FUNCTIONAL REQUIREMENTS OF INTELLIGENT OBJECT FRAMEWORK

Sasa Savicand Hao Shi

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

ABSTRACT

Intelligent Object Framework (IOF) is a new communication standard over a wireless network supporting existing multiple sets of architectural solutions. The Framework consists of a framework design that enables devices of different platforms to communicate by a common data exchange model via a device management controller. This paper provides a descriptive analysis of functional requirements for the IOF. The purpose of the proposed system is to provide a platform independent device (Intelligent Object) management by utilization of set components. The functional requirements focus on deriving primary functionality of server and client applications by description of required inputs, behaviours and outputs.

KEYWORDS

Discovery, Functional Requirements, Intelligent Object Framework, IOF, Wireless Applications

1.WIRELESS APPLICATIONS

Celine Graham stated that the use of Wi-Fi connectivity in non-PC based devices such as MP3 players, dual mode cellular and Wi-Fi VoIP phones, video games, printers, smart phones, PDA's, tablets and televisions is rapidly growing [1]. It has been reported by the In-Sat and Wi-Fi Alliance [1,2] that 56 million cellular Wi-Fi phones had been shipped (up ~52%), around 48 million consumer electronics devices (consoles, digital televisions, set-top boxes, printers) units shipped which is up 51% and staggering 71 million Portable Consumer Electronic Devices that account for hand-held games, cameras, portable music player units shipped (up 3%) only in 2008 [3]. The advancements and modularity of the Wi-Fi modules have increased productivity and sales of electronics devices drastically.

The use of wireless applications, however, stretches even further than conventional devices and has been implemented in varieties of different systems. One of typical applications outside the device spectrum has been seen in home automation systems. One particular system is Ambient Intelligence within a Home Environment [4] which aims to implement a collaborative system of wireless proportions to accommodate many devices to talk to a central control unit. The system has been implemented as an automated ambient home solution exhibiting devices such as the television, washing machine, heating system to be accessed wirelessly. The technology has been utilized accordingly to its specifications to a great advantage although the initial architecture does limit the user to be using only a certain set of appliances in accordance to the system. This exhibits future integration issues that are limited by the design.

WiiKey Smartphone application [6] is another interesting application that has been researched by applying wireless technology. The research is focused around smart objects that act as wireless

DOI :10.5121/ijscai.2013.2601

devices, potentially controlled wirelessly by a controller application. The concept of WiiKey has been thought out quite well but the fundamental framework is flawed. The author of WiiKey has not thought out the framework well enough in order to incorporate a standard set of protocols used for interconnecting these devices with a common application and providing the tools for future integrations with other devices or systems.

Certain methodologies need to be considered when implementing wireless solutions. There are also a number of standards that need to be followed in order to fully utilize capabilities of wireless technologies. Depending on the type of system design and application, environmental parameters and range need to be considered.Being an excellent wireless technology, despite having higher power consumption than other wireless technologies, when integrated into devices such as Smartphone or similar mobile phone devices it turns the device into a serious piece of equipment readily available to be integrated in a number of environments. Having wireless technology embedded in a device, with the right software readily available, it could potentially control other devices wirelessly.

An example of Wi-Fi application being integrated into a real time environment is iPhone's Augmented Reality game Drone [6]. It is a perfect example of good engineering practice utilizing multimedia streaming and device control over a Wi-Fi network. IPhone's AR.Drone attempts to bring the reality of gaming into home by creating a real-time environment and device control. A quadricopter connects to a home wireless network. An iPhone application is launched which connects to the AR.Drone.Figure 1 shows the integration between the AR.Drone and iPhone running over a Wi-Fi router. The simplicity in applications lies in the protocol design. This renders devices to be usable across many different networks ranging from wireless, to Ethernet and 3G.

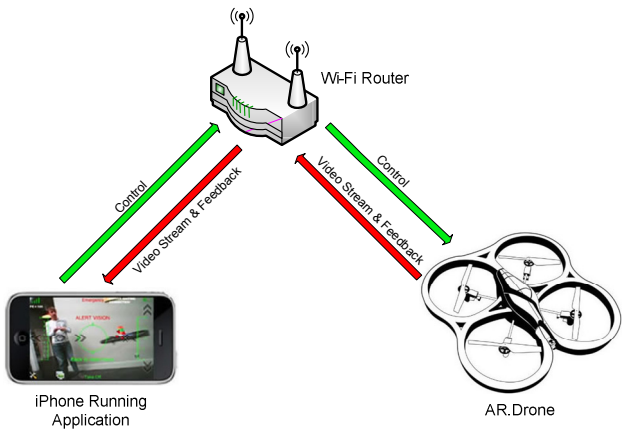

Figure 1. Augmented Reality Drone integrated with iPhone over Wi-Fi [6]

Once the communications have been established, the AR.Drone streams live video feed from two of its on-board cameras. The video stream is routed off by the router to the iPhone. The iPhone's application captures the live-stream and renders the video feed.Via the same communications channel, although a different communication port, iPhone is able to send commands that control the movement of the AR.Drone. This device infrastructure defines the necessary components needed to show the portability, efficiency and effectiveness of Smartphone application with integrated Wi-Fi modules being capable of wireless communication and task executions with other devices run on the same Wi-Fi standard.

2. DEVICE DISCOVERY

Functional requirements of the IOF consist of device discovery, server discovery, device function exchange, device execution process and asynchronous device status notification.

Primary basis of device discovery requirement states that a device must conform to a protocol standard. Within the protocol standard lies the discovery method that is, by definition, either a manual or an automated process. The rudimentary principle of device discovery is determined by the network to which a device is connected to. In order for the device to be discovered, the Intelligent Object Framework Management Application, *the server that is hosting the application*, and the Intelligent Object Device must be on the same network [7, 8]. Secondary requirement is that the device must be listening on a UDP protocol, which is detailed in the Protocol Layer of the IOF [9]. Based on the set network parameters, the developed Management Studio [7, 8] can initiate discovery process outlined as shown in Figure 2.

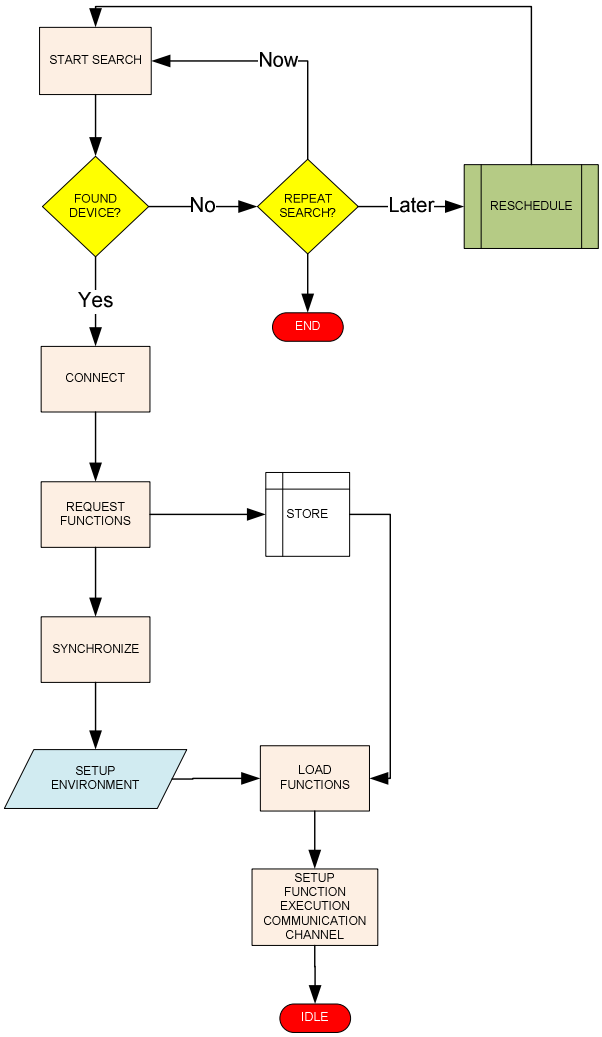

Figure 2. Device Discovery Diagram

As the search process commences, the Management Studiohas three options based on the set result. If the search result returns no devices, the search process can either repeat the operation based on the number of attempts; re-schedule the search to occur at a later time which is determined by a random function or end the process allowing a manual search. In case of successful device discovery, the next process that follows is a successful connection to the device. Once a successful connection has been initialized, the Management services request device functions, upon which the received function set is stored to a database. The process that describes the data processing layer is detailed in the Data Layer of the IOF [9].Once the Management Studio has successfully retrieved Intelligent Object functions, it synchronizes, sets up the operation environment and loads all available device functions. At this point a function execution communication channel is initiated and the user is able to control the device. The communication channel is spawned as a separate thread, away from the main service thread, which allows asynchronous data retrieval and communication to the intelligent object device and vice versa.

The next section describes server discovery mechanism. It is a similar process to device discovery although there are minor differences that the device must conform in order to gain successful authentication to the Management Studio.

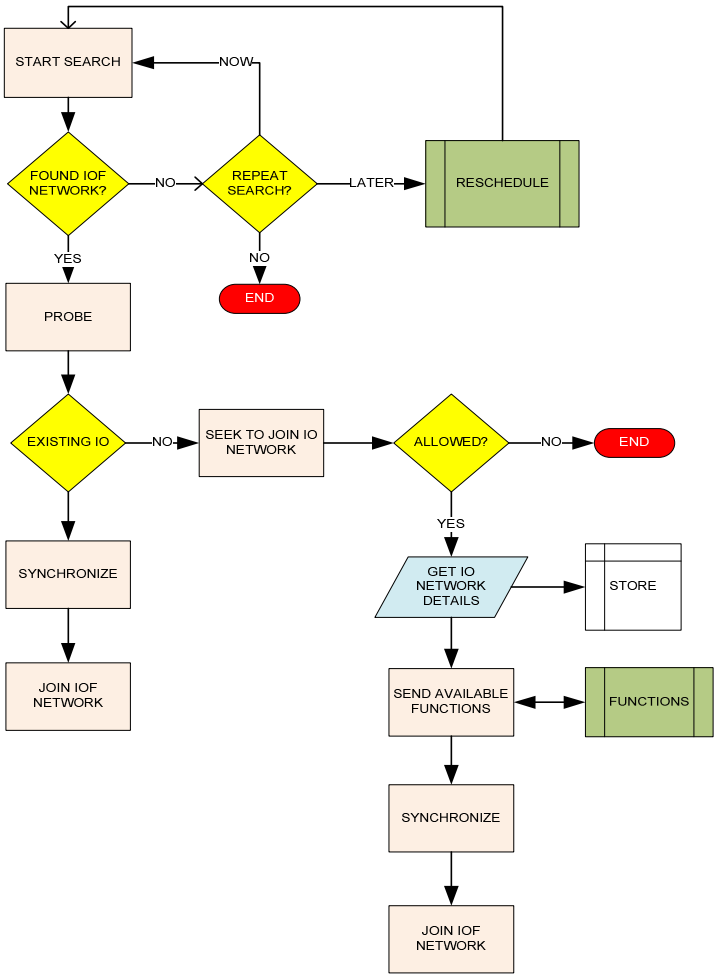

Figure 3. Server Discovery Diagram

3. SERVER DISCOVERY

The server discovery process works vice-versa to the device discovery process. In this instance the device seeks authentication from the management server. For the device to be connected to the Management Studio server, the device must also conform to a set of protocol rules [9]. In case of a successful server discovery, the device seeks an authorization to join the server. If the authorization is granted, the device is subjected to two possible operations as illustrated in Figure 3. One is to join the network as an existing device and synchronize with server, the other is to provide its set of functionalities. The process is no different to the previously described operation in Figure 2, with the difference being the server seeks authorization for access from the services. In a uniformly distributed IOF system, the process is repeated for each new device. If a device is part of the network, the authentication and function forwarding does not occur, as descriptor flags are set within the database.

4. DEVICE FUNCTION EXCHANGE

The device function exchange model describes the communication steps required to achieve function exchange from the Intelligent Object device to the services. There are a number of steps services and device need to go through for successful exchange. The next two sections show the request and response transmission from a higher level.

4.1. Function Request (Server to Device)

Following successful authentication to the IOF services, the services make a request to the Intelligent Object device to pass down the functions. The model is relatively a simple description of the request packet header [10] which holds the trigger value required by the framework to initiate function exchange. Current model is designed to request one function at a time and for each function to request its set of parameters. This model needs to be followed as UDP communication is not a reliable transport protocol, although fits appropriately within the IOF model. The UDP protocol is later described in more thorough detail and how it fits in with the framework. Figure 4 describes the high level function request model. The device initially sends a device descriptor packet, described later, followed by device data and the function data which holds the function information. Once the data is received, device initiates function counter which decrements as the functions are received.

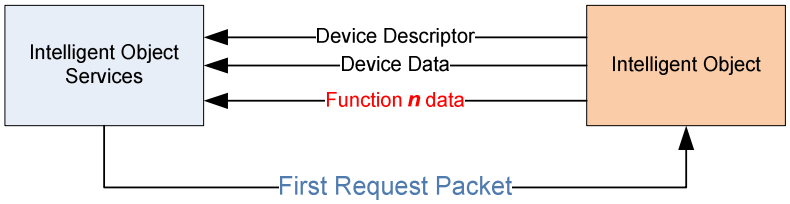

Figure 4. Server Discovery Diagram

Figure 5 further describes how the device handles the initiation of function request, and the iteration process required for each function to be received.

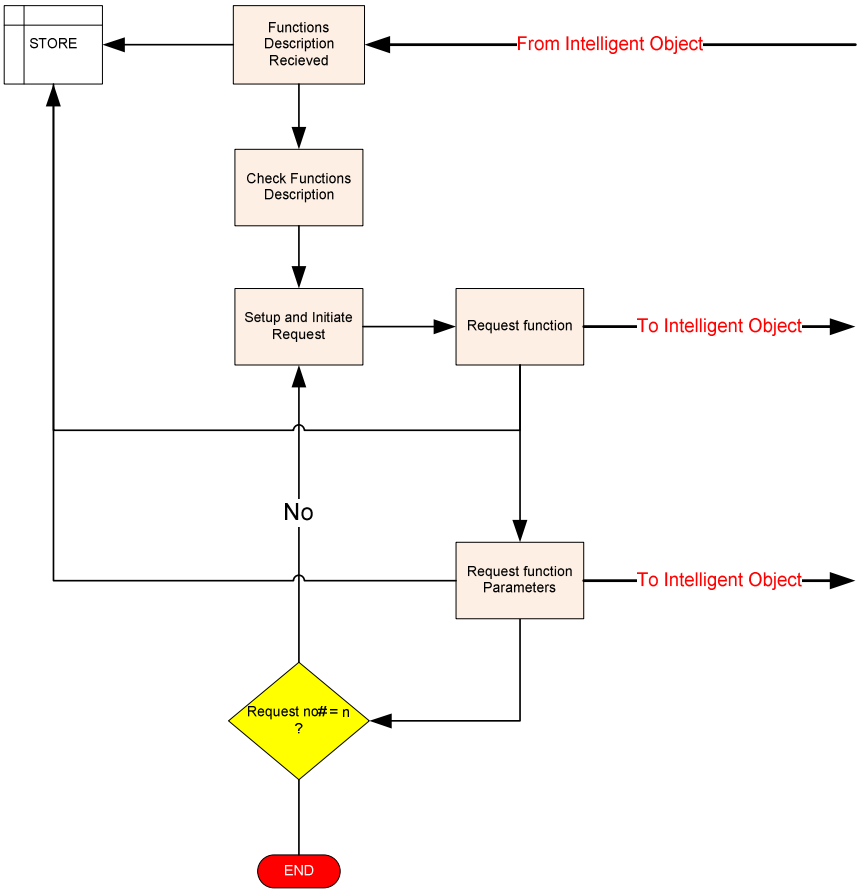

Figure 5. Server Discovery Diagram

As the data is received by the device, function counter sets up the function call stack, which is an enumerated process based on the number of iterations required to achieve successful reception of all functions and their parameters. There is a slight overhead during the function exchange communication due to the UDP packet size limitation, so the process needs to make several trips to complete the exchange.

4.2. Function Request Response (Device to Server)

Function request response model fits in tightly with the function request model as the two processes reciprocate until the device has sent all of its functions down to the services. Being a tightly coupled model mimics a typical server request, device response communication enabling reliable data exchange. Once the services receive the number of functions a particular device has, it automatically knows how many times it needs to iterate its function request process. Once the device receives the request, it sends the next function down followed by all the function parameters which bind to that function. Figure 6 loosely shows this process, and a more descriptive explanation is presented in [9, 10].

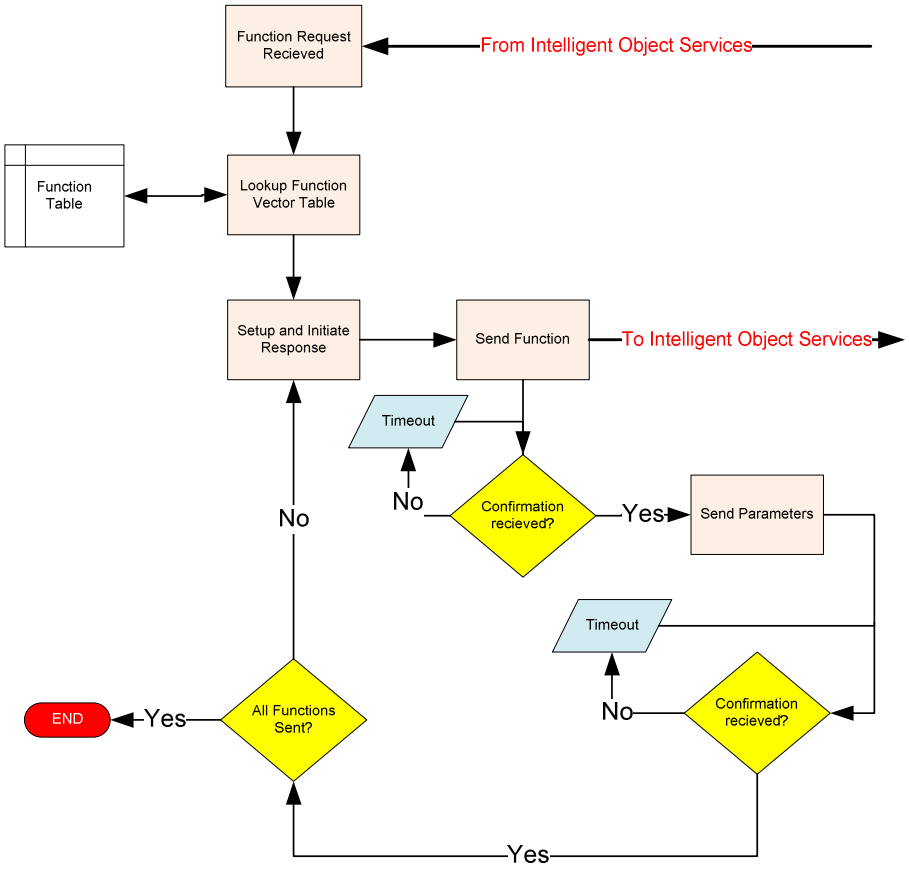

Figure 6. Server Discovery Diagram

5. DEVICE EXECUTION PROCESS

The primary and most important feature of the IOF is the ability for a user to execute arbitrary commands and functions that reside on the Intelligent Object. The device itself is preloaded with a set of available functions that are passed to the IOF services, as described in the previous sections. The device execution process conforms to the framework as it allows the execution to take place. An example of the execution process is turning the temperature up or down on an Intelligent Object enabled air conditioning system within the household environment. The user requests a function to be executed. This process takes place initially on the Intelligent Object's Management Studio, which processes the user request and creates the request packet. The request packet is then sent to the device and finally executed. The high level process is further described in the next sections.

5.1. Execution Request (Server to Device)

Initially once the application is started, the services are placed in their own thread. This maintains application and processing separation allowing asynchronous (multithreaded) application environment. The second portion of the process is to fetch the intelligent object data from the database and store it in a collection. The collection is made up of intelligent objects stored in class structures. When the user is accessing intelligent device data, it is accessing it at runtime from the collection.

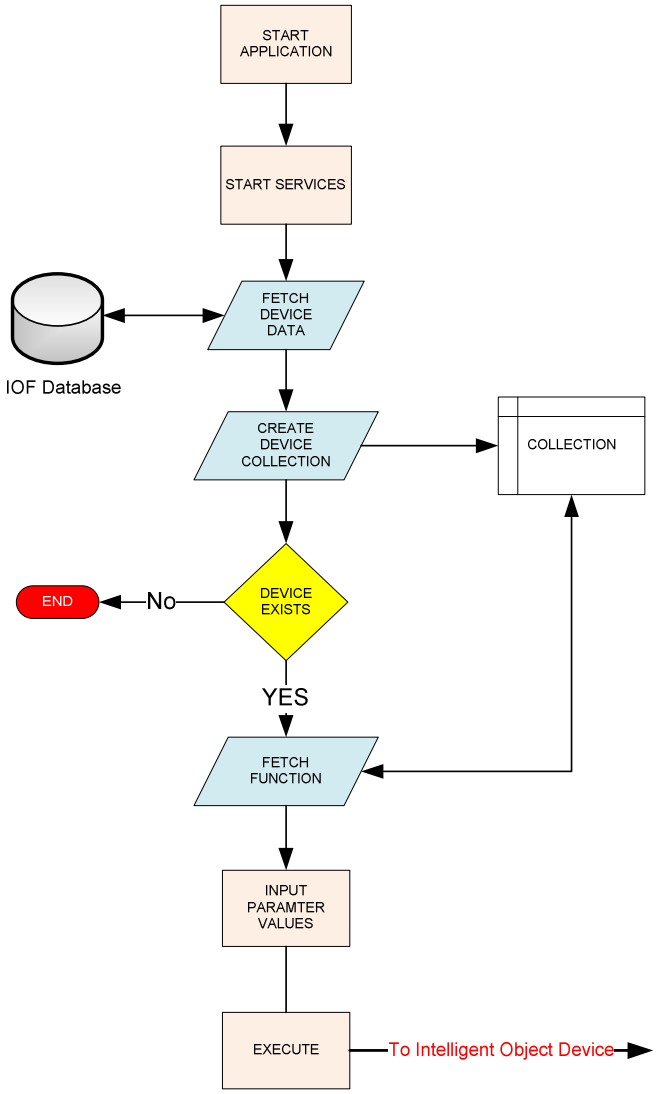

Figure 7. Function Execution Request

Upon a function execution request, the collection is checked if the device is online; the function data is then checked as shown in Figure 7. The user must also enter some parametric data for the device to execute. Once the user clicks the execute button, the data packet is constructed and sent to the Intelligent Object. The services then wait for the reply to determine if the function has been executed on the device. This process is further explained in the next section.

5.2. Execution Response (Device to Server)

Execution response is a process that conforms to protocol standards, once extracting the function name and the function parameters; it looks up the function matrix. Function matrix is a collection of device functions. The functions are native to the device. An air conditioning system might have a function matrix which holds different types of functions such as to turn on or off the air conditioner, or to set the temperature or the timer. This process is illustrated in Figure 8. Once the function has been found, the next process is to check the parameters (if any). This will set up the function to execute in a particular way, or to execute particular bit of data supplied by the user.

The process is smart enough to determine if it is a valid request. If by chance the function does not exist, or a particular parameter has not been defined correctly, the services are notified, thus notifying the user with an error message. Once the function has been successfully executed, the user is once again notified.

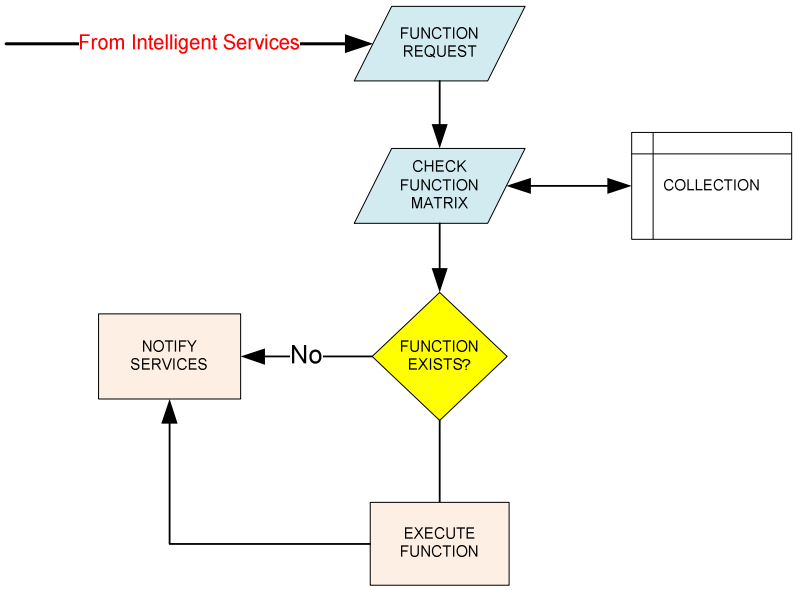

Figure 8. Function Execution Request

6. ASYNCHRONOUS DEVICE STATUS NOTIFICATION

Status notification and device callback functionality allows application to make decisions based on returned status codes. The application is also informed by the device at what state the device is currently at. With that information, it is able to dictate what the next process of execution should or should not be.The IOF makes this possible through the protocol definition allowing status codes to be sent depending on the type of operation the user or application has executed. The device is able to process requests and once the request has been completed, status codes are generated. There are two types of status codes: user status and application status. The user status is purely for user information and does not have impact on the response of the application. The application specific status replies are intended only for processing of device data.

7. CONCLUSIONS

This paper presents analysis of functional requirements for the Intelligent Object Framework which aims to provide a platform independent device management by utilization of existing components. The focus is on deriving primary functionality of server and client applications through device discovery, server discovery, device function exchange, device execution process and asynchronous device status notification.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] Zigbee Alliance (2006) "Zigbee Vision for the Home: Zigbee Wireless Home Automation", Zigbee Whitepaper
[2] Zigbee Alliance (2007) "Zigbee Enables Smart Buildings of the Future Today", Zigbee Whitepaper
[3] Graham C. (2009) "Conformity", Engineer's Reference Guide 2009; 14:178-181.
[4] Bieliková M. and Krajcovic T. (2001) "Ambient Intelligence within Home Environment.", http://www.ercim.eu/publication/Ercim_News/enw47/bielikova.html, ERCIM News No.47, October 2001.
[5] Vo N., Shi H. and Szajman J. (2008) WiiKey: An Innovative Smartphone Based Wi-Fi Application, Proceedings of International Multi-Symposiums on Computer and Computational Sciences, Shanghai, China, pp. 91-97.
[6] Parrot (2013) "AR.Drone Academy", http://ardrone2.parrot.com/ , Viewed on 8 November 2013.
[7] Savic S. and Shi, H. (2011), "An Intelligent Object Framework for Smart Living", Procedia Computer Science 5 (2011) 386–393.
[8] 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.
[9] Savic S. and Shi, H. (2013), "Detailed Design of Intelligent Object Framework", submitted tothe International Journal of Computer Networks & Communications.
[10] Savic S. and Shi, H. (2013), "Testing and Deployment of Intelligent Object Framework", submitted tothe International Journal of Next - Generation Network.

Authors

Sasa Savicis a part-time 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), Australia. He received Bachelor of 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, Robotics Vision, Visual Communications, Internet and Wireless Technologies.