DETAILED DESIGN OF INTELLIGENT OBJECT FRAMEWORK

Sasa Savicand Hao Shi

College of Engineering and Science, Victoria University, Melbourne, Australia

ABSTRACT

The design and implementation of Intelligent Object Framework(IOF) aims to unite the communication and device management through a platform independent management protocol in conjunction with a management application. The Core Framework is developed using Microsoft Visual Studio, Microsoft's .NET Framework and Microsoft's Windows Mobile SDK. Secondary Intelligent Object is developed using Tibbo Integrated Development Environment (TIDE) and T-BASIC programming language that is loaded on an EM1026 Embedded Device Platform running Tibbo Operating System (TiOS). The backend database is based on Microsoft's SQL Server.In this paper, protocols associated with Smart Living are first reviewed.The system architecture and intelligent object management studio are presented. Then device application design and database design are detailed. Finally conclusions are drawn and future work is addressed.

KEYWORDS

Intelligent Object Framework, Intelligent ObjectManagement Studio, Wireless Network

1. PROTOCOLS

1.1.SNMP (Simple Network Management Protocol)

SNMP is an application-layer protocol defined in the RFC1157, for exchanging management information between network devices. It is part of the TCP/IP protocol suite and it is one of widely accepted protocols used to manage and monitor network elements. The basic SNMP components are SNMP Manager, A managed device, An SNMP Agent, and Management Information Base (MIB) as shown in Figure 1 [1].

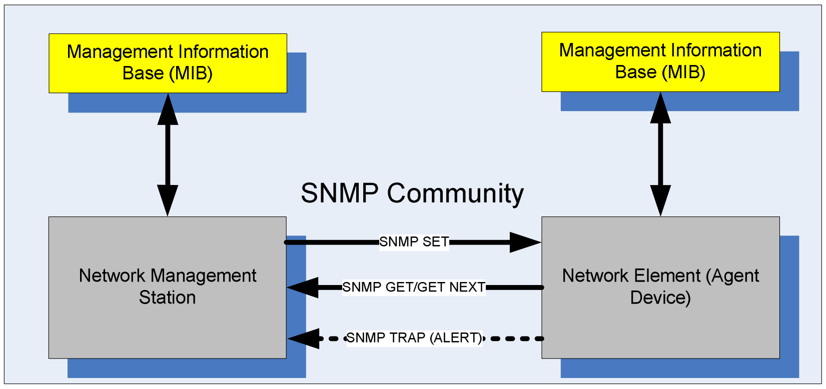

Management Information Base (MIB)
Management Information Base (MIB)
SNMP Community
Network Management Station
SNMP SET
SNMP GET/GET NEXT
SNMP TRAP (ALERT)
Network Element (Agent Device)

Figure 1: SNMP High Level Diagram [1]

Whilst being a transparent request and response protocol that communicates certain management information between two types of SNMP software entities, it fails to meet certain pre-emptive actions and response times. Although it is majorly implemented in network devices and secondary management interfaces, it has also been found on computers and printers. Despite its apparent compatibility across distributed systems, it is considered a highly complicated protocol to implement as it comes with numerous abstruse encoding rules and schemes. SNMP also lacks efficiency as network bandwidth is exhausted with useless data [2, 3]

As described in RFC3411[3]Section 4.4.3, SNMP version is transmitted in every SNMP datagram. In addition to the unnecessary data overhead at each datagram, multiple length and data descriptors are being broadcasted throughout each message. The inefficiency in packet encapsulation methodology yields in superfluous data handles which are across the network. This effect is not felt in small to medium networks, although, networks that consist of large scale distributed devices do. Furthermore, device failures are not detected in real-time and device that succumb to network failures will not be noticed until the next polling cycle.

Software applications managing the SNMP enabled devices must be aware of the available information that lies within the device. If there is a discrepancy in the schema, this must also be changed in the management application. The proposed Intelligent Object Framework differs by accepting device changes via adding functions, storing them to a database by conforming to a common data model. Once the data exchange occurs, the functions and its data types become available to any application, thus enabling a user to execute the new/amended function.

1.2. EXtensible Automation Protocol (xAP)

Another kind of network management and control protocol that has emerged since the late 90's is xAP. An open source protocol intended for support and integration of telemetry and control devices within a home environment. While the framework and protocol definition [4] enables multi-device control over UDP protocol, seen in Figure 2, it achieves this by multi broadcast messages.

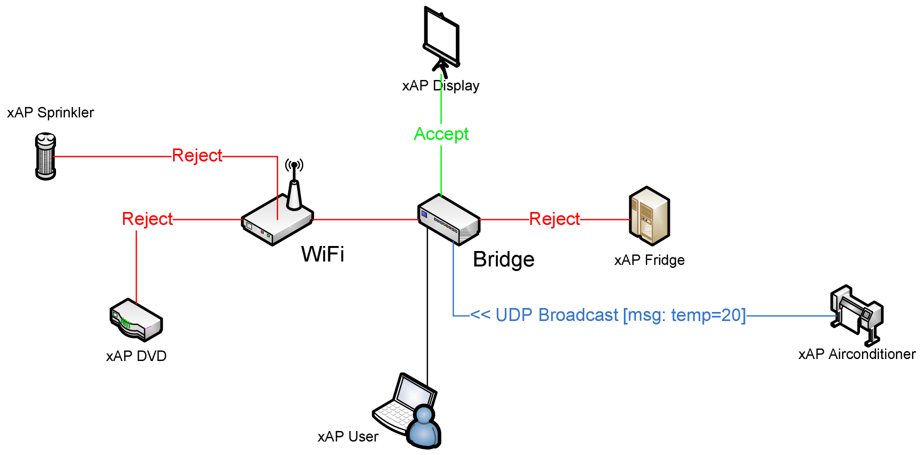


Figure 2.xAPMessage Broadcast Flow [5]

This approach must rely on independent applications and devices to drop unwanted packet data. Moreover, failure to address the issue of centralized function management yields in complex software and hardware design. XAP approach relies on each device implementation to vary its schema, thus it does not conform to a framework standard, but rather rely on protocol specification itself.

1.3. X-10 Home Automation

In the late 1970's, Pico Electronics produced an inexpensive home automation equipment. The simple idea was to use home's power wiring to communicate from a controller to small remote control modules. The X-10 technology is still widely used amongst household appliance control. The technology behind X-10 works by sending brief bursts of 120 KHz signals over the power lines, timing the bursts to coincide with the 60 Hz power line zero-voltage crossing. At the zero voltage point, the power line noise level is at a minimum and the 120 KHz signal has a better chance to be received by a remote control module [6]. At the highest level the X-10 messages are comprised of three elements, Table 1.

Table 1. X-10 High Level Message [6]

House Code	Unit Code	Function Code
Within the X-10 protocol, there are 16 possible house codes and they are identified as A through to P.	There are 16 possible unit codes, and is conventionally identified 1 through to 16.	There are 16 possible function codes which are identified by their operations such as "On", "Off", "Dim".

The X-10 home automation system is developed using MBasic, a language specifically used for PIC microcontrollers. The MBasic holds a library of functions related to X-10 protocol, thus enabling the above House, Unit and Function code specifications to be standardized in the form of **X_A** through to **X_P**, **X_1** through to **X_16** and **X_On** or **X_Off**. Controlled modules are addressed by both house and unit codes. Devices are usually controlled by rotating switches that have House and Unit code settings. By rotating the switches, the module's address is set to the desired house and unit codes.

Further looking at X-10 protocol, Parralaxstates that interface modules have limitations to how well they may work in particular area [7]. During the testing of **Xin** and **Xout** commands, Parallax found that the modules would not address some units in the same building. It was stated that this may have been due to noise in the lines or faulty electrical wiring. The units did not work well with filtering power strips either.

The rules of the protocol are hard-coded and the hardware functions must be statically assigned to the protocol's Function codes. Whilst the system may be useful for simple operations, it does however fall short of the ability to be controlled via other means such as wireless or Ethernet communication. Another issue that may arise from the operation of the units is the possibility of a fault within the special module power plugs. The danger to that is that the user is playing around high voltage devices. Since the tone bursts are scattered across the household, this in-fact scatters along the entire power line grid in the vicinity of the house, thus there is a chance of somebody else generating signal bursts and tampering with the internal house hold appliances, resulting in damages or overriding of control.

1.4. MODBUS

MODBUS is a serial communications protocol, published by Modicon in 1979[8]. It is widely used in industrial control applications. The MODBUS protocol defines the message structure and the communication rules for interconnection of control systems that exchange supervisory control and data acquisition (SCADA) information for controlling industrial processes [9]. The MODBUS comes in two different variants, serial communication and the TCP communication version. MODBUS is an application layer messaging protocol and is positioned at level 7 of the OSI model. It provides client/server communication between devices connected on different types of buses or networks. The MODBUS communication stack is depicted in Figure 3.

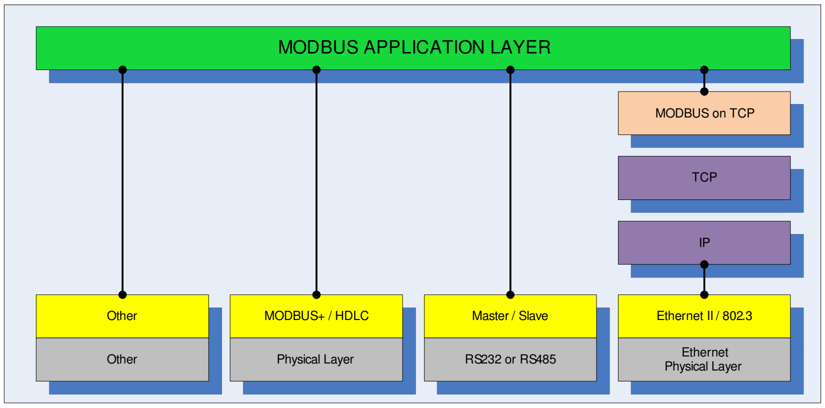


Figure 3.MODBUS communication stack [8]

1.5. HAP (Home Automation Protocol)

HAP was developed in 2002 by Sriskanthan [10] of Nanyang Technological University in Singapore. HAP facilitates the communication amongst the host and client modules in a home automation system and the protocol is constructed above Bluetooth software stack [10, 11]. The HAP protocol's initialization process utilizes the Service Discovery Protocol to query device information and services within Bluetooth's Piconet illustrated in Figure 4.

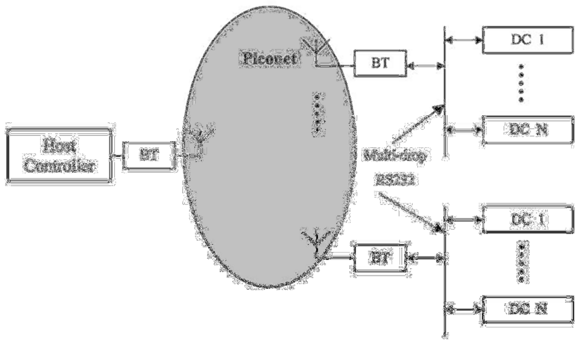


Figure 4. Host and client modules Bluetooth Piconet [10]

The individual clients are shown in Figure 5. HAP protocol uses descriptor table definition to store the information of all active network devices. The heavy lifting is done by the Device Controller, as it respectively hosts the descriptor information. The Host Controller then searches the device information from the Device Controller and it constructs the descriptor table, which the Device Controller is responsible for.

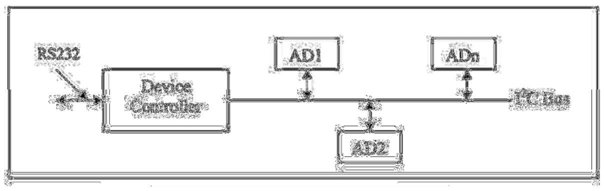


Figure 5.Individual Client Module [10]

The protocol is implemented on Bluetooth enabled devices and it requires hardware development backing in order for the devices to control a piece of appliance or equipment. The client module only acts as the medium for communication and the transaction of the HAP packets.

2. System Architecture

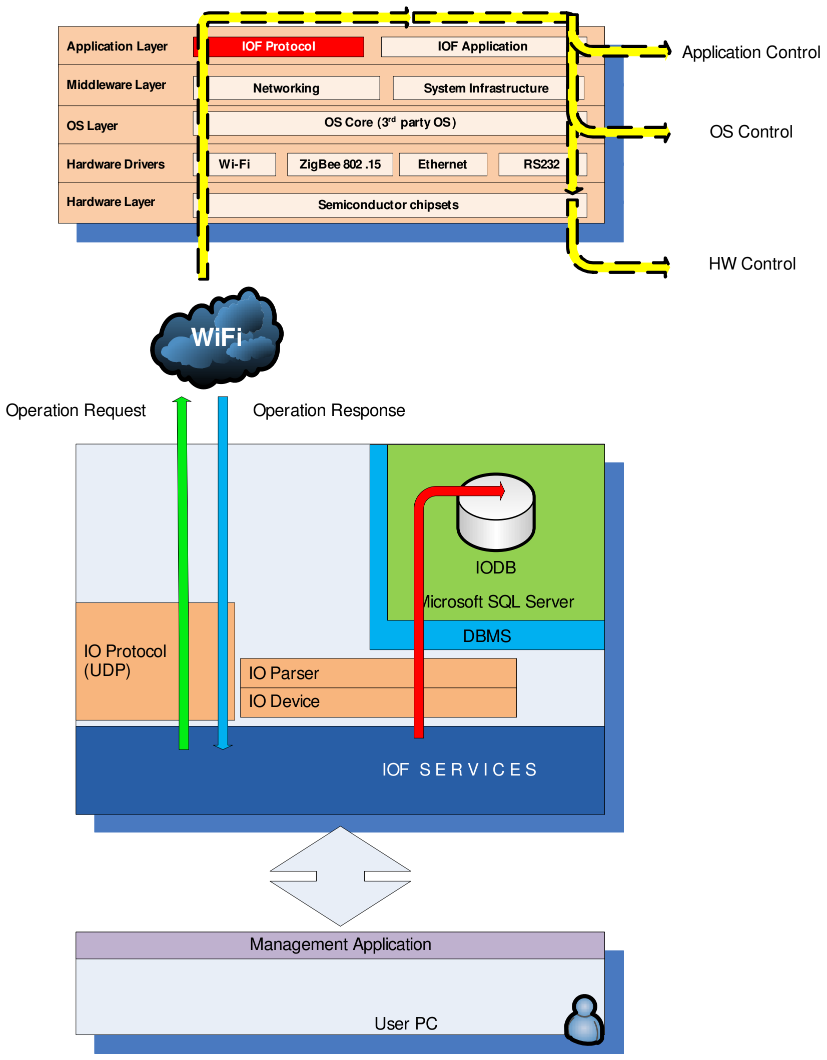


Figure 6. Intelligent Object Framework Block Diagram [12]

IOF system comprises of three component segments, the controller, service and the endpoint devices. The three layers are architecturally separated, although maintain communication via protocol services. The Services layer can further be separated in three layers outlining the core components. Figure 7 shows the separation between the internal components of the services.
The implementation of the IProtocol interface is directly linked to the Protocol class. The protocol, also defined as a service class parses to the Processing and Logic Layer. The Protocol class does this by invoking a device instance. This process is further described in the IOF Protocol section of the document. The device class inherits Device Data as the root instance with the collection of Functions and the Function Parameters. If a new device is attempting to join the IOF Service and is accepted, the device instance communicates directly with the database.
Furthermore, the separation of the networking, process and data layer allows device logic separation enabled for further possible extension of the framework. To implement a new functionality, the developer can focus on one logic section at a time. The methodology implemented here is in-turn plug-in architecture based.

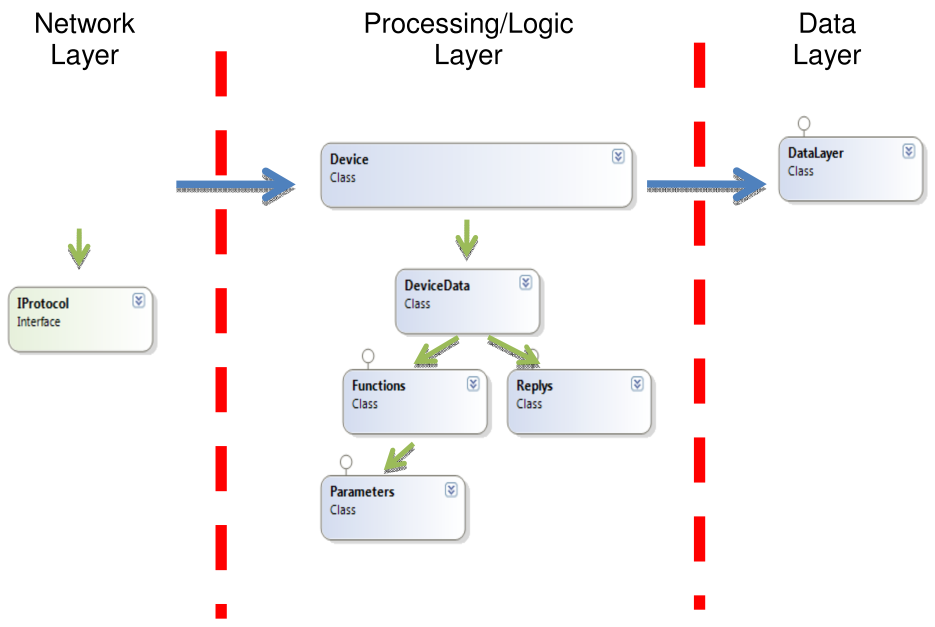
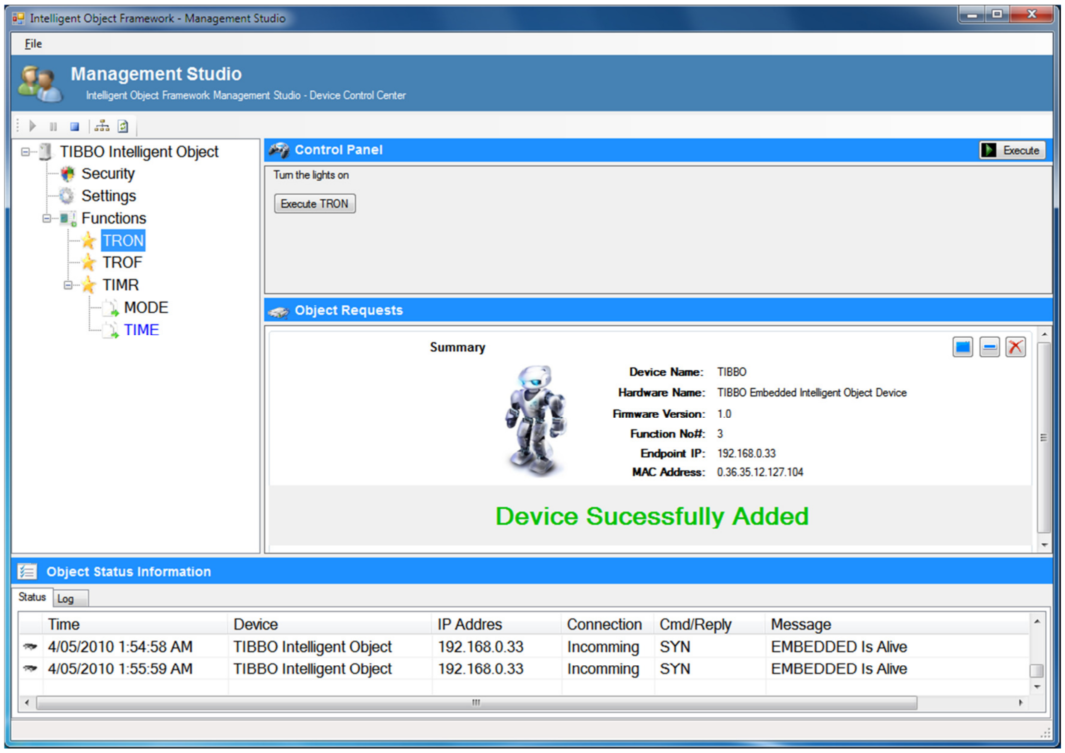


Figure 7.Framework Hierarchy

3. Intelligent ObjectManagement Studio

Intelligent Object Framework Management Studio (IOMS) provides an easy to use interface for managing devices attached to the network. Underneath the simplicity of the design lies an engine that dynamically creates the controlling environment based on the function content received from a particular device or set of devices. Figure 8 shows the Management Studio platform with the basic set of features. It renders a mainstream application design with features as control panel, status screen and the device viewing window



Device Name:	TIBBO
Hardware Name:	TIBBO Embedded Intelligent Object Device
Firmware Version:	1.0
Function No#:	3
Endpoint IP:	192.168.0.33
MAC Address:	0.36.35.12.127.104

Time	Device	IP Addres	Connection	Cmd/Reply	Message
4/05/2010 1:54:58 AM	TIBBO Intelligent Object	192.168.0.33	Incomming	SYN	EMBEDDED Is Alive
4/05/2010 1:55:59 AM	TIBBO Intelligent Object	192.168.0.33	Incomming	SYN	EMBEDDED Is Alive

Figure 8. Intelligent Object Framework Management Studio [12]

Once the application has been started, the user of the application needs to start up the services. This is accomplished by turning the services on, pausing and stopping the services fully by utilization of the menu bar shown in Figure 9.

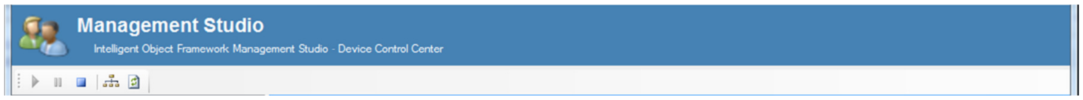
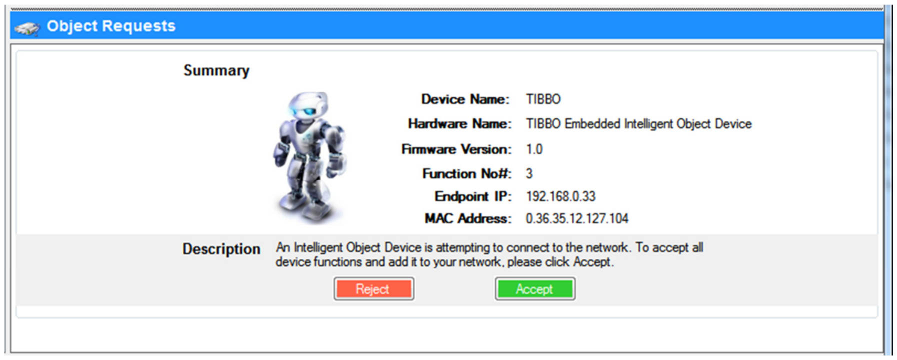

Management Studio
Intelligent Object Framework Management Studio - Device Control Center

Figure 9. Menu Bar

By clicking the Play button, a service process is spawned that starts up the protocol service which allows discovery and device communication.

The Object Requests section, shown in Figure 10 shows a device that is seeking a connection to the network. Important device information is displayed upon connection request. The user can see the device name and additional information. Another important factor is to display the function number. This shows the number of functions the particular device has on offer. The data underneath is used in setting up the function iteration loop which receives the functions from the intelligent object. The Object Request information window also displays the MAC address as it is a primary identifier that is used in distinguishing different devices from one another within the framework itself and the database.


Object Requests

Summary

Device Name: TIBBO
Hardware Name: TIBBO Embedded Intelligent Object Device
Firmware Version: 1.0
Function No#: 3
Endpoint IP: 192.168.0.33
MAC Address: 0.36.35.12.127.104

Description: An Intelligent Object Device is attempting to connect to the network. To accept all device functions and add it to your network, please click Accept.

Figure 10. Object Request window [12]

Once a user accepts the device to be added to the IOF network, the framework initiates a function download process that is described in Section 4, and further explained in the next few sub-sections. Once the functions have been successfully downloaded from the intelligent object device, and stored to the database, user can access them from the tree menu. Figure 11 shows the menu which displays all the intelligent objects that are currently online and functions the users are allowed to use on the device.

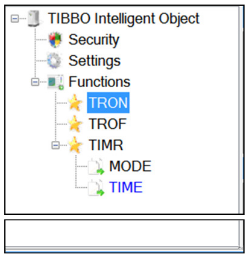
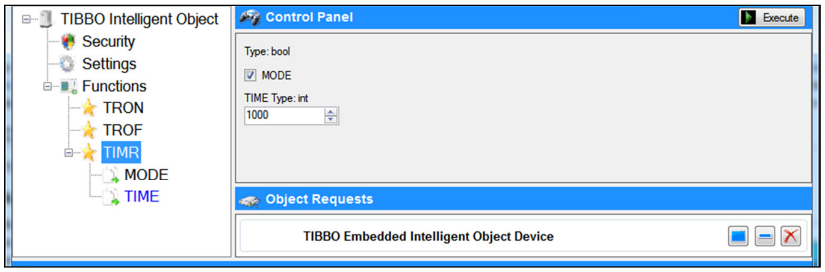


Figure 11. Available devices tree menu

Upon clicking on a particular function within a device menu, the IOF services fetch function data from the database, dynamically rendering the Control Panel section of the application seen in Figure 12.



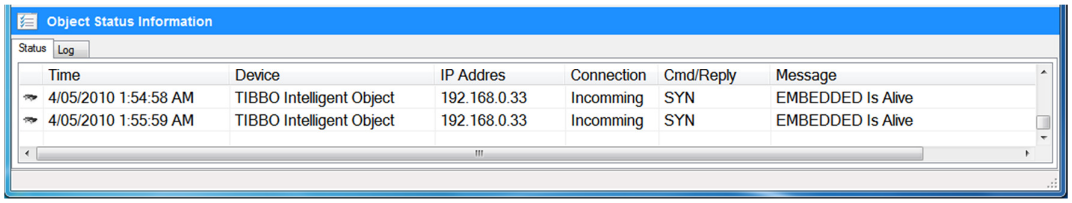
Type: bool
☑ MODE
TIME Type: int
1000

TIBBO Embedded Intelligent Object Device

Figure 12: IOF Studio Control Panel

The control panel section dynamically adds functions based on the function type which is later explained in Section3.6. In the above example, the TIMR (Timer) function, which has been derived from the Intelligent Object itself, has input parameters; Type (which is of type Boolean) and Time (which is of type integer). This particular example function allows a timer to switch on or off the Tibbo embedded device within a set time interval. The user can then execute the

function by clicking the Execute button in the top right corner. The remaining two functions that have also been exposed by the framework are TRON (Turn ON) and TROF (Turn OFF).
Another feature of the IOF framework is the Object Status Information window, Figure 13. Vital information that is being sent to the Intelligent Object (IO) device is logged in this window, but more importantly replies from the device to the services are being constantly received.



Time	Device	IP Addres	Connection	Cmd/Reply	Message
4/05/2010 1:54:58 AM	TIBBO Intelligent Object	192.168.0.33	Incomming	SYN	EMBEDDED Is Alive
4/05/2010 1:55:59 AM	TIBBO Intelligent Object	192.168.0.33	Incomming	SYN	EMBEDDED Is Alive

Figure 13 Object Status Information

In the above example, the TIBBO Embedded Device is constantly synchronizing its connectionless session with the server (since the IOF uses UDP protocol for device communication). The device also sends status messages in regards to the current operation of the device, where potential device faults can also be seen and monitored. This is an automated process. This gives users the ability to trace the communication process and potentially see if there are any communication glitches. UDP packets can sometimes be lost during the transmission, and in those particular cases, if a reply has not been received by the device, it can be a clear indication that there is a problem in the communication layer. This could further be extended with an in-depth description of errors, where particular users can drill in further to find out the exact causes of potential device disasters.

3.1. System Block Diagram

Intelligent Object Framework can be dissected in three main components from the service end as shown in Figure 14. The Network Layer; defines the protocol and services, Processing and Logic Layer; describing the device object and the parser needed to evaluate the data coming from the Intelligent Object device, and Database Layer; that interacts with the database for storage and retrieval of device functions. The Intelligent Object comprises of similar layers as the framework itself, except it does not need data layer as storage of functions is only done on the service end. The Intelligent Object does include an extra Execution Layer which interacts with applications, peripherals or abstract controls. The interactions are defined as functions that are sent to the Intelligent Object services, defined by the protocol.

3.2. UDP Protocol

User Datagram Protocol (UDP) is classified as the simplest OSI transport layer protocol that is intended for client/server networking applications that are based on Internet Protocol (IP). Being one of the oldest network protocols, it still remains one of the most popular, tuned specifically for real-time performance systems and applications where prior communication setup is not required. Intelligent Object Framework is based on UDP protocol as a simple solution to device management and monitoring. The packet structure of the UDP protocol is relatively simple. Table 2 shows the structure.

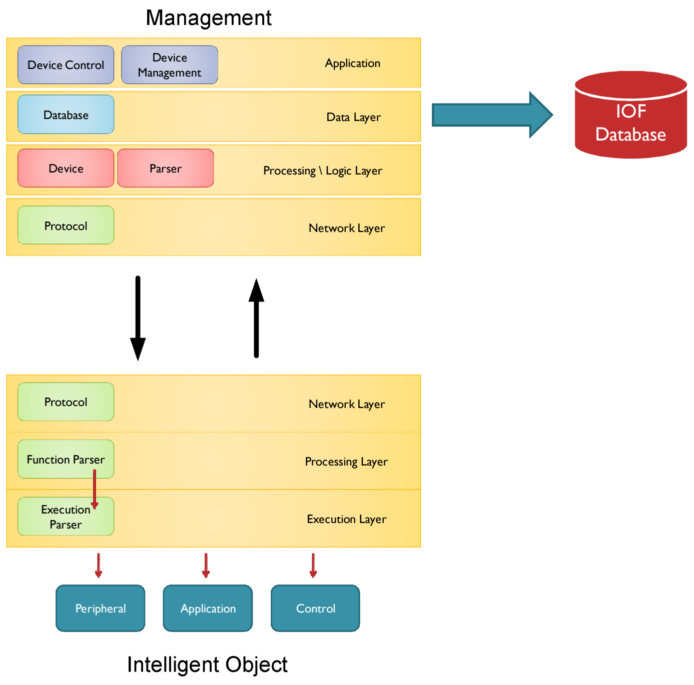


Figure 14. IOF System Architecture block diagram [12]

Table 2. UDP Packet Structure

bits	**0 – 15**	**16 – 31**
0	Source Port Number	Destination Port Number
32	Length	Checksum
64	Data	

The UDP header consists of four fields 2 bytes each field, with the fifth data field. The maximum UDP packet size is approximately 65532 bytes (8 byte header + 65527 bytes of data). The 65527 bytes of payload data is sufficient enough for function transfer exchange without putting a big overload on the transmission. The two fields highlighted in pink are optional to the header and indicate the integrity and verification of the payload. In the particular case where transmission reliability is needed, application must implement this [13]. Intelligent Object Framework implements the reliability feature by asynchronous device status implementation. As defined earlier, this allows the user and application do receive device specific replies in order to provide the status of the application. On the above UDP header sits the IP packet that indicates the destination and source address.

Dissecting a typical data transmission from Intelligent Object device to Intelligent Object server and vice-versa, we can further investigate the behaviour of the UDP packet with data payload. Figure 15 shows the packet trace of an Intelligent Object device sending device data right after a successful authentication.

```
Internet Protocol, Src: 192.168.0.33 (192.168.0.33), Dst: 192.168.0.8
(192.168.0.8)
        Version: 4
        Header length: 20 bytes
        Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        Total Length: 110
        Identification: 0x008b (139)
        Flags: 0x00
        Fragment offset: 0
        Time to live: 255
        Protocol: UDP (17)
        Header checksum: 0x397a [correct]
                Good: True
                Bad: False
        Source: 192.168.0.33 (192.168.0.33)
        Destination: 192.168.0.8 (192.168.0.8)
User Datagram Protocol, Src Port: 10183 (10183), Dst Port: ndmp (10000)
        Source port: 10183 (10183)
        Destination port: ndmp (10000)
        Length: 90
        Checksum: 0xdd18 [validation disabled]
                Good Checksum: False
                Bad Checksum: False
Data (82 bytes)
        Data: 302e33362e33352e31322e3132372e3130347c337c313932...
        Length: 118
```

```
0000  001f 3c 9b 57 8d 00 24  23 0c 7f 68 08 00 45 00   ..<.W..$ #..h..E.
0010  00 92 00 8e 00 00 ff 11  39 53 c0 a8 00 21 c0 a8   ........ 9S...!..
0020  00 08 27 c7 27 10 00 7e  39 a5 30 2e 33 36 2e 33   ..'.'..~ 9.0.36.3
0030  35 2e 31 32 2e 31 32 37  2e 31 30 34 7c 33 7c 31   5.12.127 .104|3|1
0040  39 32 2e 31 36 38 2e 30  2e 33 33 3b 30 2e 33 36   92.168.0 .33;0.36
0050  2e 33 35 2e 31 32 2e 31  32 37 2e 31 30 34 3b 54   .35.12.1 27.104;T
0060  49 42 42 4f 20 49 6e 74  65 6c 6c 69 67 65 6e 74   IBBO Intelligent
0070  20 4f 62 6a 65 63 74 3b  54 49 42 42 4f 20 45 6d    Object; TIBBO Em
0080  62 65 64 64 65 64 20 49  6e 74 65 6c 6c 69 67 65   bedded I ntellige
0090  6e 74 20 4f 62 6a 65 63  74 20 44 65 76 69 63 65   ntObjec t Device
```

Figure 15. A packet trace of an Intelligent Object device [12]

The above packet trace shows the IP + UDP structure for the example transmission. Applying packet filtering to the received data from the source port (IOF server) and destination port (IO device), yields in extended data which one can use to further analyse the function exchange. The graph in Figure 16 shows the packet throughput from the time when Intelligent Object gains authentication to the Services, to the point where the last packet of data is received.

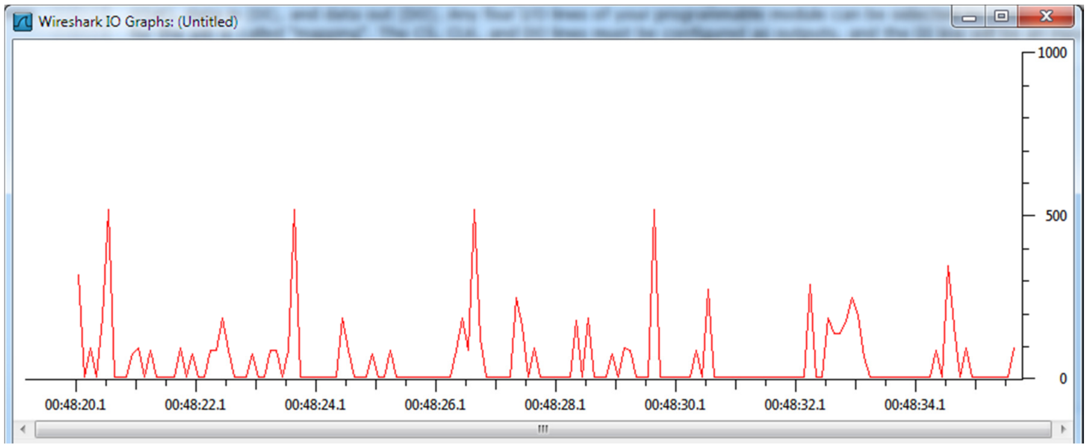


Figure 16. Function exchange transmission; byte/tick interval (0.1 sec)

Further analysing the function exchange transmission graph shows significant peak at each function exchange interval that reaches 500 bytes of data at each sending interval. In the particular example above, each function exchange response is proportional to the next response. The same statistics can be seen for function requests, where the peak request does not go over 250 bytes of data. In both request and response instances, entire bundle of packets are a frame. This means that each request and response packet sent or received includes the Data Link Layer packet (in this instance the Ethernet packet), the Internet Protocol packet followed by the UDP packet with the payload. The physical layer packet is missing from the stack as it is stripped off by the network hardware itself.

What is concluded from the analysis of function exchange is that the UDP payload inflicted by the data embedded by the Intelligent Object device and the Intelligent Object Services is not significant enough to make an impact on the network traffic, rendering this method as viable solution to service to device communication. Furthermore, the next section of chapter analyses the protocol layer from the application perspective. This includes the protocol layer of services and device.

3.3. Intelligent Object Protocol Layer

The objective of Intelligent Object Framework Protocol (IOFP) is to reliably and efficiently communicate with Intelligent Object (IO) devices. IOF Protocol is unconstrained of the selective transmission subsystem, thus only requiring an ordered data stream channel.

The design of IOF Protocol is based on the following model of communication: as intelligent object requests to communicate with Intelligent Object Services (IOS), the IOS establishes a connectionless UDP communication channel. Intelligent object then acknowledges the connection, initializing its device descriptor which holds device data. In response to the intelligent object's descriptor packet, IOS request further data from the intelligent object. Upon reception of the next packet, intelligent object initializes its next set of data to send which are the functions and function parameters. The IOS also sends IOS data to the intelligent object. This is the IOS descriptor which devices stores and uses for communication purposes.

The IOF protocol commands define the service to device communication. IOF protocol is a text-based protocol terminated by a <CRLF> (Carriage Return Line Feed). The commands themselves are numerical based and can be easily defined in any programming language as a set of enumerated properties. Each section of the corresponding command is delimited by a pipe character "|" and must conform to receiver conventions. Both the services and intelligent object conform to this standard as means of splitting streaming data and making sense out of it. Protocol configuration can be summarized by Table 3.

Table 3. Protocol Configuration

IOF Services Port	10000
IOF Device Port	10001
Data Delimiter	|
Function Delimiter	;
Parameter Delimiter	:

Intelligent Object Framework services bind and listen on port 10000, whereas the Intelligent Object devices bind and listen on port 10001.

Intelligent Object to IOF Services transactions involve several data objects that are defined as additional arguments to the command. The arguments themselves are bound by delimiters and must be transmitted together with the command. The only way the receiving system will know how to handle incoming data in a particular pattern is by knowing the first arriving field, or the command descriptor. Once the data is received in a chunk, it is split up and placed in distinct array of buffers. The buffers will further be explained in this section.

The IOF Protocol command semantics further describe Request Types, Device Status and Handshaking procedures for successful exchange and manipulation of received or sent data. The detail of Protocol Definition on Handshake, Service Response Process (Service Side), Device Request Process (Service Side), and Statuscan be defined [14].

Protocol class diagram consists of attributes, methods and events that are invoked when certain conditions are met. There are six events operating in the Protocol services layer of Intelligent Object Framework, Figure 17.

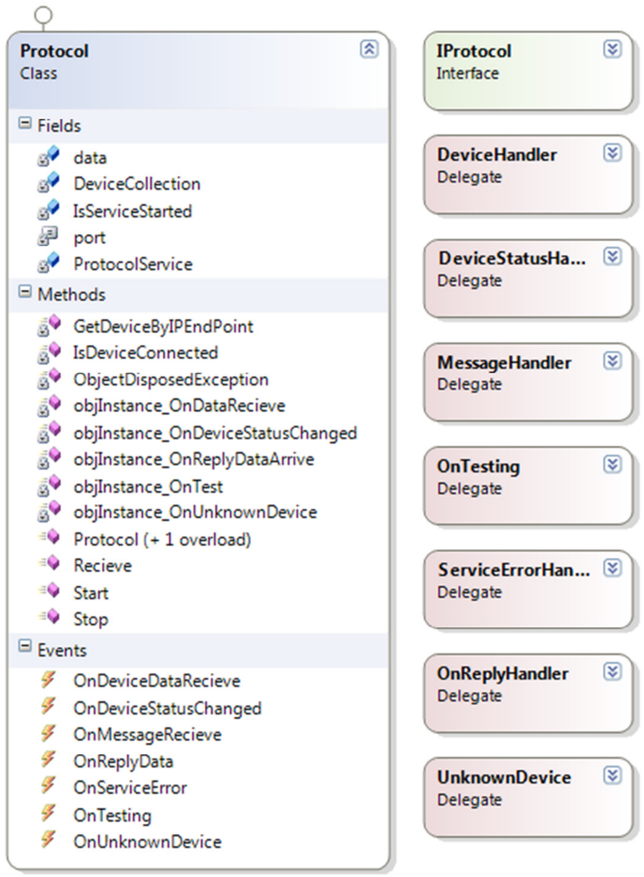


Figure 17. Protocol Services Diagram

The decision upon which event is to be fired is initiated by a ProcessData method in the Intelligent Object instance. Since each device has an instance of an Object within the DeviceCollection, they all are referenced by their separate events.

IAsyncResult interface is implemented in the Protocol class, since the class operates asynchronously. The result which is derived from the ProtocolService.EndRecieve (res, ref device) is the incoming data from the device. The objInstance.ProcessData method of the device class processes the data accordingly. The ProcessData method parses the incoming data. The data which is process is further defined in the protocol segment tables. Each table defines the data packets earlier described in this section. Their fields are also further defined.Request Type Protocol and Data Arrival Type Protocol Definitions can be found in [14].

3.4. Intelligent Object Device Layer

The Intelligent Object Framework device layer is essentially a business logic layer. The primary job of the device layer is to relay data from the IOF protocol layer and process it accordingly. It also communicates with the Data Access Layer for retrieval and storage of object information. The device layer is also responsible for raising events
To understand this process better, Figure 18 shows the interactions between different methods within the individual layer blocks. The sequence diagram illustrates the requesting procedures from the components to the corresponding activation boxes. The method-invocation occurs as a sequential process, although most processes are asynchronously defined, and are executed respectively in their threads. The diagram shows distributed workload across the IOF Services object and IOF Device object. This is due to the separation of services and device logic components. This separation is required as IOF Device object directly interrogates the Data Layer. Since the implementation of IOF Device is abstracted from the Services itself, the IOF Device layer acts as an interface for the services processes.

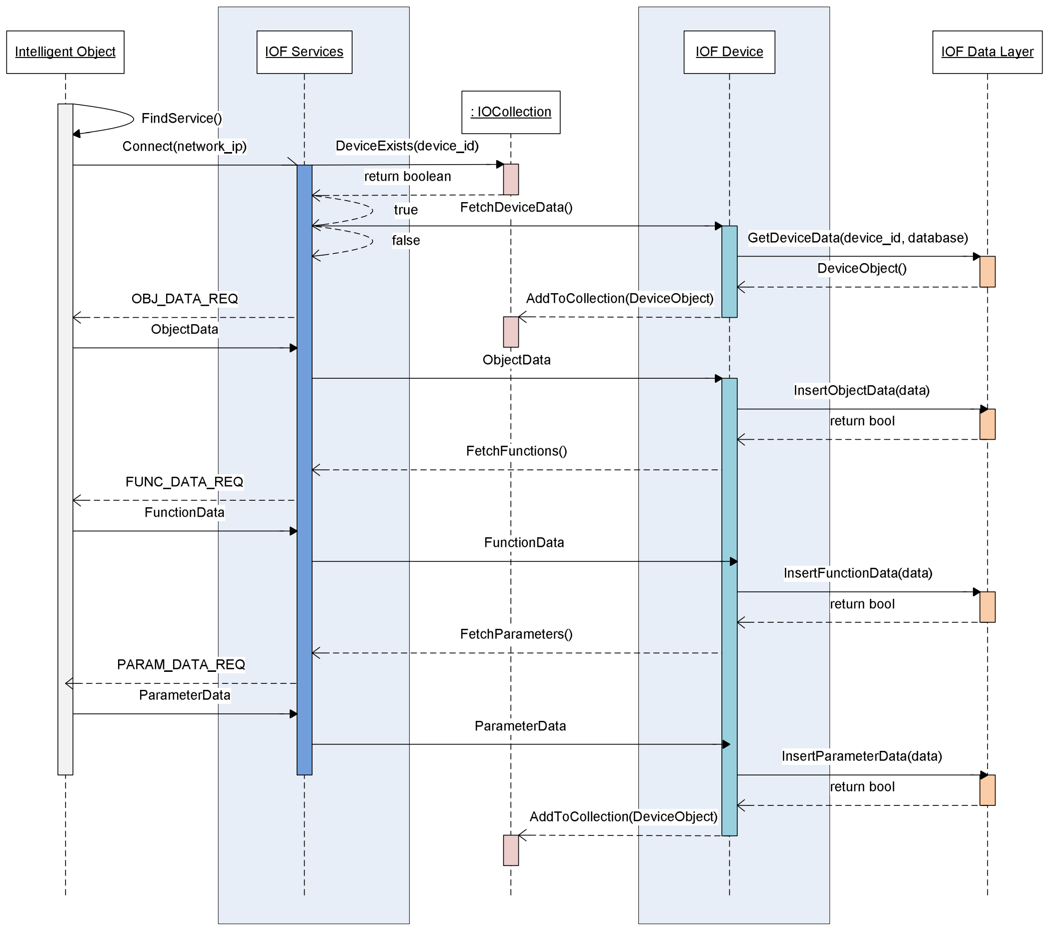


Figure 18. IOF Sequence Diagram

3.5. Intelligent Object Data Access Layer

Intelligent Object Framework Data Access Layer (DAL) is responsible for writing to and reading from database the relevant intelligent object function and device data. The specific DAL does not contain business or displaying logic but is a part of a three tier architecture design that has the following components:

- User Interface (UI) layer that contains the interface
- Business Logic Layer (BLL) which contain application's business rules.

3.6. Dynamic User Interface Control Rendering

Functions inherited from the intelligent object are stored in the IOF database. It conforms to a hierarchical structure where the function is the parent of a parameter. When the database is queried via the Data Access Layer, it forms a Function object. An important step in managing the intelligent object devices and executing functions (commands) is being able to visually present the user with the right set of controls. Part of the device control process is user input, thus the user must have a way of being able to input or select the type of information. Parameter types supported by IOF are Boolean, Integer, String and Time as shown in Figure 19. The controls displayed on the user interface are rendered based on the types supported. To better understand how the rendering model works, figure below shows the block diagram of the process. The parameter type is put through a Control Rendering Process which defines the control types. The output that is produced by the rendering algorithm is a corresponding visual control type. NumericalBox is created based on integer type, CheckBox is based on Boolean, TextBox is for a string type and Time is of TrackBar

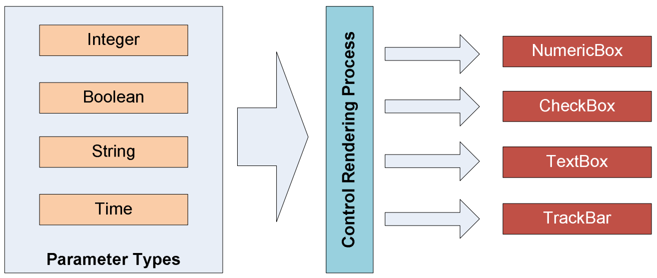
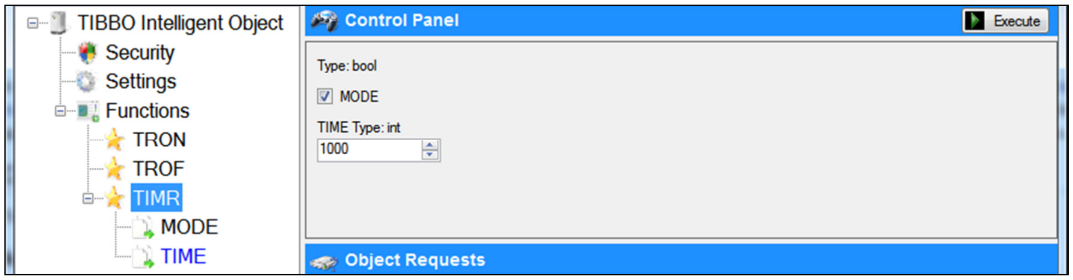


Figure 19. Parameter Type Control Rendering Process

To help support the process output, Figure 20depicts two control types based on the parameter data types from the intelligent object. The TIMR (Timer) function has two parameters. The MODE parameter and the TIME parameter are of type Integer and Boolean and are rendered accordingly. When the user makes change to the parameter, it sets the parameter value accordingly. Upon function execution, the arguments are bundled up and set to the device.



Type: bool
☑ MODE
TIME Type: int
1000

Figure 20. TIMR Function with parameters

4. Device Application Design

The HTC mobile device and the TIBBO Embedded Device design also conform to the rules of IOF Services framework, except it does not interact directly with the IOF database.There are three abstract layers on the devices as shown in Figure 21. The protocol layer is responsible for receiving data from the IOF services and is also responsible for extraction of the data. Once the data is properly parsed by the protocol process, it is passed to the Function Parser. The job of the Function Parser is to determine which Function is to be executed. The framework relies on the developer to specify the functions which are to be called. Furthermore, once the function has been determined, and the parameters extracted, the result is passed to the execution layer. The execution layer, based on the function results that have been passed to it, executes the function.

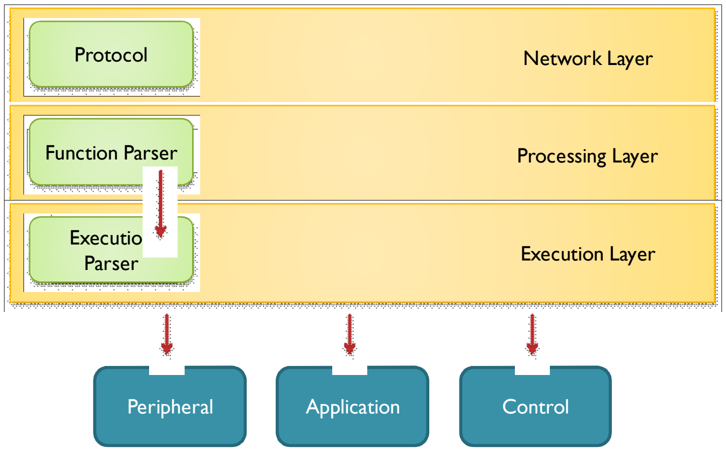


Figure 21.Device Application Design

The developer is able to control three different types of controlling layers. Peripheral control states that the device can control another peripheral on the device. This may be over an USB or RS232 port. Intelligent Object Framework prototype focused on Application specific control and generic control. Application specific control was proven by controlling HTC Mobile applications, whereas generic control was achieved by controlling LED lights on the TIBBO Basic Embedded Device.

The protocol layer of the devices mimics the IOF Service layer in its operation and it processes the data in exactly the same way. This also adheres to the parsing of the IOF packet flags. Below figures illustrate the difference in design of the portion of protocol layer. The data received from the endpoint (IOF Services) is put into the data variable, and further parsed in the function. The request flags are also checked, and `Request = … PacketData[0];` grabs the first element of the data array parsed by the protocol layer, and the request is determined. The type of request conforms to the protocol specifications explained in the Chapter 5. The way that the functions are passed from the Intelligent Object to the IOF framework is by iteration of Function object. Once the functions have been defined, and placed in the Function object, they are sent to the IOF Services.

Once the IOF services have sent a request to the Intelligent Object device to gather the device functions and parameters, a `for`loop is initialized by the device that iterates through the function collection. The post process, once the function is sent to the IOF Services, is to mark the function as sent. Once the next function is due to be sent to the Services, it skips through the function that has already been sent. The function data is set up in a function `PrepareObjectData()`.

The Function object is defined by the parameters. The function object attributes hold the very function values and the `func.Name`is mapped to the Function Parser process, enabling the particular command to be executed. Finally in the below figure, the CALL function is parsed and further processed by the Object’s internal framework.

5. Database Design

The Database consists of five tables, i.e. Response, Object, Function, AccessType and Properties as shown in Figure 22 and table schema are listed in Tables 4-8.

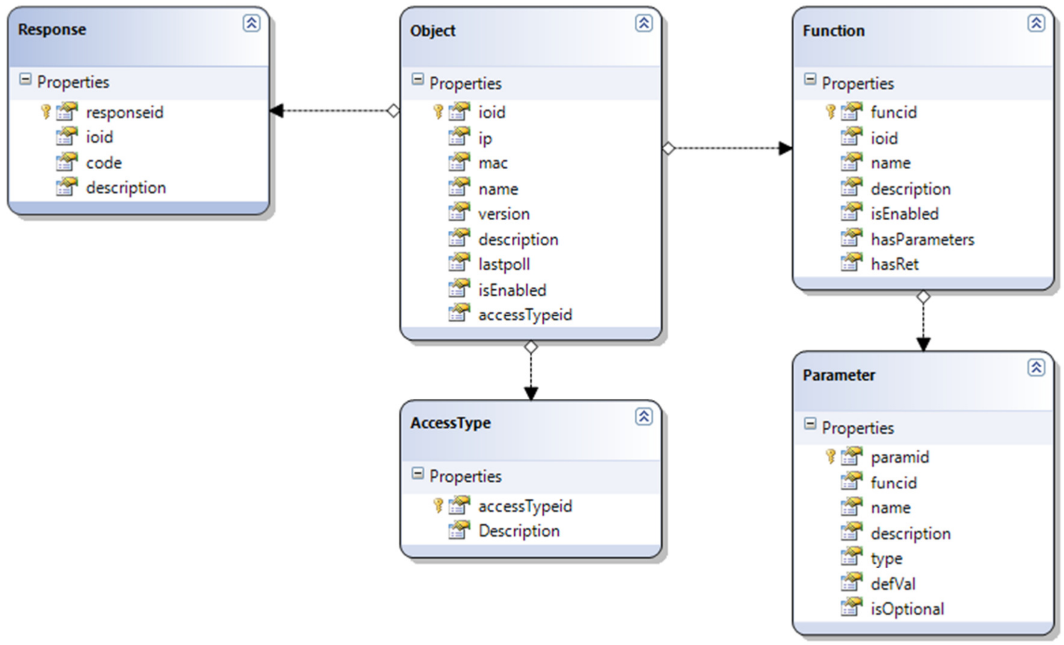


Figure 22. Intelligent Object Framework Database Model

Table 4. Objects table schema

Table – dbo.Objects		
Column Name	**Data Type**	**Description**
ioid	Varchar(MAX)	Intelligent Object ID
Ip	varchar(15)	Intelligent Object IP Address
Mac	Varchar(MAX)	MAC (Media Access Control) Address
Name	Varchar(MAX)	Intelligent Object name
Version	Varchar(MAX)	Intelligent Object software version
Lastpoll	Datetime	Last time the device polled the services
isEnabled	Bit	Check flag to see if the device is enabled or disabled
accesTypeid	Int	Access type id. Links to the AccessType table.

Table 5. Functions table schema

Table – dbo.Functions		
Column Name	**Data Type**	**Description**
Funcid	Int	Function ID
Ioid	Int	Intelligent Object ID
Name	Varchar(MAX)	Function Name
Description	Varchar(MAX)	Function Description
isEnabled	Bit	Specifies if the function is enabled or disabled
hasParameters	Bit	Has the function got parameters
hasRet	Bit	Does the function return a value

Table 6: Parameters table schema

Table – dbo.Parameters		
Column Name	**Data Type**	**Description**
Paramid	Int	Parameter ID
Funcid	Int	Function ID
Name	Varchar(MAX)	Parameter Name
Description	Varchar(MAX)	Parameter Description
Type	Varchar(MAX)	Parameter Type
defVal	Varchar(MAX)	Default parameter value
isOptional	Bit	Is the parameter optional or explicit

Table 7. AccessType table schema

Table – dbo.AccessType		
Column Name	**Data Type**	**Description**
accessTypeid	Id	Access type ID
Description	varchar(15)	Access type description

Table 8. Response table schema

Table – dbo.Response		
Column Name	**Data Type**	**Description**
Responseid	Int	Response ID
Ioid	Int	Intelligent Object ID
Code	Varchar(10)	Response code
Description	Varchar(MAX)	Response Description

6. CONCLUSIONS

The design and implementation of Intelligent Object Framework proved to be a complex task as it needed extensive research on communication models and protocol implementation specifications design. The right set of devices needed to be chosen, that have the right platform features allowing the implementation of core components, in order to design and develop the proof of concept. It has achieved by introducing a function exchange model implemented in the structure of the protocol and allowing devices to hand over their complete set of functions to be stored in a database. Users can then use the management application to query those very functions and be able to control the devices wirelessly.The IOF consists of a framework design enabling devices of varied platforms to communicate by a common data exchange model via a device management controller. The end product of the framework implementation is a single application managing multiple devices.

7. FUTURE WORK

Intelligent Object Framework is a platform independent framework allowing different devices to be controlled and monitored. Currently the model supports only the basic implementation of the services and protocol allowing functions to be downloaded from an intelligent object device and stored to a database. The framework also allows a user to load the functions of a particular device and execute them.

The framework can be further extended by implementing a secure protocol with token based authentication or equivalent. This would add an overhead on the UDP packet, although it would provide a safe communication channel. The protocol definition can further be improved by implementation of TCP protocol suite for critical devices that need a connection oriented communication to be established. This would provide a logical connection that negotiates before sending data to one another. That addition to the current solution would provide a reliable communication between Intelligent Objects and Intelligent Management Services.

XML based function exchange model would provide a more descriptive function definition that could be integrated as Web Services. This can in fact be defined using the standard Web Service Definition Language. The device function access can be made available over a range of different networks and available to different applications.

The research provided the opportunity to utilize and combine different skill sets ranging from systems design, programming, implementation, database design, embedded device programming and deployment of multiple components on multiple platforms. The skills and knowledge gained throughout the research and development far outweighed the complexity and challenges posed by the project.

ACKNOWLEDGEMENTS

Sasa Savic would like to thank Victoria University for offering the Research Training Scheme (RTS) for his PhD study.

REFERENCES

[1] ManageEngine. (2010). "A beginner's guide to SNMP." http://www.snmplink.org/snmparticles/abeginnersguide, retrieved on 2 May 2010.

[2] WTCS.ORG (2013) "An Introduction to SNMP."http://www.wtcs.org/snmp4tpc/snmp.htm, viewed on 10 November 2013.

[3] Harrington,D. and Wijnen B. (2002). "An Architecture for Describing SNMP Management Frameworks." RFC3411: pp. 34-40.

[4] Patrick (2006) "xAP an Open Standard for Home",viewed on 10 November 2013, http://www.xapautomation.org/index.php?title=Protocol_definition.

[5] XAP (2008). "Home Automation Protocol."http://www.xapautomation.org, viewed on 10 November 2013.

[6] Smith, J. R. (2005) “X-10 Home Automation. Programming the PIC Microcontroller with MBASIC”, Burlington, Newnes: 453-486.

[7] Parralax (2002) “MBASIC for PIC Microcontrollers”, M. Basic: pp 203-207.

[8] MODBUS (2010) "MODBUS Datasheet", http://www.blackoakeng.com/ds_modbuster.htm, retrieved on 2 May 2010.

[9] Huitsing, P., Chandia, R. et al. (2008). "Attack Taxonomies for the Modbus Protocols." International Journal of Critical Infrastructure Protection 1: 37-44.

[10] Sriskanthan, N., Tan, F. and Karande, A. (2002). "Bluetooth based home automation system." Microprocessors and Microsystems 26(6): 281-289.

[11] Committee, B. (1999). Profiles of the Bluetooth System. V1.0B.

[12] Savic, S. and Shi, H. (2011), "An Intelligent Object Framework for Smart Living", Procedia Computer Science 5 (2011) 386–393.

[13] IETF (1980). User Datagram Protocol, RFC 768.

[14] Savic, S. (2010) " Intelligent Object Framework", thesis for Master of Science in Computer Science, School of Engineering and Science, Victoria University, June 2010, 138 pages.

Authors

Sasa Savicis a PhD student at Victoria University in Melbourne, Australia. He obtained Advanced Diploma of Computer Systems Engineering in 2002 from Royal Melbourne Institute of Technology (RMIT). He received Bachelor of Science in Computer Science and Master of Science in Computer Science from Victoria University in 2006and 2010, respectively. He is currently working at Telstra, the largest Australian telecommunications company, as a senior software engineer.

Dr. Hao Shi is an Associate Professor in College of Engineering and Science, Victoria University, Melbourne, Australia. She completed her Bachelor of Engineering degree at Shanghai Jiao Tong University, China and obtained her PhD in the area of Computer Engineering at University of Wollongong. She has been actively engaged in R&D and external consultancy activities. Her research interests include p2p Network, Location-Based Services, Web Services, Computer/Robotics Vision, Visual Communications, Internet and Wireless Technologies.