the UPnP control function, while the publisher-subscriber model of the new specification supports UPnP event function.

The modification of IEEE1451.1 standard defines mechanisms to support the assignment of network addresses, registration and the discovery of Smart Sensor devices in networks that are absent in IEEE1451.1.

The modification of IEEE1451.1 also includes UPnP specific new classes that support the web server capabilities of the Smart Sensor devices, as well as classes necessary to communicate with XML and HTTP format files.

1.6 Thesis Organization

The remaining of the thesis is organized as follows. Chapter 2 provides a brief description of the current network Plug and Play technologies – Universal Plug and Play and Jini, followed by the description of two Smart Sensor standards – IEEE1451.1 and IEEE1451.2 in Chapter 3. In Chapter 4 we provide details of the Smart Sensor modification to make it network Plug and Play capable. Chapter 5 describes the implementation of an emulated UPnP Smart Sensor device and a UPnP NCAP control point. Chapter 6 concludes the thesis and provides a direction towards intended extensions.

2 Network Plug and Play Technologies

In this chapter we introduce the two most popular networks used today in Plug and Play technologies: UPnP and Jini [45]. Both technologies are characterized by following features:

- Ability to announce a device's presence in a network.
- Automatic discovery of a device's presence in a network.
- Ability to describe a device's capabilities.
- Device self-configuration.
- Control device's functions.
- Device's event functions.

2.1 Universal Plug and Play Architecture

2.1.1 Introduction

UPnP is a network architecture intended to connect different devices to user friendly, intelligent peer-to-peer networks [3]. UPnP can be used in all types of networks and it works on the base of TCP/IP and WWW technologies. In addition to the advantages of its Internet architecture, UPnP brings a new way to control and transfer data in home and public networks [23].

UPnP allows devices to easily connect to a network, automatically obtain IP addresses, and discover other devices and the services they provide to the network [24]. Disconnection of the device from the network is provided in a smooth manner without encountering undefined and unwanted states.

UPnP uses IP because of its proven ability to connect different physical media and also due to its high scalability. In some cases UPnP uses special bridges to connect IP with devices that cannot accommodate TCP/IP.

UPnP contains a list of protocols to provide communication between devices and control points. Control points manage attached devices by registering their services, assigning IP addresses, sending requests, and receiving devices events. Figure 1 shows a protocol stack of a UPnP network.

At the bottom of the stack lies IP. An IP *address* is obtained either through a DHCP client or, if the DHCP client is not available, by using the Auto-IP mechanism. Above the IP layer are protocols that are responsible for UPnP specific actions such as discovery, getting descriptions of devices, sending control messages, and receiving event messages. UPnP uses HTTP as the base protocol for communication [13], [21].

After getting an IP address, a device *advertises* its services to a control point. The control point, when it enters a network, searches for available UPnP aware devices. To implement the discovery mechanism, both devices and control points use SSDP. The device also sends a *description* to the control point that includes device offered services, and (optional) logical devices. In additional to the list of services, it also includes a URL for control, event messages, and presentation. The description can also provide specific vendor information such as the manufacturer's name, the serial number, and the URL of the vendor.

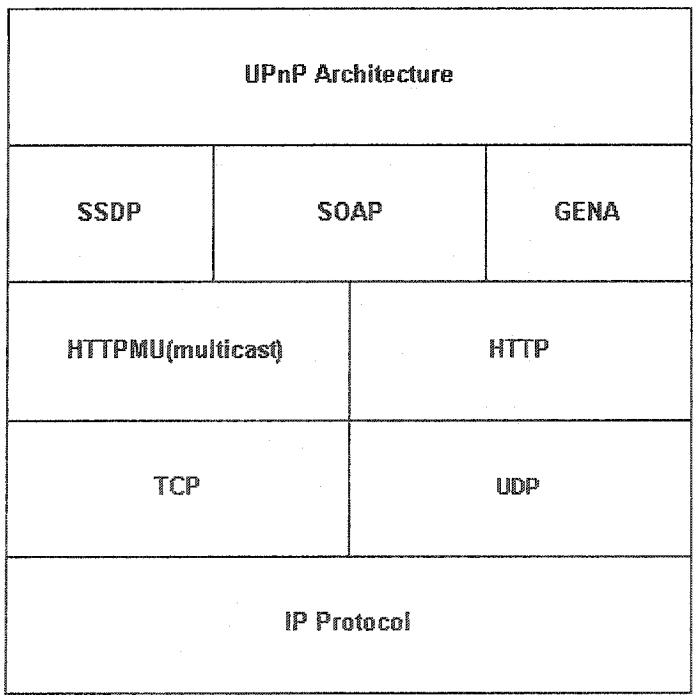


Figure 1: UPnP Network Stack

When the control point registers the device and gets the device description, it can receive event messages and can send control messages. The description of the device includes a list of event variables and their states. If an event variable changes its state, the device sends an event message. The control point can subscribe to a particular variable in order to get an event message when the variable changes. The device sends event messages using the General Event Notification Architecture (GENA) protocol.

Every UPnP device can have a presentation page. After loading this page from the device, a control point can present the page in a browser. By viewing this page, a user can determine capabilities of the device.

2.1.2 Addressing

The first step to connect to a network is to get an IP address. The device gets an IP address either through a DHCP server or, if a DHCP server is not available, by using the default assigned address by the Auto-IP mechanism.

2.1.2.1 Dynamic Host Configuration Protocol

DHCP is a protocol used to dynamically assign an IP address to a network node. A DHCP network consists of two parts: DHCP servers that have a repository of available IP addresses and provide distribution of IP addresses to clients, and DHCP clients who require IP addresses from the DHCP server. Information that is sent from the DHCP server to a DHCP client includes: an IP address, a subnet mask, a default gateway, and a domain name for the server. The communication transport protocol from the DHCP server to the DHCP client is provided by UDP. Port 67 of UDP is used for communication from the DHCP client to the DHCP server, while port 68 is used for communication in the opposite direction (from DHCP server to DHCP client). A DHCP server takes an available IP address from its pools and sends it to a DHCP client. When the DHCP client returns ownership of the IP address, the DHCP server can then reuse it.

There are several mechanisms of assigning IP addresses:

- Automatic allocation: the DHCP server assigns an IP address to the DHCP client.
- Manual allocation: a network administrator assigns a particular IP address to the DHCP client.
- Dynamic allocation: the DHCP client gets an IP address for a limited duration of time. After expiration, the DHCP client may get a different IP address.

An UPnP device may use all three methods, but the more preferable mechanism of assigning IP addresses is the dynamic allocation mechanism, which provides more flexibility.

When the device gets an IP address, it can use it for its communication. However, a user may prefer to use a static allocated name for the device. In this case the IP address and the host name for the device must map as an entry of a Domain Name Server database according to RFC 2136 [9], [15].

2.1.2.2 Auto-IP

If after a certain amount of time the device does not receive a response from the server, it will auto-configure itself. Usually UPnP devices use addresses in the 169.254/16 ranges, where the first and the last addresses are reserved and not used. The chosen address is tested to determine if it is in use. If in use, the device randomly selects the next address to avoid collisions of multiple devices attempting to allocate addresses. The device tests the chosen address by sending an Address Resolution Protocol (ARP) request. Usually the device sends two or four ARP probe requests within an interval of two seconds to ensure that the address is not allocated to another device.

If the device gets a response from the address, it considers the address as not available and chooses another. Even after choosing the IP address, the device should continue to detect address collisions. If at any time the device gets an ARP packet with its own address, it assumes that a collision has occurred, and sends a broadcast message with its address. If during the time interval the device gets an ARP packet with its address, it performs the Auto-IP algorithm again.

When an IP address is chosen, it can be kept in case the computer needs to be rebooted and attempts to reuse the address. If the UPnP device has an address in the range 169.254/16, it assumes that the address is valid within the boundaries of its local network and it does not send messages outside the network.

When the device uses Auto-IP, it has to periodically test the network for the existence of a DHCP server. The device sends, with an interval of five minutes, a DHCPDISCOVERY message. If the UPnP device receives DHCPOFFER message, it releases the IP address obtained by Auto-IP and cancels all advertisements performed using the old address. Figure 2 depicts the algorithm for configuring the IP address.

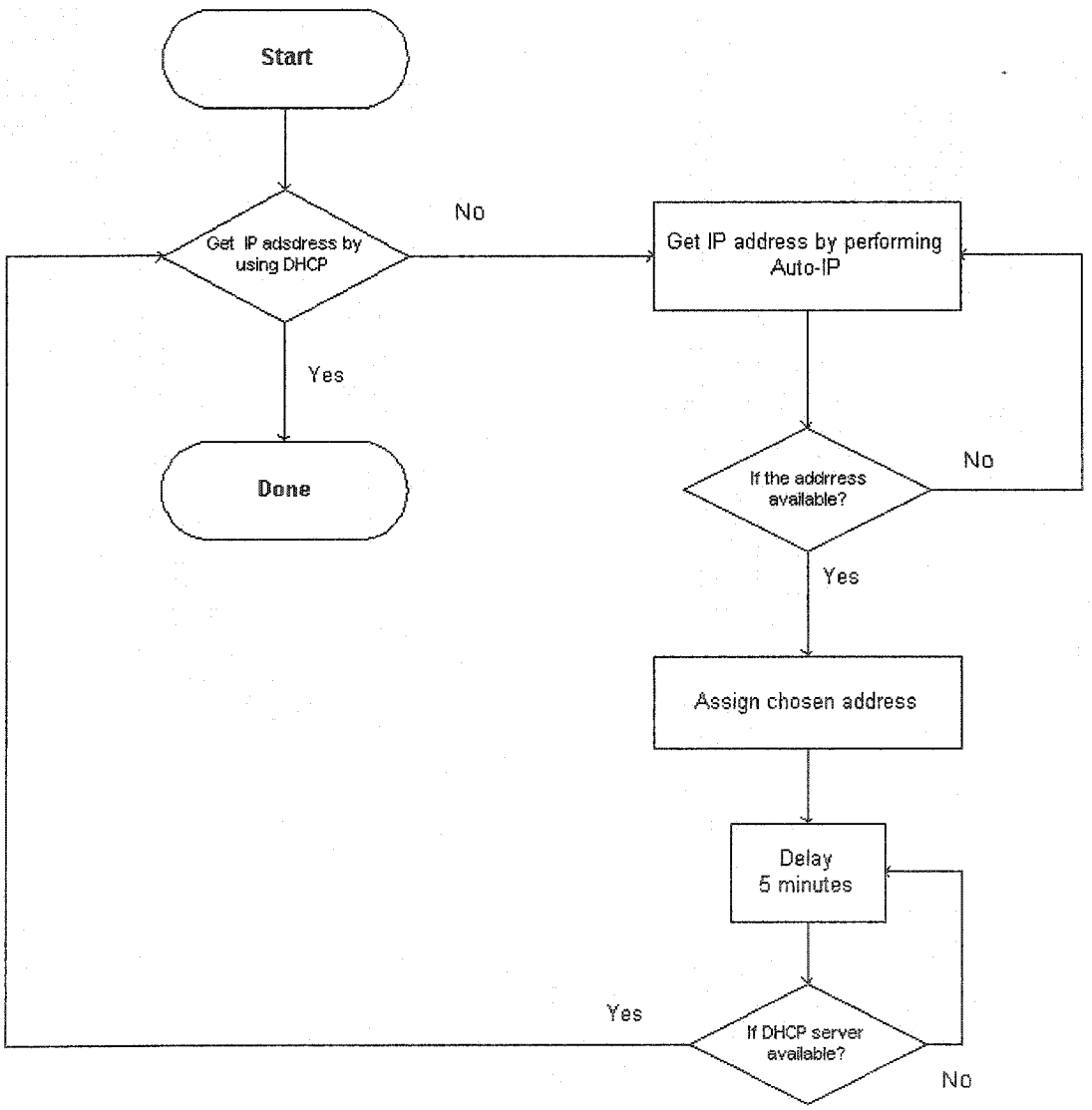


Figure 2. Configuration of IP Address in UPnP Device

2.1.3 Discovery

Discovery is a mechanism that allows an UPnP device to advertise itself and its services so that a UPnP control point could discover and select services that it needs. Discovery is the step that comes after Addressing (when the UPnP device gets an IP address) and it is the base for the rest of UPnP services. All UPnP services such as Control, Event, and Presentation depend on Discovery.

This service uses the decentralized approach to discovering services, as well as their resources, locations, and availabilities [25].

The Discovery service consists of two parts: UPnP device discovery and UPnP control point discovery. When a UPnP device comes into network, it advertises itself, its embedded devices, and its services by sending a message in a SSPD format. The message includes information about the type and the name of the device, its IP address, a list of embedded devices, a list of services, and the other information necessary to work with the device. All control points in the network listen to multicast messages and reply to the UPnP devices with which they intend to work.

When the control point comes into networks, it sends a search message to find UPnP devices with services of interest. The UPnP devices present in the network listen for search messages and reply with SSDP announcement messages to the control point. In the case of removing the device from the network or changing IP address, the UPnP device should send a message that cancels all previously advertised services. After setting up the new IP address, the device sends SSDP discovery messages again.

If an IP address is configured by DHCP, the time to live (TTL) variable should be set to the minimum value (default 4) to prevent congestion. TTL is not needed if the IP address is determined by Auto-IP, because the network space is restricted by the device's local network domain.

2.1.3.1 Advertisement

When an UPnP device joins the network it announces its presence. The device sends a multicast message to the specific address and port 239.255.255.250:1900 [17][37]. To send the message the device uses the GENA NOTIFY method. There are two types of advertisement messages: device available and device unavailable. The device available message is the *ssdp:alive* message and the device unavailable message is the *ssdp:bye-bye* message.

For each root device, embedded device, and service provided by the device, separate *ssdp:alive* and *ssdp:bye-bye* messages are sent. Each message contains information about a specific embedded device or service and also about its enclosed devices (services).

The control point listens to advertisement messages to detect new capabilities when they are available in the network.

2.1.3.1.1 Device Available and Removed Message

When the device joins the network, it advertises its root device, all embedded devices, and all device services by sending a NOTIFY message with *ssdp:alive*. For each type of service the UPnP device must multicast a predefined number of multicast messages. For the root device, the UPnP device sends three messages.

If the UPnP device or its embedded devices have multiple instances of a particular service, it sends only one message for each type . The order of service messages, however, is not important.

Choosing the appropriate value for the duration of advertisement involves a trade-off between minimizing network traffic and maximizing freshness of device status. The recommended value, 1800, compromises frequency of updating the device status with loading network traffic. In general, an UPnP device vendor should chose an appropriate value that corresponds to the device usage. For example, mobile devices that often leave the network should have shorter duration to more accurately reflect their device status.

To send an advertisement message, the UPnP device uses the UDP protocol, which is unreliable. To ensure that control points receive advertisements, the device should send messages more than once but less than thrice in order to avoid network congestion. When the advertisement expires, the device must send the advertisement again.

When a device is removed from a network it sends multicast NOTIFY messages with *ssdp:byebye*. Each NOTIFY *ssdp:byebye* messages must correspond to a NOTIFY *ssdp:alive* message.

The device removed message has a similar format to the NOTIFY *ssdp:alive* message except the NTS field that has the value *ssdp:byebye* Table 10 specifies the header of the NOTIFY *ssdp:byebye* message.

2.1.3.2 Search Method

2.1.3.2.1 Search Request Method

When an *UPnP control point* joins the network, it searches the *devices* of interest on the network. It sends the special HTTP M-SEARCH message to this address: 239.255.255.250:1900. The message includes the name of the device of interest

After receiving the M-SEARCH method a UPnP device sends a respond message.

2.1.4 Description

After discovering a UPnP device, a control point knows little about its functionality and services. The control point only gets the device's name, its service types, the device's *uuid,* and the URL of the device description document. To interact with the device, the control point should get more information about the capabilities of the device. This information is included in the device and services description documents. The device description document contains vendor specific information such as a manufacturer name, a vendor Web site URL, a model name, a serial number, a list of

services, a service name and a type, a service description URL, a service control URL, a service event URL, and a device presentation URL.

The service description document contains a list of actions that the service responds to and the list of parameters for each action. The service description document also includes the list of variables, which reflect the current state of the UPnP device. Each variable has a range, data type, and event characteristic.

Both the device and the service description documents are written on the base of the template, designed by UPnP Forum working committee, in XML format.

After receiving a device advertisement, the control point assumes that the device is presented in network and it can at any time request a description document. The HTTP GET command performs this request. The HTTP response contains the description document. After the advertisement expires, the control point assumes the description document is not longer valid. If the advertisement does not expire and the control point gets a new advertisement, it should request the description document again.

2.1.4.1 Service Description Document

The service description document includes detailed information about a service, such as the actions supported by the service, the action arguments, the state variables, and their data type, range, and events characteristics. Each service must to have one or more state variables as well as zero or more actions.

The major section in the service description document is the service state table. The service table contains a list of service state variables. The service state variables reflect the service state machine at run time. The service must have at least one state variable.

2.1.5 UPnP Control Function

UPnP architecture has Control function that allows a control point to call a remote function located on the UPnP device. Because of the HTTP and XML nature of

communication, UPnP uses SOAP to provide remote procedure calls [31]. SOAP uses HTTP to send messages and XML to describe the content of messages.

SOAP can not only use HTTP, but can also use other transport protocols to transport its messages. However, each transport protocol should establish its convention that allows carrying SOAP messages over the protocol. Using SOAP over HTTP is simple, because SOAP request/respond messages match HTTP commands. SOAP also defines an additional HTTP header to be sure that SOAP messages are not confused with other HTTP extensions.

SOAP transfers messages in an envelope consisting of two parts: a header and a body. The SOAP envelope's header consists of auxiliary information for authentication, transaction, and so on. The body of the envelope is an essential element. It includes a remote procedure call and the procedure's arguments.

The UPnP SOAP control message is sent as a HTTP POST message. The content type of the message is set in *text/XML* format indicating that the content of the message is an XML document. The header of the message also includes the element SOAPACTION that defines it as a HTTP SOAP message. The element also contains information about a service and the service's method to be invoked.

When the UPnP device gets a HTTP SOAP control request it must complete the request within 30 seconds.

2.1.6 Event Function in UPnP

When a control point starts, it receives the Device Description and Services Description documents. The Service Description document contains a list of state variables. The control point could subscribe to get notifications about changes in the states of these variables. Notifications are sent in the form of special UPnP events. UPnP uses GENA to support its event mechanism [33]. Despite the fact that there is not an official standard of GENA (there was only a draft), all UPnP publications and designs make reference to GENA as the major source of UPnP event notification mechanisms.

UPnP event notification uses a publisher/subscriber model. The publisher is the source of events and the subscriber is the receiver. In the UPnP architecture, the publisher is a UPnP device and the subscriber is a UPnP control point. To get events, the subscriber must subscribe to the publisher. A subscription is usually granted for a specific amount of time. To renew the subscription, the subscriber must subscribe again. If the subscriber is no longer interested in receiving event notification, it may unsubscribe. The publisher can also cancel the subscription.

GENA, as the UPnP event mechanism, uses IDs to control subscriptions. The subscriber gets the ID after sending a subscription message to the publisher. If the subscription is accepted, the publisher generates the ID and sends it, along with the duration of the subscription, to the subscriber. Subscription operations, such as renewal and cancellation, use the ID as reference for subscription.

The first subscription message from the publisher to subscriber is an initial notification that contains the current values of all subscribed state variables. If one or more state variables are changed the publisher sends an event notification message. The message contains the name of the changed variable and its value expressed in XML format.

If the state variable changes too rapidly, the publisher could enable a filter to moderate the flow of event notifications.

The current implementation of UPnP event mechanism assumes that the subscriber cannot select variables to get events. The control point can only subscribe on all event messages of a UPnP device's service.

2.1.6.1 Subscription Request

The publisher or UPnP device maintains the Subscriber list, which keeps all necessary information about subscribers. The list is updated upon receiving subscription, renewal, or cancellation messages from subscribers.

The Subscription list contains following fields:

Unique Subscription Identifier - is generated by a publisher upon receiving a subscription message. It must be unique during the subscription. It is recommended to be used for universal unique identifiers (URI)

URL for event messages – is the URL that the subscriber sends in a subscription message. It tells a publisher where to deliver event notification messages.

Event key - is an integer that keeps the number of event messages sent to the subscriber. After receiving subscription messages, the publisher sets it to 0, with increments for each subsequent event message.

Duration of subscription – is amount of time for which the subscription is valid.

After receiving the Device Description document, the control point retrieves the event subscription URL and the ID of the service from the document. The control point then sends a subscription message to the UPnP device. If the publisher has enough resources it should accept the subscription. To accept the subscription, the publisher generates the subscription ID, assigns the duration for subscription, and sends the initial message to the subscriber. The message must be sent within 30 seconds.

2.1.6.2 Subscription Renewal

The subscription expires according to the Timeout value in the response subscription message. To renew the subscription, the subscriber must send a renew subscription message to the publisher. The message is sent to the same address as the original SUBSCRIBE message. To renew the subscription, the control point uses the same GENA SUBSRIBE message as the subscription request, but with a different type of header. The header of the renew message does not have *Callback* sub-element, but has *SID* sub-element. *SID* sub-element indicates which subscription is to be renewed. The publisher accepting the renewal sends the renew subscription response back to the

subscriber. The renew response has the same format as a subscriber response request and must be sent within 30 seconds.

Unlike the subscribe response, the renew response does not generate initial messages.

2.1.6.3 Subscription Termination

If the subscriber no longer wants to receive events, it sends a cancel subscription message to the publisher. If the control point leaves the network but does not send a cancel subscription message, the publisher continues to maintain the subscription and sends subscriber event notifications, which unnecessarily consumes network bandwidth. The publisher receives the unsubscribe message and sends the response to the control point.

The publisher may also cancel the subscription. If the device or device's services cancel their advertisements, the control point assumes that the subscriptions to that device and its services are canceled as well.

2.1.7 Sending Event

When one of the state variable changes its state, the publisher sends an event notification. The notification is sent as soon as possible, and includes the name of the variable and its state. If more than one variable is changed simultaneously, the publisher should include all changed variables in one message.

As described earlier, the publisher receives the subscription request and the initial message that contains the current state of the variables sends back to the subscriber.
Each event messages have an event key. The event key is maintained to uncover lost packets.

If the subscriber becomes aware of one or more lost events, it should unsubscribe from the publisher and subscribe again.

When the publisher sends the initial message, it sets the event key to 0. The publisher increments an event key for each subsequent event message. The event key

takes up four bytes and is incremented until it reaches 4294967295. After reaching this highest possible number, it reverts to 1.

The event notification message consists of a header and a body. The header consists of control information for the message while the body is presented in XML format and contains a list of names of changed parameters and their values.

If the subscriber gets an error event message it responds with one of the error messages in Table 1.

HTTP Error code	UPnP Error type
412 Precondition Failed	Invalid SID
412 Precondition Failed	Missing SID
400 Bad Request	Missing NT or NTS header
412 Precondition Failed	NT header doesn't equal *upnp:event*
412 Precondition Failed	Invalid NTS header

Table 1. Event Error Codes

2.1.8 Presentation

The presentation function of a UPnP device provides an easy way to retrieve information from the device to a control point. The control point could use a standard web browser to work with the device by loading a device's presentation page. To get the presentation page, the UPnP control point issues the HTTP GET command.

Through the presentation page, the UPnP control point can determine the whole functionality of the UPnP device. Well-designed presentation pages can not only allow monitoring of the current device status by receiving event notification messages, but can also provide functions to control the UPnP device.

Figure 3 depicts the communication schema between a UPnP device and a control point in the case of using the device's presentation page.

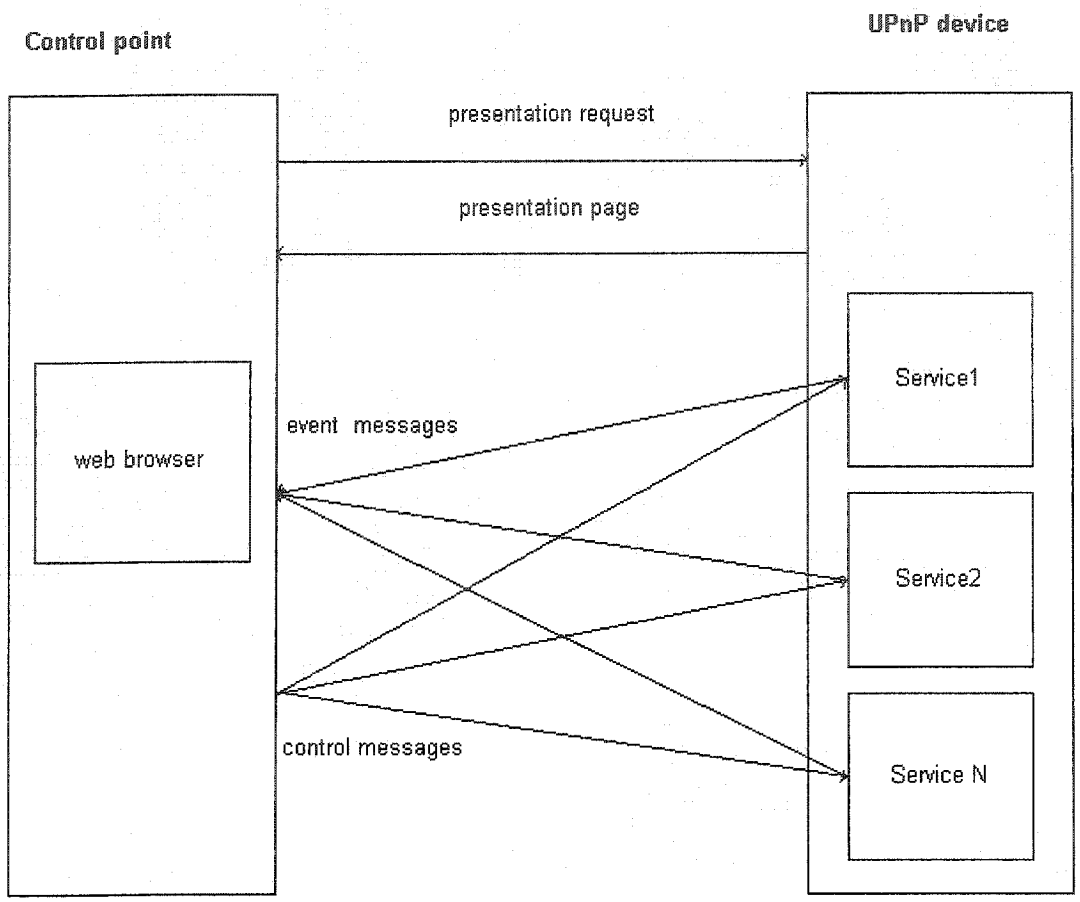


Figure 3. Presentation Page Communication Schema

There are two types of user interface that can be used to handle a device presentation page. The first approach processes the device's messages on a client or control point side. In this case, more work could be done in a web browser that uses client-side scripting or applets. On the other hand, the implementation of a UPnP device function that handles presentation requests would be simple. The device would only need a function that sends a presentation page to the control point.

The second approach uses the device side to process the messages in the presentation page. In addition to its standard functions, the device must implement a mechanism to handle messages received from the presentation page and translate them to device standard functions. The UPnP device would be more complex, but the

implementation of the presentation page would be simpler. The presentation page could be written using only HTTP code [28].

There is a trade-off in deciding between the efforts to design an UPnP device and its presentation page.

The UPnP working committee does not specify the UPnP presentation function. The committee only specifies that the page must be in HTML format, version 3.0 or more. The implementation of the presentation function is left to UPnP device vendor.

The presentation function must support localization. Localization of the presentation page is defined by Language Tags – Accept-Language and Content-Language that are present in the HTTP header. Both tags use two or three letter codes specified in the ISO-639 and ISO-3166 standards.

The Accept-Language tag specifies the set of languages that are preferred by the control point as a response to the request. An example illustrating the use of the Accept-Language tag is as follows:

GET /presentation.html HTTP 1.1

Accept-Language: *language codes*

2.2 JINI

JINI is a distributed system that combines groups of users and required resources, such as hardware devices or software programs [4]. The main goal of JINI is to provide a flexible environment in which new services can be used, added, or deleted in an easy manner.

The system relies on the following components:

- A set of devices that provide infrastructure in a distributed system
- A programming model, which allows for quickly developing applications that could work in any JINI device without modification.
- Services that offer functionality of devices to users.

Although the components are separate pieces of the system, they strongly depend on each other. The devices and services use the programming model for communication and presentation of their functionality. The programming model is supported by the components of the system.

From the point of view of users, the JINI systems offer following features:

- Easily enabling users to have access to resources of the network.
- Ubiquitous access to the resources from anywhere in the network, allowing the users to change their locations.
- Easy maintenance of the network.

JINI extends the Java Virtual Machine from one computer to a network of machines. It allows building a distributed system where codes and data can easily be moved from one computer to another. JINI relies more on the Java virtual machine rather than the Java language. Any computer language that can produce Java byte code can be used for building JINI applications.

In addition to the considerable advantages of using Java technology, JINI offers a flexible mechanism to attach and detach devices and services to and from the network.

Each JINI device has memory and processing power. Devices without processing power or memory can be connected to the network through special components, proxies, that can work as a bridge to the JINI world.

2.2.1 JINI Service

To control devices on the network, JINI uses services, which are the core of the JINI network. Services can be as a whole device or as a part of functions of the hardware, and can be added to or deleted from the network at any time.

To provide ease of communication among services JINI uses a service protocol that is a set of interfaces. JINI also provides a mechanism for service construction and lookup services.

Lookup services bridge users with services. They find and register services in the network by playing two functions – discovery and join. The discovery function (or protocol) finds a service and the join function (protocol) registers the service in the lookup service.

To provide communication among services, Remote Method Invocation (RMI), an extension of the JAVA language, is used. RMI finds, activates, and cleans objects in the JINI network. Not only can it send data between two objects, it can also send a full object code encapsulated as another object.

JINI provides a lease-based approach to object access. If a client wants to use an object, it negotiates with a service to lease the object for certain period of time. A lease can be exclusive or non-exclusive. An exclusive lease guarantees that no other client can have access to this type of object during the lease period.

JINI supplies a protocol for performing transactions, but the implementation of transaction relies on the service using the interface.
The architecture provides an event mechanism that allows an object to be registered. It also receives events and their notification when the events occur.

2.2.2 Components of the JINI Architecture

The JINI architecture consists of three components: infrastructure, programming model, and services. The infrastructure is a set of components that are used to build JINI networks. Services are the entities of the infrastructure. The programming model allows for the development of interfaces that represent services. Although all three components are independent from the point of view of JINI the boundaries between components are often not clear. It is possible to develop a system with one or more components absent, but to use JINI at its best, developers should use all three components. This allows a developer to build a system where components could be changed minimally when modifying the system or moving it to a new project.

The infrastructure includes the following elements:

- Protocols to provide discovery and join services in JINI networks. It also advertises discovered services to the others members of the network.
- Lookup services, which are repositories of discovered services. Entities of lookup services are objects of their services. The object can be downloaded from the lookup service and can play a proxy role of services that supply their interfaces.
- Distributed security systems that can extend RMI security to the level of the network.

The programming model contains the following :

- The leasing interface that defines a way to allocate and free resources.
- The event and notification mechanism.
- The transaction mechanism that allows a group of objects to synchronize their actions.

The programming model is used to build services, and services use the interface to communicate with each other. In terms of programming language, services are the objects and they are defined by the interfaces of services. The interfaces define lists of operations that objects are to perform.

2.2.3 JINI Discovery and Lookup Protocols

At the core of JINI architecture are three protocols – discovery, join, and lookup. The discovery protocol is invoked when a service searches for a lookup service. When the lookup service is found, the service joins the network by registering itself in a repository of the lookup service. A client uses the lookup protocol when it wants to get an object of the registered service.

Figures 4, 5, and 6 describe the discovery, joint and lookup processes.

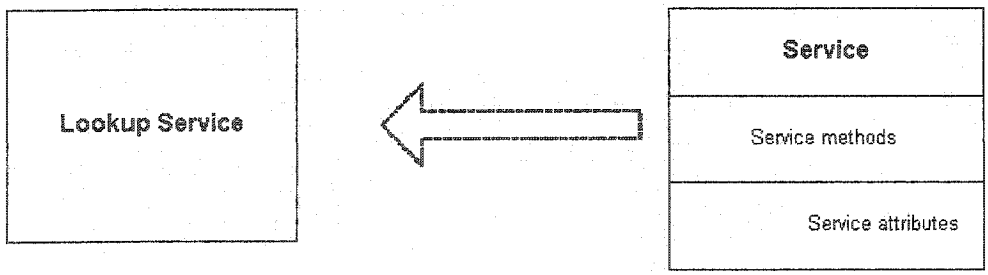


Figure 4: Discovery Phase of JINI

A service can either be a hardware device or software. The service first multicasts a request to find any existing lookup services (as shown in Figure 4). When a lookup service is found, the service joins the JINI network by loading its methods and attributes on to a lookup table (as shown in Figure 5). The methods and attributes are presented using a Java interface.

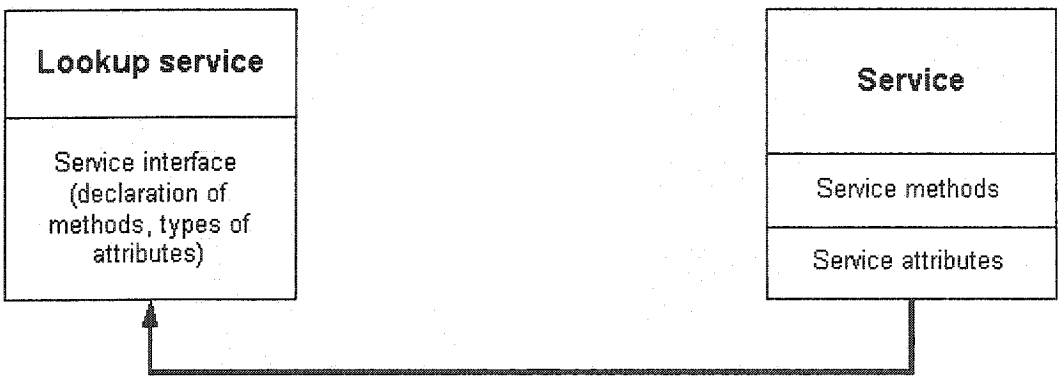


Figure 5: Join Phase of JINI

When a client wants to find a service with particular methods and attributes, it sends a request to the lookup service [26]. The lookup service searches its table, and if it finds the type of service requested by the client, a reference to a service location is sent. The client makes a request to this service and loads the service's object (as shown in Figure 6).

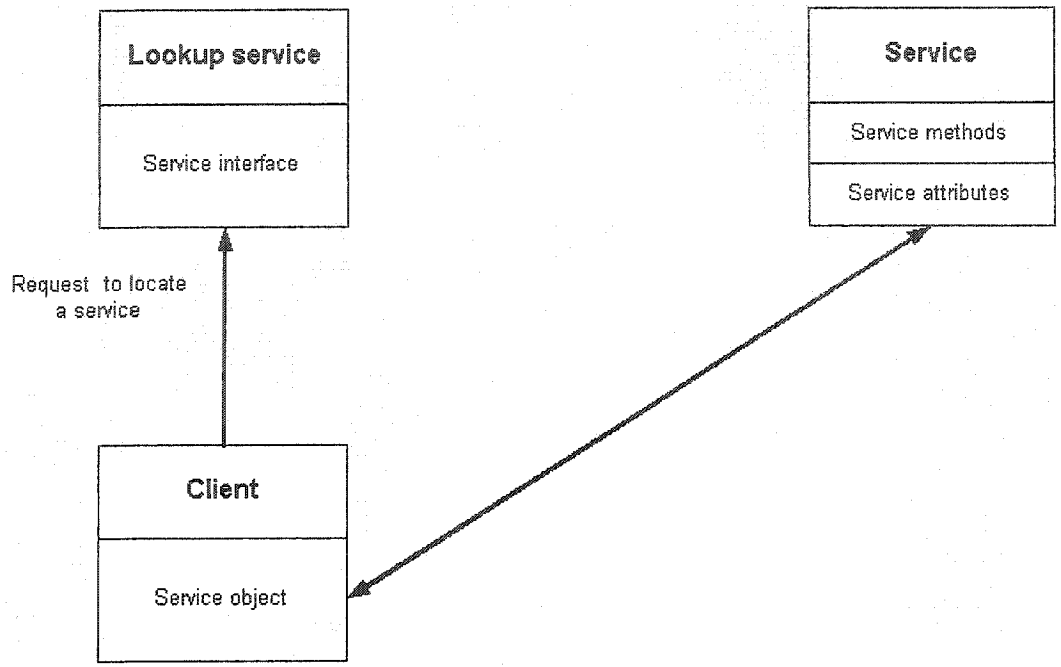


Figure 6: Lookup Phase of JINI

2.3 UPnP versus JINI

Currently, UPnP and JINI are the two major technologies used to design and implement a device distribution system that can be used with the Internet easily. Deciding upon which technology to use affects not only the current implementation efforts, but also future design.

Although JINI offers the better discovery mechanism, it has one significant disadvantage: JINI relies on an API that is written in Java and requires the use of Java and Java support tools that are, to some degree, operating system and language dependent [44].

Unlike JINI, UPnP is independent from operating systems and language. For instance, it is possible to use a Windows based control point that interacts with a non-Windows UPnP device.

3 Smart Sensor Standards

Smart Sensor includes four standards, but only two, IEEE 1415.1 and IEEE 1451.2, are related to network communication [10][12]. The next two sections comprise of a concise overview of these two standards.

3.1 IEEE 1451.1 Standard

This standard was developed to standardize the connection between Smart Transducer devices and networks [5][18].

The standard defines a common object model of the Networked Smart Transducer components and their interfaces [14]. It can be presented in two views: physical and logical. Figure 7 depicts the physical view, while Figure 8 shows the logical view of the standard.

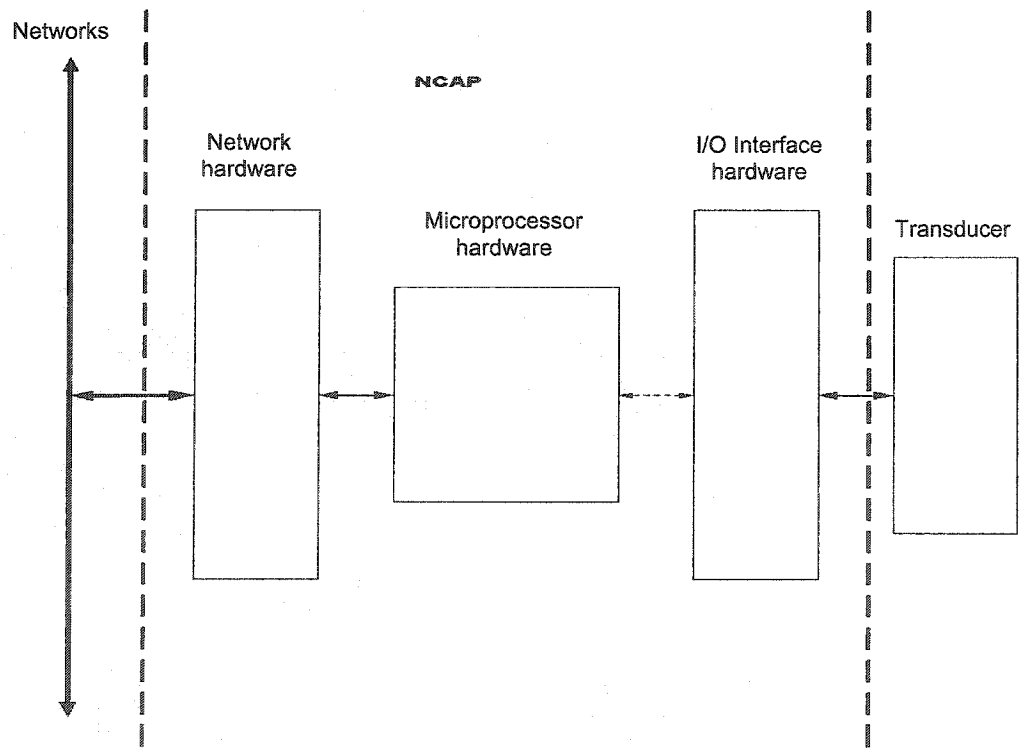


Figure 7: Physical View of NCAP

The physical view depicts access to networks, NCAP hardware, the I/O interface and to the Transducer hardware. The NCAP hardware contains the microprocessor and its peripherals, such as RAM, ROM, and the input/output interface.

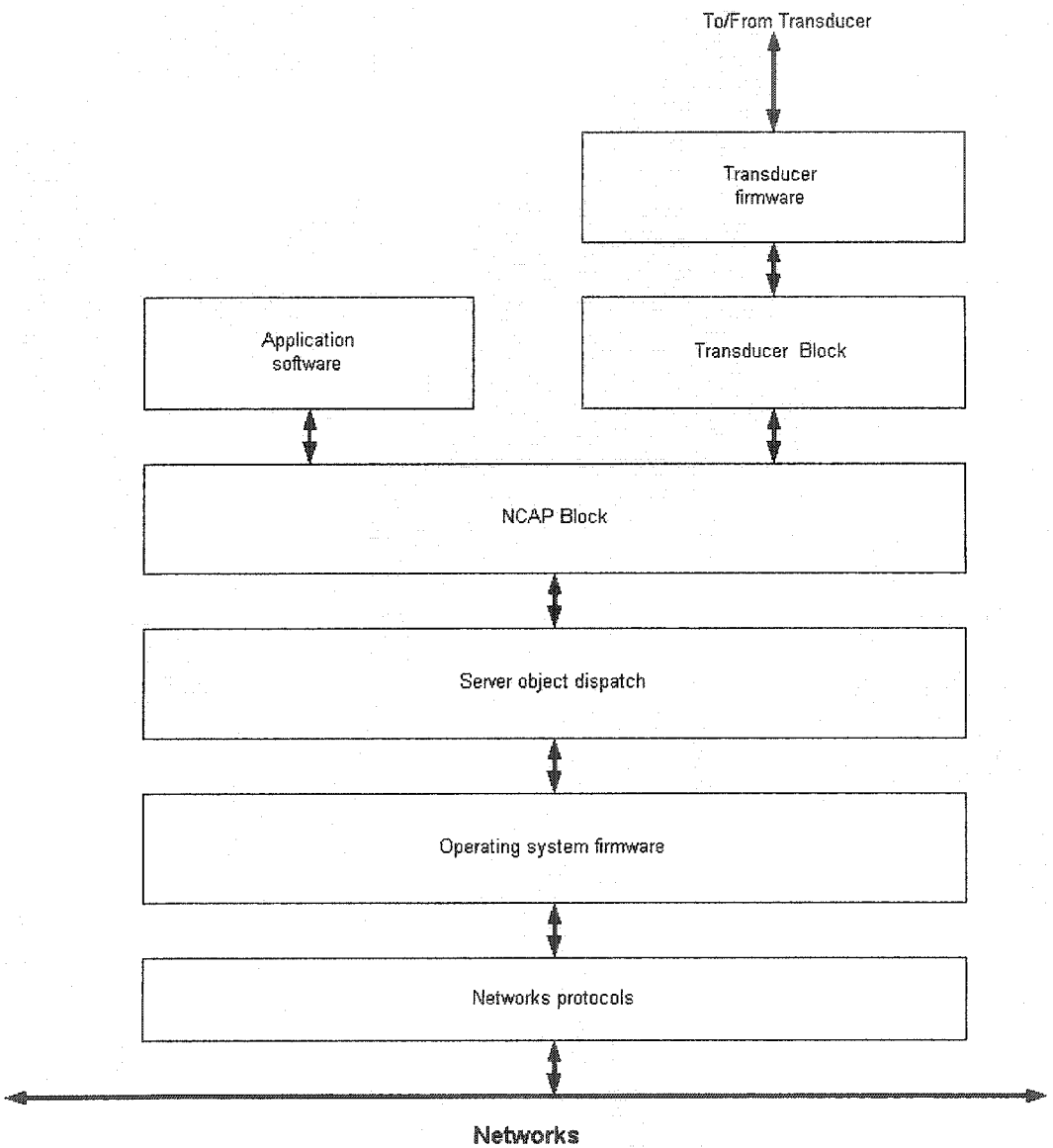


Figure 8: Logical View of NCAP

The logical view can be split into three parts: Network Logical Specification, NCAP Application, and Transducer Logical Specification. The Network Logical Specification includes networks protocols and a part of Operating system firmware components, while hiding specific details of communication over networks.

The NCAP Application contains Operating system firmware, the Server object dispatch, a part of the NCAP block, and an Application Software. It provides treatment of data from the Transducer and synchronization of communication through the network and Transducer Block. The Server object dispatch is responsible for presenting transducer data as a list of objects. These objects can be used to develop a client–sever distributed system.

The third part – Transducer Logical Specification - includes the NCAP block, the Transducer Block and the Transducer Firmware. The essential goal is to provide an interface to the Transducer hardware.

3.1.1 The Information Model

The standard specifies software architecture for the hardware that connect standard networks with transducers. One or many NCAP represent the hardware.

The standard specifies the interface of NCAP functions to networks as well as the interface of NCAP functions to transducers. The networks and transducers are presented in IEEE 1451.1 as abstraction layers to provide communication to/from these layers an independent way.

The standard defines three types of models: object, data, and networks.

3.1.1.1 Object Model

The Object model specifies four types of object classes: block, component, service, and Non-IEEE 1451.1 object classes. Figure 8 depicts the hierarchy of these classes.

Each class is defined by an interface and by class behaviour.

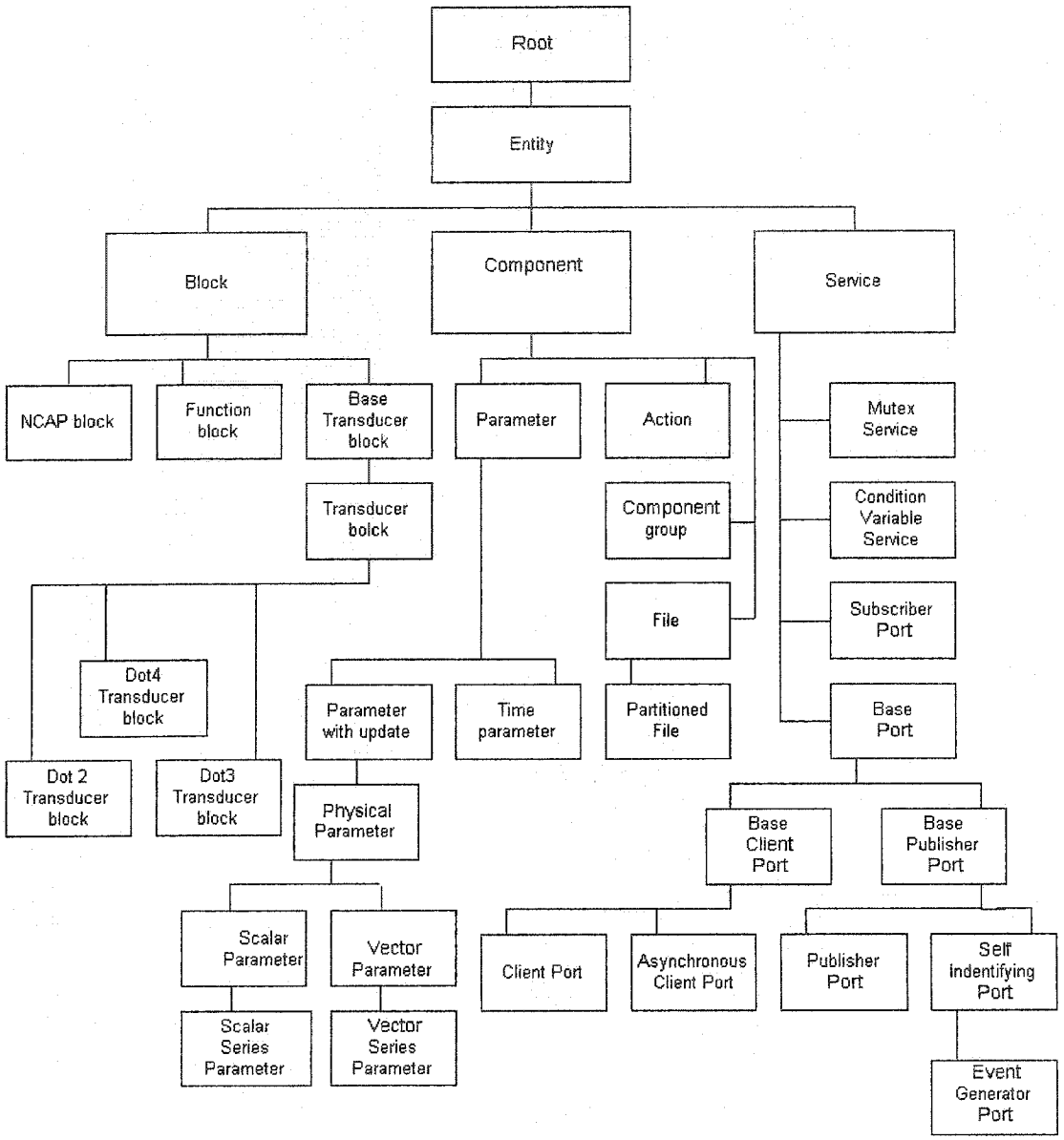


Figure 9: IEEE1451.1 Object Class Hierarchy

The Block group consists of three types of classes: NCAP block classes, Transducer block classes and Application block classes. The NCAP block classes provide a software interface for network communication and system configuration. The Transducer block classes provide a software interface between transducers and applications. The Application block classes implement specific application functions.

The Component group includes classes that provide structural information, collections of objects, and actions such as events or other changes of states of objects.

The Service group provides communication among NCAP objects and synchronizes system services.

The standard also specifies relations among the NCAP objects. There is always at least one NCAP object that contains at least one software process. Each process must have one object of the "NCAP block", zero or more objects of the "Function block", and zero or more objects of the "Base Transducer block". Each object must be associated with exactly one NCAP process. Ownership relations are used mostly to cleanly shut down the systems well as to dynamically invoke and register objects.

- To communicate through a network, an object has to be network visible. Network communication means that the object can send a message through the network or that another part of the network infrastructure can invoke the object.

The NCAP block is always made visible.

3.1.1.2 Data Model

The data model specifies the forms and types of both local and network communications. The model has primitive types(such as Boolean, string, integer and float types) of representing of data, as well as more complex ones, such as structure types.

3.1.1.3 Network Communication Model

The standard defines two models for NCAP capable devices: one-to-one client-servers and one-to-many or many-to-many communications. The network communication is defined by subclasses including Client, Publisher, and Subscriber Ports of the Service class.

The standard does not specify any network protocol. It relies on network communication libraries supplied by a network vendor. The libraries, in addition to the

network protocols, must have marshaling routines to ensure transfer of objects between a server and a client. The marshaling routines allow transfer of memory dependant parameters, such as pointers to memory, through networks.

3.1.1.3.1 Client-Server Communication Model

The client–server model relies on two methods: *Execute* of a client Port Object and *Perform* of a Server Object. Figure 10 depicts the relationship between the Client Object and the Server Object.

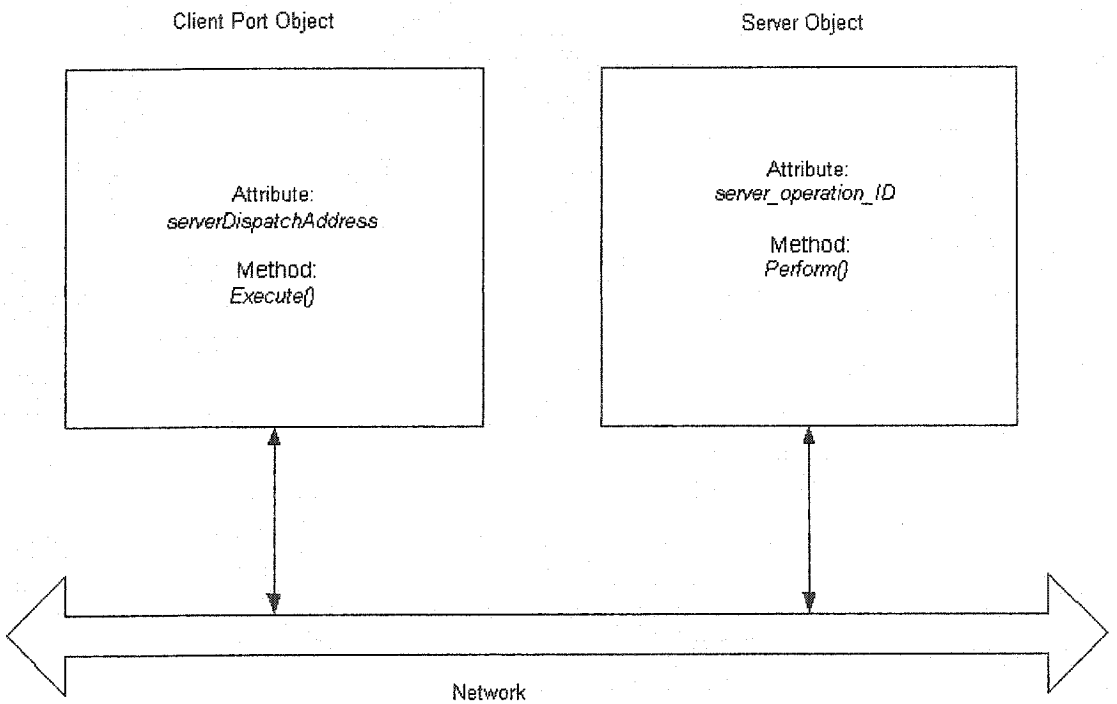


Figure 10: The Client–Server Components

The communication between the Client Port and the Server Object occurs in several steps:

1. A *serverDispatchAddress* attribute of a Client Port Object is bound with server_operation_ID of a Server Object.

2. A Client Object calls the method *Execute()* of the Port Object. One parameter is made an ID of the server, while the rest of the parameters are organized as an Argument Array.
3. Using the network protocol calling *Execute()* on the client side invokes the method *Perform()* on the server side.
4. When the server finishes the *Perform()* routine it sends a result to the client. The result is sent using the network protocol.
5. The client, after obtaining the result, returns from the *Execute()* method.

3.1.1.3.2 Publisher-Subscriber Communication

The Publisher–Subscriber relationship is one–to–many or many–to–many communication.

The Publisher plays the role of server and the Subscriber acts as a client. The Publisher is not aware of Subscriber objects, which are receivers. The Subscriber consists of two parts – a Subscriber Port and a Subscriber Object. If the Subscriber wants to obtain specific information from the publisher, it registers with the Publisher. When the Publisher prepares the information, it sends it to all subscribed clients. The Subscriber Ports receive the data, filter it, and invoke callback functions of the Subscriber Objects that will process the data. Figure 11 depicts the components of Publisher–Subscriber communication.

During initialization, a *subscriber_key* of a Subscriber Port is bound to a *publisher_key* of a Publisher. When a Subscribed Object wants data, it calls the *AddSubscriber()* method of the Subscribed Port. The Subscribed Port binds the *subscriber_domain* and *subscriber_topic* with the *publisher_domain* and *publisher_topic* of the Publisher. The Publisher then starts to send information to all the Subscribers that were registered.

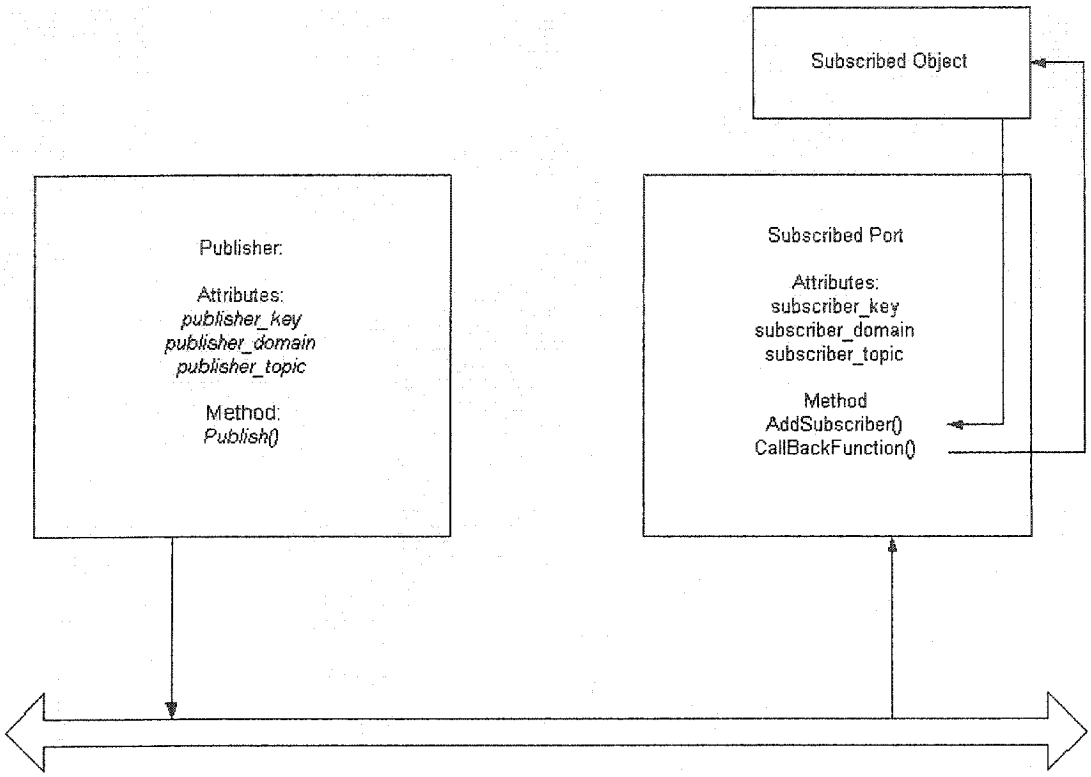


Figure 11: Publisher-Subscriber Communication Components

3.1.2 Data types of IEEE 1451.1

The standard defines data types that are used to design and develop an IEEE 1451.1 system. There are two classes of data types: primitive and derived. The primitive data class includes the following types of data: 8-, 16-, 32-, 64-bit integers, a Boolean, an octet of bits, a string, and an ObjectDispatchAddress.

The derived data class is based on primitive data types. It includes the following data types:

- Arrays of primitive data types.
- TimeRepresentation structure that consists of two integers field: seconds and nanoseconds.
- Structures that describe Objects in IEEE 1451.1 system. These are ObjectProperties and ClientPortProperties structures.
- Return Code types.

- Defined enumeration data types.
- Unions and Arguments data types. These are used for network operations.
- Physical Parameters structures. These are used to represent transducers parameters.

3.1.3 Properties of IEEE 1451.1 Classes

The standard defines rules and templates to design classes of an IEEE 1451.1 system. It specifies three types of common properties: class designators, object operations, and expressions of uncertainty in values.

The abstract class defines a specification for all its subclasses. It cannot be initiated.

The class ID uniquely defines a position of the class in a class hierarchy. IEEE 1451.1 also describes rules to define a class ID.

The object operation properties specify major properties and implementation rules of IEEE 1451.1 classes, and include the following groups:

- Requirements of implementation.
- The designator for the operation.
- The signature of the operation.
- The Behaviour of the operation.

Implementation requirements define the types of operations and how they should be implemented. The operation can be mandatory or optional. The implementation can be full or may only include an interface.

The standard defines general rules to create a Return Code. It also includes the list of predefined Return Codes for both local and network operations.

The Behaviour of operations specifies two major types of operations: Get and Set.

Get type operations allow the observation, either locally or through networks, of any portion of an object's data. The recommended name for a Get operation is Read.

Set operations allow for modification of any internal data of an object. The recommended name for a Set operation is Write.

The Behaviour of operations also specifies concurrency requirements for different cases of intra-NCAP executions, such as multiple process concurrency, multiple operations on an object within a single process, and interfering operations.

Block Cookies are used to notify a client about changing the state of a server. Only Network Visible Objects of class IEEE 1451.1_Block can have a Block Cookie.

3.1.4 Concise description of IEEE1451.1 Classes

The hierarchy of the IEEE1451.1 classes was shown earlier in Figure 9.

The Root abstract class is the root class of all classes defined in the IEEE1451.1 standard.

The abstract Entity class is the root of all IEEE1451.1 classes visible through networks.

The Block abstract class is the root in the hierarchy of Block classes and defines all common properties and methods of Block classes.

The NCAP Block class manages the system configuration and registration of Network visible objects.

The Function Block Class is the root for all classes and is inherited from the Block class.

The Base Transducer Block is the root in the hierarchy for all Transducer classes.

The Transducer Block abstract class is the root in the hierarchy of all Transducer classes specified in the IEEE 1451.X standards.

The Dot2 Transducer Block class provides an interface to transducers defined in the IEEE1451.2 standard.

The Dot3 Transducer Block class provides an interface for transducers defined by the IEEE1451.3 standard.

The Dot4 Transducer Block class provides an interface for transducers defined by the IEEE1451.4 standard.

The Component abstract class is the root for all Component objects.

The Parameter class is used to define parameters and ways to access them.

The Parameter with Update class is the class with which to model Parameter classes that provide updating operations.

The Physical Parameter abstract class is used to model Network visible variables that represent sensor data.

The Scalar Parameter class represents Parameters that do not have any dimension or orientation associated with them. The class supports three types of Parameters: Scalar Analog, Scalar Discrete, and Scalar Digital.

The Scalar Series Parameter class represents Scalar Parameters that are distributed in some dimension. An example of such a Parameter is a time series.

The Vector class is used to represent parameters that have multiple dimensions. Examples of Vector Parameters are electrical fields and the temperature/pressure relationship.

The Vector Series Parameter class represents a series of Parameters with dimensions.

The Time Parameter class is used to represent a time value.

The Action class is used to design classes that require a significant amount of time to execute compared to other activities in the system.

The File class is used to design a class that represents data resources.

The Partitioned File class is used to design a File class that is split over a number of partitions.

The Service abstract class is the root for all classes in the Service class hierarchy.

The Base Port abstract class is the root for all Port Objects used for communication through networks.

The Base Client Port abstract class is the root for all client side Objects in client-server communications.

The Client port class and the Asynchronous Client Port are used to design the client side in client-server communications. There are two models of client-server communication: blocking and non-blocking. The Client port class presents the blocking model. The Asynchronous Client Port class is used to design the client side in non-blocking client-server communications.

The Publisher Port abstract class provides the basic functionality needed to design Publisher classes and is used to design publisher side classes in publisher–subscriber communications.

The Self Identifying Publisher Port class provides the publisher–subscriber model of communications, where a subscriber sets up communication with a publisher and the publisher notifies the subscriber about changes in subscriber policy.

The Event Generator Publisher Port class provides a mechanism to manage an Event Generator, which generates events as a result of the publication of data.

The Subscriber Port class provides a mechanism for subscribing with the publisher–subscriber model. The class has been defined to use in modern Event notification mechanisms [27].

The Mutex class provides a mutually exclusive mechanism for the Objects in the concurrent environment of the system.

Condition Variable Service Class provides a mechanism for ordering concurrent activities.

3.2 IEEE 1451.2 Standard

The main objective of the IEEE 1451.2 standard is to develop a transducer's interface where sensors or actuators would be isolated from a network interface [6][16]. This goal is achieved by the following:

- Develop sensor and actuator devices that have plug and play capabilities.
- Develop transducers that have network capabilities.
- Support multiple networks in transducer devices.

The standard simplifies developing transducer devices by defining hardware and software parts [20][22]. IEEE 1451.2 offers a hardware interface to connect STIM to NCAP processors. The core of the standard is the transducer electronic data sheet (TEDS) that defines a mechanism for the communication between STIM modules and NCAP [40].

- TEDS allows specifying a wide range of transducers that can work in a standard network interface. Transducer data sheets define types of transducers, their attributes and operations.

3.2.1 STIM Specification

The concise functional schema of STIM is depicted in Figure 12. The STIM can be represented by a collection of transducers that can either be sensors or actuators.

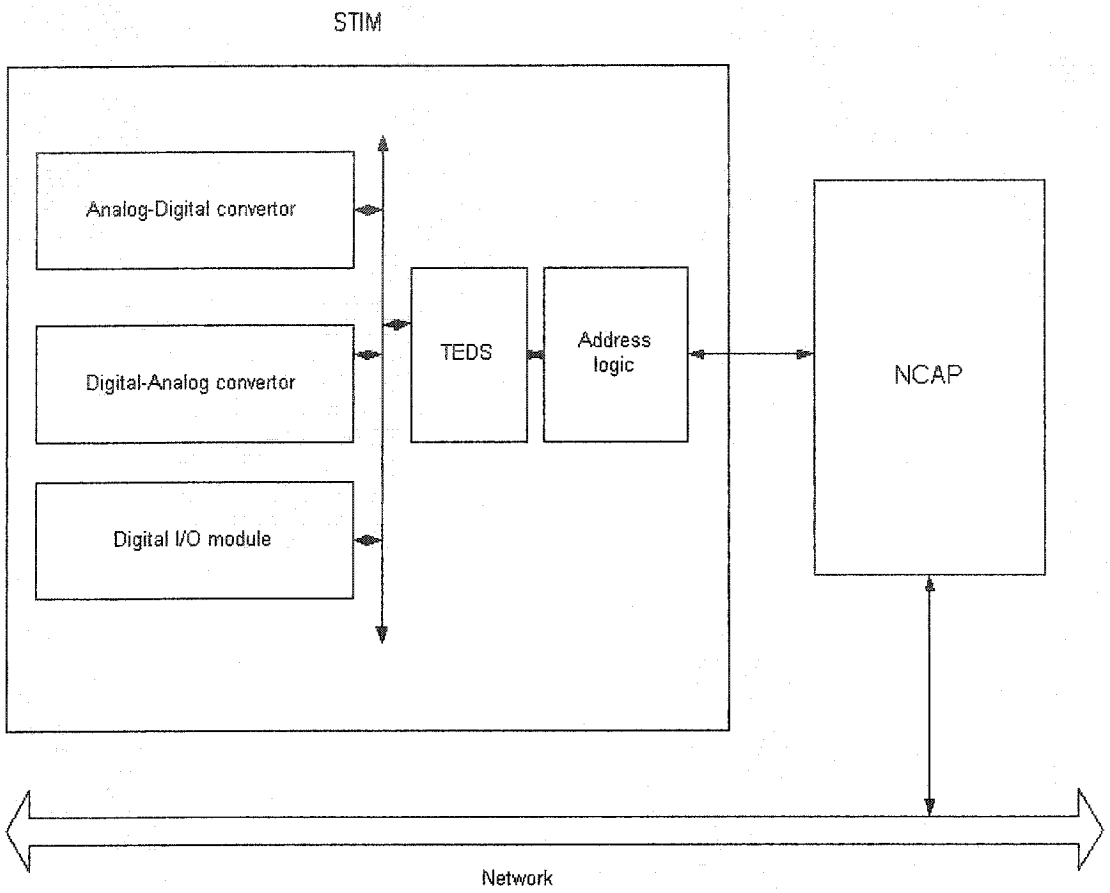


Figure 12: Simple STIM Functional Schema

3.2.1.1 Transducer Types

There are seven types of transducers used in STIM. They are listed below:

- Sensors.
- Actuators.
- Buffered sensors.
- Data sequence sensors.
- Buffered data sequence sensors.
- Event sequence sensors.
- General transducers.

A sensor measures a physical parameter and returns a digital value representing a parameter. An actuator accepts some digital value and performs an action according to this value. A buffered sensor differs from a sensor in that it has a buffer for output data.

A data sequence sensor and buffered data sequence sensor provides continuous measurement of data. An event sequence sensor measures data whenever a specified event occurs. Finally, a general transducer includes all transducers except those specified above.

3.2.1.2 STIM Functions

Each STIM implements the following functions:

- Addressing.
- Interface data transport.
- META-TEDS.
- Global status.
- Global control.
- Triggering.
- Hot-swap capability.
- Interrupt.
- Interrupt masking.

Each channel of a STIM implements the following functions:

- Channel TEDS.
- Transducer data.
- Status.
- Control.

In addition to these functions, a channel may implement the following optional functions:

- Calibration TEDS.
- Calibration identification TEDS.
- Channel Identification TEDS.
- User-End Identification TEDS.
- Generic Extension TEDS.
- Self-calibration.
- Self-test.

Addressing is used to specify the type of transport operation (read or write) and the address of the function or channel. Data transport provides the physical interface for communications between NCAP and STIM modules. It specifies the data format and data functions. Triggering is a function that allows sending, from NCAP to STIM, a command for the action to take place. Triggering is provided by a physical line and can be applied to either one channel or to all channels. The Control function allows sending a command from NCAP to STIM. The command can be sent to one designated channel or to all channels. Table 2 shows the major standard commands.

Value	Description
1	Reset (STIM or channel)
2	Initiate self test
3	Calibrate channel
4	Zero channel
5	Enable event sequence sensor
6	Disable event sequence sensor
7	Set event sequence sensor
9	Enable data sequence or buffered data sequence sensor
10	Disable data sequence or buffered data sequence sensor

Table 2. Major Control Functions of STIM

The status function allows NCAP to determine the status of NCAP or a channel. The status information are kept in the standard and auxiliary status registers. Table 3 shows the standard status register and Table 4 depicts the auxiliary status registers.

Bit	Description
msb	STIM or channel operational bit
	STIM or channel hardware error bit
	STIM or channel event bit
	STIM or channel auxiliary status available bit
	STIM or channel has been reset bit
	STIM or channel trigger acknowledged bit
lsb	STIM or channel request bit

Table 3. The Standard Status Register Contents

Bits	Description
msb	Channel data over range or under range bit
	Channel consumables exhausted bit
	Channel failed self-test bit
	Channel failed calibration bit
lsb	Channel busy bit

Table 4. The Auxiliary Status Register Contents

The Interrupt function allows the STIM to request service from NCAP. The function is controlled by a digital signal from the physical interface. The interrupt masking function includes the standard and auxiliary interrupt mask registers. They allow the masking of corresponding bits in the standard and auxiliary status registers. The Hot-swap function allows detection of an event when the STIM is inserted and extracted from a powered NCAP.

4 Modifications to the IEEE1451.1 Standard

4.1 Motivation to Modify the IEEE 1451.1 Standard

The goal of this thesis is to provide modifications to the IEEE 1451.1 standard in order to make it work in a system that could very easily connect at device to a network, and manage it. Given that UPnP is more flexible in nature compared to JINI, the former architecture was chosen as a basis to provide network Plug and Play capabilities to Smart Sensor devices (for more details see Section 2.3).

Although IEEE 1451.1 supports two network models – client-server and publisher-subscriber, it does not provide a standard network mechanism that could allow easy connection of a Smart Sensor device into the Internet. UPnP offers the specification of how to make a network device Internet capable. A device with UPnP capabilities can be easily connected to the Internet. Any user with access to the Internet can manage the UPnP devices.

The UPnP specification is similar to the IEEE 1451.1 networks model. Both client-sever and publisher-subscriber classes of IEEE 1451.1 will be used as a base for network classes of the new modified standard. These classes will be extended to implement UPnP specific protocols such as DHCP, Auto-IP, SOAP, and GENA.

In addition to modification of the network classes, the File and Partitioned File classes were modified to support HTML and XML files, which are the base document formats required to present data in UPnP communications.

4.2 Modification of the Network Model of IEEE 1451.1

The network model of IEEE1451.1 consists of two parts – the client-server and the publisher-subscriber classes. The first group contains the following classes: Base port (Class ID: 1.1.3.1 [5]), Base Client Port (Class ID: 1.1.3.1.1 [5]),

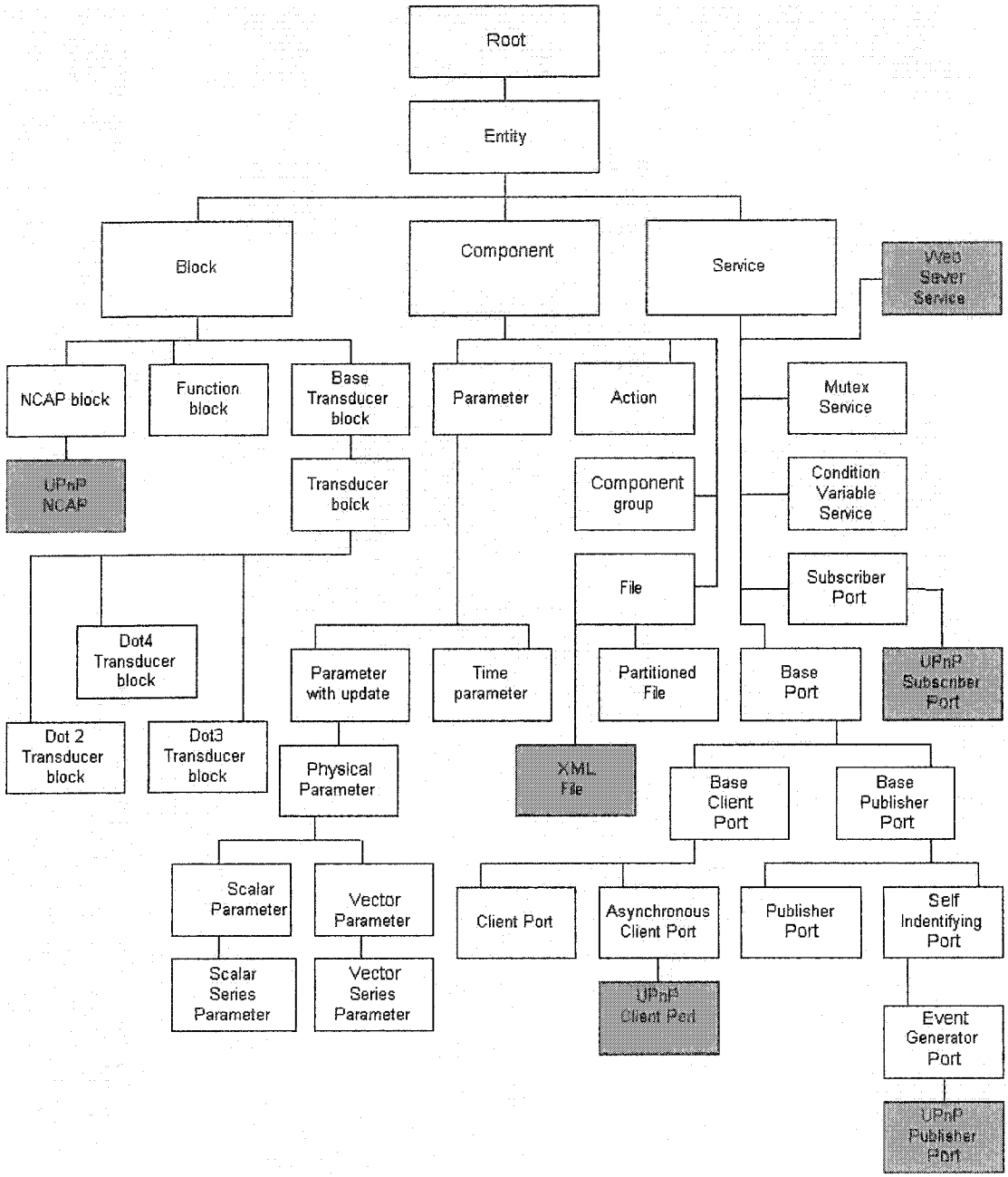


Figure 13: Object Class Hierarchy of IEEE1451.1 with UPnP Capabilities

Client Port (Class ID: 1.1.3.1.1.1 [5]) and Parameter (Class ID: 1.1.2.1 [5]). The second group consists of the following classes: Base Publisher Port (Class ID: 1.1.3.1.2 [5]), Publisher Port (Class ID: 1.1.3.1.2.1 [5]), Self Identifying Publisher Port class (Class

ID: 1.1.3.1.2.2 [5]), Event Generator Publisher Port class (Class ID: 1.1.3.1.2.2.1 [5]), and Subscriber Port class (Class ID: 1.1.3.2 [5]).

In addition to the Service group classes, we modified the NCAP block class that provides registration of the NCAP system. Figure 12 depicts the Object class hierarchy IEEE1451.1 with UPnP capabilities.

4.3 Modification of the NCAP Block

The NCAP block enables registration of the system into the network and also manages the network functions of an IEEE 1451.1 device. The standard provides common definitions for registration of NCAP systems, but does not provide a mechanism for making them. The standard leaves this mechanism to the specific network implementation.

First, to make any network device work we should decide how this device needs to be addressed. The standard does not specify a mechanism to define network addresses and assumes that they are fixed or hard coded. The UPnP specification offers a more flexible solution. To address a device, UPnP uses IP addressing and DHCP or Auto-IP protocols [7]. Both protocols automatically assign IP addresses. In general, UPnP devices should implement support for both DHCP and Auto-IP protocols.

DHCP assigns an IP address, a subnet mask, a default gateway, and a domain name server. DHCP communication uses the UDP protocol and ports 67/68 to send and receive information from the DHCP server.

If the UPnP device does not get a response from a DHCP server, it assumes that no DHCP server is present and switches to the Auto-IP protocol. Auto-IP is the method by which a network device can choose an IP address and subnet mask in the absence of a DHCP server. The Auto-IP method does not replace DHCP, but it makes the device more robust in case the DHCP server is not present. The address selection algorithm is implementation-dependant, but the chosen IP address should generally fall in the range

169.254.XX.XX with the subnet mask 255.255.0.0. The chosen address will work only in a local network domain and not in cross gateway boundaries.

In addition to the dynamic assignment of IP addresses, the new NCAP block class should provide an UPnP discovery mechanism. To register, a UPnP device uses the SSDP. The device identifies itself with two parameters: the type of service and Unique Service Name (USN). The UPnP committees for each standard device type define the type of service. The USN uniquely identifies the instance of a particular service. Part of the USN is a universal unique identifier (UUID). UUID allows recognition of different devices that belong to the same type of service.

The third function of the UPnP NCAP Block is to provide the Description of the UPnP device and its services. The Description of the UPnP device is an XML document that fully describes the functionality that the device offers. The Description document contains specific information about the vender and type, name and URL of services that the device supports. Each service section of the Description document includes the URL to control the service and the URL to subscribe event messages. The service description is another document and it is sent in a separate message. The service description contains a list of actions or variables that are available for control points and a list of events on which the control point could subscribe.

Using the above functional description, a new class of IEEE 1451.1 - UPnP NCAP Block inherited from the NCAP Block class is introduced.

4.3.1 UPnP NCAP Block Class

Class: IEEE1451_NCAPBlock_UPNP

Parent class: IEEE1451_NCAPBlock

Class ID: 1.1.1.1.1

Description: The UPnP NCAP class provides UPnP functionalities for the NCAP Block class. In addition to NCAP Block functions, the UPnP NCAP class provides following:

- Network addressing of a NCAP device.
- Registration of UPnP control point and advertising available services for the UPnP device.

Class summary:

Since UPnP NCAP performs all communications through the network using HTML messages, the class does not have Network Visible operations.

Local visible operations: See Table 5.

Publication: See Table 6

Subscription: See Table 7.

Operation name	Requirement
GetUPnPVersion	(m)
GetIPAddress	(m)
GetNetworkPortNumber	(m)
GetURI	(m)
IsDHCPSupport	(o)
GetDHCP_Ipaddress	(o)
AssignAuto_Ipaddress	(o)
GetSSDP_Portaddress	(m)
SetSSDP_Portaddress	(m)
SetControlURL	(m)
GetControlURL	(m)
SetEventURL	(m)
GetEventURL	(m)

Table 5: UPnP NCAP Block Local Operations

Publication Key	Requirement
SSDP_bye_bye_announcement	(m)
SSDP_alive_announcement	(m)
SSDP_Discovery_respond	(m)
Device_Description	(m)
Service_Description	(m)

Table 6: UPnP NCAP Publication

Subscription Key	Requirement
DHCP_Discover	(o)
DHCP_Request	(o)
SSDP_M_Search_request	(m)
Get_Device_Description	(m)
Get_Service_Description	(m)

Table 7: UPnP NCAP Subscription

4.3.1.1 Publication

4.3.1.1.1 Publication SSDP_bye_bye_announcement

Description: This publication provides a notification about removing an UPnP NCAP device from a network.

Publication Key: SSDP_bye_bye_announcement

Publication Topic: UPnP_discovery

Publisher Port Object Name: SSDP_bye_bye_announcement

Publication format: HTTP *NOTIFY* message

Publication Content: See Table 8.

Header	Requirement	Type	Description
Host	(m)	Multicast address and host	Must be 239.255.255.255.250:1900
NT	(m)	Notification type	One from the table 11
NTS	(m)	Single URI	Must be `ssdp:bye-bye`
USN	(m)	Single URI	URI of UPnP NCAP device

Table 8: SSDP_bye_bye_announcement Publication Content

Type of Search	Syntax	Description
All devices	ssdp:all	Searches for all UPnP devices
Root devices	upnp:rootdevice	Searches for all UPnP root devices. Only root devices will respond.
Specific device	uuid:*device_uuid*	Searches for a particular device by the device's unique ID.
Device of a specific type	urn:schemas-upnp-org:device:*deviceType-version*	Searches devices of a given type.
Services of a specific type	urn:schemas-upnp-org:service:serviceType-version	Searches services of a given type

Table 9: UPnP Search Types

4.3.1.1.2 Publication SSDP_alive_announcement

Description: This publication provides a notification about the presence of a UpnPNCAP device.

Publication Key: SSDP_alive_announcement

Publication Topic: UPnP_discovery

Publisher Port Object Name: SSDP_alive_announcement

Publication format: HTTP *NOTIFY* message

Publication Content: See Table 10.

Table 10. SSDP_alive_announcement publication content.

Header	Requirement	Type	Description
Host	(m)	Multicast address and host	Must be 239.255.255.250:1900
Cache-Control	(m)	*max-age* directive	Number of second that the advertisement is valid. Should be >1800 second
Location	(m)	URL	URL of UPnP NCAP description document
NT	(m)	Notification type	One from Table 11
NTS	(m)	URI	Must be ssdp:alive
Server	(m)	String	*os_name/os_version/UPnP/1.0 /product_name/product_version*
USN	(m)	URI	One from Table 11

4.3.1.1.3 Publication SSDP_Discovery_respond

Description: This publication provides a response to an M-Search message.

Publication Key: SSDP_Discovery_respond

Publication Topic: UPnP_discovery

Publisher Port Object Name: SSDP_Discovery_respond

Publication format: HTTP response message – HTTP/1.1 200 OK

Publication Content: See Table 11.

Header	Requirement	Type	Description
Cache - Control	(m)	Max-age	>=1800
Date	(o)	RFC 1123 date	
Ext	(m)		Confirmation to *Man* header in the request (ssdp:discover)
Location	(m)	URL	URL of the UPnP NCAP root device
Server	(m)	String	*os_name/os_version/UPnP/1.0 /product_name/product_version*
ST	(m)	URI	A target for discovery message. Should be the same as in the request message
USN	(m)	URI	Service name for the discovered device

Table 11: SSDP_Discovery_respond Publication Content

4.3.1.1.4 Publication Device_Description

Description: This publication provides a response to a Device Description Request.

Publication Key: Device_Description

Publication Topic: UPnP_description

Publisher Port Object Name: Device_Description

Publication format: HTTP response message – HTTP/1.1 200 OK

Publication Content: See Table 12.

Header	Requirement	Type	Description
Content-Language	(o)	RFC1766 language tag	
Content-Length	(m)	Integer	Length of body in byte
Content-type	(m)	text/xml	Type is XML document
Date	(o)	RFC1123 date	Date of generating the response
Response data	(m)	String	Description document in xml format

Table 12. Device_Description and Service_Description Publication Content

4.3.1.1.5 Publication Service_Description

Description: This publication provides a response to a Service Description Request.
Publication Key: Service_Description
Publication Topic: UPnP_description
Publisher Port Object Name: Service_Description
Publication format: HTTP response message – HTTP/1.1 200 OK
Publication Content: See Table 13.

4.3.1.2 Subscription

4.3.1.2.1 Subscription DHCP_Discover

Description: This subscription discovers a DHCP server when registering as a DHCP client.
Publication Key: DHCPOFFER
Subscription Key: DHCPDISCOVER
Subscription Qualifier: Not applicable
Subscriber Port Object Name: DHCP_Discover

Subscription format: DHCP format message. (See [7]).
Subscription content. DHCPDISCOVER message. (See [7].

4.3.1.2.2 Subscription DHCP_Request

Description: This subscription confirms the selection of a DHCP server and receiving IP address.

Publication Key: DHCPOFFER

Subscription Key: DHCPREQUEST

Subscription Qualifier: Not applicable

Subscriber Port Object Name: DHCP_Request

Subscription format: DHCP format message. (See [7]).

Subscription content. DHCPREQUEST message. (See [7]).

4.3.1.2.3 Subscription SSDP_M_Search_request

Description: This subscription searches for UPnP NCAP devices.

Publication Key: SSDP_Discovery_respond

Subscription Key: SSDP_M_Search_request

Subscription Qualifier: Not applicable

Subscriber Port Object Name: SSDP_M_Search_request

Subscription format: M-SEARCH HTTP message

Subscription Content: See Table 13.

Header	Requirements	Type	Description
Host	(m)	IP address and port	For UPnP the value should equal 239.255.255.255:1900
Man	(m)	ssdp:discover	It is discover message
MX	(m)	Integer	Maximum number in second a UPnP NCAP device should response
ST	(m)	URI	URI of a search UPnP NCAP device

Table 13: SSDP_M_Search_request Subscription Content

4.3.1.2.4 Subscription Get_Device_Description

Description: This subscription requests the description of a UPnP NCAP device

Publication Key: Device_Description

Subscription Key: Get_Device_Description

Subscription format: GET HTTP message

Subscription Content: See Table 14.

Header	Requirements	Type	Description
Host	(m)	IP address and port	The address of the Description service of UPnP NCAP device
Accept-Language	(o)	RFC1766	Language preferred by control point

Table 14: Get_Device_Description and Get_Service_Description Subscription Content

4.3.1.2.5 Subscription Get_Service_Description

Description: This subscription requests the description of a UPnP NCAP device.

Publication Key: Service_Description

Subscription Key: Get_Service_Description

Subscription Qualifier: Not applicable

Subscriber Port Object Name: Get_Service_Description

Subscription format: GET HTTP message

Subscription Content: See Table 14.

4.3.1.3 UPnP NCAP Block Behaviour

4.3.1.3.1 GetDHCP_IPaddress Behaviour Specification

The GetDHCP_IPaddress operation starts to discover a DHCP server in its domain by sending a DHCPDISCOVERY message to a network. The DHCP server receives the DHCPDISCOVERY message and sends back a DHCPOFFER message containing the IP address. The UPnP NCAP device that plays the role of a DHCP client gets the DHCPOFFER message and sends a DHCPREQUEST message that includes a server identifier to the chosen DHCP server. The server identifier indicates that the DHCP client has accepted the offer and implicitly tells other servers that it has declined their offer.

The chosen DHCP server receives the DHCPREQUEST message and sends either: DHCPACK, in the case of accepting the chosen IP address, or DHCPNAK, in the case of declining the chosen IP address (IP address may be allocated to another host) to the client. The DHCPACK message includes additional configuration parameters such as a subnet mask, a default gateway, and the domain name of the server. The DHCP client must check the IP address to ensure that another host does not use it. If the received address has already been used, the client sends a DHCPDECLINE message and reinitiates the DHCP protocol. Figure 14 depicts the communication between the UPnP NACP device and the DHCP server.

If the UPnP NCAP device is unable to receive an IP address from a DHCP server it returns an error code.

A UPnP NCAP device uses the Dynamic allocation of IP addresses requiring renewal of the address lease. When the lease is close to expiration, the device sends a DHCPREQUEST message to the server. The DHCP server responds with a DHCPPACK message. The message contains the new duration of the lease.

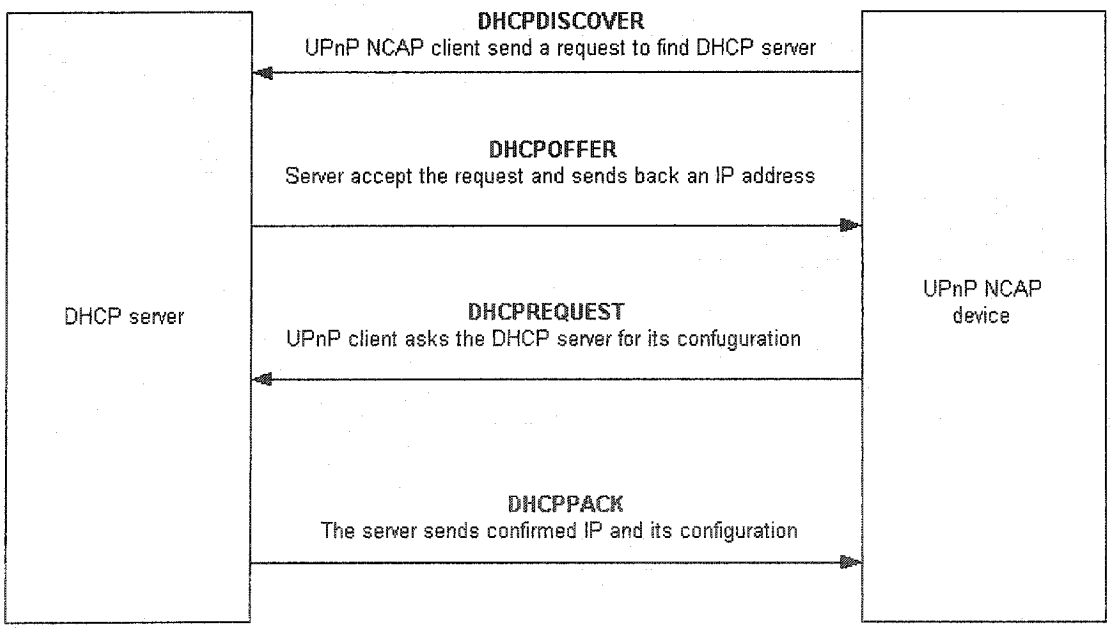


Figure 14: The Communication Between UPnP NCAP Device and DHCP Server

4.3.1.3.2 Operation AssignAuto_IPaddress Behaviour Specification

This method launches the Auto-IP algorithm. The algorithm startsif a UPnP NCAP device determines that a DHCP server is not available.

The Auto-IP algorithm involves three major steps. The first step defines IP address. The second step tests the chosen IP address, and the final step continues to check for the availability of a DHCP server. The Auto-IP chooses an IP address in the range 169.254/16 with the subnet mask 255.2555.0.0. It uses the pseudo-random algorithm to choose the IP address. The start point for the pseudo-random algorithm is the UPnP NCAP Ethernet MAC address. If the chosen address is used by another device, the UPnP NCAP repeats the algorithm .

After choosing the IP address, the UPnP NCAP device checks the address by sending the ARP probe message. The probe message is an ARP request that includes the hardware address of the sender and the IP address that set up in zeroes.

The UPnP NCAP device gets a response and analyzes the received IP address. If the received IP address matches with the chosen address, the UPnP NCAP device assumes that the IP address is already chosen by another device and performs the pseudo-random algorithm again. If nobody replies with an IP address that matches the chosen address, the UPnP NCAP device assumes that it is free to use. Despite successfully choosing an IP address, the UPnP NCAP device continues to check for the availability of a DHCP server by sending, a DHCPDISCOVERY message every 5 minutes . If a DHCPOFFER message is received, the device executes a DHCP allocation and releases the previous IP address allocated by Auto-IP. The device also cancels any advertisements made at the old address and reissues the advertisements to the new IP address.

When the device joins the network, it advertises its presence by sending a NOTIFY message.

For each type of device (the root, embedded or device's service) the publication sends a certain number of NOTIFY messages. For the root device, the publication sends three messages, while sending two messages for the embedded device, and one message for services. For further information see Section 2.1.3.1.1

When the device is removed from the network, it sends out *ssdp:bye-bye* messages corresponding to all advertised *ssdp:alive* messages.

The UPnP NCAP control point requests the device Description document by sending a HTTP request GET. The request is sent using the URL, which the control point obtains from the device announcement message.

4.3.1.3.3 Publication Device_Description Behaviour specification

When the UPnP NCAP device gets a Get_Device_Description message it responds with the publication Device_Description. The publication then sends the Device Description document containing relevant information needed to work with the UPnP NCAP device to the control point.

The response is sent in the form of a HTTP response message. The body of the message consists of the Device Description document in xml format.

The format of Device Description document is as follows:

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <URLBase> the base URL of the UPnP NCAP device </URLBase>
  <device>
      <device type>urn:upnp-ncap:device: NCAP_type of NCAP: </device type>
      <friendlyName> NCAP_short name of NCAP device </friendlyName>
      <manufacturer>manufacturer name of NCAP device</manufacturer>
      <modelName>name of model</modelName>
      <modelNumber>model number</modelNumber>
      <serialNumber>NCAP device serial number</serialNumber>
      <UDN>uuid:UUID of the NCAP device<UDN>
      <serviceList>
            <service>
              <serviceType>urn:upnp-ncap:service:ControlStatusSTIM.1
              </serviceType>
              <serviceId> urn:upnp-ncap:serviceId: ControlStatusSTIM.0001
              </serviceId>
              <SCPDURL> ControlStatusSTIM /scpd.xml</SCPDURL>
              <controlURL> ControlStatusSTIM /control</controlURL>
              <eventSubURL> ControlStatusSTIM /event</eventSubURL>
            </service>
            <service>
```

```
            <serviceType>urn:upnp-ncap:service:ParameterSTIM1.1</serviceType>
            <serviceId> urn:upnp-ncap:serviceId:ParameterSTIM1.0001
            </serviceId>
            <SCPDURL> ParameterSTIM1/scpd.xml</SCPDURL>
            <controlURL> ParameterSTIM1 /control</controlURL>
            <eventSubURL> ParameterSTIM1 /event</eventSubURL>
          </service>
          The other Parameter's services
      </serviceList>
      <presentationURL> the URL for theUPnP NCAP device presentation
      </presentationURL>
  </device>
</root>
```

Note: The numbers and names of ParameterSTIM services depend on the particular implementation of a UPnP NCAP.

4.3.1.3.4 Subscription Get_Service_Description Behaviour Specification

After receiving the device Description document, the control point retrieves, for each defined service, a Service Description document. The Services Description documents are requested by sending the HTTP command GET to the UPnP NCAP device.

The request is sent using the URL, which the control point obtains from the device announcement message.

4.3.1.3.5 Publication Service_Description Behaviour Specification

When the UPnP NCAP device receives a Get_Service_Description message, it responds with the publication Service_Description. The publication sends, to the control point, a Service Description document containing all information necessary to work with the UPnP NCAP device service.

The response is sent in the form of a HTTP response message. The body of the message consists of the Service Description document in xml format. To learn the general requirements of a Device Description document see Section 2.1.4.2 .

The Service Description document has the following format:

```
<?xml version="1.0"?>
<scpd xmlns="urn:upnp-ncap:service-1-0">
   <specVersion>
     <major>1</major>
     <minor>0</minor>
  </specVersion>
  <actionList> List of actions </actionList>
  <serviceStateTable> List of state variables </serviceStateTable>
</scpd>
```

<actionList> - Consists of a list of actions for the service. Each service possesses different numbers and types of actions. For more information please see Section 2.1.4.2.1.

<serviceStateTable> - Consists of a list of state variables that reflect the current state of the service. For more information refer to Section 2.1.4.2.2.

4.3.1.3.5.1 The actionList *Section of the UPnP NCAP ControlStatusSTIM Service*

The *actionList* section of the ControlStatusSTIM service has the following format:

```
<actionList>
   <action>
      <name> ResetSTIM</name>
   </action>
   <action>
      <name>CalibrateAllChannels</name>
   </action>
   <action>
       <name>EnableAllSensors</name>
    </action>
    <action>
        <name>DisableAllSensors</name>
    </action>
    <action>
```

```
        <name> STIMhardwareErrorBit</name>
        <argumentList>
                <argument>
                    <name>STIMHardwareError</name>
                    <direction>out </direction>
                    <retval />

                <relatedStateVariable>STIMHardwareErrorBit</relatedStateVariable>
                </argument>
        </argumentList>
    </action>
    <action>
        <name> STIMEventBit</name>
        <argumentList>
                <argument>
                    <name>STIMEvent</name>
                    <direction>out </direction>
                    <retval />
                    <relatedStateVariable>STIMEventBit</relatedStateVariable>
                </argument>
        </argumentList>
     </action>
     <action>
        <name> STIMMissedEventBit</name>
        <argumentList>
                <argument>
                    <name>STIMMissedEvent</name>
                    <direction>out </direction>
                    <retval />
                    <relatedStateVariable>STIMMissedEventBit
                     </relatedStateVariable>
                </argument>
        </argumentList>
      </action>
</actionList>
```

ResetSTIM action provides reset of all channels of STIM.

CalibrateAllChannels action provides calibration of all channels of STIM.

EnableAllSensors action enables all sensors.

DisableAllSensors action disables all sensors.

STIMhardwareErrorBit action returns the *STIMHardwareError* argument that reflects the state of the hardware error bit of the STIM. The *STIMHardwareError* argument is set to TRUE if the STIM has a hardware error.

STIMEventBit action returns the *STIMEvent* argument. The argument is set to TRUE if all channels (parameters) have set up their data sample time event.
STIMMissedEventBit action returns the *STIMMissedEvent* argument. The argument is set to TRUE if all channels (parameters) have missed data sample time events.

4.3.1.3.5.2 The actionList Section of the UPnP NCAP ParameterSTIM Service

The *actionList* section of the ParameterSTIM service is of the following format:

```
<actionList>
    <action>
      <name> ResetChannel</name>
  </action>
  <action>
     <name>CalibrateChannel</name>
  </action>
  <action>
     <name>EnableChannel</name>
  </action>
  <action>
      <name>DisableChannel</name>
  </action>
  <action>
      <name>ReadParameter</name>
      <argumentList>
            <argument>
                  <name>ReadParameterValue</name>
                  <direction>out </direction>
                  <retval />
                  <relatedStateVariable>ParameterValue</relatedStateVariable>
            </argument>
      </argumentList>
  </action>
  <action>
      <name>UpdateParameter</name>
      <argumentList>
            <argument>
                  <name>UpdateParameterValue</name>
                  <direction>out </direction>
                  <retval />
                  <relatedStateVariable>ParameterValue</relatedStateVariable>
            </argument>
      </argumentList>
  </action>

  <action>
      <name>WriteParameter</name>
      <argumentList>
            <argument>
                  <name>WriteParameterValue</name>
```

```
                    <direction>in </direction>
                    <retval />
                    <relatedStateVariable>ParameterValue</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name> ChannelHardwareErrorBit</name>
            <argumentList>
                <argument>
                    <name>ChannelHardwareError</name>
                    <direction>out </direction>
                    <retval />
                     <relatedStateVariable>ChannelHardwareErrorBit
                     </relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name> ChannelEventBit</name>
            <argumentList>
                <argument>
                    <name>ChannelEvent</name>
                    <direction>out </direction>
                    <retval />
                    <relatedStateVariable>ChannelEventBit</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name> ChannelMissedEventBit</name>
            <argumentList>
                <argument>
                    <name>ChannelMissedEvent</name>
                    <direction>out </direction>
                    <retval />
                    <relatedStateVariable>ChannelMissedEventBit
                     </relatedStateVariable>
                </argument>
            </argumentList>
        </action>

</actionList>
```

*Reset*Channel action provides reset of a channel.

CalibrateChannes action provides calibration of a channel.

EnableSensor action enables a sensor of a channel.

DisableSensor action disables a sensor of a channel.

ReadParameter action returns the data of the parameter's last data sample.

UpdateParameter action causes the immediate reading of a parameters value from a sensor and returning a result to the control point.

WriteParameter action writes data to the channel sensor.

ChannelHardwareErrorBit action returns the *ChannelHardwareError* argument that reflects the state of the hardware error bit of the channel. The *ChannelHardwareError* argument is set to TRUE if the channel has a hardware error.

ChannelEventBit action returns the *ChannelEvent* argument. The argument is set to TRUE if a channel (parameter) has set up its data sample time event.

ChannelMissedEventBit action returns the *STIMMissedEvent* argument. The argument is set to TRUE if channel (parameter) has missed a data sample time event.

4.3.1.3.5.3 The serviceStateTable Section of the UPnP NCAP ControlStatusSTIM Service

The *serviceStateTable* section of the ControlStatusSTIM service is of the following format:

```
<serviceStateTable>
        <stateVariable   sendEvents="yes">
                <name> STIMHardwareErrorBit </name>
                <dataType>boolean</dataType>
        </stateVariable>
        <stateVariable   sendEvents="yes">
                <name> STIMEventBit </name>
                <dataType>boolean</dataType>
        </stateVariable>
        <stateVariable   sendEvents="yes">
                <name> STIMMissedEventBit </name>
                <dataType>boolean</dataType>
        </stateVariable>
</serviceStateTable>
```

STIMHardwareErrorBit variable keeps the state of the STIM Hardware Error Bit. A change in the state of the variable causes an event.

STIMEventBit variable keeps the state of the STIM Event Bit. A change in the state of the variable causes an event.

STIMMissedEventBit variable keeps the state of the STIM Missed Event Bit. A change in the state of the variable causes an event.

4.3.1.3.5.4 The serviceStateTable Section of the UPnP NCAP ParameterSTIM Service

The *serviceStateTable* section of the ParameterSTIM service has the following format:

```
<serviceStateTable>
        <stateVariable    sendEvents="yes">
                <name> ParameterValue </name>
                <dataType>the type of the state variable depends on the type of the
                channel </dataType>
                <allowedValueRange>Value range</allowedValueRange>
        </stateVariable>
        <stateVariable    sendEvents="yes">
                <name> ChannelHardwareErrorBit </name>
                <dataType>boolean</dataType>
        </stateVariable>
        <stateVariable    sendEvents="yes">
                <name> ChannelEventBit </name>
                <dataType>boolean</dataType>
        </stateVariable>
        <stateVariable    sendEvents="yes">
                <name> ChannelMissedEventBit </name>
                <dataType>boolean</dataType>
        </stateVariable>
</serviceStateTable>
```

The *ParameterValue* variable keeps the last data sample of the Parameter. The data type as well as the allowed value range of the parameter depend on the type of the channel (parameter) and the vendor's implementation of the channel.

The *ChannelHardwareErrorBit* variable keeps the state of the channel Hardware Error Bit. A change in the state of the variable causes an event.
The *ChannelEventBit* variable keeps the state of the channel Event Bit. A change in the state of the variable causes an event.
The *ChannelMissedEventBit* variable keeps the state of the channel's Missed Event Bit. A change in the state of the variable causes an event.

4.3.2 Modification of the Asynchronous Client Port Class

The Asynchronous Client Port Class of IEEE1451.1 provides a non-blocking client side in the client-server communication model. To use this class efficiently, objects of the class must work in a multitasking program environment.

The new UPnP Client Port class, inherited from the Asynchronous Client Port Class, includes additional operations and attributes to support the Control functions of a UPnP device. In general, the UPnP Control functions are based on SOAP. SOAP uses both XML and HTTP for its remote call mechanism procedure and for Web messaging. Communication using UPnP is simple because SOAP's request/respond messages easily match the HTML model.

4.3.2.1 UPnP Client Port Class

Class: IEEE1451_UPnP_ClientPort
Parent class: IEEE1451_AsynchronousClientPort
Class ID: 1.1.3.1.1.2.1
Description: The UPnP Client Port class provides client side or control point functionality for UPnP communications.
Class summary:
Local operations: See Table 15.

Operation name	Requirements
UPnPControlMessageRequest	(m)
GetResultUPnPControlMessage	(m)

Table 15: Local Operations of UPnP Client Port Class

4.3.2.1.1 UPnP Client Port Class Behaviour

4.3.2.1.1.1 Operation UPnPControlMessageRequest Behaviour Specification

The UpnPControlMessageRequest operation sends a request to a UPnP NCAP device to execute a method on the device side. The UPnP NCAP control point gets the names of the method for the requested service together with input arguments from the Device and Services description document. The request is sent in the form of an HTTP SOAP message. The message consists of two parts: a header and a body. The header contains control information, while the body is presented in an xml document format describing the request to execute the method and its input arguments.

The operation UPnPControlMessageRequest is a non-blocking function and it returns immediately after sending the message to the UPnP NCAP device. If the control point expects a result from the UPnP NCAP device, it executes the operation GetResultUPnPControlMessage instead.

4.3.2.1.1.2 Operation GetResultUPnPControlMessage Behaviour Specification

The GetResultUPnPControlMessage operation returns the result of executing an UPnP NCAP requested method.

When the UPnP NCAP device receives an HTTP SOAP request message, sent by a UPnPControlMessageRequest, it executes the requested method and sends the result back to the control point. The result is sent in the form of a UPnP HTTP SOAP response message. The message contains two parts – a header and a body. The header contains control information of the message, such as the type of message, the destination URL, content languageetc., while thebody of the message, written in xml document format, consists of the result arguments of the requested method.

The GetResultUPnPControlMessage function is used to obtain a result after running the UPnPControlMessageRequest operation. This function is non-blocking. Its internal implementation uses the instance of the Conditional Variable Service class of IEEE 1451 that implements concurrent activities in IEEE 1451.1.

4.3.3 *Modification of Event Generator Publisher Port Class*

The Event Generator Publisher Port class provides a Publisher side functionality of Publisher-Subscriber model of NCAP. In this model, the publisher is the source of events and it registers a subscriber. Upon the occurrence of an event, the publisher sends an event notification to the subscriber. The subscriber may cancel the subscription if it is no longer interested in receiving event notification.

The UPnP model uses one similar to NCAP publisher-subscriber model. The UPnP publisher-subscriber model relies on GENA. In addition to the base functions of NCAP, GENA offers features that allow subscription for a duration of time. To maintain such a subscription, the subscriber must periodically renew the subscription. GENA uses HTTP as a transport protocol in publisher-subscriber communication. Three HTTP messages are used in GENA. They are SUBSCRIBE, UNSUBSCRIBE, and NOTIFY.

The SUBCRIBE message is used to receive event notification and to renew an existing subscription. UNSUBSCRIBE terminates a subscription.

The NOTIFY message sends notifications to a client or a subscriber.

4.3.3.1 *UPnP Publisher Port Class*

Class: IEEE1451_UPnP_PublisherPort

Parent class: IEEE1451_EventGeneratorPublisherPort

Class ID: 1.1.3.1.2.2.1.1

Description: UPnP Publisher Port class provides publisher capabilities for a NCAP UPnP device. Local operations: See Table 16.

Operation name	Requirements
PublishUPnP_Event	(m)
Publish_InitialUPnP_Event	(m)
Get_XMLEvent_scema	(m)
SetMaximumRate	(m)
GetMaximumRate	(m)
SetMinimumRate	(m)
GetMinimumRate	(m)

Table 16: Local Operations of UPnP Publisher Port Class

4.3.3.1.1 UPnP Publisher Port Class Behaviour

The UPnP NCAP publisher-subscriber model combines features of the UPnP and IEEE1451.1 publisher-subscriber models. The Dot2 Transducer block class presents a STIM service and the Parameter's classes present Parameter services. Figure 14 depicts the UPnP NCAP event schema.

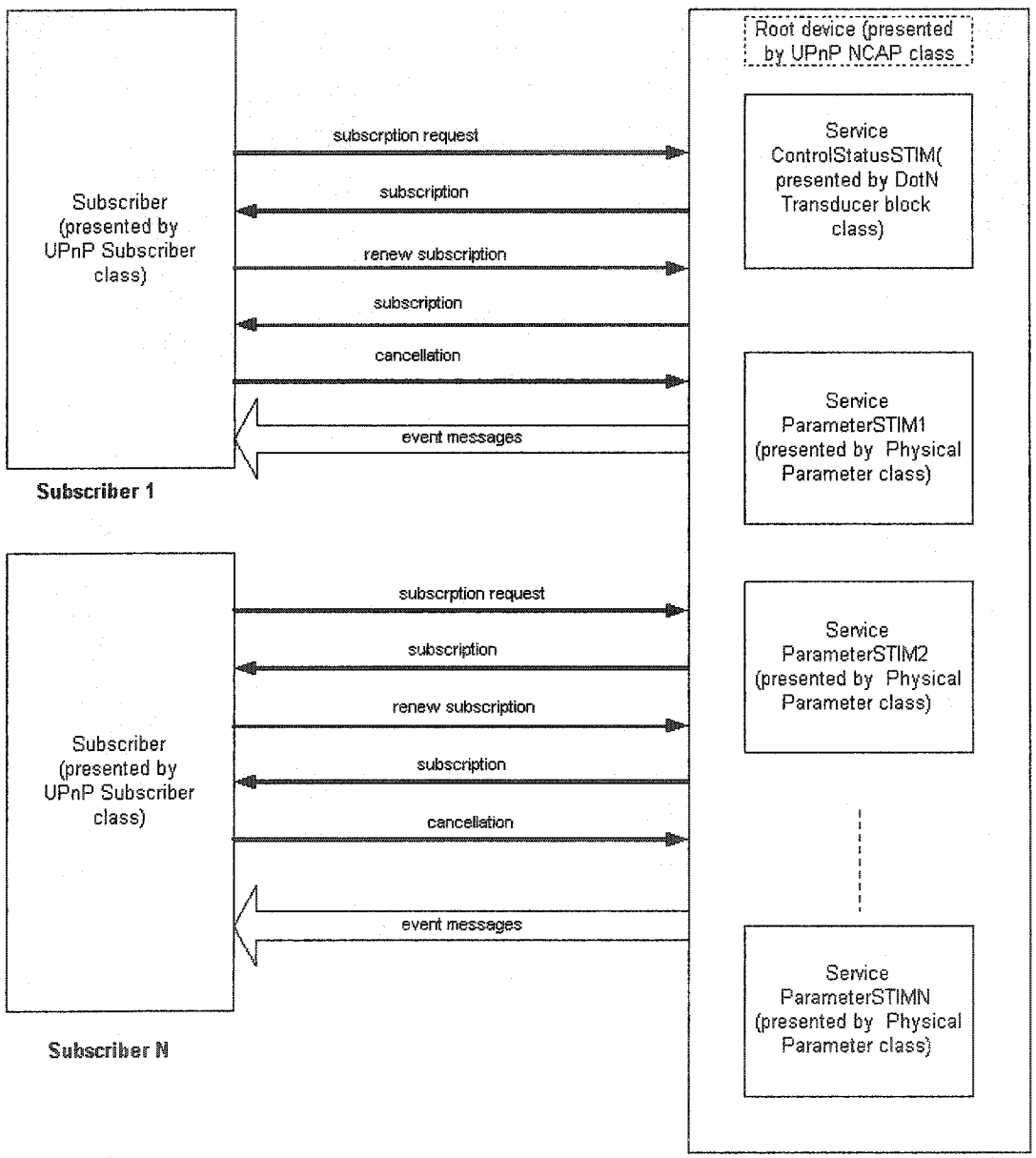


Figure 15: UPnP NCAP Communication Event Schema

In the UPnP NCAP event communication schema, the subscriber plays the role of a UPnP NCAP control point, while the publisher plays the role of a UPnP NCAP device.

4.3.3.1.1.1 Operation PublishUPnP_Event Behaviour Specification

The PublishUPnP_Event class provides event notification to registered subscribers. The event notification is sent as a HTTP document, consisting of two parts: a header and a body. The header includes control information such as the URL path of the service generating the event, the IP and the port of destination subscriber, the ID of the subscription, and the sequence number of the event being sent to the subscriber.
The body of the event message is an *xml* document that contains the name of the changed state variable and its new value.

The first part of the message from the **NOTIFY** element to the **SEQ** is the header, whilethe second part is the body and includes *xml* document.

4.3.3.1.1.2 Operation Publish_InitialUPnP_Event Behaviour Specification

The *Publish_InitialUPnP_Event* operation sends the initial event notification message. The message is sent after receiving the subscription and contains all state variables of the device's service, which are enabled to get events. The format of initial event notification message is similar to the regular event message.

4.3.3.1.1.3 Operation Get_XMLEvent_schema Behaviour Specification.

The Get_XMLEvent_schema returns a template of the event notification message. The function is used to create an event notification message.

4.3.4 Modification of Subscriber Class

The Subscriber Port class of IEEE 1451.1 provides a mechanism to subscribe to a publication. The Subscriber Port class is used on the client side of the publisher-subscriber model. An object that needs to subscribe sends a subscribe message to a publisher. When the subscriber receives an event notification message, it executes, a call back function for all those registered for this type of event. The UPnP subscribing

mechanism is similar to IEEE1451.1, except that messages in UPnP are sent by an HTTP protocol in an XML format.

This class is inherited from the Subscriber Port class and it keeps all methods and algorithms needed to register and run callback functions,making the process of obtaining events much more efficient than the UPnP specification.

4.3.4.1 UPnP Subscriber Class

Class: IEEE1451_UPnP_SubscriberPort.

Parent class: IEEE1451_SubscriberPort

Class ID: 1.1.3.2.1

Description: The UPnP Subscriber Port class provides the UPnP subscriber's functionality to the IEEE 1451.1

Local operation: See Table 17.

Operation name	Requirements
SubscribeUPnP_event	(m)
UnSubscribeUPnP_event	(m)
RenewSubscription	(m)
ExecuteCallBackFunction	(m)
EventNotificationReplay	(m)

Table 17. The Local operation of UPnP Subscriber Class

4.3.4.1.1 UPnP Subscriber Port Class Behaviour

The Subscriber class provides the subscriber functionality of a UPnP NCAP control point and combines the functionalities of the IEEE 1451.1 and UPnP specifications. For each successful subscription, the subscriber gets a subscription ID from a publisher. This subscription ID is matched with a callback function that is used in the case of an event.

4.3.4.1.1.1 Operation SubscribeUPnP_event Behaviour Specification

The *SubscribeUPnP_event* operation provides a subscription request to a publisher.

The UPnP NCAP control point gets the publisher IP address and the event URL from the Device Description document. The subscription request is sent in the form of an HTTP document.

The publisher receiving the subscription request responds with a subscription response within 30 seconds.

4.3.4.1.1.2 Operation UnSubscribeUPnP_event Behaviour Specification

The *UnSubscribeUPnP_event* operation provides the cancellation of a subscription.

When the UPnP NCAP control point no longer needs a subscription, it sends an unsubscribe message to the publisher. The message is sent as an HTTP message.

4.3.4.1.1.3 Operation RenewSubscription Behaviour Specification

The *RenewSubscription* operation allows for the renewal of a subscription. Each subscription has a time during for which the subscription is valid. To renew the subscription, a subscriber sends a renew subscription message. The renewal message is sent as an HTTP message and must be delivered before the subscription expires. The *SID* element is the subscription ID to renew.

4.3.4.1.1.4 Operation EventNotificationReply Behaviour Specification

The *EventNotificationReplay* operation sends a reply message after receiving an event notification message. The format of the message is as follows:

HTTP/1.1 200 OK

4.3.4.1.1.5 Operation ExecuteCallBackFunction Behaviour Specification

The *ExecuteCallBackFunction* operation executes a callback function in the case of receiving an event notification message. After receiving a response to a subscribe message, a control point matches the received subscriber ID with the desired callback function. The callback function treats the event message and calls the *EventNotificationReplay* operation after competing its work.

4.3.5 Support Web Interface in a NCAP Device

A UPnP device must have Web server capabilities to communicate with UPnP NCAP control points. Almost all functions of a UPnP device, such as description, control and event reporting, use the HTTP protocol requiring the using of a Web server.
To support the Web interface, NCAP uses a new class – Web server. In addition to supporting HTTP, the class is responsible for running the Presentation function of the UPnP NCAP device.

In the current version of the UPnP specification, the Presentation function is optional, but very useful in terms of the convenience of using a device. The presence of this function allows a user to have easy access to the device.
The Presentation function maintains a presentation page of the device. The presentation page is developed using*html* and can be accessed through a standard Web browser.

There are two approaches to design and implement the Presentation function. The first approach does not require modification of the device. It uses existing mechanisms of communication but the control point is more complex. The client or the control point needs some form of scripting to compose and send SOAP control messages to the device.

The second approach requires a more complex device design, but the client side uses only a web browser without any extra scripting code. The device should not only handle standard SOAP messages but also presentation device messages.

To minimize device implementation, we chose the first approach. This approach could also decrease the network traffic because in this case the device would send only the changed states of the system.

4.3.5.1 Web Server Class

Class: IEEE1451_WebServer
Parent class: IEEE1451_Service
Class ID: 1.1.3.5
Description: The Web server class provides Web server capabilities necessary to implement UPnP functions in a NCAP device.
Local operations: See Table 18.

Operation name	Requirements
StartWebServer	(m)
StopWebServer	(m)
SetPathPresentationPage	(o)
GetPathPresentationPage	(o)

Table 18: Local Operations of Web Server Class

4.3.6 Support of XML File

All major communications in UPnP use XML files [36]. XML have become the de-facto Internet standard for information representation. These meta-languages allow specification of the structure of data in a document as well ashow elements of the structure relate to one another. XML also provides easy parsing of the document because it contains self-describing information about the rules used to compose them.

Because of using XML for communication, the UPnP NCAP device needs a mechanism to compose and retrieve information contained in a XML document. To treat XML documents, the UPnP NCAP device uses an XML file class inherited from the File class.

4.3.6.1 XML File Class

Class: IEEE1451_XMLFile

Parent class: IEEE1451_File

Class ID: 1.1.2.3.2

Description: The XML file class provides the mechanism to work with a XML file.

Local operations: See Table 19.

Operation name	Requirements
OpenXMLfile	(m)
CloseXMLfile	(m)
GetNextXMLNode	(m)

Table 19: Local Operations of XML File Class

4.3.6.1.1 XML File Class Behaviour

The XML class provides parsing of XML documents. It has three methods: *OpenXMLfile*, *CloseXMLfile,* and *GetNextXMLNode.*

The *OpenXMLfile* prepares an XML document for parsing. It creates an internal XML_DOCUMENT structure that contains all necessary information for parsing. The XML_DOCUMENT structure has the following look:

```
structure XML_DOCUMENT
{
        String          xml_document,
        Uinteger32      length_string,
        Uinteger32      current_offset,
        Uinteger32      handle_xml_document
}
```

xml_document – contains the XML document.

length_string – contains the length of the XML document.

current_offset – contains the current offset in the XML document. The current offset represents the beginning of a XML element.
handle_xml_document – contains the handle of the open XML document. The handle returns as the *xml_handle* output argument of the *OpenXMLfile* operation.

The *CloseXMLfile* closes the XML document and destroys the XML_DOCUMENT structure associated with this document.

The *GetNextXMLNode* function processes the XML document. It returns an XMLNode that contains information about a processed the XML element. The XMLNode structure has the following look:

```
structure  XMLNode
{
        String  Name;
        Uinteger32 NameLength;

        String  Prefix;
        Uinteger32  PrefixLength;

        String  Value;
        Uinteger32 ValueLength;
}
```

Name – contains the name of the xml element.
NameLength – contains the length of the *Name* string.
Prefix – contains, if applicable, the name of a prefix of an xml element.
PrefixLength – contains the length of the *Prefix* string.
Value – contains the name of the value of an xml element.
ValueLength – contains the length of the *Value* string.

5 Implementation of UPnP NCAP

The purpose of the implementation of the UPnP NCAP model is to evaluate the design of the UPnP Smart Sensor. The evaluation model consists of the major components of the Smart Sensor, while supporting all the components and the protocols of the UPnP specification.

The implementation of the UPnP NCAP model consists of two parts: a UPnP NCAP device and a UPnP Control Point. The UPnP NCAP device provides a network interface to the IEEE 1451.2 transducer and contains three groups of IEEE 1451.2 signals: Meta-TEDS, an Actuator Channel and a Sensor Channel [29]. The UPnP NCAP Control Point is used to present the device capabilities to the user and it is implemented as an application program. The UPnP protocols are used for communication between a UPnP NCAP device and the UPnP Control Point. Figure 16 depicts a schema of the major components of the UPnP NCAP device and the UPnP NCAP Control Point.

Both the UPnP NCAP Device and the UPnP NCAP Control Point are designed as Windows dialog applications and work on Windows 95/98/NT/2000/XT. To implement these devices "Intel® Authoring Tools for UPnP Technologies" may be used, providing a suitable environment for the framework to develop UPnP devices [28],[35]. Both the UPnP NCAP device and the UPnP NCAP Control Point were developed by using Visual Studio C++ version 6.

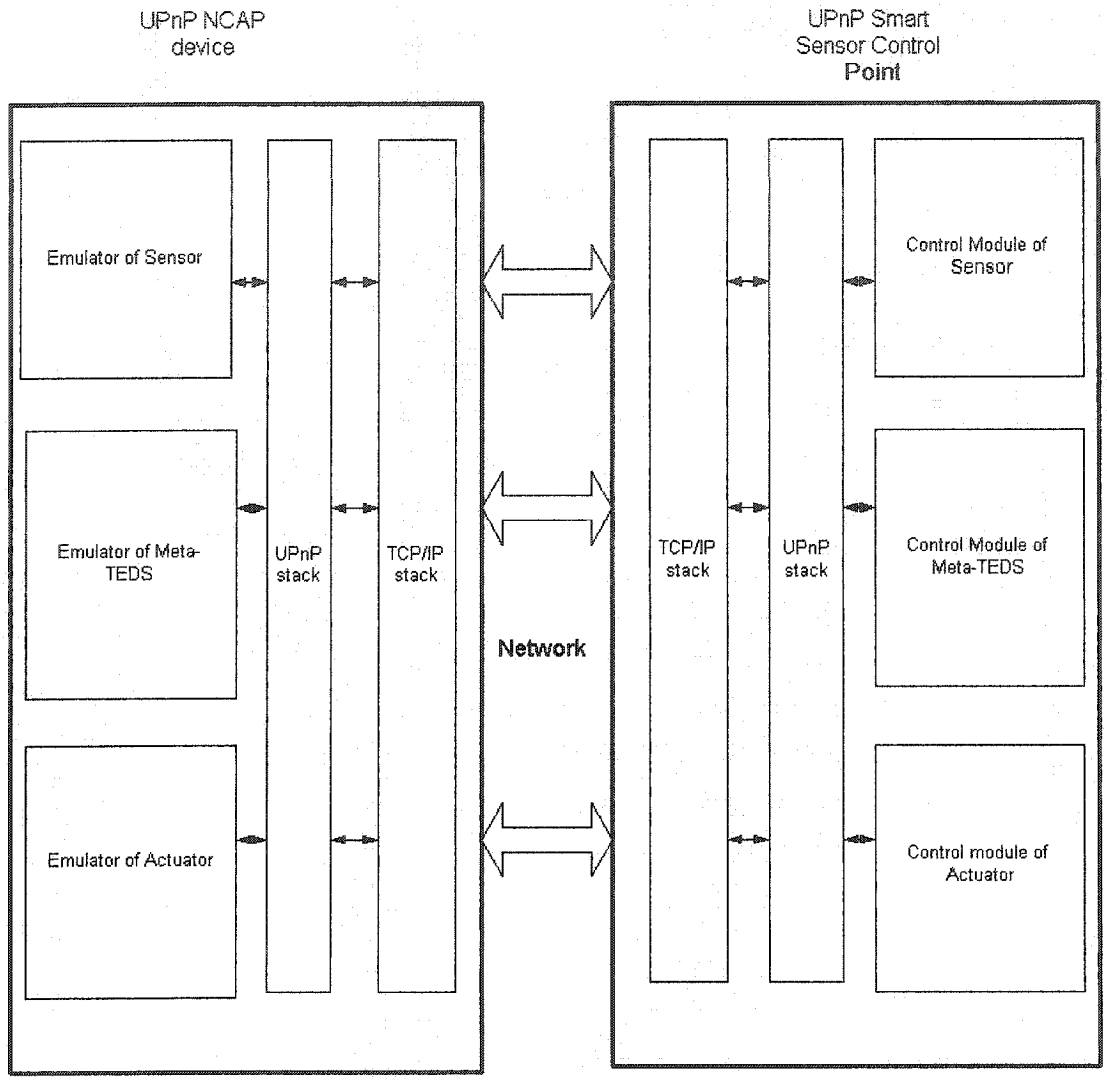


Figure 16. Implementation of emulation model of UPnP NCAP Device and Control Point

5.1 Design the communication between an UPnP NCAP Device and a NCAP Control Point

To communicate with each other, a UPnP Device and a NCAP Control Point use the TCP/IP and UPnP protocols (Figure 1). On the bottom of the stack lies the IP, followed by the TCP and the UDP protocols. UDP is used for the discovery mechanism and TCP is used for transferring the NCAP parameter's data.

The current implementation of the NCAP Device and Control Point does not use DHCP and Auto-IP protocols directly. They rely on the network functionality supplied by the Windows OS. To define an IP address that can be used for the UPnP communication, both the NCAP Device and the Control Point use the Windows SDK function *gethostbyname()* that returns the lists of the IP addresses used by the PC Workstation.

5.1.1 Discovery of NCAP Device

When a NCAP Device starts to work it sends two types of messages: *byebye* and *alive*. The point of sending *byebye* messages is to cancel all opened UPnP communications that were opened in the previous session of the Device work. The *alive* messages advertise the presence of the Device and all available Device services. Both *byebye* and *alive* messages are sentusing multicast addresses.

When a NCAP Device starts to work it sends two types of messages: *byebye* and *alive*. The point of sending *byebye* messages is to cancel all UPnP communications that were opened in the previous session. The *alive* messages advertise the presence of the Device and all available Device services. Both *byebye* and alive messages are send by using the multicast address: 239.255.255.250:1900. This address is a standard address and it is defined by the UPnP committee.

The first message to send is *byebye*, followed by alive. The NCAP Device sends two identical groups of *byebye* and *alive* messages. Each group consists of six messages. The first message is send to *the rootdevice* that defines the main access point to the device. The next two messages are sent to the NCAP device itself – one message is send

using the Device UUID and the second using the Device name. The last three messages are sent for each of these device services - Actuator STIM service, ControlSTIM service and SensorSTIM service.

The format of *the byebye* message is as shown below:

NOTIFY * HTTP/1.1

HOST: 239.255.255.250:1900

NTS: ssdp:byebye

USN: uuid:*<uuid of NCAP device>* ::upnp:*<name of device or service>*

NT: upnp: :*<nameof device or service>*

Content-Length: 0

where :*<name of device or service>* can have the following name : rootdevice, *<uuid of device>*, ActuatorSTIM, ControlStatusSTIM and SensorSTIM.

The format of *alive* messages is as follows:

NOTIFY * HTTP/1.1

LOCATION: http://192.168.62.65:54376/

HOST: 239.255.255.250:1900

SERVER: WINDOWS, UPnP/1.0

NTS: ssdp:alive

USN: uuid:*<uuid of device>*::upnp:*<name device or service>*

CACHE-CONTROL: max-age=1800

NT: upnp:rootdevice

where :*<name of device or service>* can have following name : rootdevice, *<uuid of device>*, ActuatorSTIM, ControlStatusSTIM and SensorSTIM.

If the NCAP Control Point has already started it requests for the Device's Description. Otherwise the NCAP Device switches itself to the waiting mode.

When the NCAP Control Point starts, it sends the "M-SEARCH" message for the discovery of an NCAP Device. The format of the "M-SEARCH" method is as follows:

M-SEARCH * HTTP/1.1

MX: 3

ST: urn:schemas-upnp-org:device:NCAP_device:1

HOST: 239.255.255.250:1900

MAN: "ssdp:discover"

If the NCAP Device has already started it responds with following message:

HTTP/1.1 200 OK

LOCATION: http://192.168.62.65:50823/

EXT:

SERVER: WINDOWS, UPnP/1.0, Intel MicroStack/1.0.1545

USN: uuid:13c921d3-8413-46ed-914d-e02db930ee09::urn:schemas-upnp-org:device:NCAP_device:1

CACHE-CONTROL: max-age=1800

ST: urn:schemas-upnp-org:device:NCAP_device:1

Figure 17 shows the discovery phase of the NCAP Device. The NCAP Device starts and sends two types HTTP messages – *byebye* and *alive*. The *byebye* messages are sent to cancel all previous opened connections to NCAP Control Points. The *alive* messages are sent to notify that the NCAP Device starts to work. Getting a Description request message the NCAP Device goes to the Description phase.

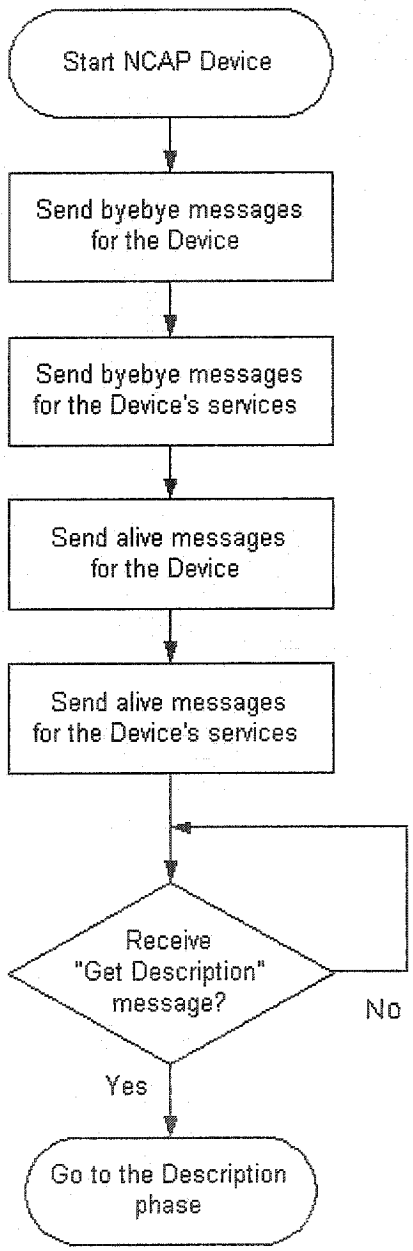


Figure 17. Algorithm of NCAP Device Discovery phase.

Figure 18 shows the discovery phase of the NCAP Control Point.

The NCAP Control Point starts and sends "M-SEARCH" message to discovery a NCAP Device. After receiving an *alive* message the NCAP Control Point goes to the Description phase.

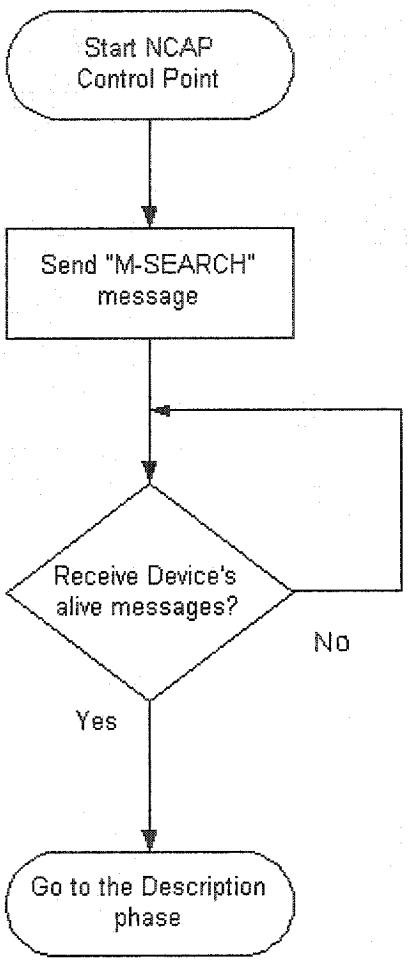


Figure 18.Algorithm of NCAP Control Point discovery phase

5.1.2 Description phase

In the emulation model the NCAP Device offers three services: ControlStatusSTIM, ActuatorSTIM and SensorSTIM.

The ControlStatusSTIM emulates the META-TEDS. Table 20 contains the list of ControlStatusSTIM signals.

N	Name of signal	Direction	Description
1	Disable Hardware Error Bit	out	Switch off the Hardware Error Bit
2	Enable Hardware Error Bit	out	Switch on the Hardware Error Bit
3	Enable all channels	in	
4	Disable all channels	in	
5	Calibrate all channels	in	
6	Reset STIM	in	Reset NCAP device
7	STIM event	out	
8	STIM event missed	out	Send in the case when the device could not send a STIM event

Table 20 Signals of ControlStatusSTIM service

The ActuatorSTIM service emulates an actuator channel. Table 21 contains the list of ActuatorSTIM signals.

N	Name of signal	Direction	Description
1	Disable Hardware Error Bit	out	Switch off Hardware Error Bit of the channel
2	Enable Hardware Error Bit	out	Switch on Hardware Error Bit of the channel
3	Enable channel	in	
4	Disable channel	in	
5	Calibrate channel	in	
6	Parameter value	in	Value of Actuator's parameter. The value is send from NCAP Control Point to NCAP Device

Table 21 Signals of ActuatorSTIM service

The SensorSTIM service emulates a sensor channel. Table 22 provides the list of SensorSTIM service signals.

N	Name of signal	Direction	Description
1	Disable Hardware Error Bit	out	Switch off Hardware Error Bit of the channel
2	Enable Hardware Error Bit	out	Switch on Hardware Error Bit of the channel
3	Enable channel	in	
4	Disable channel	in	
5	Calibrate channel	in	
6	UpdateAndRead	in	NCAP Device receives this signal immediately update the current value of sensor and send to NCAP Control Point
7	Sensor parameter	out	Value of sensor
8	Sensor read event	out	Event is send when NCAP Device finishes to read sensor value
9	Sensor missed read event	out	Event is send if the device fail to send Sensor read event

Table 22 Signals of SensorSTIM service

After completing the discovery phase, the NCAP Control Point sends a request to obtain Descriptions of the Device and its services. Such requests are sent in the form of HTTP commands. Requests to obtain the NCAP Device Description resemble the following commands:

GET / HTTP/1.0
Host: 192.168.62.65:61745
User-Agent: WINDOWS, UPnP/1.0

A request to get the Device service's Description has following look:

GET /<*name of device's service*>/scpd.xml HTTP/1.0
Host: 192.168.62.65:61745
User-Agent: WINDOWS, UPnP/1.0

where *<name of device's service>* - can be either SensorSTIM, ControlStatusSTIM or ActuatorSTIM.

The NCAP Device receiving the requests responds by sending HTTP messages that contain Descriptions for the Device and the services of each . The HTTP message that sends the Device Description has following look:

```
HTTP/1.0 200  OK
CONTENT-TYPE:  text/xml
Server: WINDOWS, UPnP/1.0
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="urn:schemas-upnp-org:device-10">
<specVersion><major>1</major><minor>0</minor></specVersion>
<device><deviceType>urn:schemas-upnp-org:device:NCAP_device:1</deviceType>
<friendlyName>%s</friendlyName>
<manufacturer>University of Ottawa</manufacturer>
<manufacturerURL>http://www.uottawa.com</manufacturerURL>
<modelDescription>NCAP UPnP emulation device W
indows version</modelDescription>
<modelName>NCAP UPnP </modelName>
<modelNumber>A1</modelNumber>
<serialNumber>0000001</serialNumber>
<UDN>uuid:13c921d3-8413-46ed-914d-e02db930ee09</UDN>
<serviceList><service><serviceType>urn:schemas-upnp-org:ActuatorSTIM::1
</serviceType>
<serviceId>urn:upnp-org:serviceId:ActuatorSTIM.0001</serviceId>
<SCPDURL>ActuatorSTIM/scpd.xml</SCPDURL>
<controlURL>ActuatorSTIM/control</controlURL>
<eventSubURL>ActuatorSTIM/event</eventSubURL></service><service>
<serviceType>urn:schemas-upnp-org:ControlStatusSTIM::1</serviceType>
<serviceId>urn:upnp-org:serviceId:ControlStatusSTIM.0001</serviceId>
```

<SCPDURL>ControlStatusSTIM/scpd.xml</SCPDURL>
<controlURL>ControlStatusSTIM/control</controlURL>
<eventSubURL>ControlStatusSTIM/event</eventSubURL>
</service>
<service><serviceType>urn:schemas-upnp-org:SensorSTIM::1</serviceType>
<serviceId>urn:upnp-org:serviceId:SensorSTIM.0001</serviceId>
<SCPDURL>SensorSTIM/scpd.xml</SCPDURL>
<controlURL>SensorSTIM/control</controlURL>
<eventSubURL>SensorSTIM/event</eventSubURL>
</service></serviceList></device></root>

The NCAP Device Description includes general information about the Device and the list supported services.

The HTTP message that sends the SensorSTIM services Descriptions has the following look:

HTTP/1.0 200 OK
CONTENT-TYPE: text/xml
Server: WINDOWS, UPnP/1.0
Content-Length: 1882
<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
<specVersion><major>1</major><minor>0</minor></specVersion>
<actionList>
<action><name>CalibrateChannel</name></action>
<action><name>DisableChannel</name></action>
<action><name>EnableChannel</name></action>
<action><name>ReadParameter</name>
<argumentList><argument><name>ReadParameterValue</name>
<direction>in</direction>
<relatedStateVariable>ParameterValue</relatedStateVariable></argument>

</argumentList></action>
<action><name>ResetChannel</name></action>
<action><name>SensorEvent</name>
<argumentList><argument><name>SensorReadEventBit</name>
<direction>in</direction>
<relatedStateVariable>SensorReadEvent</relatedStateVariable>
</argument></argumentList></action>
<action><name>SensorHardwareError</name><argumentList>
<argument><name>HardwareErrorBit</name><direction>in</direction>
<relatedStateVariable>HardwareError</relatedStateVariable></argument>
</argumentList></action><action>
<name>SensorMissedEvent</name>
<argumentList><argument><name>SensorReadMissedEventBit</name>
<direction>in</direction>
<relatedStateVariable>SensorReadMissiedEvent</relatedStateVariable></argument>
</argumentList></action>
<action><name>UpdateAndReadParameter</name>
<argumentList><argument><name>UpdateAndReadParameterValue</name>
<direction>in</direction>
<relatedStateVariable>ParameterValue</relatedStateVariable></argument>
</argumentList></action></actionList>
<serviceStateTable><stateVariable sendEvents="yes">
<name>SensorReadMissiedEvent</name><dataType>boolean</dataType>
</stateVariable>
<stateVariable sendEvents="yes"><name>HardwareError</name>
<dataType>boolean</dataType></stateVariable>
<stateVariable sendEvents="yes"><name>SensorReadEvent</name>
<dataType>boolean</dataType></stateVariable>
<stateVariable sendEvents="no"><name>ParameterValue</name>
<dataType>i4</dataType></stateVariable></serviceStateTable></scpd>

The HTTP message that sends the ControlStatusSTIM service description has a following look:

HTTP/1.0 200 OK

CONTENT-TYPE: text/xml

Server: WINDOWS, UPnP/1.0

Content-Length: 1310

<?xml version="1.0" encoding="utf-8"?>

<scpd xmlns="urn:schemas-upnp-org:service-1-0"><specVersion><major>1</major>

<minor>0</minor></specVersion>

<actionList><action><name>CallibrateAllChannels</name></action>

<action><name>DisableAllSensors</name></action>

<action><name>EnableAllSensors</name></action>

<action><name>ResetSTIM</name></action>

<action><name>STIMEvent</name>

<argumentList><argument><name>EventBit</name>

<direction>in</direction>

<relatedStateVariable>STIMEvent</relatedStateVariable></argument></argumentList>

</action>

<action><name>STIMEventMissed</name>

<argumentList><argument><name>EventMissedBit</name><direction>in</direction>

<relatedStateVariable>STIMEventMissed</relatedStateVariable></argument>

</argumentList></action>

<action><name>STIMHardwareError</name>

<argumentList><argument><name>HardwareError</name>

<direction>in</direction><relatedStateVariable>HardwareErrorBit

</relatedStateVariable></argument></argumentList></action></actionList>

<serviceStateTable><stateVariablesendEvents="yes">

<name>STIMEventMissed</name>

<dataType>boolean</dataType></stateVariable>

<stateVariable sendEvents="yes"><name>STIMEvent</name>

<dataType>boolean</dataType></stateVariable>
<stateVariable sendEvents="yes"><name>HardwareErrorBit</name>
<dataType>boolean</dataType></stateVariable></serviceStateTable></scpd>

The HTTP message that sends the ActuatorSTIM service Description is presented in the following:

HTTP/1.0 200 OK
CONTENT-TYPE: text/xml
Server: WINDOWS, UPnP/1.0
Content-Length: 971

<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0"><specVersion><major>1</major>
<minor>0</minor></specVersion><actionList><action><name>
CalibrateChannel </name></action>
<action><name>DisableChannel</name></action><action><name>EnableChannel
</name></action>
<action><name>SensorHardwareError</name><argumentList><argument>
<name>HardwareErrorBit</name><direction>in</direction><relatedStateVariable>
HardwareError</relatedStateVariable></argument></argumentList></action>
<action><name>WriteParameter</name>
<argumentList><argument><name>ActuatorParameterValue</name>
<direction>in</direction><relatedStateVariable>ParameterValue</relatedStateVariable>
</argument></argumentList></action></actionList>
<serviceStateTable><stateVariable sendEvents="yes"><name>HardwareError
</name><dataType>boolean</dataType></stateVariable>
<stateVariable sendEvents="no"><name>ParameterValue</name>
<dataType>i4</dataType></stateVariable></serviceStateTable></scpd>

Figure 19 depicts the algorithm of the NCAP Device's Description phase.

When the NCAP Device goes to the Description phase it waits to get a "Get Description" message. After getting the message the NCAP Device analyzes the type of the message. The message can be either a request to get the Device Description or one of the Service Descriptions. The Service Description can be Control/StatusSTIM, ActuatorSTIM or SensorSTIM. For each request the NCAP Device sends a respective Description.

Figure 20 depicts the algorithm of the NCAP Control Point Description phase. After discovery the NCAP Device The NCAP Control Point sends the requests to get Descriptions of the NCAP Device and Services. After sending the requests the NCAP Control Point waits the messages from the NCAP Device that contain the Device and Services Descriptions. The NCAP Control Point keeps and uses the Device and Services Descriptions to communicate with the NCAP Device.

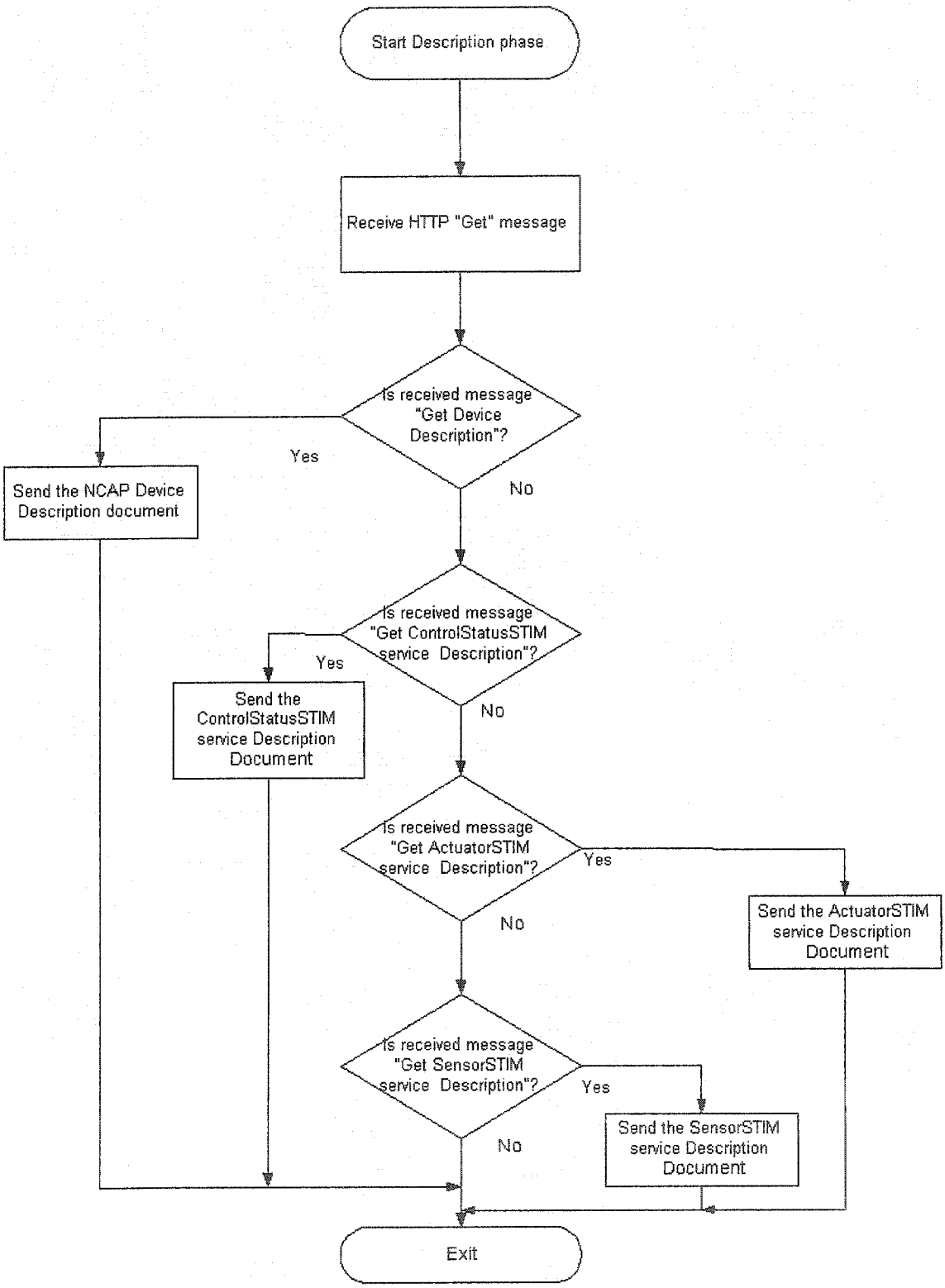


Figure 19. Algorithm of NCAP Device Discovery phase

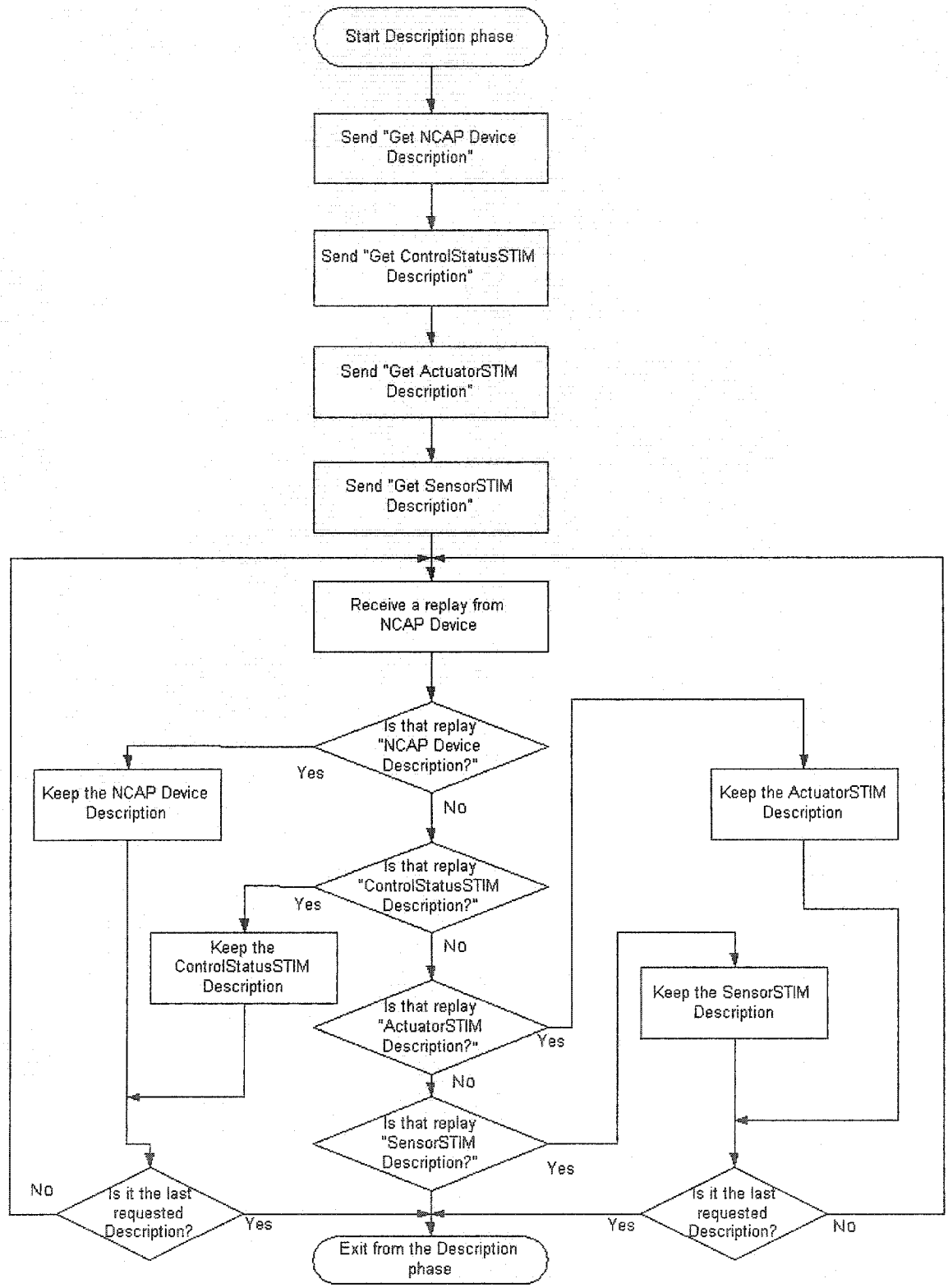


Figure 20 Algorithm of Control Point Description phase

5.1.3 Design of Publishing/Subscription and Event communication models

An NCAP Device plays the role of a Publisher and an NCAP Control Point as a Subscriber. The Control Point sends a Subscription request for each NCAP Device service. The NCAP Device accepts the request and sends an "Event message" to the Control Point when an event occurs.

The Subscription request sent by the Control Point, is similar :

Send: SUBSCRIBE /<*name of a Device service*>/event HTTP/1.0

HOST: <*IP address of the Device*>:50519

NT: upnp:event

TIMEOUT: Second-180

User-Agent: WINDOWS, UPnP/1.0

CALLBACK: <http://<*IP address of the Control Point*>:52902/<*name of a Device service*>/event>

Where <*name of a Device service*> can be either ControlStatusSTIM, ActuatorSTIM or SensorSTIM.

The NCAP Device receiving the Subscription request sends a confirmation of registration of the Subscription and all initial states of parameters that trigger events. An event message has a following look:

NOTIFY /ControlStatusSTIM/event HTTP/1.1

HOST: <*IP address of the Control Point*>:61125

Content-Type: text/xml

NT: upnp:event

NTS: upnp:propchange

SID: uuid:2

SEQ: 3

Content-Length: 162

<?xml version="1.0" encoding="utf-8"?>

<e:propertyset xmlns:e="urn:schema-upnp-org:event-1-0"><e:property>

<*name of parameter*>*value of parameter*</ *name of parameter* ></e:property>

</e:propertyset>

Figure 21 depicts the algorithm of the NCAP Control Point Subscription phase.

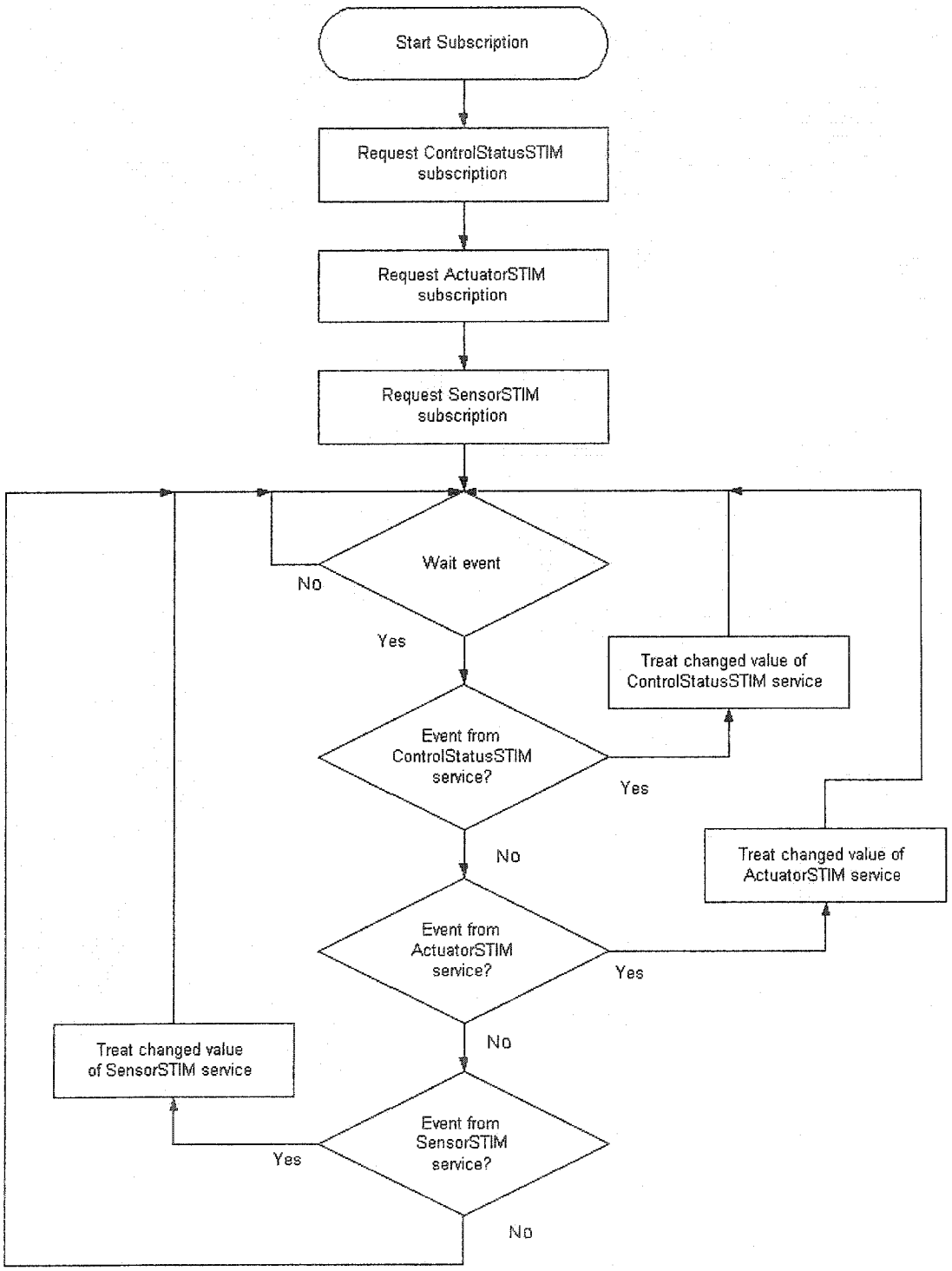


Figure 21 Algorithm of NCAP Control Point Subscription

Figure 22 depicts the algorithm of the NCAP Device Publisher phase.

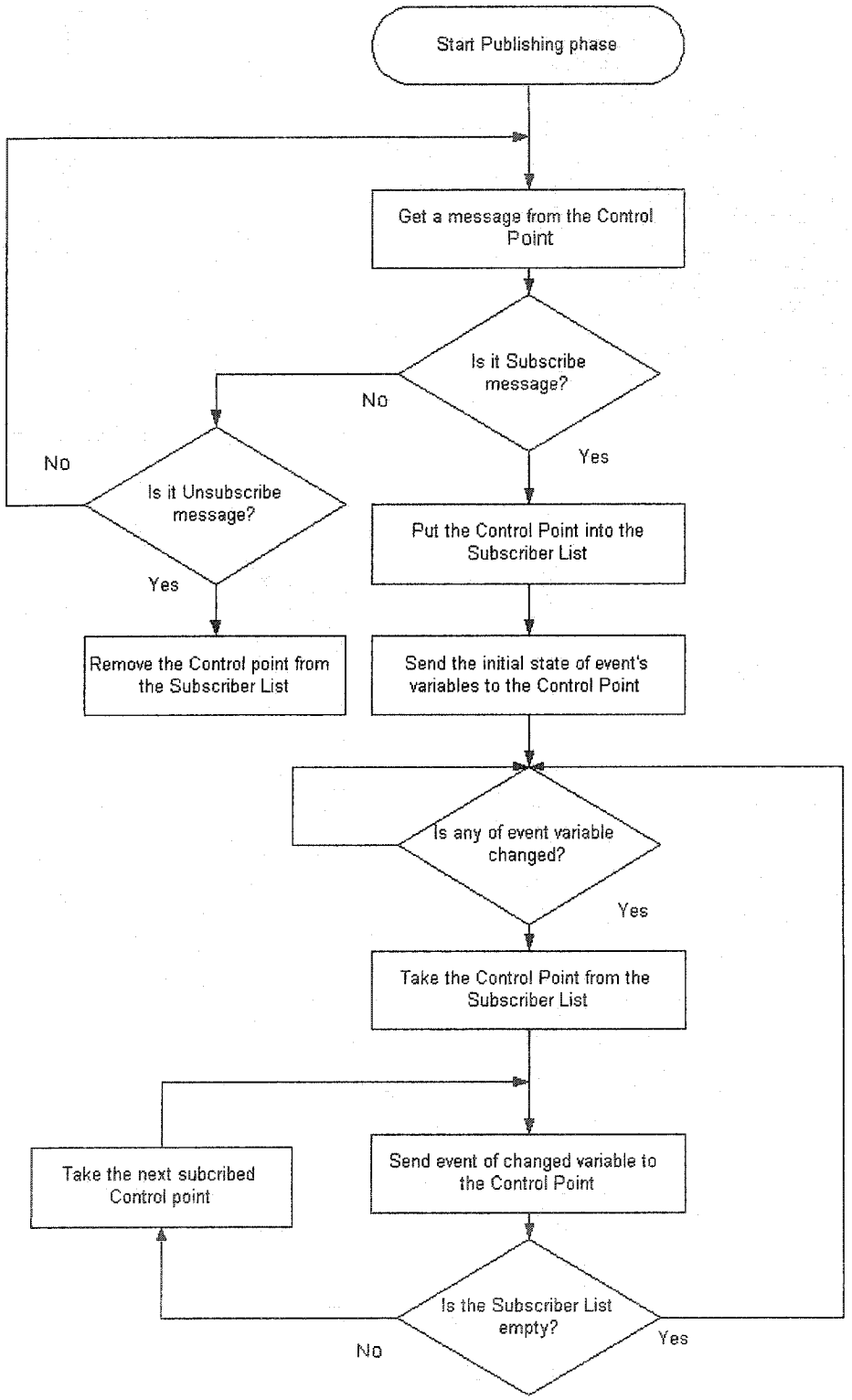


Figure 22. Algorithm of NCAP Device Publishing phase

To unsubscribe from the reception of events, the Control Point sends an HTTP message of following format:

UNSUBSCRIBE /<*name of service*>/event HTTP/1.0
HOST: <*IP address of the Control Point*>:63128
User-Agent: WINDOWS, UPnP/1.0
SID: uuid:5

The NCAP Device receiving the Unsubscribe message then removes the NCAP Control Point from the Subscriber List and sends an HTTP OK message.

5.2 Implementation of NCAP Device

The NCAP Device is developed as a Dialog Windows Application and emulates an NACP UPnP Device. The application has three services – ControlStatusSTIM, ActuatorSTIM and SensorSTIM.

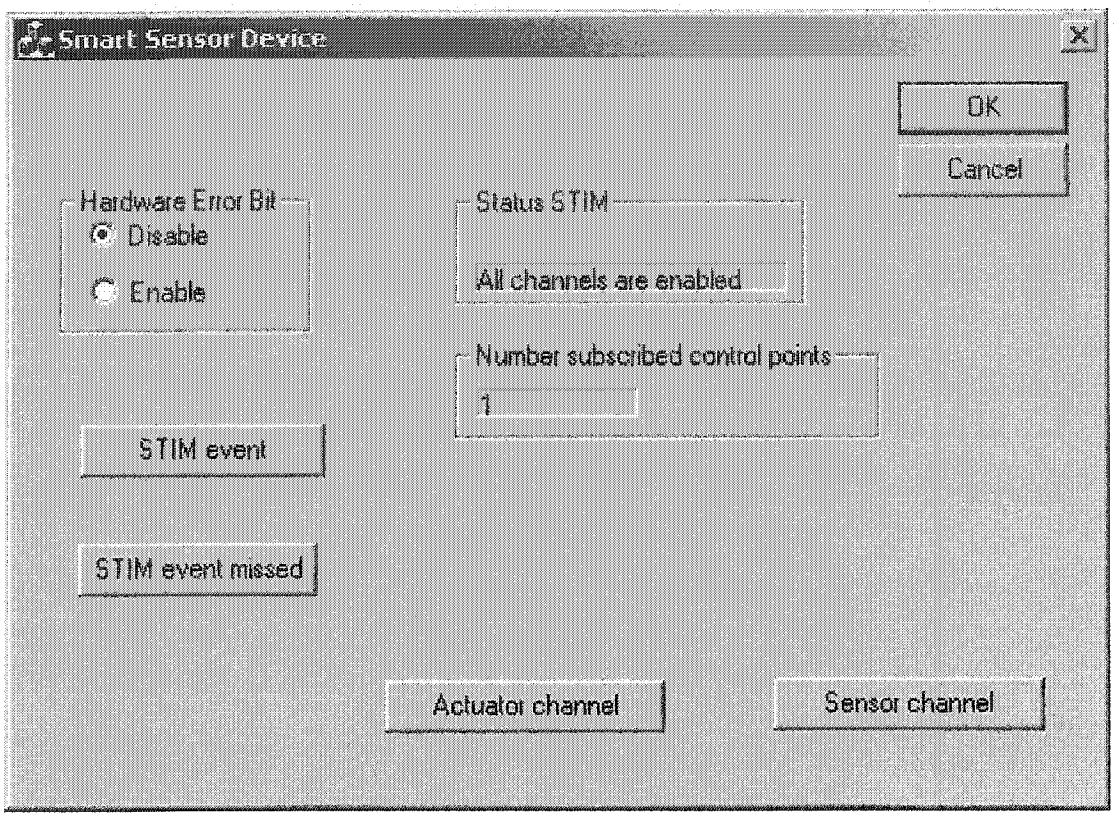
The ControlStatusSTIM contains signals belonging to Meta – TEDS: HardwareErrorBit, STIMEvent, STIMMissedEvent and StatusSTIM. The HardwareErrorBit represents an event signal that indicates the state of all hardware in the system. If signal is high it indicates that some part of hardware is not functioning well.

STIMEvent is an event signal generated when an event occurs in any of the system channels. STIMMissedEvent is an event signal, generated when an event occurs in any of the channels, but the system fails to generate a STIMEvent.

StatusSTIM is an input signal that changes the status of STIM according to a received variable of the status from an NCAP Control Point. StatusSTIM manages four defined states of STIM: EnableAllChannel, DisableAllChannel, CalibrateAllChannel and RestSTIM. Figure 23 shows the main dialog of the application containing a ControlStatusSTIM service.

The HardwareErrorBit is presented by a GroupBox that contains two Radio Buttons - Enable HardwareErrorBit and Disable HardwareErrorBit. A user can change the status of the Hardware Error Bit by clicking on Enable or Disable Radio Buttons.
The STIMEvent and STIMEventMissed signals are presented by Buttons. A user clicking on one of the Buttons can generate an event to connect to the NCAP Control Point. The Dialog contains the StatusSTIM window showing the current status of the Device. Any NCAP Control Point that is connected to the NCAP Device can change the status.

In addition to the described signal the Dialog has a window that shows the number of connected NCAP Control Points and two Buttons that call the ActuatorSTIM service and SensorSTIM service Dialogs windows.



The Figure 23 Dialog Window of ControlStatusSTIM

The ActuatorsSTIM dialog emulates an Actuator channel and contains HardawareErrorBit, ActuatorState and ValueToWrite signals. The HardawareErrorBit is an event signal that indicates the state of channel hardware. The ActuatorState is an input variable and presents the current state of the Actuator channel. It can have following values: Enable Channel, Disable Channel and Calibrate Channel. The ValueToWrite is an input signal and presents a value to write into an actuator. The value is sent by an NCAP Control Point. Figure 24 shows the ActuatorSTIM dialog.

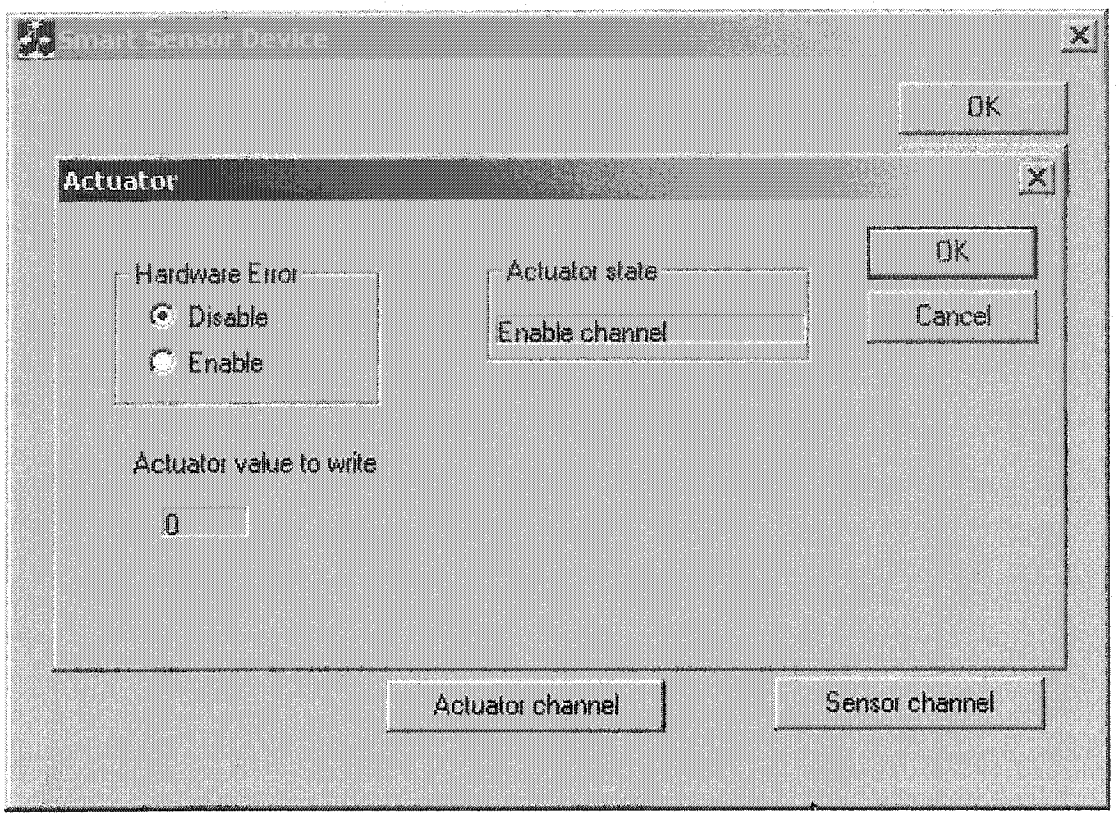


Figure 24 ActuatorSTIM dialog

The SensorSTIM dialog emulates a Sensor channel. It contains HardwareErrorBit, SensorReadEvent, SensorMissedEvent, SensorState, UpdateAndRead and SensorParameter signals.

The HardwareErrorBit is an event signal indicating the state of the sensor hardware.

The SensorReadEvent is an output event signal, which is sent to an NCAP Control Point when the sensor finishes its read cycle. The SensorMissedEvent is sentwhen the NCAP Device fails to send SensorReadEvent. . The SensorState is an input variable and presents the current state of the Sensor channel. It can have following values: Enable Channel, Disable Channel and Calibrate Channel.

The UpdateAndRead is an input signal that tells the Device to force the sensor to read its parameter and sends new values to the subscribed NCAP Control Points.

The SensorParameter emulates the sensor hardware and is presented as a Slide Control in the SensorSTIM dialog. The position of the Slide Control defines a sensor value. Figure 25 shows the SensorSTIM Dialog:

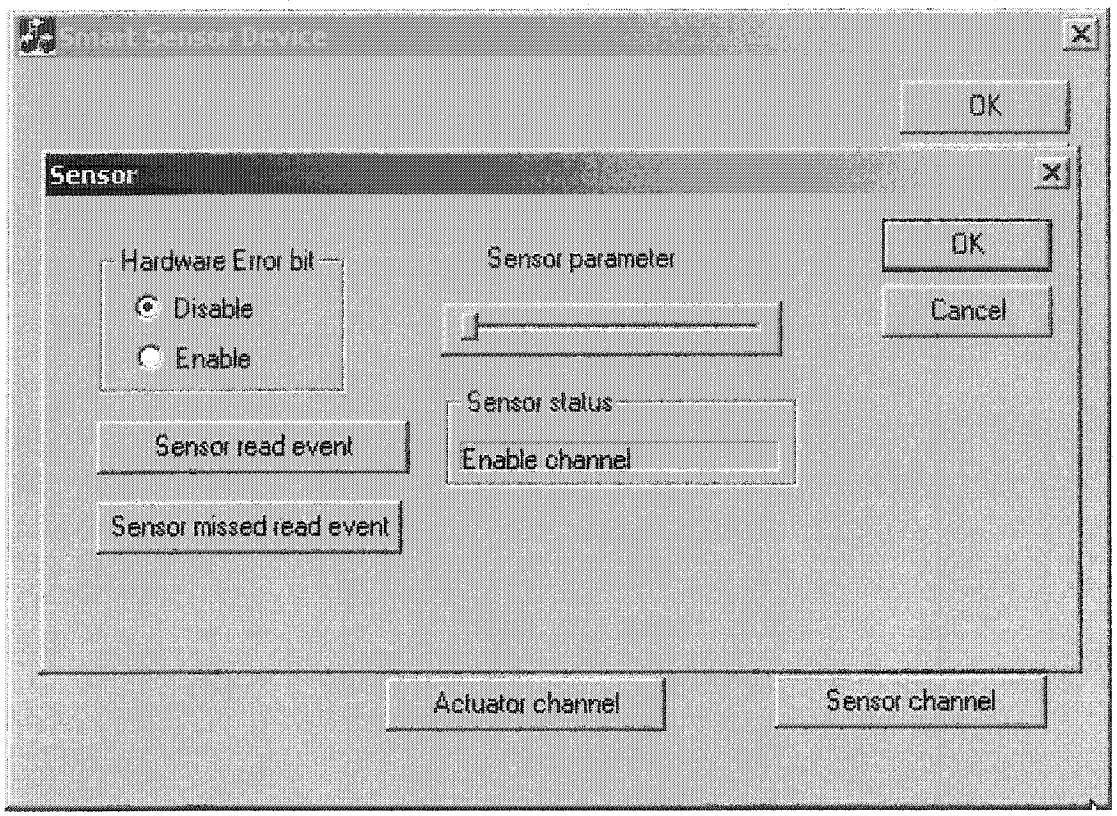


Figure 25 SensorSTIM dialog

5.2.1 Program design of NCAP Device

The NCAP UPnP Device is developed as a Windows OS application. The Application uses the Windows TCP/IP stack to provide Internet communication and its own UPnP library to make UPnP communication. During the design of the Device we do not focus on creating multitasking applications, and consider doing it in the next version of the Device. Thus the Device application uses the blocking functions of the TCP/IP stack.

To simplify the design we do not use native classes of IEEE 1451.1 standard as parents classes. Instead, we declare new inherited classes of the UPnP Smart Sensor (the section 4.2) as base classes. The implementation of emulated UPnP NCAP Devices consists of following classes:

- UPnPNCAP class. The class presents a functionality of UPnP NCAP (see the section 4.3.1).
- UpnPWebServer class. The class presents a functionality of Web Service class (see the section 4.3.5).
- UpnPPublisher class. The class presents a functionality of UPnP Publisher Port class (see the section 4.3.3).
- UPnPXMLSupport class. The class presents a functionality of XML File class (see the section 4.3.6).
- ActuatorSTIM class. The class presents a functionality of Parameter with Update Class (see the section 3.1.4.6.3) and also plays a role of an applied class of the ActuatorSTIM dialog.
- SensorSTIM class. The class presents a functionality of Parameter with Update Class (see the section 3.1.4.6.3) and also plays a role of an applied class of the SensorSTIM dialog.
- CGuiDialogNCAPDeviceDlg class. The class is an applied class of ControlSTIM dialog.
- CGuiDialogNCAPDeviceApp class. The class is the class that starts and runs UPnP NCAP Device application.

The application also includes the list of supported functions that are not parts of designed classes for a efficiency reasons. Some of these functions are CommunicationTread and callback functions. The CommunicationTread function is a function that runs UPnP communication in a separate thread. Callback functions are used as a convenient and efficient method of treating UPnP messages. Because of the blocking nature of used TCP/IP functions, all UPnP communications are organized in a CommunicationTread function that runs as a Windows thread and has an infinite loop to manage TCP/IP connections. The infinite loop sends HTTP send messages and gets HTTP received messages. For each message received or those intended to be sent, a

UPnP message CommunicationThread calls a call back function. The Figure 26 depicts the algorithm of the CommunicationThread function.

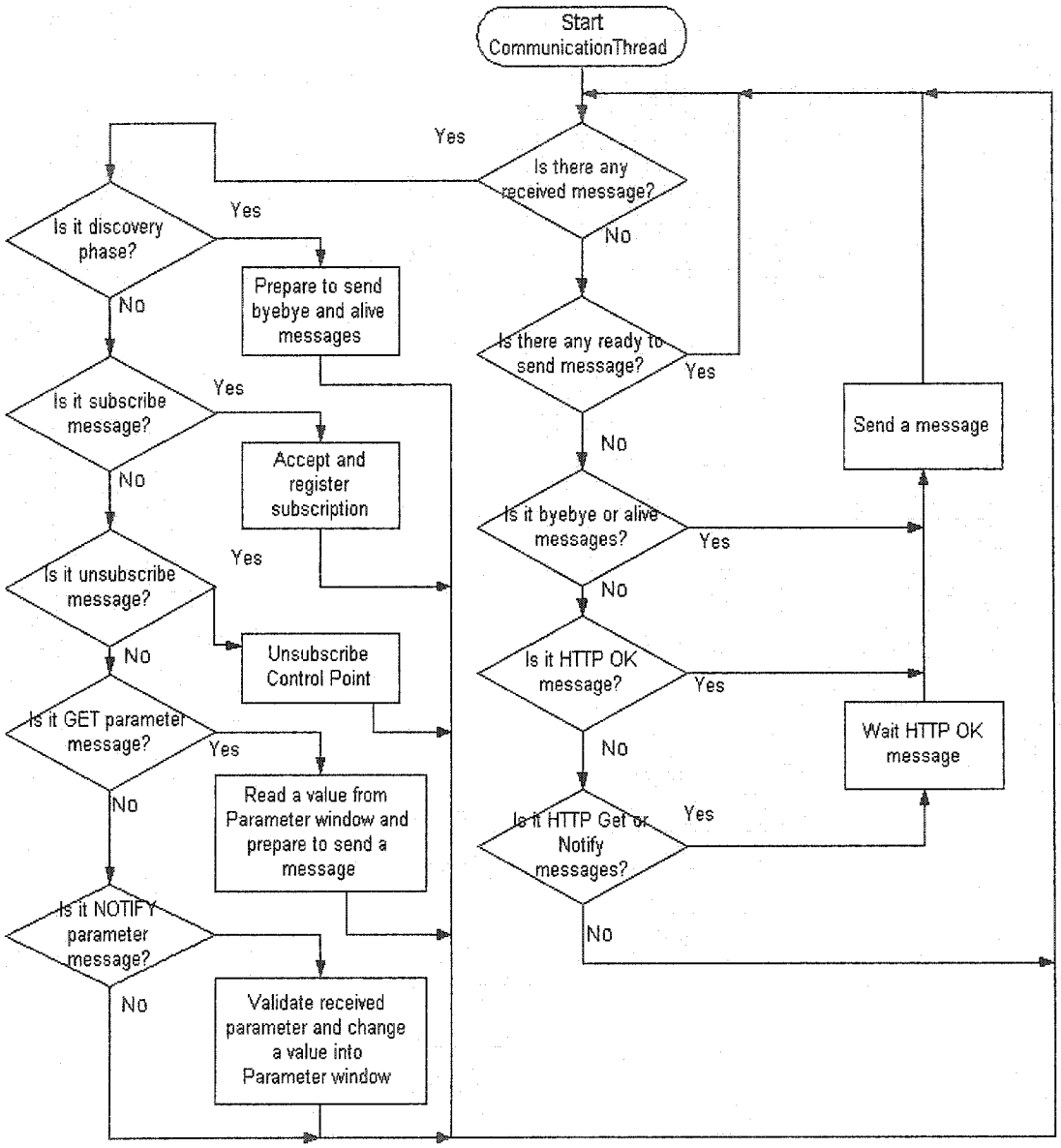


Figure 26. Algorithm of CommunicationThread function of NCAP Device

5.2.1.1 ControlStatusSTIM dialog window

The ControlStuatusSTIM window is a GUI interface of the ControlStatus service. The dialog contains graphical controls that present all parameters of the service. They are Hardware Error Bit, Status STIM, STIM event and STIM missed event. In addition to the

service parameters controls, the dialog contains a Static control to show the number of connected NCAP Control Points and two Buttons controls to run Actuator and Sensor service dialogs. Figure 27 presents a working algorithm of the ControlStatusSTIM dialog.

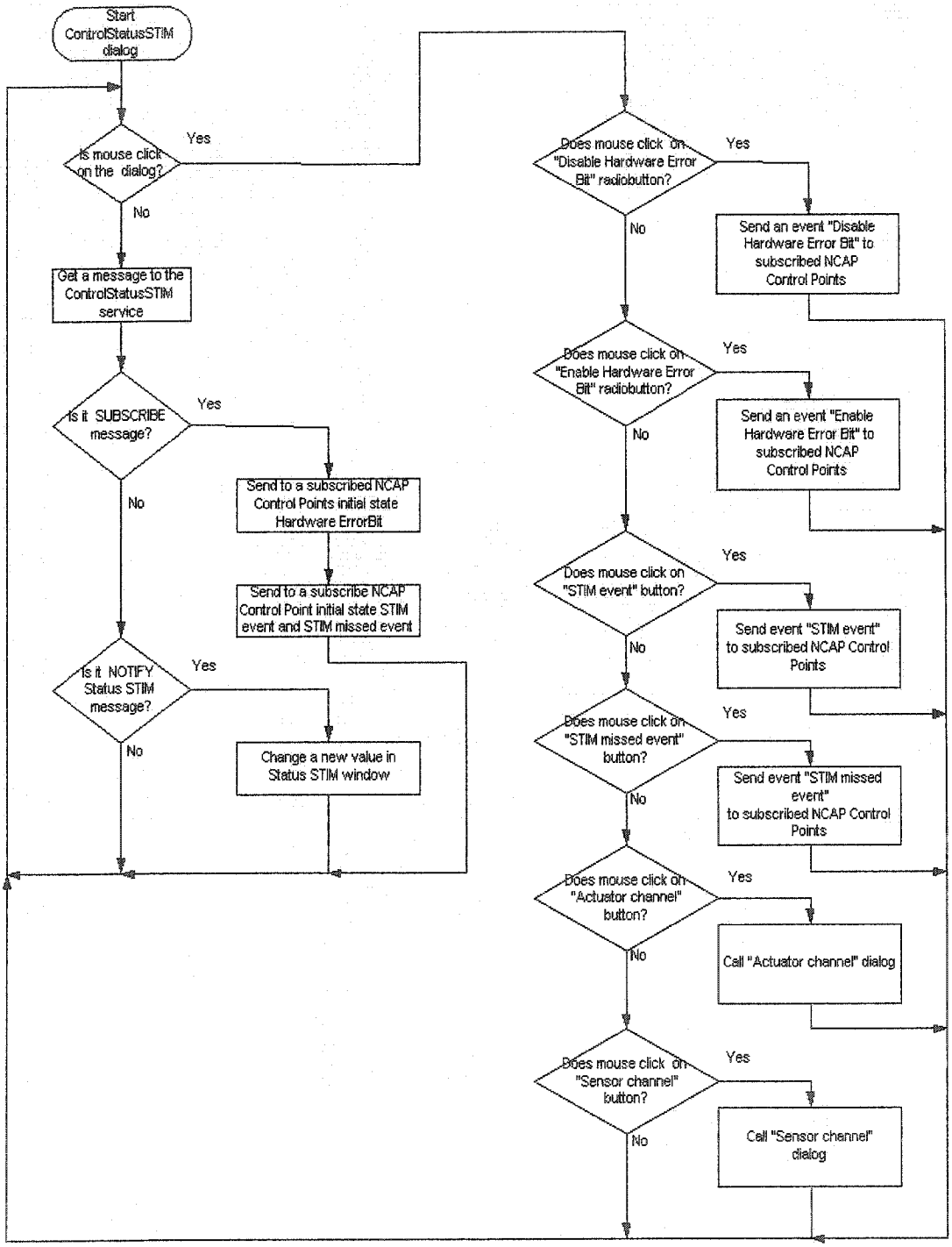


Figure 27. Algorithm of ControlStatusSTIM dialog window

5.2.1.2 Actuator channel window

The Actuator channel dialog provides a GUI interface of the ActuatorSTIM service. The dialog contains graphical controls that represent ActuatorSTIM parameters and provides the opportunity to a user to control these parameters. Figure 28 depicts an algorithm of the Actuator channel dialog.

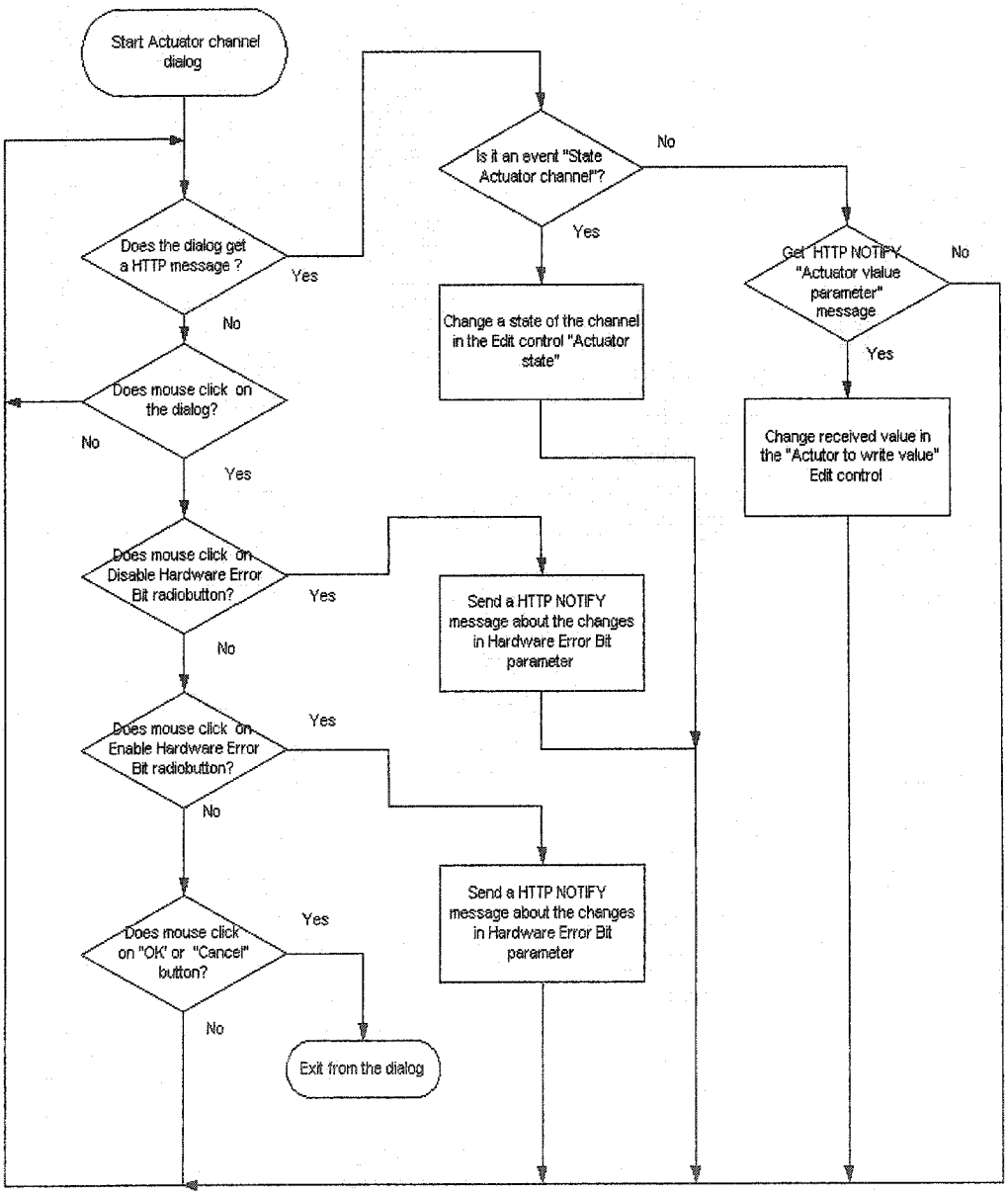


Figure 28. Algorithm of Actuator channel dialog.

The dialog is called from the ControlStatusSTIM dialog window and returned to its parent window when the work finishes.

5.2.1.3 Sensor channel dialog window

The Sensor channel dialog window presents the GUI of the SensorSTIM service. The GUI contains Windows controls that allow control the parameters of the service. The dialog includes the following controls:

-Hardware Error Bit. This parameter is presented by two Radio Buttons – Disable and Enable. Each Radio Button generates an event to the subscribed NCAP Control Points.

- Sensor Status. This is an input parameter and is presented by an Edit control. A subscribed NCAP Control Point could change a value of the control.

- Sensor parameter. This is Slide control. The current position of the control is defined by the sensor parameter.

- Sensor read event. This event is represented by a Button control. Clicking on the button generates an event. The event reads the Sensor Slide control and sends the sensor value to the subscribed Control Point.

-Sensor missed event. Like the Sensor read event, this is presented by a Button Control. This event is sent when the NCAP Device fails to send the Sensor read event.

The following page shows the algorithm of Sensor channel dialog.

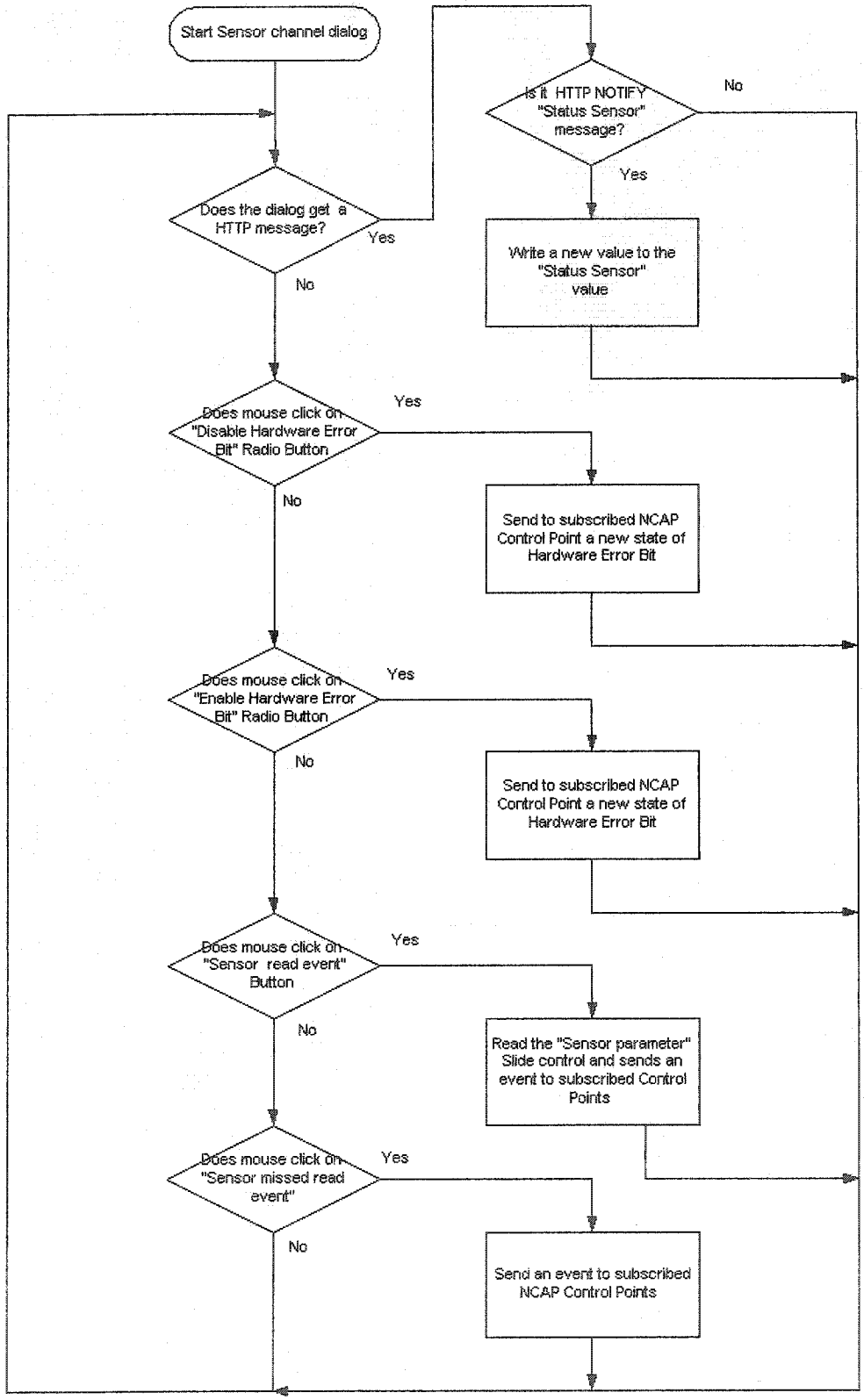


Figure 29. Algorithm of Sensor channel dialog window

5.3 The implementation of NCAP Control Point

The NCAP Control Point is designed and developed as a Windows dialog application. Unlike the NCAP Device, it has only one window that is split into three parts: Control/Status STIM panel, Actuator channel panel and Sensor channel panel. Figure 30 shows how the NCAP Control Point dialog looks.

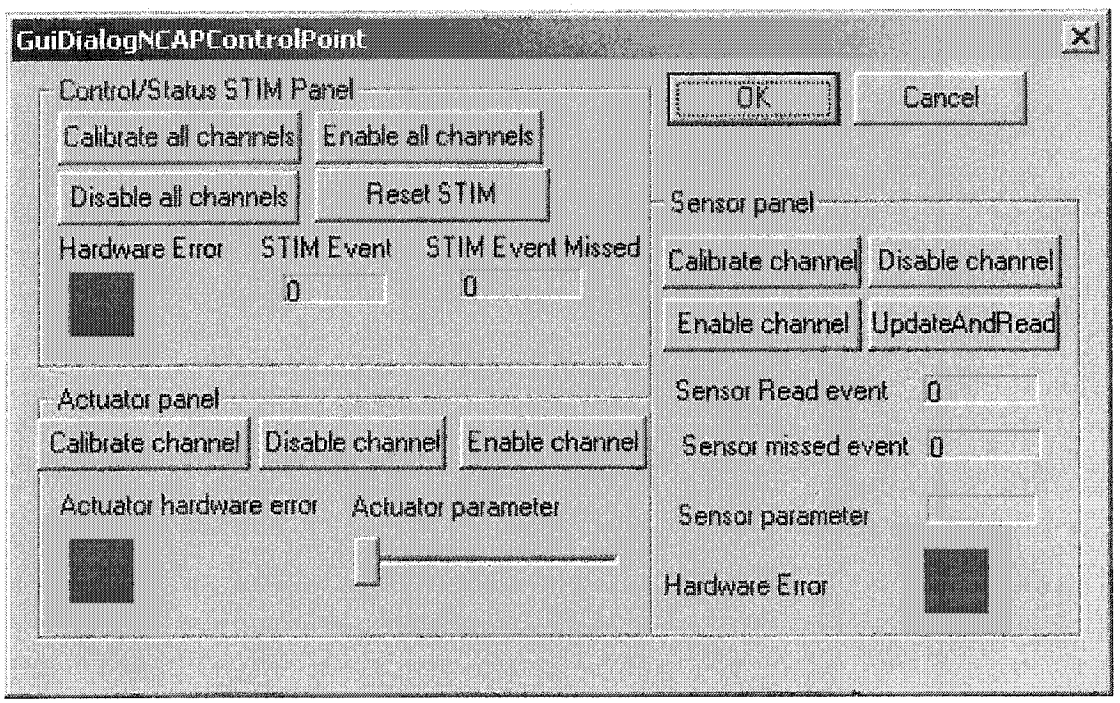


Figure 30. NCAP Control Point dialog

The Control/Status STIM panel allows a user to work with the ControlStatusSTIM service of NCAP Device. It contains four control Button controls, one graphical image window and two Edit windows. The Button controls are used to manage the status of STIM. One control is useed for each state of the status. There are four Buttons: Enable all channels, Disable all channel, Calibrate all channel and Reset STIM.

The graphical image window shows a state of the Hardware Error Bit of the ControlStatusSTIM service. The green color represents the disable state of the Hardware Error Bit and the red color represents the enable state of the parameter.

The Edit windows shows the number of received STIM and STIM missed events. When the NCAP Control Point receives an event it increases the number in a window.

The Actuator panel is presented a GUI to work with the ActuatorSTIM service of the NCAP Device. The panel has three Buttons, one graphical image window and a Slide controls. The Button controls are used to manage the status of Actuator channel. There are three Buttons: Calibrate channel, Disable channel and Enable channel.

The graphical image window shows a state of the Hardware Error Bit of the ActuatorSTIM service. The green color presents the disable state of the Hardware Error Bit and red color represents the enable state of the parameter.

The Slide control is used to set up an Actuator parameter. When a user moves the Slide, the NCAP Control Point sends a notification to the NCAP Device about the change of the Actuator parameter.

The Sensor panel is a GUI that is used to control a Sensor channel. It has four Button controls, one graphical image window and three Edit controls.

The Button controls are used to manage the status of Sensor channel. There are four Buttons: Calibrate channel, Disable channel, Enable channel and UpdateAndRead. Clicking on the UpdateAndRead Button, a user sends a notification to the NCAP Device. The notification tells the Device to read the sensor parameter and to send a read parameter to the NCAP Control Point.

The graphical image window shows a state of the Hardware Error Bit of the SensorSTIM service. The green color represents the disable state of the Hardware Error Bit and red color presents the enable state of the parameter.

Two Edit windows show the numbers of received Sensor read and missed Sensor read events. The third Edit window shows a value of the Sensor parameter. The value is received from the Sensor read event message.

5.3.1 Program design of NCAP Control Point

NCAP Control Point is designed according to section 4 "Modification to the IEEE 1451.1 Standard". To simplify the design for current implementation of the NCAP Control Point we do not provide the whole hierarchy of the standard classes. Instead we design the modified UPnP classes from section 4 as base classes. The implementation of NCAP Control Point includes following classes:

- UPnPNCAPClient. This class represents a UPnP NCAP class
- UpnPClientPort. This class represents a UPnP Client Port (see the section 4).
- UpnPSubscriber. This class represents a UPnP Subscriber Port(see the section 4).
- UPnPClientXML. This represents an XML class (see the section 4).

In addition to the base UPnP NCAP classes, the implementation contains classes and functions that provide an interface to the Windows OS. They are:

- CGuiDialogNCAPControlPointApp class. The class provides the starting functions of the application.
- CGuiDialogNCAPControlPointDlg. This is a dialog interface class that provides program communication with the controls of the dialog.
- CommunicationThread function. This is a thread function that runs all UPnP communication of the NCAP Control Point. To simplify our design, we use blocking version of the TCP/IP socket. Figure 31depicts the algorithm of the CommunucationThread function.
- Call back functions that are used in UPnP communication but are not parts of NCAP Control Point functions. These functions are used as an efficient method of treatment of UPnP messages.

As we mentioned earlier the NCAP Control Point is designed as a Window application containing one dialog. The dialog is a GUI and it is split into three panels. Each panel presents one of the NCAP Device services: ControlStatusSTIM, ActuatorSTIM and SensorSTIM.

Figure 31depicts the algorithm of the CommunucationThread function. The function starts during the initialization of the NCAP Control Point. After starting the function sends the "M-SEARCH" message and goes to the infinite loop. The loop contains two receiving and sending messages branches. If there is any message in the TCP/IP socket the function goes to the receive messages branch. If there is any ready to send message the function goes to the sending messages branch.

The receive message branch analyzes the type of the received message. If the message belongs to the group of "HTTP alive message" the CommunicationThread function prepares the "Get Description" message. If the received message is the "HTTP NOTIFY Description" message the function prepares the "HTTP Subscribe" message. If the received message is the "HTTP GET" message the function reads a requested Parameter and prepares to send the "HTTP NOTIFY" message that contains the value of the Parameter. If the received message is the "HTTP NOTIFY" message the function validates the received Parameter, changes the value into the Parameters window and prepared to sends the "HTTP OK" message. All prepared messages the function puts into the ready to send message queue.

The sending message branch sends all available messages from the ready to send message queue.

Figure 32 depicts an algorithm of the NCAP Control Point Dialog. The dialog starts during the initialization of the NCAP Control Point application and goes to the infinite loop. The dialog consists of three panels,one panel for each service. In the beginning of the loop the dialog checks if the mouse is clicked. If it is, the dialog checks on which panel the mouse is clicked. If the mouse is clicked on the ControlStatusSTIM panel, the dialog goes to the function that treats the ControlStatus STIM service Parameters. If the mouse is clicked on the ActuatorSTIM panel, the dialog goes to the function that treats the ActuatorSTIM service Parameters. If the mouse is clicked on the SensorSTIM panel, the dialog goes to the function that treats the SensorSTIM service Parameters. The dialog terminates if a user clicks on either "OK" or "Cancel" button.

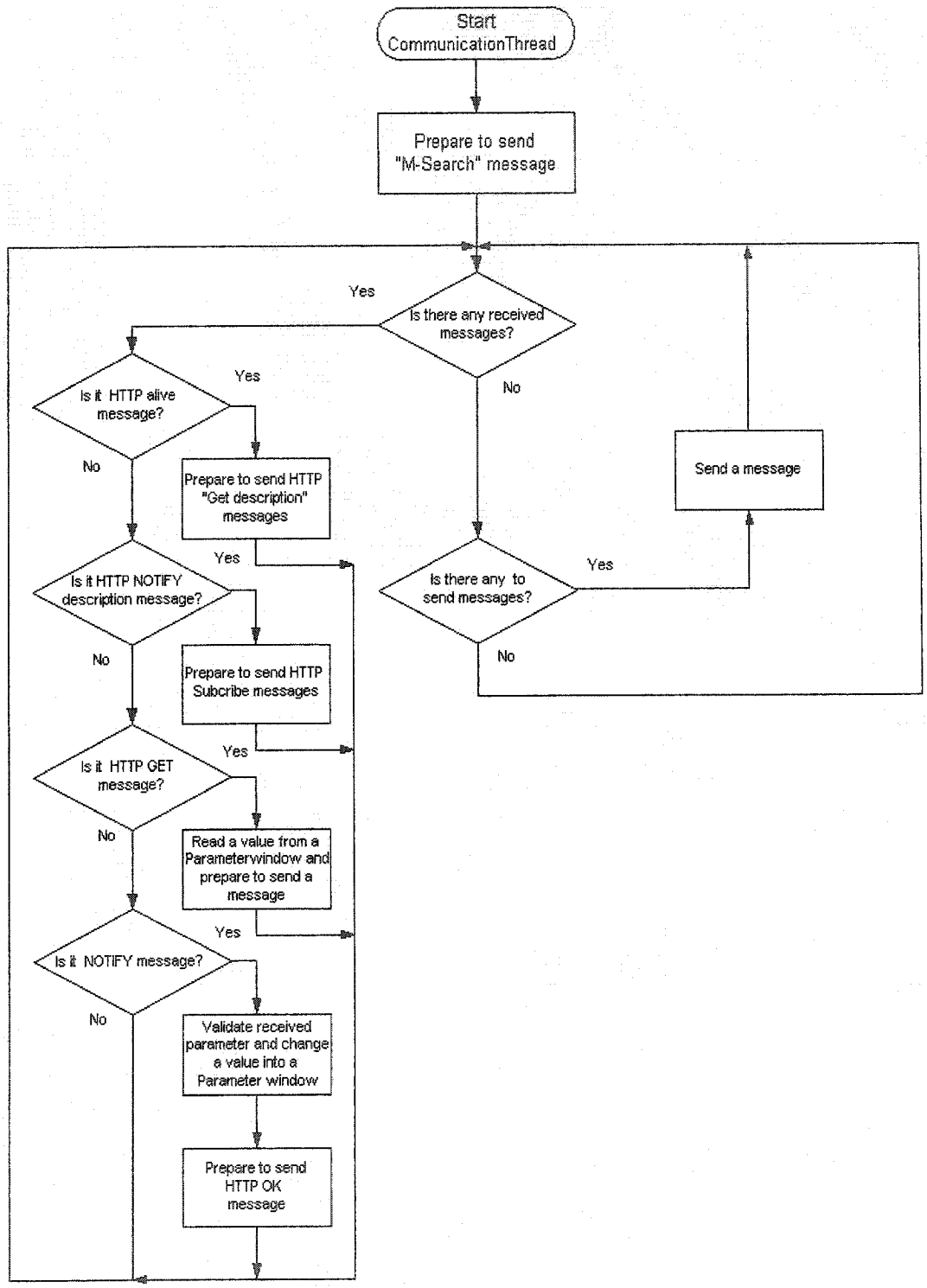


Figure 31 Algorithm of NCAP Control Point's CommunicationThread function

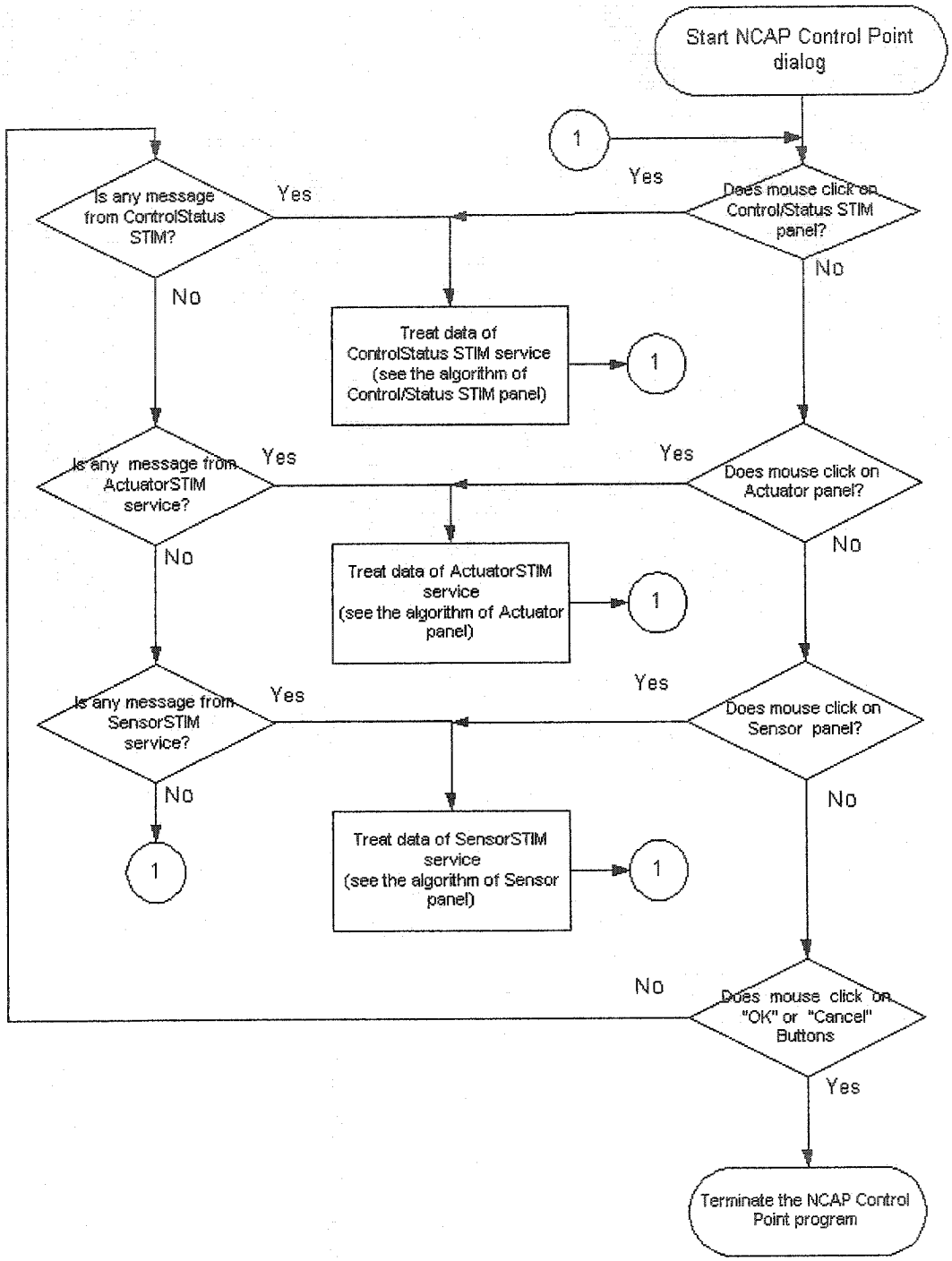
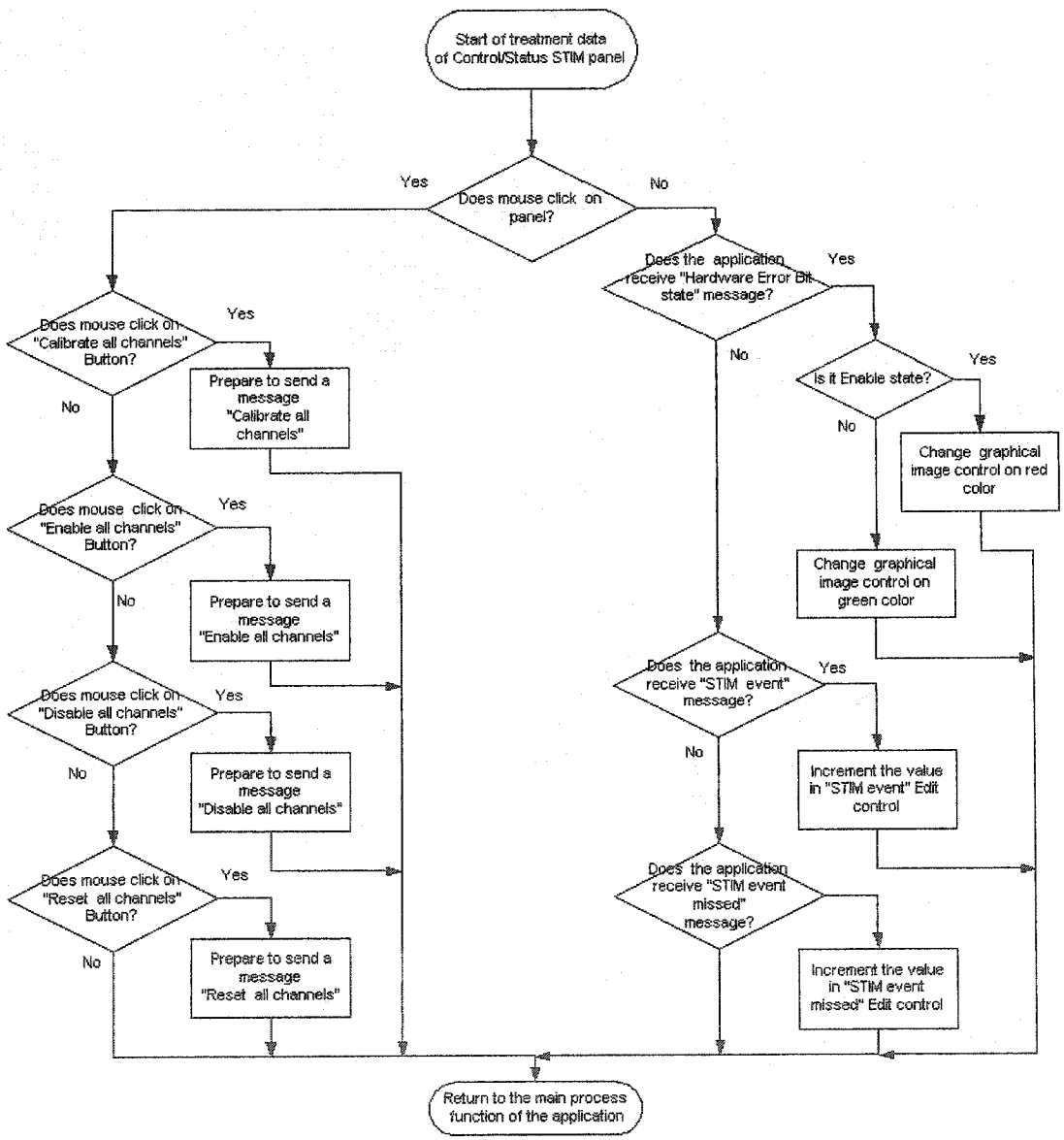


Figure 32 Algorithm of NCAP Control Point dialog

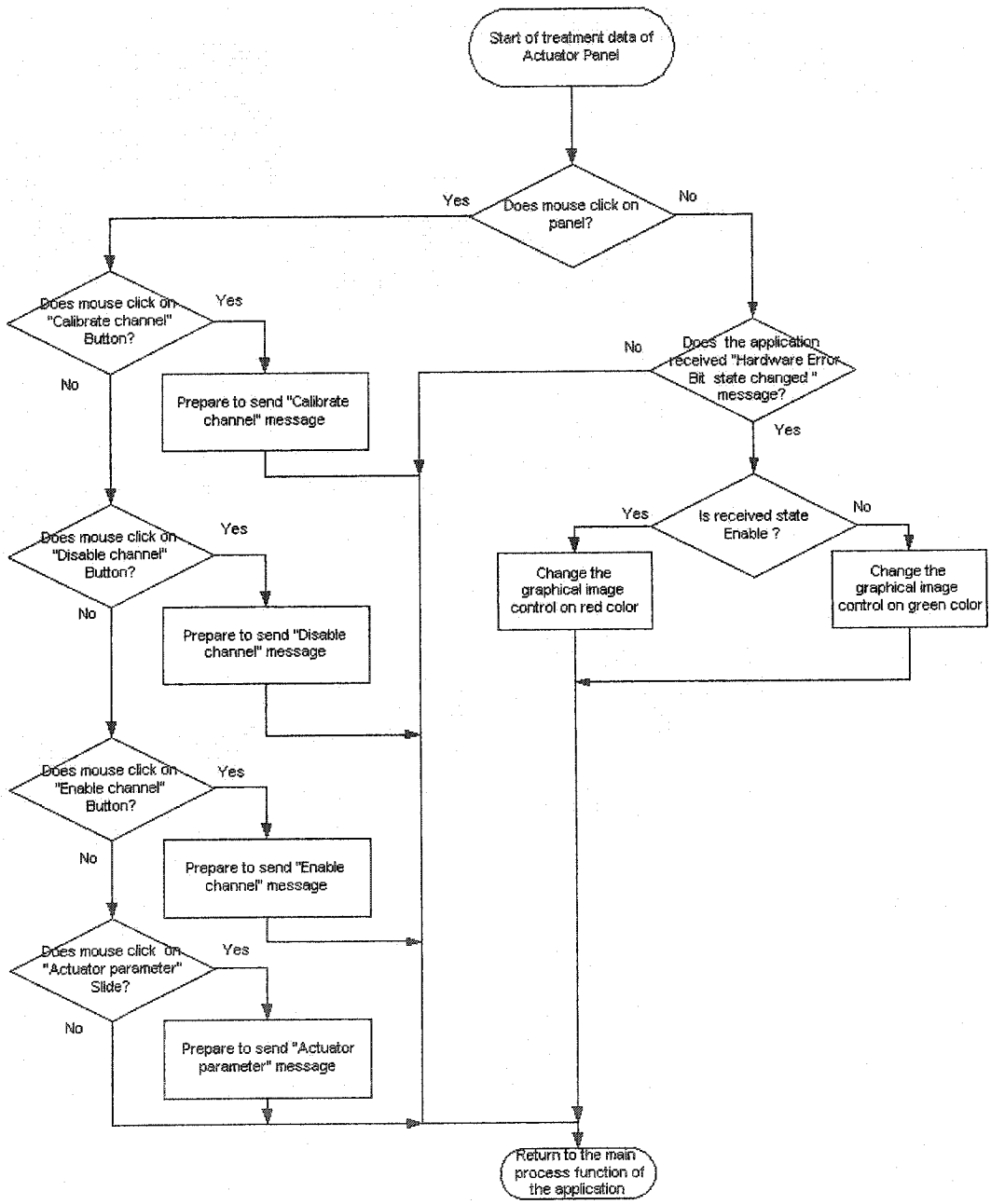
Figure 33 depicts an algorithm of Control/Status STIM panel of NCAP Control Point application. The algorithm contains two branches. The first branch checks received messages that belong to the Control/Status Service. The second branch checks mouse's clicks on the panels Windows Controls.

The panel could receive "Hardware Error Bit State", "STIM event", or "STIM missed event" messages. If the panel received the "Hardware Error Bit State" message it changes the colour of the Image Windows Control according to the received value. If the panel receives the value equal one it changes the colour to green. Otherwise it changes the colour to red. If the panel receives the "STIM event" message it increments the value in the "STIM event" Windows Control. If the panel receives the "STIM missed event" message it increments the value in the "STIM missed event" Windows Control.



The Figure 33 Algorithm of NCAP Control Pont's Control/Status STIM Panel

If a user clicks on one of panels Button the panel generates a respective "HTTP NOTIFY" message to the NCAP Device. The message could be either "Calibrate all channels", "Enable all channels", "Disable all channels" or "Reset all channels".



The Figure 34 Algorithm of NCAP Control Pont's Actuator panel

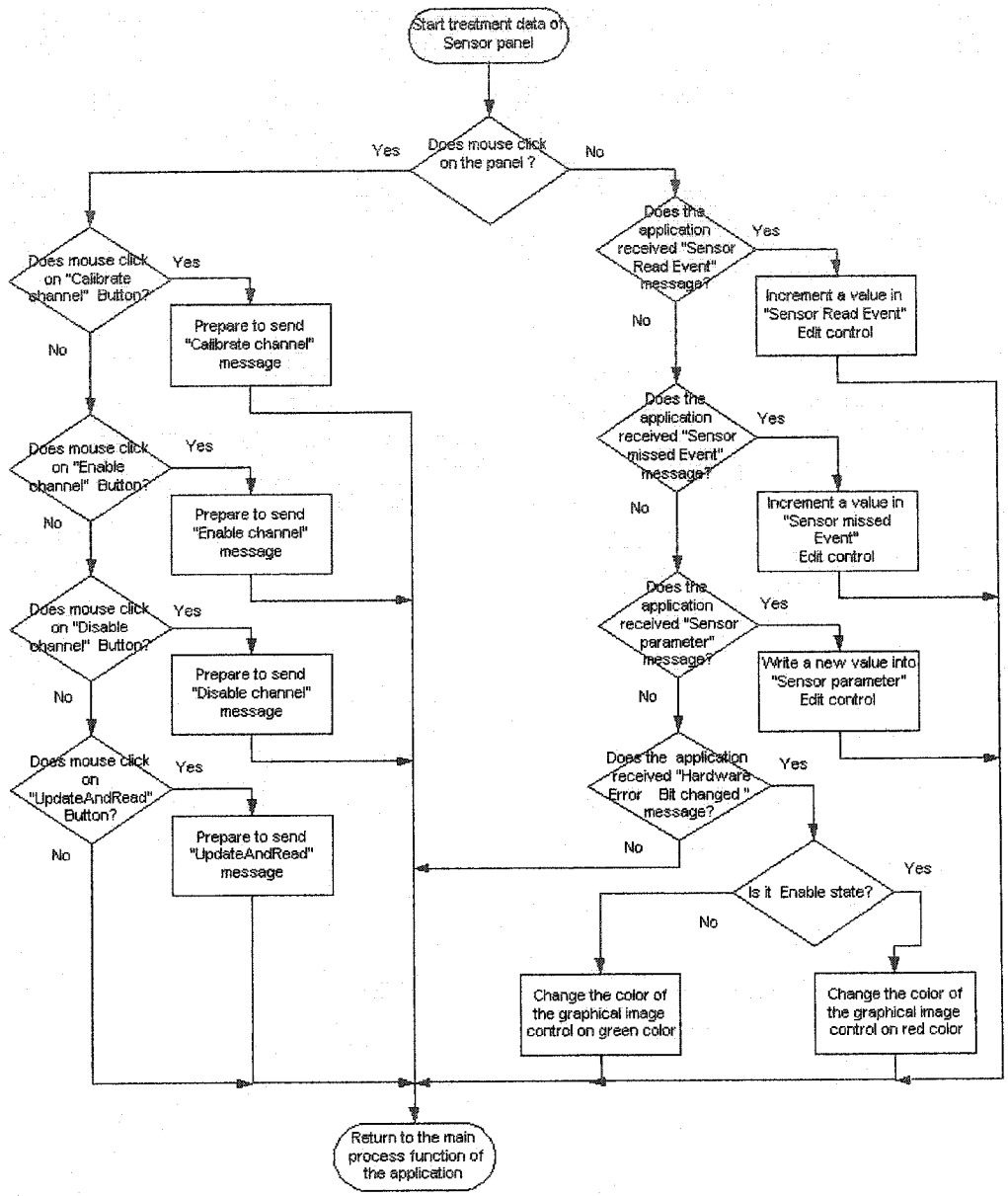
Figure 34 depicts an algorithm of Actuator panel of NCAP Control Point application. The algorithm has two branches. The first branch treats the panels received panel messages. The second branch treats users mouse's clicks. The panel can receive only one type of a "HTTP NOTIFY" message. It is "Hardware Error Bit state change" message that signals about the changing of the "Hardware Error Bit" ActuatorSTIM Parameter. The panel changes the colour of the Image Windows Control according to the received value. If the panel receives the value equal one it changes the colour to green. Otherwise it changes the colour to red.

If a user clicks on either one of panels Button or moves the "Actuator Parameter" Slide the algorithm goes to the second branch. If one of panels Button is pushed the panel generates a respective "HTTP NOTIFY" message to the NCAP Device. The message could be "Calibrate channel", "Enable channel" or "Disable channels". If the "Actuator Parameter" Slide is moved the panel generates the "HTTP NOTIFY" message that is sent to the NCAP Device. The message contains the value of the changed Parameter.

Figure 35 depicts an algorithm of Sensor panel of NCAP Control Point application.

The algorithm includes two branches. The first branch treats the received Sensor panel messages. The second branch treats users mouse's clicks. The panel could receive "Sensor event", "Sensor missed event", "Sensor parameter" or "Hardware Error Bit state change" messages. If the panel receives "Sensor parameter" message it updates the value in the "Sensor parameter" window. If the panel received "Sensor event" message it increments the value in the "Sensor event" window. If the panel receives "Hardware Error Bit state change" message it changes the colour in the Image Window Control according to the received value. If the panel receives the value equal one it changes the colour to green. Otherwise it changes the colour to red. If the panel received "Sensor missed event" message it increments the value in the "Sensor missed event" window.

If a user clicks on one of panels Button the panel generates a respective "HTTP NOTIFY" message to the NCAP Device. The message could be either "Calibrate channel", "Enable channel", "Disable channels" or "Update and read".



The Figure 35 Algorithm of NCAP Control Pont's Sensor Panel

6 Conclusions and Recommendations for Future Work

The focus of this thesis was on the enhancement of the network capabilities of Smart Sensor devices. The IEEE 1451.1 standard, which belongs to the Smart Sensor group of standards, is responsible for network communications of th Smart Sensor. It defines two communication models, the client-server and the publisher-subscriber, but it does not specify a mechanism to register and manage the devices in the network. There are several technologies that are capable of bringing these devices into a network. One such technology is the Universal Plug and Play architecture. We chose UPnP as the network mechanism used to enhance Smart Sensor devices because it is platform and programming language independent.

To provide network Plug and Play capabilities to Smart Sensor devices, we modified the Common Object Model of IEEE 1415.1, which provides the specification to design the network communications of IEEE 1451 devices. Modifications were made in this paper by creating new classes for IEEE 1451.1 that support UPnP features. We kept the existing IEEE 14151.1 client-server and publisher-subscriber communication models, but made them UPnP capable. In our proposal, the client-server model used the UPnP Control function while the publisher-subscriber model used the UPnP Event function. We also added new classes that supported the Addressing and Discovery functions that are absent in IEEE 14151.1. The Addressing function of a Smart Sensor device defines the mechanism allowing for t he reception of automatically generated IP addresses. The Discovery function provides the registration of a Smart Sensor device within the network. This function not only announces the presence of the device but also, if required, provides the device specification.

UPnP uses HTTP protocol for communication, and the data transferred is in XML format. We defined a new class for IEEE 1451.1 that treats XML documents. The class composes an XML document for sending data.. When the Smart Sensor device receives data from the network, the class retrieves the useful information from the received XML document.

Because the UPnP device works as a web server, we introduced an IEEE 1451.1 class that supports web server capabilities for Smart Sensor. To make Smart Sensor work as an UPnP device, a Smart Sensor Device template meeting the requirements of the UPnP committees was developed.

To prove our work, we developed prototypes of a UPnP Smart Sensor device and a UPnP Smart Sensor control point that work in a Windows environment.

These modifications allow design Smart Sensor devices, which can be easily connected to the Internet, while minimizing implementation effort.

We also consideredthe possible extensions of our work. In addition to the developed prototypes, it is possible design and develop a concrete implementation of a UPnP Smart Sensor device. One of the hardware options under consideration is an FPGA. Implementing UPnP in Wireless Smart Sensor devices may also be considered [42][43].

Having realized a real-world Smart Sensor implementation, we may consider expanding our efforts towards designing a solution that will provide UPnP capabilities to a wide range of devices. This solution will include both hardware complete with a network interface, and an input/output bus to connect devices that do not have network capabilities. We can also develop software that can bring UPnP features to devices connected to the input/output bus.

This could bring enormous benefits to companies that want to add UPnP features to their manufactured devices.

7 References

[1] M.Weiser, J. Seely Brown, *"Calm technology"*, Xerox PARC, 1996, available at: http://www.ubiq.com/hypertext/weiser/acmfuture2endnote.htm

[2] S. Deering, R. Hinden "IPv6 Specification" http://ds.internic.net/rfc/rfc1883.txt, December 1995

[3] "Universal Plug and Play Forum", http://www.upnp.org

[4] "Jini Technology Architectural Overview", Source: *http://wwws.sun.com/software/jini/whitepapers/architecture.html*

[5] "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Network Capable Application Processor (NCAP) Information Model", IEEE Std 1451.1 – 1999

[6] "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators-Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats", IEEE Std 1451.2

[7] "Dynamic Host Configuration Protocol. RFC 2131"

[8] V.Groza, I.Sharovar, and D.Ionescu "Smart Sensors with Universal Plug and Play Capabilities", Buletinul Stiintific al Universitatii "Politehnica" din Timisoara, Romania, Seria AUTOMATICA si CALCULATOARE, 2004

[9] P.Mockapetris"RFC 1034.Domain Names- Concepts and Facilities",1987

[10] R. Frank, Understanding Smart Sensors, Second Ed., Artech House, April 2000

[11] R.Braden"RFC1123. Requirements for Internet Hosts – Application and Support", 1989.

[12] K.Lee, "IEEE 1451: A standard in support of Smart Transducer Networking," Proceedings of the 17th IEEE Instrumentation and Measurement Technology Conference, pp. 525 –528, 2000

[13] R.Fielding, H.Frystyk, and T. Berners-Lee" RFc 1945. Hypertext Transfer Protocol", 1996

[14] Smart Transducers Interface Specification," Version 1.0 formal/03-01-01, OMG – Object Management Group, January 2003

[15] P.Vixie, S.Thomson, Y.Rekheter, and J.Bound "RFC 2136. Dynamic Updates in the Domain Name System (DNSUP-DATE)", 1997

[16] H. Kopetz, M. Holzmann, W. Elmenreich, "A Universal Smart Transducer Interface: TTP/A," ISORC 2000 - The 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing, Newport Beach, California, USA, March 15 - 17, 2000

[17] D.Meyer "RFC2365 Administratively Scoped IP Multicast", July 1998

[18] "IEEE P1451 Draft Standard for a Smart Transducer Interface for Sensors and Actuators", http://ieee1451.nist.gov/

[19] T. Berners-Lee, R. Fielding, L. Masinter "RFC2396 Uniform Resource Identifiers (URI)", August 1998

[20] R. N. Johnson, S. P. Woods, "Overview and Status Update for IEEE 1451.2: Transducer to Microprocessor Communications Protocols and Transducer Electronic Data Sheet (TEDS) Formats," Proc. of Sensors Expo, http://www.telemonitor.com/doc/dot2.pdf , May 2000

[21] R.Fielding, J.Gettys, J.Mogul, H.Frystyk, L.Masinter, P.Leach, and T.Berners-Lee "RFC 2616. Hypertext Transfer Protocol –1.1", 1999

[22] R. N.Johnson, "Building Plug-and – Play Networked Smart Transducers. The Proposed IEEE P1451.2 Standard Enables Transducer Manufacturers to Support Multiple Fieldbuses", http://www.telemonitor.com/doc/sensors.pdf

[23] B.Miller,T.Nixon, C.Tai, and M.Wood "Home Networking with Universal Plug and Play" IEEE Communication Magazine, December 2001

[24] UPnP Device Architecture 1.0.Version 1.0.1," Dec., 2003. http://www.upnp.org/resources/documents/CleanUPnPDA101-20031202s.pdf

[25] "Understanding Universal Plug and Play" Microsoft Windows ME white paper, 2000

[26] C. Bettstettter, C. Rennder, "A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol," Institute of Communication Networks, D-80290 Munich, Sept. 2000

[27] J.Waldo, A.Wollrath, G.Wyant, and S.Kendall "Events in an RPC-Based Distibuted System." Sun Microsystems Lab Technical Report, November 1995

[28] M. Jeronimo, J. Weast "UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play," Intel Press, 2003

[29] D. Potter, "Overview and applications of the IEEE P1451.4 smart sensor interface standard," AUTOTESTCON Proceedings, IEEE, Oct. 2002, Pages: 777 – 786, 2002

[30] D.Commer, "Internetworking With TCP/IP Volume 1: Principles Protocols, and Architecture", Fourth edition, Upper Saddle River, NJ: Prentice Hall,2000.

[31] "Simple Object Access Protocol (SOAP) 1.1," W3C Note 08, www.w3.org/TR/2000/NOTE-SOAP-20000508, May 2000

[32] D.Commer (with D.Stevens), "Internetworking With TCP/IP Volume II:Design, Implementation, and Internals", Third edition, Upper Saddle River, NJ: Prentice Hall,1999.

[33] J. Cohen, S. Aggarwal, "General Event Notification Architecture (GENA) Base," available at http://www.ietf.org/internet-drafts/draft-cohen-gena-p-base-01.txt

[34] D.Commer(with D.Stevens), "Internetworking With TCP/IP Volume III:Client-Server Programming and Applications, BSD Socket Version", Second edition, Upper Saddle River, NJ: Prentice Hall,1996.

[35] "Intel Software for UPnP Technology", http://intel.com/ technology /UPnP/doc.htm

[36] S.Holzner "Inside XML", Indianapolis: New Rides Publishing, 2001

[37] D.Kosiur "IP Multicasting", New York: John Wiley & Sons, Inc, 1998

[38] R.Pressman, "Software Endineering: A Practitioner's Approach", New York:MCGraw-Hill, 2001

[39] B.Travis, "Smart-sensor standard will ease networking woes", EDN, June 22 1995, pg 49

[40] T.Miao "IEEE 1451.2, A Network-Independent Standard for Smart Transducers," presentation, Analog Devices Inc.

[41] R. N. Johnson "Building Plug-and-Play Networked Smart Transducers" source: http://ieee1451.nist.gov/edc-pap.pdf

[42] M.R. Moore and S.F. Smith
Kang Lee "The Next Step - Wireless IEEE 1451 Smart Sensor Networks", Oak Ridge National Laboratory, National Institute of Standards and Technology,2001, source: http://www.sensorsmag.com/articles/0901/35/

[43] M.Dunbar "Where Wireless Sensor Communications and the Internet Meet", source: http://www.sensorsmag.com/articles/0900/89/index.htm

[44] E.Freeman " Is UPnP Plugging Jini's Bottle?" source: http://www.ericfreeman.com/stories/2002/04/28/isUpnpPluggingJinisBottle.html

[45] "UPnP, Jini and Salutation - A look at some popular coordination frameworks for future networked devices." source: http://www.cswl.com/whiteppr/tech/upnp.html

October 2004

Letter of Intent to complete Trainer's Certification

This confirms my intent to complete, by November 30, 2004, the trainer's certification required to be a trainer for a Nepean Girls Hockey Association team. I understand that failure to complete the required course will result in my inability to continue on as a trainer.

Signature: _____

Name: _____

Address: _____

Telephone: _____