



[Date]

OPC UA Implementation on Android and ROS based devices

[Document subtitle]

Niranjan Shahane
[COMPANY NAME]

Declaration

We hereby declare that the project entitled **“Implementation of OPC-UA on Android and ROS based devices”** written and submitted by us to Otto von Guericke Universität Magdeburg is an authentic record of our genuine work done under the guidance of **apl. Prof. Dr.-Ing. habil. A. Lüder** and **Dr.-Ing. Christoph Steup**.


Group members:

1. Niranjan Shahane (224652) _____
2. Saurabh Pandey (224240) _____
3. Nishad Pawaskar (225992) _____

Date: **29 January 2020**

Place: **Magdeburg**

Acknowledgement

This masters project was supported by Otto von Guericke Universität. We would like to pay our sincere gratitude to **apl. Prof. Dr.-Ing. habil. A. Lüder** and **Dr.-Ing. Christoph Steup** for providing us with their valuable insights and expertise. Without their precious support it, it would be not possible to conduct this project. 

Last but not the least, the dedicated efforts and cooperation between all the group members, has crucially helped in concluding the project.

Table of Contents

1	Introduction	7
1.1	Background	7
1.2	Objectives and scope	7
1.3	Sources.....	7
1.4	Structure of work	8
2	OPC- Unified Architecture	9
2.1	What is OPC-UA	9
2.2	Structure of OPC-UA	9
2.2.1	Functional equivalence	9
2.2.2	Platform Independence.....	10
2.2.3	Information Modelling	10
2.2.4	Extensibility	10
2.2.5	Security	10
2.3	OPC UA Capabilities	11
2.3.1	Address Space	11
2.4	Concepts of Address Space	11
2.4.1	Object Model	11
2.4.2	Node Model	12
2.4.3	Attributes	12
2.4.4	Variables.....	12
2.4.5	Methods.....	12
2.4.6	View	13
2.4.7	Data Types.....	13
2.4.8	Interaction of the eight node classes	13
2.5	OPC UA services	14
3	Android Operating System.....	15
3.1	Why Android OS?	15
3.2	Perspective of Android with OPC UA	15
3.3	Android app components.....	15
3.3.1	Activities.....	16
3.3.2	Services	16
3.3.3	Broadcast Receivers	16
3.3.4	Content Providers	16
4	OPC environment Set up in Android	17

4.1	Java JDK 8.....	17
4.1.1	Modifications in Environment Variable	17
4.2	Maven	18
4.3	UPC UA Java Stack.....	19
4.4	SLF4J.....	19
4.5	Spongy Castle	20
4.6	OPC UA Simulation Servers	20
4.7	Android Studio	21
4.8	Emulator	22
5	Android Client Application	23
5.1	Sequential Procedure.....	23
6	Robot Operating System (ROS)	25
6.1	What is ROS?.....	25
6.2	ROS and Industry 4.0.....	25
6.3	ROS Core Concepts.....	25
6.3.1	ROS Node	25
6.3.2	ROS Topics.....	26
6.3.3	ROS Messages	26
6.3.4	ROS Services	26
6.3.5	ROS_launch	26
6.3.6	ROS_Run	26
6.3.7	ROS_cd	26
6.3.8	ROS_core.....	26
7	Ubuntu Virtual Box setup for ROS.....	27
7.1	Ubuntu Virtual Box.....	27
7.2	Setting up the virtual box and installing ubuntu 16.04	27
7.3	Employed Network types	27
7.4	ROS installation	28
8	OPC-UA server realization with ROS	29
8.1	Cloning ROS based OPC-UA server	29
8.2	Initializing ROS on OPC- UA server	29
9	Final Implementation (Server & Client).....	31
9.1	Procedure for Implementation	31
9.1.1	Browsing the Address Space:	31
9.1.2	Calling the Method:.....	31

9.1.3	Setting Input Arguments:	32
9.2	Designed UI	32
9.3	Motion of Turtlebot	33
10	Conclusion and Recommendation	34
11	Abbreviations	35
12	References	36

Table of Figures

Figure 2-1 (Multi-Layer Architecture of OPC-UA).....	10
Figure 2-2 (Object Model OPC-UA)	11
Figure 2-3 (Node Model).....	12
Figure 2-4 (Node classes interaction).....	13
Figure 4-1 (Java path in system).....	Error! Bookmark not defined.
Figure 4-2 (Apache Maven Version).....	18
Figure 4-3 (UPC UA Java Stack)	19
Figure 4-4 (Maven package build success)	19
Figure 4-5 (SLF4J version)	20
Figure 4-6 (Spongy Castle)	20
Figure 4-7 (Prosys OPC-UA).....	21
Figure 4-8(Prosys OPC-UA simulation server)	21
Figure 5-1 (Basic UI)	24
Figure 8-1 (Turtlebot).....	30
Figure 9-1 (Address Space Browsing)	31
Figure 9-2 (Method Calling)	31
Figure 9-3 (Input Argument Setting)	32
Figure 9-4 (Designed UI)	32

List of Tables

Table 2-1 (OPC-UA service sets and their use cases)	14
---	----

1 Introduction

This unit gives a primitive overview of the intended task and provides the necessary initial information for deciphering the solution.

1.1 Background

In recent years of vast industrial growth and technological developments, a significant increase in **OPC-UA** (Open Platform Communications Unified Architecture) applications for service-oriented architecture with a platform independent framework has been observed. OPC-UA has made a substantial contribution to integrate the wide spectrum of hardware and software systems accompanied by data security, scalability and extensibility. In particular, the platform independence provides adaptability to different operating systems. **Android**, being the most popular operating system along with a wide variety of compatible devices (Smartphones, Watches, Smart TV's etc) comes out to be an optimized OS for implementation of OPC-UA.

The entire idea of implementing OPC-UA on handheld devices is to incorporate these systems into an industrial landscape. A fully integrated digital factory, incorporating a digital monitoring system is what the present industrial environment (**Industry 4.0**) requires. An entirely automated factory depicts the involvement of smart robots and industrial tools. This can be sufficed by integration of a software platform that has the capabilities for perception, manipulation and planning. This is where **ROS** (Robot Operating System) emerges out to be a significant infrastructure for factory automation.

1.2 Objectives and scope

The goal of the **master's project** is to investigate and evaluate the compatibility of an OPC interface architecture for Robot based control systems. Moreover, to validate how OPC-UA could be implemented on Android Operating system and create a generic concept for the client. Requirements for creating the application are:

- Connectivity to an OPC-UA server
- Navigation in Address space
- Subscribing and writing node values
- ROS Server (Topics, messages, services nodes)
- Alarms and events interaction.

1.3 Sources

The main sources for the foundation alongside the literature were, the **OPC-UA platform websites** and research papers. The android development websites are updated more rapidly because of the constantly evolving user interface. Whereas, ROS provides us with an extensive number of packages and repositories according to the user's application.

Due to lack of references and citations for OPC-UA and android compatibility, a great deal of issues had to be resolved and verified by experimentation with trial runs. Testing the client application played a crucial role in gaining the experience and applying it onto our upcoming decisions.

1.4 Structure of work

The forthcoming units summarize the relevant information regarding the operating systems involved in this project. The chapter-2 deals with the fundamentals & features of OPC-UA framework. The chapter 3, 4 & 5 represent the basics of Android and how it can be set up as an OPC-UA client.

The basic description of ROS and its features is presented in chapter-6. Chapter-7 & 8 comprised of the prerequisites for initializing ROS environment and how it can be realized with OPC-UA. The connection establishment between all three software platforms, is presented in chapter-9. The last chapter involve some suggestions for future development possibilities and the conclusion of the project.

In addition to the sequential steps, the required installation links and the respective citations, an additional reference document is also provided for error handling and their resolution. The prerequisites and the system configurations mentioned in the reference document, play a significant role to expedite the process for installation and connection. The validation of entire connection process between the Android based client and ROS based server can be verified by the demonstration video included in the documentation.

2 OPC- Unified Architecture

This unit will provide a brief introduction to OPC-Unified architecture, its capabilities for versatile interoperability, requirements and its scope as an industrial communication framework [1].

2.1 What is OPC-UA

OPC is the interoperability standard for secure and reliable exchange of data in an industrial automation space. It ensures a seamless information flow across a wide array of devices from multiple vendors with utmost security and reliability. OPC classic, the predecessor of OPC-UA was originally developed for Microsoft's distributed component object model. So, its applications were limited only for Windows OS. In order to standardize the data exchange between different software components from different manufacturers, a unified structure was required.

Hence, OPC-UA a platform independent specification was developed. OPC-UA overcame all the shortcomings of the prevailing OPC classic and provided a better security, extensibility and comprehensive information modelling.

2.2 Structure of OPC-UA

OPC-UA was introduced in 2006, is a service-oriented architecture and unifies all the features of OPC classic and into an extensible framework [2].

- Functional equivalence
- Platform independence
- Information modelling
- Extensibility
- Security

2.2.1 Functional equivalence

Overlooking the shortcoming of the OPC classic, it can still be considered a successful communication framework. Yet an enhanced architecture was demanded for elimination of these limitations. An advanced version of OPC classic was developed which equalized not only the functionality, but also brought something more applicable for the user. In addition to the OPC classic specification, OPC-UA has a potential to provide something more:

- **Discovery:** Find and detect available OPC servers in local systems.
- **Address space:** Representation of all simple and complex data in a hierarchical structure for OPC clients.
- **On demand:** Read and write information based on access permission.
- **Subscriptions:** Monitor the information and provide relevant data according to client's request.
- **Events:** Notify important information based on the requests.
- **Methods:** Taking actions according to the procedure required or selected by the client.

2.2.2 Platform Independence

An equally significant aspect of OPC-UA is versatility towards the diverse hardware and software systems currently available. OPC-UA provides the necessary infrastructure and interoperability for an efficient and reliable communication between an enterprise and machines.

2.2.3 Information Modelling

OPC-UA uses the object-oriented capabilities to convert most of the multilevel complex data into information. It provides the fundamental building blocks necessary to represent the information model. OPC-UA defines all the fundamentals to the information models:

- Look up mechanisms
- Read and write operations
- Method execution
- Notification of events and data

2.2.4 Extensibility

The multi-layered architecture of OPC-UA provides innovative technologies and methodologies, for instance security algorithms, network protocols, services or encoding standards.

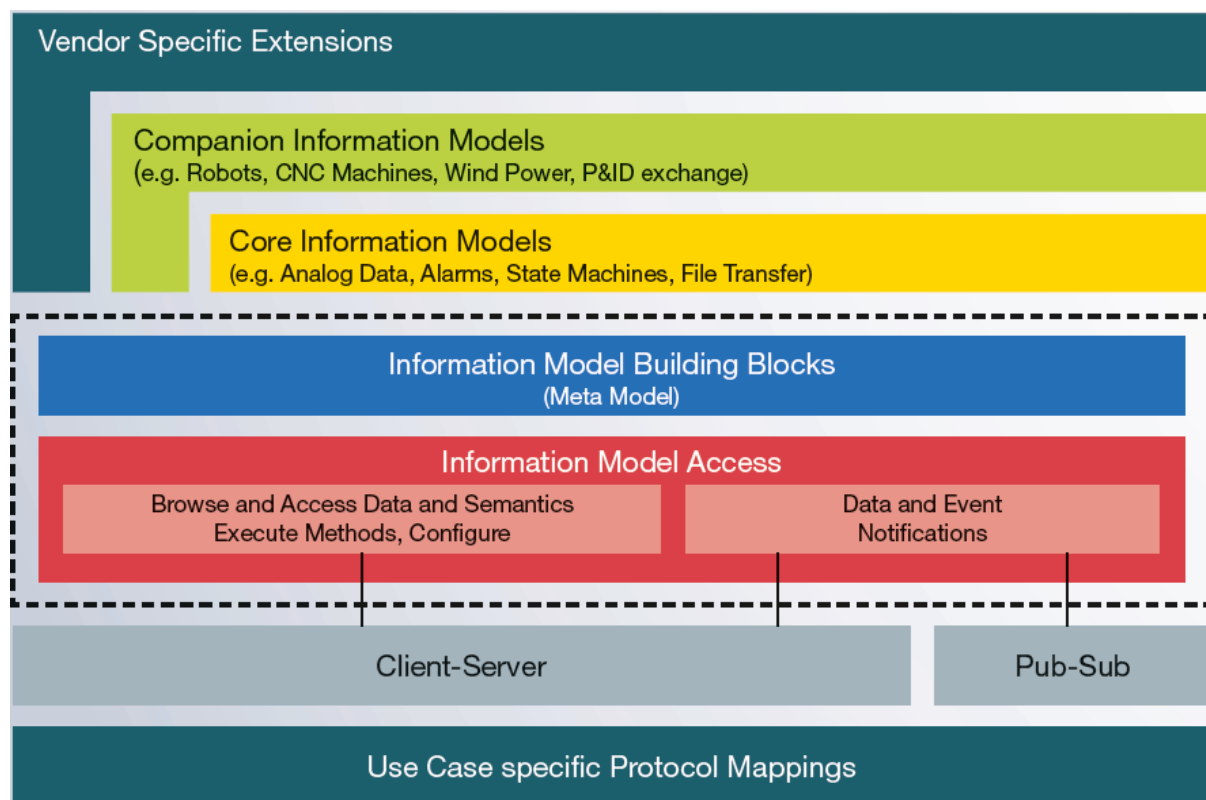


Figure 2-1 (Multi-Layer Architecture of OPC-UA)

2.2.5 Security

One of the most essential components of any technology is to provide an infallible security protocol. OPC-UA is firewall friendly while addressing security concerns by proving suit of controls:

- **Transport:** Ultra-fast OPC binary transport is used for handling numerous protocols.

- **Session Encryption:** Secure transaction of messages over various encryption levels.
- **Message Signing:** The origin and integrity of messages can be verified by the recipient.
- **Sequenced Packets:** Exposure to message replay attacks is eliminated with sequencing.
- **Authentication:** Each UA client and server are identified by X509 certificate, over which applications and systems are permitted to connect with each other.
- **User control:** An authentication is required (certificates, credentials etc) for accessing the applications in the address space.
- **Auditing:** Activities of user and systems are logged for providing it to audit trial.

2.3 OPC UA Capabilities

This section represents how an Address Space is utilized for acquisition of information and how it can be accessed from reliable external sources.

2.3.1 Address Space

OPC UA server provides visibility of information to the clients my means of an **address space**. The address space is composed of multiple **Nodes** representing the relevant information in the form of **References**. These references define the hierarchies in the address space and the intercommunication process involved in multiple nodes.

The real entities such as devices, servers and communication systems represent **objects**, which defines the scheme for organization of the information within the address space.

Certain common Nodes interacting in between the OPC UA servers & Address space act as entry points to address space while their access.

2.4 Concepts of Address Space

Accessibility for address space relies entirely upon its Model-based architecture and respective representation of objects to the clients.

2.4.1 Object Model

Representation of objects to clients in a standardized manner defines the basic objective of an **Object Model**. An object-oriented approach can enhance the data & feature representation, but a simple address space model can also be employed for depiction of objects and variables [1].

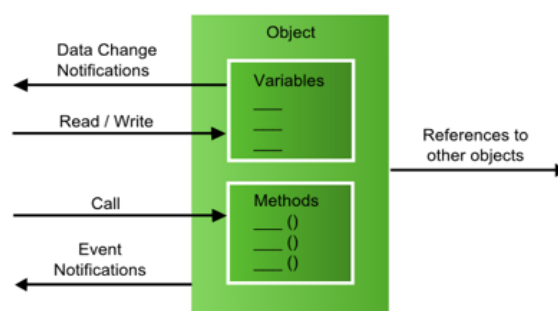


Figure 2-2 (Object Model OPC-UA)

An object model comprises of components for reading or writing a variable value or calling a method or receiving an event. Exploration of these features can be executed by UA services.

2.4.2 Node Model

Several co-related objects work simultaneously to represent a common entity known as Node Model. It describes the objects and its components by means of their attributes and references [1].

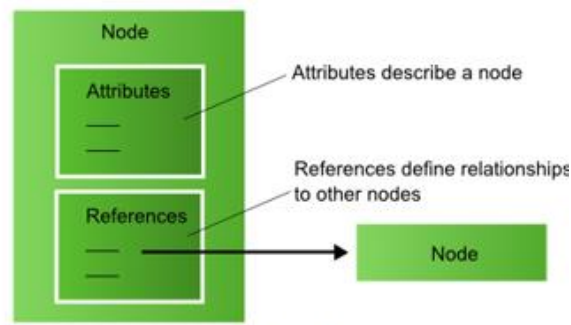


Figure 2-3 (Node Model)

2.4.2.1 Node Classes

It defines attributes and references for different nodes. It includes 8 node classes and neither clients nor servers are allowed to define additional node classes. UA Node classes are in serialim:

1. Object
2. Variables
3. Method
4. View
5. Object Type
6. Variable Type
7. Reference Type
8. Data type

2.4.3 Attributes

The data elements that describe nodes. The accessibility to attributes can be executed by the clients using Read, Write, Query or Subscription. The attributes are elementary terms which are known by the clients and are not directly visible in the address space.

2.4.4 Variables

They represent values and basically include properties and data variables. The data represented by each individual depends upon their individual functions. It can be used to represent the distances, temperatures or position (in our model). It is basically responsible for displaying all the varying aspects in the address space. The simple readable data available at the client side that depicts the general details (for e.g. orientation, direction, status) come under variables

2.4.5 Methods

These are callable operations that clients can access in the address space through Objects and acquire desired results. The list of input parameters necessary for output actuation or information retrieval. Unlike Variables, Methods are incorporated in the Objects and can be only be called by the client. Any alterations via the client interface is not possible for Methods. As, Methods define set of commands which are more complex, hence any intended changes should be executed at server level. For

example, the directions chosen by the operator in our project for the robot motion is what that triggers the methods in Objects and provides the desired output.

2.4.6 View

Visibility of certain nodes and references in the address space can be restricted by view. User specific data can be represented in the address space, similar to use case. For example, View can be programmed to monitor the motion of Youbot on the set map or track.

2.4.7 Data Types

The basic entities represented in Nodes such as integers, string, bytes etc. Additional data types can be defined in the server based on the required operations.

2.4.8 Interaction of the eight node classes

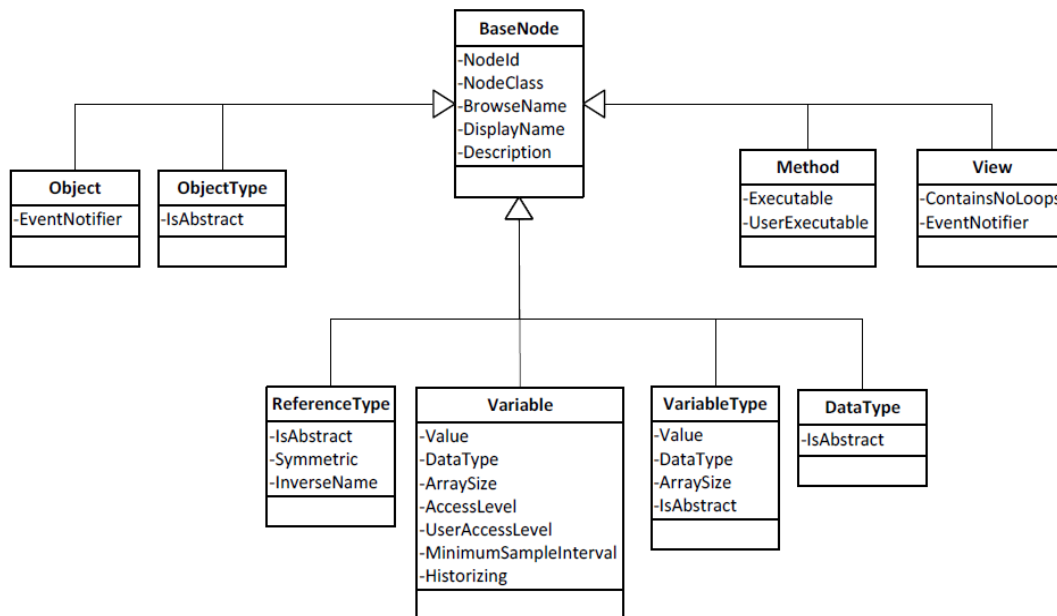


Figure 2-4 (Node classes interaction)

The Base Node in the **figure2-4** represents the common attributes for each node. An equally significant aspect in OPC UA is **NodeID**. As the name suggests, it is used to identify the nodes in the server with a unique ID. It basically comprises of three parts: an **address space**, an **enumerated identifier type** & **identifier** itself. There are four type of identifiers:

1. Numeric
2. String
3. Globally unique identifier
4. Opaque

2.5 OPC UA services

A secure data access and information retrieval is the core segment of the OPC UA client/server communication. Each interaction between the server/client is carried out through the services. For any action initiated by the client, service parameters are sent to the server and the deployed results are reverted to the client. In the case of any failures the server responds with a reason message.

Based on the enormous number of service possibilities, Services are categorized into Service Sets. They depend upon the assigned tasks and their operability.

Use case	Service Set	Services
Find servers	Discovery	FindServers, GetEndpoints, RegisterServer
Manage the connections between clients and servers	SecureChannel	OpenSecureChannel, CloseSecureChannel
	Session	CreateSession, Activate Session, CloseSession, Cancel
Modify the structure of the server Address Space	NodeManagement	AddNodes, AddReferences, DeleteNodes, DeleteReferences
Find information in the Address Space	View	Browse, BrowseNext, TranslateBrowsePathsToNodeIds, RegisterNodes, UnregisterNodes
Find information in a complex Address Space	Query	QueryFirst, QueryNext
Read and write data and metadata, access history data	Attribute	Read, HistoryRead, Write, HistoryUpdate
Calling Methods defined by the server	Method	Call
Subscribe for data changes and Events	MonitoredItem	CreateMonitoredItems, ModifyMonitoredItems, SetMonitoringMode, SetTriggering, DeleteMonitoredItems
	Subscription	CreateSubscription, ModifySubscription, SetPublishingMode, Publish, Republish, TransferSubscription, DeleteSubscriptions

Table 2-1 (OPC-UA service sets and their use cases)

3 Android Operating System

This unit provides a general introduction to the Android OS and its capabilities. The scope of developing a generic application, based on android architecture and testing its compatibility to the OPC UA framework.

3.1 Why Android OS?

Android is an open source mobile operating software based on Ubuntu/Linux and developed by Google. Developed basically for touch screen smart-devices and for implementing a more user-friendly environment.

As mentioned in the previous unit, OPC UA's compatibility feature makes it a reliable communication framework for current automation requirements (**Industry4.0**). Furthermore, Android having an estimate of 80% devices in market share, comes out to be the best OS for OPC UA client development.

Google offers all the relevant tools for development and deployment of applications. The official IDE (Integrated Development Environment) is distributed with an open source licence. Being an **Open Source** software, one can easily download, develop, modify or customize any android application based on their requirements. With an abundance of multiple Android development tools with interactive UI's, makes it easier for an android developer to design an application according to the desired operation.

3.2 Perspective of Android with OPC UA

Android apps are written in java programming language and its development is done by Android SDK (Software Development Kit) tools. It compiles the Java code into APK (Android package) and can be installed into android device. OPC UA framework represents a collection of sensitive information, hence a secure client environment is a must. Android app runs in its own security sandbox and provides a unique user ID. This user ID can be used to permit/deny accessibility to certain third-party applications. Hence, the java script can be developed based on the task requirements and third-party app assist. For example, certain network access protocols would be necessary for an uninterrupted connection with OPC UA server.

3.3 Android app components

Android app is generally based on four components:

1. Activities
2. Services
3. Content providers
4. Broadcast Receivers

Each component has a distinct role, based on their preferred usability. A significant part of app development involves in defining how each component is either created or destroyed and what manipulations are essential in the Manifest file [3].

3.3.1 Activities

As the title implies, activities represent the display for the UI (User Interface). An application is collection of multiple interrelated and independent activities. Therefore, with correct permissions, an app can launch activities from other applications. For example, in order to access the files in the OPC UA server we must launch the P2P Wi-Fi Direct.

Activities includes an additional component namely: **Fragments**, which represents a portion of the interface in an activity. Collection of multiple fragments can be used to develop an activity and hence build the UI. The versatility of these fragments and assist the app's interface for all possible outcomes. Instead of creating multiple activities separately, we can develop a single activity with numerous fragments for all required functionalities.

3.3.2 Services

General purpose entry points for keeping an application running in background. An equally significant role of background task operation is essential for performance of the long running or remote processes. It does not provide a UI but plays a crucial role for the main application to compile efficiently. For example, in the current project, the primary objective is Robot motion, wherein the services play a role to fetch the necessary data from the OPC server without blocking the user interaction. This kind of communication between these activities is known as a Bounded one and shuts down immediately as the client turns off.

3.3.3 Broadcast Receivers

As the name suggests, it enables system to deliver events happening outside the application, so that it could respond to the system wide broadcast announcements. Applications can be programmed to initiate the Broadcast to let other components know about the whereabouts of the system. In order to optimize the function of the running application, one must be notified about the upcoming events that could be a possibility of hinderance. For example, the user must be aware of the credibility of current network, the battery status of the device, triggering retrieval of a relevant data or any other red flags. This will assist in the validating the background services or activities. This will not only support the app execution, but also eliminate the possible security threats.

3.3.4 Content Providers

Accessibility with authority for data from other components of system plays a vital role for managing shared sets of app data. With the help of content providers, other apps can access and change the shared data as long as the permissions are granted for the content providers. To a system, a content provider is an entry point into an app for publishing or manipulating the items in it. With relevant permissions for content providers, our system can start another app's component. For example, in order to set up a successful connection with the OPC server, our app must launch the WiFi services. Rather than doing it separately/manually, the app itself can execute it with content provider access.

4 OPC environment Set up in Android

This unit includes all the essentialities regarding the programs, libraries and steps needed to set the working environment for OPC based Android app development. In general, it is based on OPC Foundation's guide, moreover some modifications have been made according to the project requirement.

4.1 Java JDK 8

OPC UA needs **Apache Maven** and the prerequisite for Maven is **JDK 1.7(or higher)**. So, for initial step, install download Java SE Development Kit 8 (JDK 8) [4]. The setup procedure for OpenJDK involves **Oracle software**, but a specific version can be downloaded based on your system configuration. Do refer to the **Reference Document**, in case of any **Build Failure**.

Java SE Development Kit 8u241		
You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.		
<input type="radio"/> Accept License Agreement <input checked="" type="radio"/> Decline License Agreement		
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.94 MB	jdk-8u241-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.83 MB	jdk-8u241-linux-arm64-vfp-hflt.tar.gz
Linux x86	171.28 MB	jdk-8u241-linux-i586.rpm
Linux x86	186.1 MB	jdk-8u241-linux-i586.tar.gz
Linux x64	170.65 MB	jdk-8u241-linux-x64.rpm
Linux x64	185.53 MB	jdk-8u241-linux-x64.tar.gz
Mac OS X x64	254.06 MB	jdk-8u241-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	133.01 MB	jdk-8u241-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	94.24 MB	jdk-8u241-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	133.8 MB	jdk-8u241-solaris-x64.tar.Z
Solaris x64	92.01 MB	jdk-8u241-solaris-x64.tar.gz
Windows x86	200.86 MB	jdk-8u241-windows-i586.exe
Windows x64	210.92 MB	jdk-8u241-windows-x64.exe

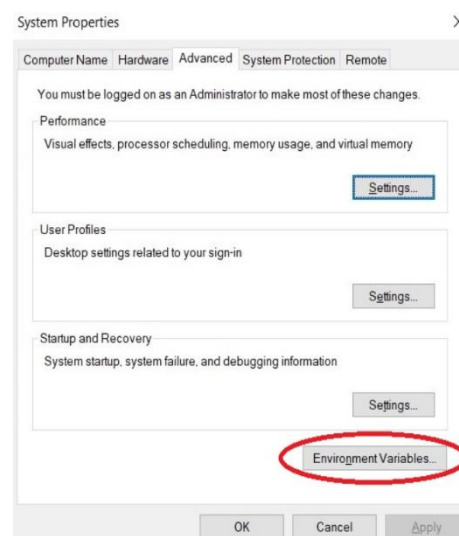
Figure 4-1 (Java JDK 8 downloads)

On the download page, accept the licence agreement and download the preferred version for your operating system. Then follow the installer's instructions.

4.1.1 Modifications in Environment Variable

The environment variables must be modified to implement Java in Windows:

1. Search for **Environment Variables** in your system
2. Select **System Variables** in the Environment Variables and click **New**
3. In **Variable Name** field, enter **JAVA_HOME**
4. In **Variable Value** field, enter the JDK installation path, if not changed (**C:\Program Files\Java\jdk.8.0_162**)
5. Click **OK**, the environment variables are updated correctly.



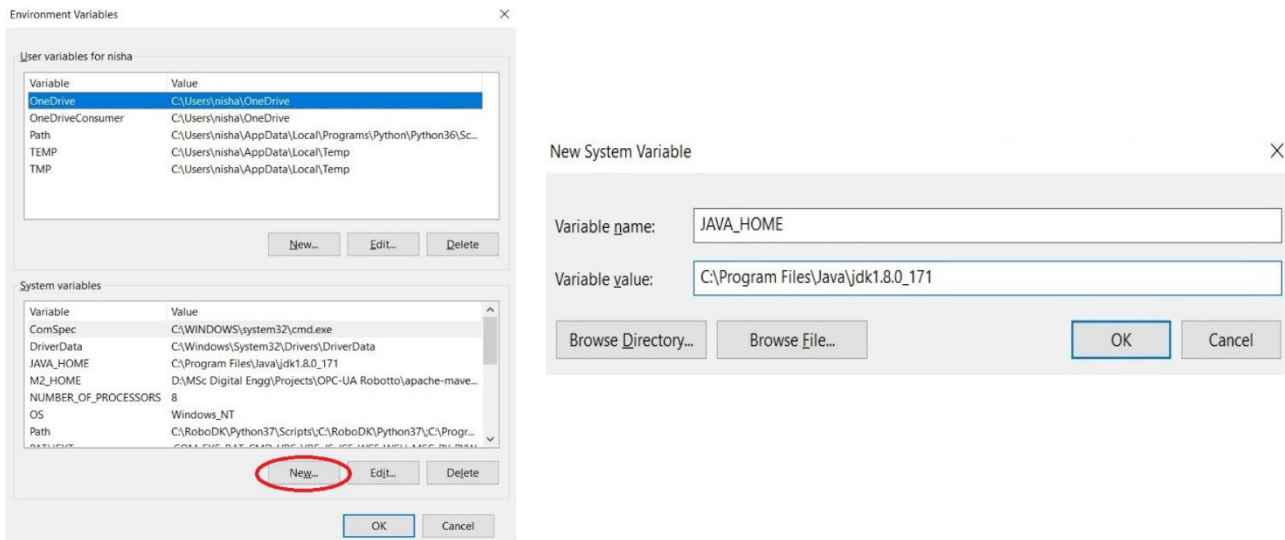


Figure 4-2 (Java path in system)

The JDK version and environment variable were configured according to the above-mentioned system. If not similar, search for the compatible version for your system on web.

4.2 Maven

Project's build, reporting and documentation can be managed by Apache Maven. Based on POM (Project Object Model), Maven can be used as a comprehension tool from a central piece of information. The installation procedure is mentioned in serialtim:

1. Download Maven from official website [5]. If your OS is Windows, download the Binary Zip archive (**apache-maven-3.6.2-bin.zip**). Please refer to the guidelines for OS's apart from Windows.
2. Extract the archive to **C:\Program Files** or in any other directory, you prefer.
3. Add the bin directory in the newly extracted directory to the **PATH** environment variable:
 - In the Environment Variables window, select **PATH** variable and click **Edit** or **New** (if not existing).
 - Click **New** and enter the new PATH to Bin folder (e.g. **C:\Program Files\apache-maven-3.6.2\bin**), then click **OK**
4. To confirm whether Maven works, Open **Command Prompt** and write **mvn -v** and press Enter. The result must be Similar to the image below.

```
Apache Maven 3.6.2 (40f52333136460af0dc0d7232c0dc0bcf0d9e117; 2019-08-27T17:06:16+02:00)
Maven home: D:\MSc Digital Engg\Projects\OPC-UA Robotto\apache-maven-3.6.2\bin\..
Java version: 1.8.0_171, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_171\jre
Default locale: en_IN, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Figure 4-3 (Apache Maven Version)

4.3 UPC UA Java Stack

After the successful installation of Maven, the OPC UA Java stack has to be build. Download the stack directly from the GitHub website for OPC Foundation [6].

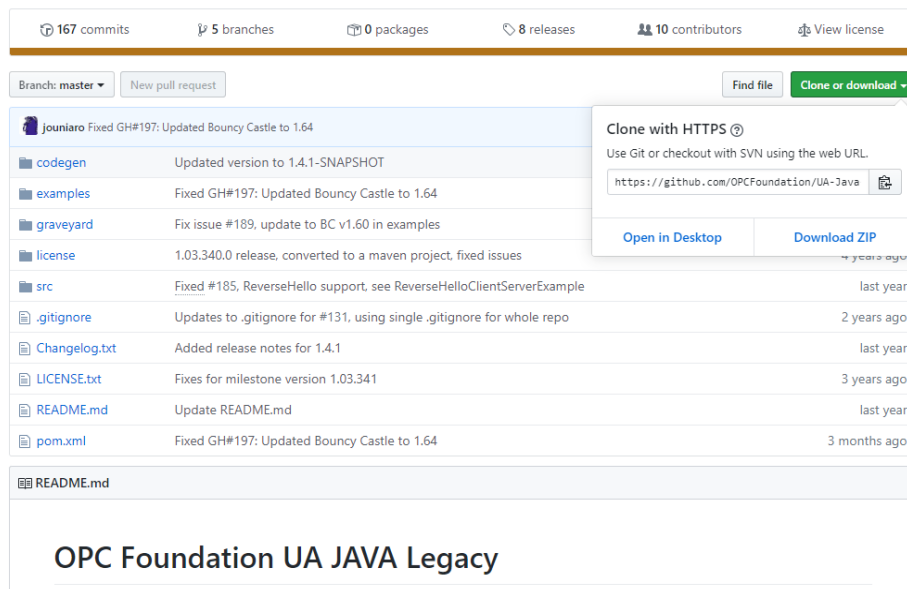


Figure 4-4 (UPC UA Java Stack)

1. Extract the folder **UA-Java-Master**, then open the Command Prompt and go to the newly created directory (**cd Downloads\UA-Java-master**).
2. Now, type **mvn package**, then wait for the process to complete. If all the packages and commands are correctly compiled, a **BUILD SUCCESS** message will appear.

```
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:49 min
[INFO] Finished at: 2018-04-18T23:18:15+02:00
[INFO] -----
```

a.

Figure 4-5 (Maven package build success)

3. In the UA-Java-master folder, a new folder **Target** will be created. It contains the required jar file (**opcustack1.3.344SNAPSHOT.jar**). Create a new folder with name **libs** and copy the jar file into it. This will import it into Android Studio together with other jars.

4.4 SLF4J

Additional libraries would be needed apart from the stack. SLF4J (Simple Logging Facade for Java) is a façade for logging systems, allowing the end-user to plug-in the desired logging system at the time of deployment.

1. Download the zip file (slf4j-1.7.30.zip) from download page or the available latest and stable version [7]. Extract the contents, then copy the file **slf4j-api-1.7.30.jar** to the **libs** folder.

Latest STABLE version

The latest stable SLF4J is version 1.7.30.

EXPERIMENTAL/UNSTABLE version

SLF4J version 2.0.0-alpha1 is the most recent but it is also experimental/unstable.

Maven central

SLF4J artifacts such as `.jar`, `.sources.jar` and `.javadoc.jar` files can be downloaded from [Maven central](#). The corresponding groupId is `org.slf4j`.

Figure 4-6 (SLF4J version)

4.5 Spongy Castle

As OPC UA implementation requires transmission of highly sensitive information, that should not be deciphered over transmission easily, so employment of certain crypto capabilities is necessary. These special classes are known as **Bouncy Castles**. These comprise of a collection of API cryptography to encrypt the information to ensure security.

Android does not support these libraries, so **Spongy Castle** was created to stock the Bouncy Castle libraries and make it compatible for Android.

You must download these three files directly from the main page [8]:

- Core
- Prov
- Bcpkix-jdk15on

Spongy Castle artifacts are published on Maven Central. Use Gradle or the [android-maven-plugin](#) to make the most of this, or click the `jar` links below - make sure you include all dependencies:

- [core \(jar\)](#) - Core lightweight API
- [prov \(jar\)](#) - JCE provider (requires core)
- [bcpkix-jdk15on \(jar\)](#) - PKIX, CMS, EAC, TSP, PKCS, OCSP, CMP, and CRMF APIs (requires prov)
- [bcpg-jdk15on \(jar\)](#) - OpenPGP API (requires prov)

Figure 4-7 (Spongy Castle)

Copy the downloaded files to the **libs** folder.

4.6 OPC UA Simulation Servers

In order to successfully implement the OPC UA communication framework, certain demonstrations and test runs must be carried out. This can only be sufficed by Demo or Simulation servers. The Demo servers provide simulated data and information models for unlimited run time demonstrations. Though a great extent of OPC UA Simulation servers are available online, yet we found **Prosyst Simulation Server** best suited for implementation in Android OS.

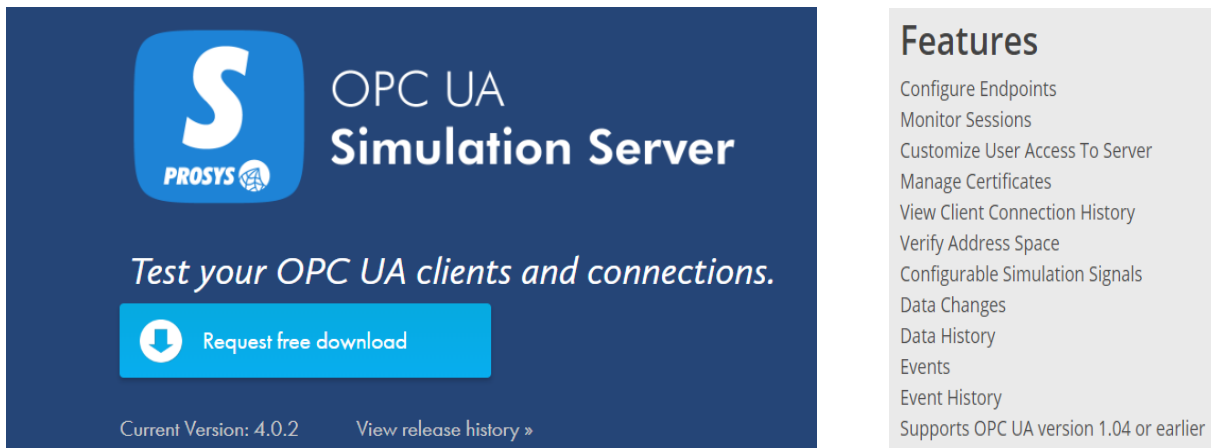


Figure 4-8 (Prosys OPC-UA)

1. Request for the free download [9] and fill the redirected form accordingly. After submitting your form, you will receive a confirmation mail from Prosys comprised of the download link. Choose the correct version based on your system's configuration.
2. Once you have downloaded and installed the software, the interface might look like **fig4-7**. The connection address contains OPC address of server.



Figure 4-9(Prosys OPC-UA simulation server)

4.7 Android Studio

The final step is to install the suitable version for Android Studio in the system [10]. Follow the installer steps with default settings (recommended). At the initial launching, some necessary components must be downloaded for the app development:

- Android SDK
- Support Libraries
- Emulator

Once the installation of the components is complete, a new window called New Project will appear:

1. Click on Start a new Android Studio project. Rename the project, as **OPC UA Client**, and then press **Next**.
2. In the new window **Target Android Devices** leave everything unchanged and press **Next**.

3. In the **Add an Activity to Mobile** window, select Empty Activity, then select **Next**.
4. In the last window leave everything unchanged, press **Finish** and wait for the Gradle configuration.

After these last steps Android Studio will finally open.

4.8 Emulator

Android Studio features the possibility to employ an emulator to test our applications. The AVD manager provides options for selecting from wide range of virtual devices.

1. Choose or create suitable the virtual device.
2. Choose the template, you prefer and press **Next**.
3. Choose the version of android for the installed Emulator. Then press **Finish**.

5 Android Client Application

This unit describes, how to create a simple client application and demonstrating a trial connection with the OPC UA server.

5.1 Sequential Procedure

The following section describes the steps to create a Client application to read data from the Prosys Server:

1. **Add Libraries:** Before starting to code, it is necessary to imported the downloaded libraries in the Android project.
 - Copy the earlier created libs folder in the project directory (e.g. *D:\Projects\OPC-UA Robotto\ OPC-UA-Client\app*).
 - Now, in Android Studio, Change the View to **Project**, then select the jar files you copied. Right-click and select **Add as Library**.
 - Now you have all necessary libraries required to create the Client application. You can now change the view back to **Android**.
2. **Permissions:** According to the Android policy terms, it is necessary to request permission to access sensitive user data and certain system features (Internet in our case). Depending on the feature, the system grants permission automatically or prompt the user to approve the request.
 - Now, open **AndroidManifest.xml** file & add the following permission inside

```
<manifest>
<uses-permission android: name="android.permission.INTERNET" />
```
3. **Creating a Client:** To create the Client, we first need to create a try-catch block to handle various exceptions (ServiceResultExceptions, OutOfBoundsExceptions). We need to create a client certificate every time we establish a connection.
The certificate requires following parameters:
 - Application Name
 - Application URI
 - Application Type
4. **Discovering Endpoints:** To find the endpoints, you need to know the TCP address of the server (*opc.tcp://serveraddress:port/OPCUA/SimulationServer*) and the security levels adopted.
 - Considering the server has **no security**, order the endpoints according to security level and choose the one with lowest security.
 - For you to have the server with no security, in Prosys Simulation Server, check the option of **None** in security modes.

5. **Activating the session:** Create the session through the client and the chosen endpoint, and then activate the session. After successful creation of session, do the required operations like Reading the values or Writing to the variables. It is very important to close the session after you are done with the operations.
6. **Creating the User Interface:** Create a simple interface to read an integer value from the Server with a Button to execute the read operation and a **TextView** to display the read value. It should look similar to figure below.

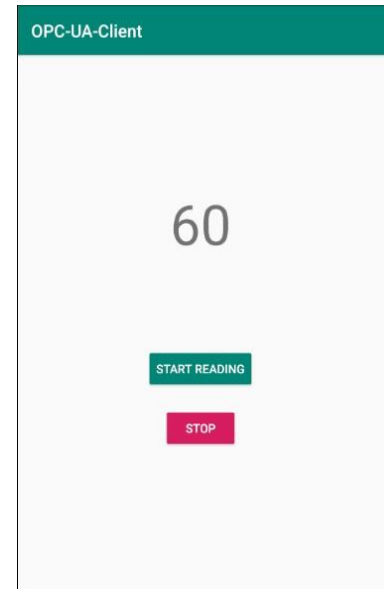


Figure 5-1 (Basic UI)

6 Robot Operating System (ROS)

This unit provides the significant insight to a Robot Operating System and its capabilities to simplify the complex tasks and encourage the collaborative robotics software development [11].

6.1 What is ROS?

A flexible framework comprised of software tools, libraries and conventions for influencing and dealing with robust robot behaviour across the wide variety of robotic platforms. ROS is an open-ended framework involving integrative and embodies AI. In spite of a rapid development, ROS presents significant challenges to the software developers

Initially, ROS was developed by multiple institutes based on their robot configurations and respective functionalities. ROS has successfully emerged as a “Federated Model” and has provided more control to its operators. It enables programmer to create their ROS repository on their own servers and maintain its access permissibility and ownership.

6.2 ROS and Industry 4.0

A reliable, safe and multifunctional operation is what we expect from the current generation of robots. The production systems can be optimized by having an operating system with analysis and monitoring capabilities. Robots are a system of distributed software components communicating with each other. These communication nodes are responsible for the intended functionalities.

As the current trend is inclined more towards the industrial automation and data exchange, implementing an OS that enables all the robotic platforms to work simultaneously and optimally is what Industry 4.0 needs.

6.3 ROS Core Concepts

These concepts are implemented in ROS_comm repository for basic computation:

1. Nodes
2. Topics
3. Messages
4. Services
5. ROS_launch
6. ROS_run
7. ROS_cd
8. ROS_core

6.3.1 ROS Node

It is a command line tool for displaying debug information about ROS nodes, including publications, subscriptions and connections. ROS nodes are written using a ROS client library:

- Roscpp- c++ client library
- Rospy - python client library

Node publish or subscribe to Topics. Node could also provide or use the Services for Actions.

6.3.2 ROS Topics

These are the named stream of messages with a defined type. These are responsible for Node communication by publishing the messages. Topics are intended for unidirectional streaming communication. Nodes that need certain data can subscribe to the relevant topics and the nodes that generate data, publish the relevant topic.

6.3.3 ROS Messages

Simplified descriptive language as messages is used to describe the data values that are published by nodes. This description makes it easy for ROS tools to automatically generate source code for the message type in several target languages.

There are two parts to a .msg file: fields and constants. Fields are the data that is sent inside of the message. Constants define useful values that can be used to interpret those fields.

6.3.4 ROS Services

Though the publish/subscribe communication model is very flexible, yet it is not appropriate for reply interactions. Request/Reply is done by services, which is defined by pair of messages, one is for request other for reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Client libraries usually present this interaction to the programmer as if it were a remote procedure call.

6.3.5 ROS_launch

ROS_launch is a tool for easily launching multiple ROS nodes locally and remotely via SSH, as well as setting parameters on the **Parameter Server**. It includes options to automatically respawn processes that have already died. ROS_launch takes in one or more XML configuration files (with the launch extension) that specify the parameters to set and nodes to launch, as well as the machines that they should be run on.

6.3.6 ROS_Run

ROS_run allows you to run an executable in an arbitrary package from anywhere without having to give its full path or cd/ros cd there first.

6.3.7 ROS_cd

ROS_cd allows you to change directories using a package name, stack name, or special location.

6.3.8 ROS_core

ROS_core is a collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a ROS_core running in order for ROS nodes to communicate. It is launched using the ROS_core command.

7 Ubuntu Virtual Box setup for ROS

This unit deals with the basic prerequisites and installation procedure for the successful setup of ROS environment in a Windows as host operating system. It also includes the chronological steps with installation commands for the **Ubuntu Virtual Box**. In order to further the installation process without any failures, do refer the reference document.

7.1 Ubuntu Virtual Box

In case, if an Ubuntu based system is unavailable, a virtual box can be utilized, wherein a windows system will play the role of host and the Ubuntu as the guest. It is an open source virtualization software from Oracle [12], which enables installation of other OS on the virtual machines.

7.2 Setting up the virtual box and installing ubuntu 16.04

Before installing the Virtual Box, we need to make sure that the virtualization feature is enabled on the host machine. For this purpose, we need to first enter the bios setup, then check for virtualization in system configuration. If the virtualization parameters are already enabled then we can go ahead with the installation of the Virtual Box or else we need to enable it first.

After the Virtual Box has been installed follow the below procedure;

1. Set up a new machine.
2. Make sure to download Ubuntu(64-bit) 16.04 version as it is the only version which is compatible with ROS kinetic kame.
3. In the general settings select basic and then provide the Name, select the type to be Linux and the version as Ubuntu(64-bit).
4. In System settings select Motherboard and assign the Base Memory. The base memory assigned was 2048MB. Then select Acceleration and keep the Paravirtualization Interface as default and for Hardware Virtualization enable both VT-x/AMD-V and Nested Paging.
5. Then select Storage and select the option under Controller: IDE. Under Attributes you will find Optical Drive where you have to select the downloaded Ubuntu file from the dropdown menu. You also need to assign the storage space to the drive. The storage space was set to 64MB.
6. In Network settings for Adapter 1 select the Bridged Adapter option for Attached to. No changes are to be made for the remaining sections.
7. Once you are done with this Start the machine and go ahead with the installation of Ubuntu 16.04.

7.3 Employed Network types

When using Virtual Box to run **Ubuntu 16.04** as a guest OS change the network settings to **Bridged Adapter**.

There are two important network modes:

1. NAT: NAT masks all the activity of the Virtual Box and assigns the VM in a different subnet wherein the VM would be able to access the outside network but the other computers would not be able to locate the VM. It can be used as a security mode.
2. Bridged Adapter: BA keeps the VB in the same subnet as that of the Host OS and hence the VM would be able to access the outside network and will be discoverable by the other computers.

7.4 ROS installation

There is a variety of ROS distributors available online, so we have employed **ROS Wiki** for installation and development of the suitable ROS package. A prerequisite for configuring the ROS repositories, the system should be either Ubuntu based or have a virtual box for Ubuntu. So, the installation link for ROS packages can be used to download the appropriate package [13].

8 OPC-UA server realization with ROS

This unit explains how the OPC-UA server can be realized for the ROS operating software.

8.1 Cloning ROS based OPC-UA server

The communication package for ROS and OPC-UA can be found on GitHub [14]. The download link is available in the references. The installed package used is **ros_opcua_impl_python_opcua**. For cloning the package follow the below steps,

1. Check the installed python version on the Ubuntu system. Python version 2.7 and above are compatible for use with the above-mentioned package.
2. Before cloning the repository, we first need to navigate into the **catkin_ws** created during ROS installation and source the file into the **src** file.
3. First open terminal, then type the command "**cd ~/catkin_ws/src**" to navigate into the src file of the catkin workspace.
4. For cloning the repository use the command "**git clone https://github.com/iirob/ros_opcua_communication.git**".
5. The repository will be cloned in the **src file**. The reason why we need to clone the repository in the src file is because when we try to initiate the server the terminal executes and searches for files in the **catkin workspace** and it does not recognize any files that are out of the catkin workspace.
6. After the repository has been cloned, we need to install or update the OPC-UA packages in the system, for this use the command "**sudo apt-get install opc ua**". Once the OPC-UA packages are updated. We can initialize the server.
7. The command mentioned in step 4 installs a package **ros_opcua_communication**, which has different sub-packages. Out of these sub-packages we need to work with "**ros_opcua_impl_python_opcua**".

The scripts file of the above-mentioned package has a python program of server named as **ros_server**. A few changes must be made in the program before launching the server. The server address must be updated according to the **IPv4 address** assigned to the virtual box which can be found in the network options. This address is updated as "**opc.tcp://IpV4 address: port number/**".

8.2 Initializing ROS on OPC- UA server

In order to initialize the ROS OPC-UA server the following steps should be followed:

1. Before initializing the ROS OPC-UA server, we need to initialize a few nodes with topics, services & messages, for this we use the general **Turtlesim** node as an example.
2. We first need to initialize the "**ROS Core**" which can be done by using the basic ROS command "**roscore**".
3. After the ROS_core has been initialized we need to initiate the **Turtlesim** node which can be done by using the command "**roslaunch turtlesim turtlesim_node**".
4. We need to source the current workspace into the devel in order to access the ROS commands such as "**roslaunch**". This can be done by using the command "**source ~/catkin_ws/devel/setup.bash**".

5. After the current workspace has been sourced in the devel we can use the command **"roslaunch ros_opcua_impl_python_opcua rosopcua.launch"** to initialize the server.

After the server has been initialized the following Topics and Services are initialized and can be used by the client to control the Turtlebot.

Services:

1. Clear
2. Kill
3. Reset
4. Rosopcua
5. Rosout
6. Spawn
7. Turtle1
8. Turtlesim

Topics:

1. Rosout
2. Rosout agg
3. Turtle1

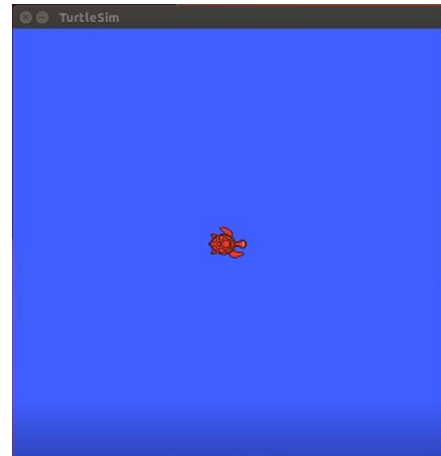


Figure 8-1 (Turtlebot)

The sub-services included and initialized in Turtle1 such as **"turtle1/set_pen"**, **"turtle1/teleop_absolute"**, and **"turtle1/teleport_relative"** are essential for the movement of turtlebot.

The above-mentioned sub-services are described as methods in OPC UA i.e. we need to call these methods in the client and then assign their respective input arguments. As explained in Chapter 2, these methods cannot be overwritten in the client. The instruction set that has been used for calling these methods are explained in Chapter-9 Connection Establishment (Client and Server).

The above server can be accessed by different OPC UA based clients such as **UA Expert** and **Prosys**. In order to establish the connection, we need to provide the client with the endpoints which is mentioned as **"opc.tcp://ipV4 address: port number"**. Both the mentioned clients were able to call the methods generated by the server and provide the input arguments which led to the movement of the Turtlebot. The same procedure has been followed for connecting the ROS OPC UA server and Android based OPC UA client.

9 Final Implementation (Server & Client)

This unit involves the final implementation of server and client after their final successful installation.

9.1 Procedure for Implementation

The sequential steps for calling the relevant functions in order to move the Turtlebot are in serialtim.

9.1.1 Browsing the Address Space:

1. In order to setup a connection between an Android device and the ROS based OPCUA server, you should know the **opc.tcp address** of the OPCUA server (**opc.tcp://Ipv4 address:Port**).
2. Discover the server using the server endpoints of the ROS based OPCUA server then create and activate the client session. Now, **browse** the ROS based OPCUA server address space using **BrowseDescription**.
3. Initially, start browsing with the **ObjectsFolder** which shows all the objects in the form of topics, services and actions.
4. Now, select the **NodeID** of the Object, that is supposed to be subscribed. If the object does not have any method, you can directly write the value of the variable using **WriteValue()** function.
5. In the demonstrated version, the **Turtlesim** object was subscribed. It comprises of methods such as spawn, set_pen, teleop_absolute, teleop_relative and call one of the methods.
6. Here, the **ObjectID** is set as the **NodeID** of Turtlesim and set the **MethodID** as that of the NodeID of above-mentioned methods.

```
//Browse
BrowseDescription browse = new BrowseDescription();
NodeId n = new NodeId( namespaceIndex: 3, value: "turtle1"); //ObjectID
NodeId n1 = new NodeId( namespaceIndex: 3, value: 2); //MethodID
browse.setNodeId(n);
browse.setBrowseDirection(BrowseDirection.Forward);
browse.setIncludeSubtypes(true);
BrowseResponse res3 = mySession.Browse( RequestHeader: null, View: null,
    RequestedMaxReferencesPerNode: null, browse);
System.out.println(res3);
```

Figure 9-1 (Address Space Browsing)

9.1.2 Calling the Method:

1. To make a **CallMethodRequest** to the server for a specific method, it is necessary to make a **CallRequest** to the server first. The call request should be as follows.

```
//Call Method
CallRequest callRequest = new CallRequest();
CallMethodRequest methodRequest = new CallMethodRequest();
callRequest.setMethodsToCall(new CallMethodRequest[] {methodRequest});
methodRequest.setMethodId(n1);
methodRequest.setObjectId(n);
```

Figure 9-2 (Method Calling)

9.1.3 Setting Input Arguments:

1. To operate the Turtlesim with absolute co-ordinates, we need the X-coordinate, Y-coordinate and angle Theta of the turtle.
2. Define these coordinates as float objects and convert them into Variant and set the **InputArguments** as a Variant array.

```
//Input Arguments for Turtlesim teleop_absolute
Float cords_x = 4.0f;
Variant cords_x_InVariant = new Variant(cords_x);
Float cords_y = 2.0f;
Variant cords_y_InVariant = new Variant(cords_y);
Float cords_ang = -45.0f;
Variant cords_ang_InVariant = new Variant(cords_ang);
methodRequest.setInputArguments( new Variant[] {cords_x_InVariant,
        cords_y_InVariant, cords_ang_InVariant});
```

Figure 9-3 (Input Argument Setting)

9.2 Designed UI

The designed user interface for executing the implementation process and move the Turtlebot to the designed coordinates.

The figure displays two screenshots of the Robotto-OPCUA application interface. Both screens are titled 'Robotto-OPCUA' and 'TELEPORT_ABSOLUTE'. The left screenshot shows the initial state with input fields for X, Y, and Theta, each containing the placeholder text 'Enter a float value'. The right screenshot shows the same interface after data entry, with X set to 2.7, Y set to 7.32, and Theta set to 0.0. A green 'ENTER' button is visible at the bottom of both screens.

Figure 9-4 (Designed UI)

9.3 Motion of Turtlebot

The ultimate purpose of calling the methods and executing the input arguments, was to initiate the motion of Turtlebot based on the inputs in the UI. It can be observed in the demonstration video, how the input arguments, manipulate the motion of Turtlebot.

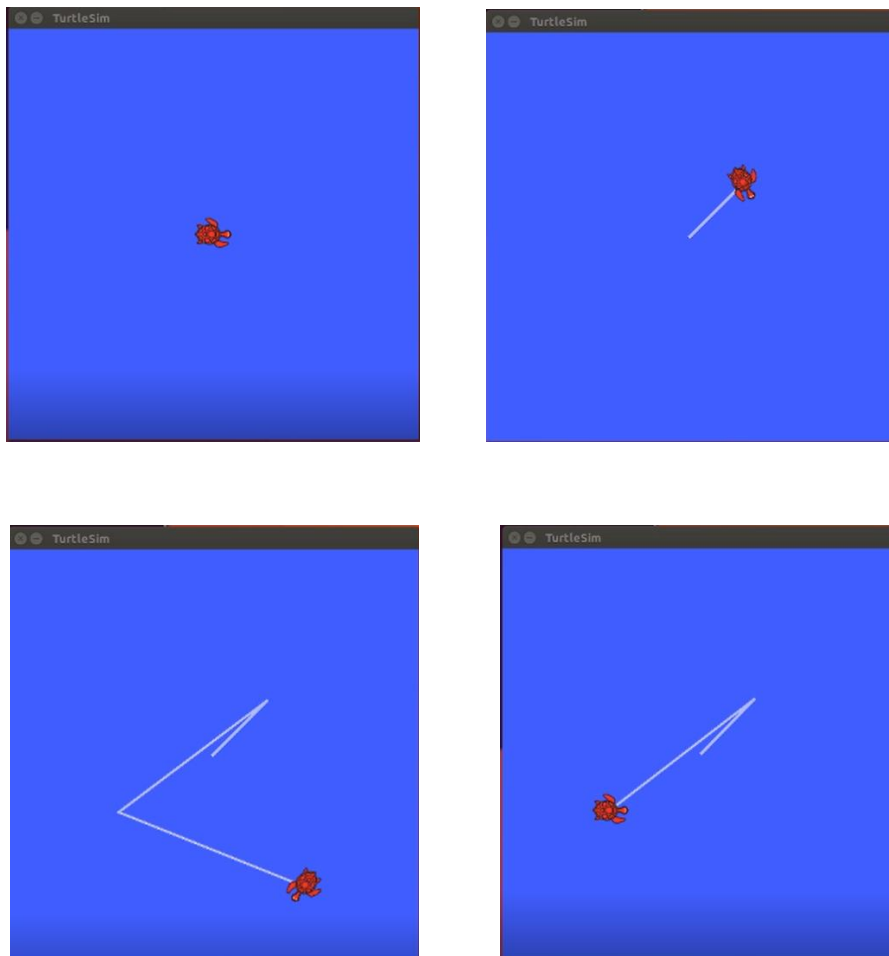


Figure 9-5 (Motion of Turtlebot)

10 Conclusion and Recommendation

1. The Android interface requires the input arguments to be entered **numerically**. Further developments can be made in the interface to control the motion of the robot using **directional keys**.
2. The Android based OPC UA client and the ROS based OPC UA server must be connected to the **same network** for identification and data transfer purposes.
3. While using Virtual Box, **Bridged Adapter** network type must be used for establishing a stable and successful connection between the Server and the Client.
4. The connection between the client and server maintains itself, even if both the server and the client remain static for certain duration of time.
5. Linear motion of the **Turtlebot** been accomplished. Further advancements & upgradations for executing the curvilinear path can be done by modifications in Android and server.
6. It has been determined that the origin for the Turtlebot, lies at the bottom left corner of the Turtlesim display. So, the movement can only be made for the **positive X and Y axes**, whereas the angular movement of the robot along the Z axis through the Turtlebot is possible in both clockwise and anticlockwise direction. Further development can be made for motion of the Turtlebot in the negative axes.
7. At the time executing the application, when no value is entered for the coordinates (not even zero), no motion in the Turtlebot is observed i.e. (**Bad_nullpointexception error**).
8. The absolute position of the Turtlebot is displayed on the terminal, where Turtlesim node has been initialized. The calling of methods can also be observed on the server display.
9. Whenever the methods are called by the client with the input arguments, even though the Turtlebot moves according to the executed method, yet an error "**Bad_UnexpectedError**" is displayed in the message terminal of the client.

11 Abbreviations

AI: Artificial Intelligence	24
API: Application Program Interface.....	20
APK: Android Package	15
AVD: Android Virtual Device	22
IDE: Integrated Development Environment.....	15
IPv4: Internet Protocol Version 4	28
OPC-UA: Open Platform Communication Unified Architecture	7
OpenJDK: Open Java Development Kit	17
P2P: Point to Point	16
POM: Project Object Model	18
ROS: Robot Operating System	7
SDK: Software Development Kit.....	15
SLF4J: Simple Logging Facade for Java	19
SSH: Secure Shell	25
UI: User Interface.....	15
VM: Virtual Machine	26
XML: Extensive Markup Language	25

12 References

- [1] Unified Automation GmbH, “Embedded OPC UA stack,” 2019. [Online]. Available: https://documentation.unified-automation.com/uasdkhp/1.0.0/html/_12_ua_address_space_concepts.html. [Accessed November 2019].
- [2] OPC Foundation, “Unified Architecture,” 2020. [Online]. Available: <https://opcfoundation.org/about/opc-technologies/opc-ua/>. [Accessed December 2019].
- [3] Android Developers, “Application fundamentals,” [Online]. Available: <https://developer.android.com/guide/components/fundamentals>. [Accessed September 2019].
- [4] O. T. Network. [Online]. Available: <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>.
- [5] A. M. Project. [Online]. Available: <https://maven.apache.org/>.
- [6] G. U. Java). [Online]. Available: <https://github.com/OPCFoundation/UA-Java-Legacy>.
- [7] S. L. F. f. Java. [Online]. Available: <http://www.slf4j.org/download.html>.
- [8] G. Castle). [Online]. Available: <https://rtyley.github.io/spongycastle/>.
- [9] P. OPC. [Online]. Available: <https://www.prosysopc.com/products/opc-ua-simulation-server/>.
- [10] G. D. A. Studio). [Online]. Available: <https://developer.android.com/studio>.
- [11] R. Wiki. [Online]. Available: <https://www.ros.org/about-ros/>.
- [12] O. V. VirtualBox. [Online]. Available: <https://www.oracle.com/virtualization/technologies/vm/downloads/virtualbox-downloads.html>.
- [13] R. W. Kinetic). [Online]. Available: <http://wiki.ros.org/kinetic/Installation/Ubuntu> .
- [14] R. W. & O.-U. package). [Online]. Available: http://wiki.ros.org/ros_opcua_impl_python_opcua .