An Automatic Protocol Verification Framework for the Development of Wireless Sensor Networks

Taehyun Kim Jaeho Kim Sangshin Lee Ilyeup Ahn Minan Song Kwangho Won
Korea Electronics Technology Institute
#68 Yatapdong Bundanggu
Seongnamsi Gyeonggido REPUBLIC OF KOREA
+82-31-789-7515
{thkim, jhkim, sslee, iyahn, mhsong, khwon}@keti.re.kr

ABSTRACT
In recent years, there are many active researches on Wireless Sensor Networks (WSNs) as a way to collect diverse context information around the world. A lot of new WSN protocols have been proposed and implemented for the various application fields such as military, environmental, habitat monitoring, health, home and office, and other applications. When we compose a WSN protocol stack using several layers which have been designed and implemented individually, some uncertain protocol layer modules with malfunctions may cause the serious faults of their own or entire WSNs. Therefore, it is very important to verify functions and interoperability of each layer as well as to make well-defined protocol specifications. In this paper, we propose an automatic protocol verification framework for WSNs. The proposed framework consists of a test procedure description language written in XML and a test harness which executes test procedures. We have implemented the proposed framework and used it to verify some of our own WSN protocol layers. And our test framework has performed nicely. Therefore, we think that this framework would make it possible for WSN protocol developers to verify protocols easily.

Categories and Subject Descriptors
B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance.

General Terms
Experimentation

Keywords
WSN, Test Automation, XML

1. INTRODUCTION
As a way to collect context information of real world environment, the Wireless Sensor Networks (WSNs) are in the limelight. A lot of versatile application specific WSN protocol stacks are designed and implemented to maximize the performance of WSNs under the restricted environment in terms of low power capacity, low bandwidth, low computing power, and so on. Even though there are some famous sensor network protocols such as IEEE802.15.4[1], TinyOS[2] and ZigBee[3], there are little generic sensor network protocols supporting all different kind of applications. Because of these characteristics of WSNs, the protocol designer should make lots of efforts to overcome its restriction. And it is too difficult to make an entire protocol stack which may have several layers such as PHY, MAC, NWK, APP and so forth. In this reason, we are apt to adopt not only a layered architecture but also some ready-made layers from others.

Because of the flexibility and extensibility of layered architecture, it is possible to implement each protocol layers as an independent module. Due to the characteristic of functional transparency, we can compose a WSN protocol stack using several layers which have been designed and implemented individually. But some uncertain protocol layer modules with malfunctions may cause the serious faults of their own or entire WSNs. Therefore, it is very important to verify functions and interoperability of each layer.

In this paper, we propose a well defined test procedure description language and a test harness to make sure whether the implemented WSN protocol stack works properly or not. The test procedure description language is easily understandable to a test operator as well as the test harness. The test harness of our framework interprets the given test procedure description document to run test procedure step by step in the way of the test procedure description.

We present in Section 2 detailed descriptions about the XML schema for the test procedure description language and section 3 overviews the WSN protocol stack verification framework. The mechanism for generating a part of test harness program source code is explained in section 4. And section 5 depicts how the proposed test framework works on the test procedure with case study on ZigFest Spain 2006. Last three sections are conclusions, acknowledgments and references.

2. The Test Procedure Description Language
In software engineering, a test suite is a collection of test cases that are intended to be used as input to a software program to show that it has some specified set of behaviors.

Tridentcom 2008, March 18 – 20, 2008, Innsbruck, Austria.

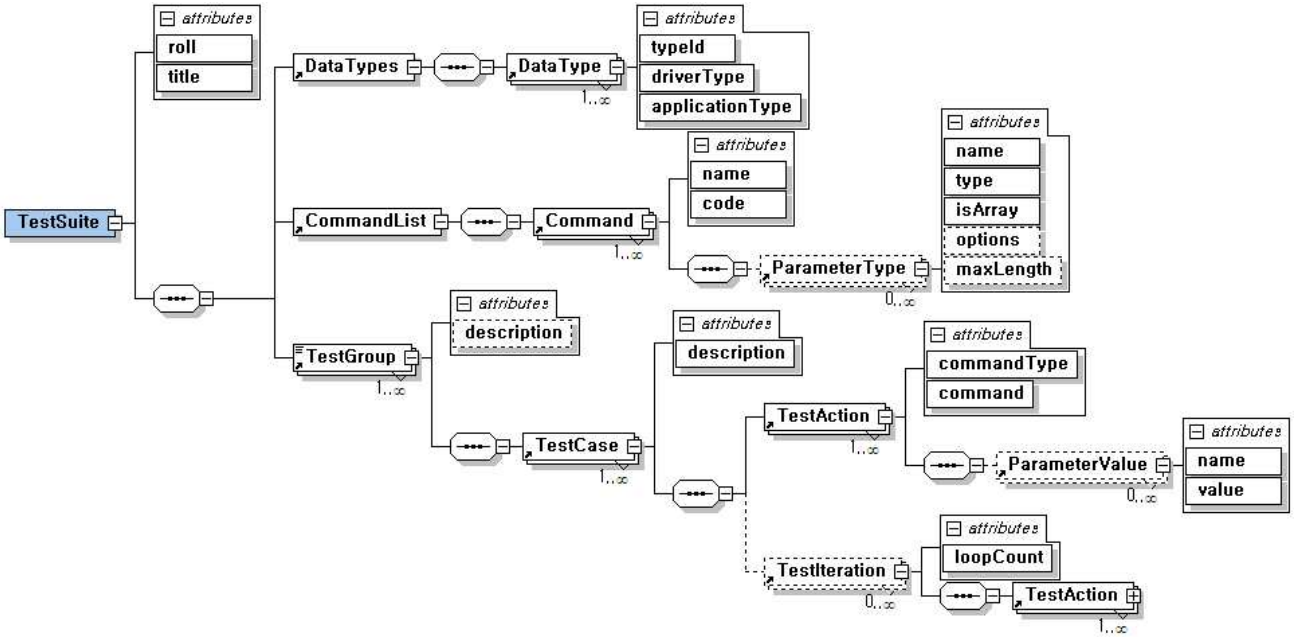

Figure 1. The schema for the test suite description language

A test suite often also contains detailed instructions or goals for each collection of test cases and information on the system configuration to be used during testing.

In this section, we present how you can write a test procedure description language using XML. The XML has the hierarchical structure that is good for us to define test suite systematically and it is relatively immune to changes in technology because of its platform-independent characteristics. Above all, the biggest strength of the XML based test suite description language lies in extensibility. The test operator can extend some XML elements as occasion demands. Figure 1 illustrates whole structure of the XML based test suite description language.

2.1 The TestSuite element

The "TestSuite" element is the root element of the test procedure description XML document that consists of a "DataTypes" element, a "CommandList" element and one or more "TestGroup" elements. And this element has the "title" attribute and "role" attribute. The "title" attribute offers advisory information about the whole Test Suite and "role" attribute specifies role of this sensor node on the test scenario.

2.2 "DataTypes" and "CommandList" element

The "DataTypes" element and "CommandList" element are a container element that contains "DataType" element and "Command" element respectively. These elements contain the basic information to be used by source code generator.

2.2.1 DataType element

To prevent a mistaken interpretation of the command message frames between the Test Application and the Test Driver, the "DataType" element defines platform neutral data type and its mapping to the Test Application and the target WSN sensor node platform. This way we can get a more portable source code.

The "DataType" element has three attributes. The "teypId" attribute specifies the ID of a platform neutral data type definition and the value of the "applicationType" element and "driverType" attribute determines the data type which translated into each platform.

2.2.2 "Command" element

The "Command" element can be used to define the interface message frame format and the message handler function prototypes. This element can contain the two attributes and zero or more "ParameterType" element. The "name" attribute and the "code" element specifies the human readable name and the code value respectively, and "ParameterType" element defines detailed parameter information of this command.

2.2.3 "ParameterType" element

The "ParameterType" element contains information that describes the data structure of a parameter of the containing "Command" element. This element can have six attributes. The "name" attribute and "type" attribute identify the name and data type of this parameter. The value of the "name" attribute must be unique within containing "Command" element and the value of the "type" attribute must refer "DataType" element. The "isArray" attribute specifies whether this parameter is an array data or not and the value of the "maxLength" attribute means maximum array length when the value of the "isArray" attribute equals to "TRUE."

2.3 "TestGroup" element

The "TestGroup" element allows several associated Test Cases classify into small groups and attach a tag to each groups. This element can have one or more "TestCase" element and a "description" attribute. The value of the "description" attribute is a brief description of the small group.

2.4 “TestCase” element

In software engineering, a test case is a set of conditions or variables under which a tester will determine if a requirement or use case upon an application is partially or fully satisfied. The “TestCase” element is used to define a test case. A “TestCase” element has a “description” element which offers advisory information about this test case, and one or more “TestAction” element or “TestIteration” element.

2.5 “TestAction” element

The “TestAction” element defines a standard function to support the containing test case. In other words, it has detailed information to execute an action which cannot be divided into smaller than. Information for test action comprises a reference to the “Command” element and actual parameter list to pass dynamic setting on runtime.

When the Test Execution Engine component meets a “TestAction” element, it makes command message frame and send through the Sensor Node Interface Adaptor component.

2.6 “TestIteration” element

If you want to repeat any test action or group of test actions, then use the “TestIteration” element. The “TestIteration” element wraps one or more “TestAction” elements to execute repetitively. The total number of test repetition is specified by the value of the “loopCount” attribute.

3. WSN Stack Verification Framework

The proposed WSN stack verification framework is a software framework which performs WSN protocol stack test procedure for the target WSN protocol stack or a WSN protocol stack layer using test suite document written in XML. This framework has two software blocks. One block is “Test Application” which operates on a personal computer environments and the other is “Test Driver” which operates on a sensor node. These two blocks communicate with each other to exchange messages and commands going on test procedure. The framework provide diverse channels that commonly used wired communication methods for example RS232, RS485, USB and Ethernet. The system architecture of the test framework is depicted on Figure 2.

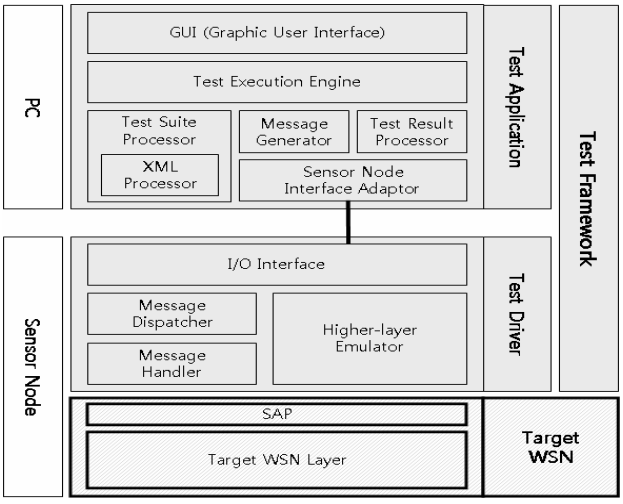

Figure 2. WSN Stack Verification Framework Architecture

3.1 The Test Application block

The Test Application block is a Graphic User Interface (GUI) application running on a personal computer. This block is responsible for interacting with user and driving the execution of each step of the selected test case described on XML based test description document. The Graphic User Interface component allows the test operator to select the test case to be executed and to change test parameter values of the test actions which belongs to the selected test cases. The primary function of the Test Execution Engine component is to parse and interpret the test case definitions that are part of the test procedure description document. Even when these Test Cases involve the Test Suite Processor, Message Generator and Test Result Processor components, the interpretation of the Test Cases is under the control of the Test Execution Engine.

The Sensor Node Interface Adapter component gives flexibility to the Test Framework. The APIs of the Sensor Node Interface Adapter component are separate from communication channel specific mechanism.

3.2 The Test Driver block

The Test Driver represents the higher layer for the test target WSN protocol stack. It receives command message from Test Application and event notifications from the WSN Protocol stack, and issues the SAP request primitives which manage the WSN protocol stack and send RF messages.

The command message from the Test Application through I/O Interface of the Test Driver will be routed to the Message Handler component by the Message Dispatcher. Then, the Message Handler updates settings that are managed by the Higher-layer Simulator or issues SAP primitive of the test target WSN protocol stack.

4. Interface Source Generator

In the Test Development step of the test process, the Test driver source code can be generated automatically using the source code generator from a test suite document. In this section, the underlying principles are depicted to generate source code. Figure 3 shows a part of the test suite document example that helps to describe our source code generation procedures.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xtaf:TestSuite  ... >
3    <xtaf:DataTypes>
4      ....
5    </xtaf:DataTypes>
6    <xtaf:CommandList>
7      ....
8      <xtaf:Command  xtaf:name="set_data_payload"
9                xtaf:code="14">
10       <xtaf:ParameterType  xtaf:name="payload_size"
11           xtaf:type="int"     xtaf:isArray="false"/>
12       <xtaf:ParameterType  xtaf:name="payload"
13           xtaf:type="hexa"    xtaf:isArray="true"
14           xtaf:maxLength="100"/>
15     </xtaf:Command>
16     ....
17   </xtaf:CommandList>
18 ....
19 </xtaf:TestSuite>
```

Figure 3. Example of the test suite document

4.1 Test driver header file

The test driver header file provides some definitions easy to port generated source code to the target sensor node platform. It consists of constant definition, message handler prototype definition and etc.

Figure 4 presents culled lines from the generated test driver header file.

```
1  #define CMD_SET_DATA_PAYLOAD     14
2
3  void     call_set_data_payload(uint8_t     payload_size,     uint8_t
4  payload[]);
```

Figure 4. Generated Test driver header file

4.2 Message dispatcher source file

Message dispatcher parses the command message from the test application and routes command to the message handler. The facture below illustrates how the message dispatcher parses and routes the command. The Message dispatcher makes variables for message handler parameter and calls the corresponding message handler with these variables.

```
1   bool_t   messageDispatcher(
2   uint8_t * buffer, uint8_t nLength )
3   {
4   ...
5     switch( buffer[nIdx ++] )
6     {
7     case  CMD_SET_DATA_PAYLOAD:
8       {
9         uint8_t payload_size = buffer[nIdx ++];
10
11        uint8_t nLen = buffer[nIdx ++];
12        uint8_t payload[100];
13        for( aryIdx=0; aryIdx < nLen; aryIdx ++ )
14          payload[aryIdx] = buffer[nIdx++];
15        uint8_t payloadLength = buffer[nIdx++];
16
17        call_set_data_payload(payload_size, payload);
18
19        return TRUE;
20      }
21      break;
22    ...
23
24    default:
25      return FALSE;
26      break;
27    }
28
29    return FALSE;
30  }
```

Figure 5. Generated Message dispatcher source file

4.3 Message handler skeleton source file

The last job for test developer is implementing the message handler. The test developer just put the program body into the message handler skeleton code generated by source code generator. The program body must include the features to control WSN protocol stack or to send RF message through test target WSN protocol stack.

```
1  void     call_set_data_payload(uint8_t     payload_size,     uint8_t
2  payload[])
3  {
4    // TODO: Add your command handler code here
5  }
```

Figure 6. Generated Message handler source file

5. Test Framework application

This section depicts how this test framework can be applied to test procedure with case study on ZigFest Spain 2006. We made the best use of this test framework to test interoperability features of our IEEE 802.15.4 Wireless Medium Access Control (MAC) protocol stack in ZigFest Spain 2006 (April 25-27, 2006).

5.1 Test suite design

5.1.1 Test command definition

To make test suite document, we derived command list as a result of analysis IEEE 802.15.4 MAC specification and "ZigFest Level 1 Interoperability Test Procedures" document.

Table 1. Examples of the Test Action command list

Category	Command	Description
Config-urations	set_device_type	MAC properties setting
	set_mac_address	
	set_short_address	
	set_pan_id	Network properties setting
	set_channel	
	set_beacon_order	
	set_superframe_order	
	set_beacon_payload	
	set_data_payload	NWK layer simulation
	set_assigning _device_address	
Primitives	RESET.request	MLME
	SCAN.request	
	ASSOCIATE.request	
	DISASSOCIATE.request	
	POLL.request	
	START.request	
	SYNC.request	
	GTS.request	
	DATA.request	MCPS

The configuration part commands used to set the values for calling primitives later, and the primitive part commands used to call the MAC primitives.

5.1.2 Test Group and Test Case definition

In this case, all of the test cases to test interoperability features were given by ZigBee Alliance. So we just translated the test documents into XML scripting language.

5.2 Run the test suite

To run the test suite, test operators only executes the test application and load the test suite document to run. If you need to change parameters, change the parameter values on Test Application GUI.

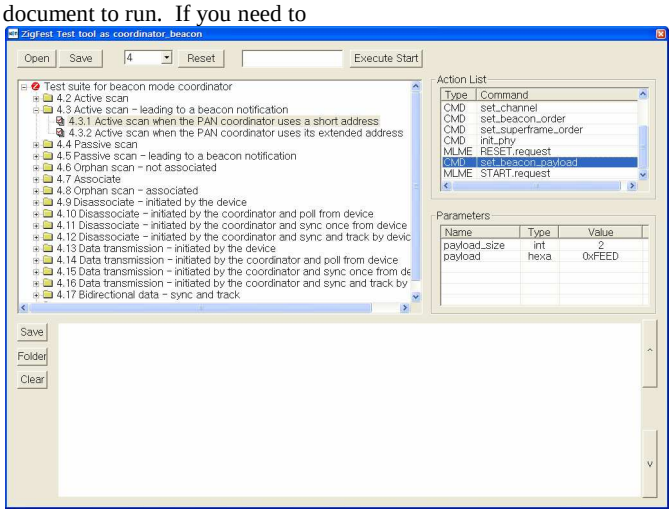

Figure 7. Test Application GUI

6. Conclusions

The main contribution of this paper is the proposal of a test procedure description language and framework for a layer of WSNs protocol stack. The test suite description language based on XML to define test conditions and test procedures in formal way and gives information to generate source code for exchanging messages and commands between test application and test driver. The test framework is a great help to verify correctness of the WSN protocol stack automatically.

We have shown that an application of our proposal is feasible by putting it into interoperability test for verifying a IEEE 802.15.4 MAC protocol implementation on ZigFest Spain 2006. Our test framework would make it possible for us to verify MAC protocol stack implementations without Network layer protocol as a higher layer of the target MAC layer. We dispute that the test framework allows for inspecting WSN protocol stack implementations on development stage step by step and building more robust WSN applications.

We plan to improve the performance of test framework by promising extension of the automated test and combining test result from several test frameworks to extend capabilities of our frameworks. The collected test result data from more than one test frameworks helps to make decision whether the target WSN protocol stack is interoperable or not. It is possible for the test framework to collect the test result data from there colleagues by means of connecting a group of the test applications of the test frameworks.

7. ACKNOWLEDGMENTS

This research is partially supported by the ubiquitous Computing and Network (UCN) Project, the Ministry of Information and Communication (MIC) 21st Century Frontier R&D Program in Korea. And this work is also partially supported by the Medium-term Strategic Technology Development Program funded by the Ministry of Commerce, Industry and Energy(MOCIE, Korea).

8. REFERENCES

[1] IEEE 802.15.4 Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), http://www.ieee802.org/15/pub/TG4.html

[2] TinyOS, http://www.tinyos.net

[3] ZigBee Alliance, http://www.zigbee.org

[4] J. Beutel, M. Dyer, R. Lim, C. Plessl, M. Wöhrle, M. Yücel and L. Thiele, “Automated Wireless Sensor Network Testing”. Proc. 4th International Conference on Networked Sensing Systems (INSS 2007), IEEE, Piscataway, NJ, June, 2007, page 303.

[5] Ian F. Smith, “Test automation for embedded products”, 2004.06

[6] TaeHyun Kim, SangShin Lee, JaeHo Kim, IlYeup Ahn, MinHwan Song, KwangHo Won, “Design of XML based interface definition language between embedded device and host server”, The 9th Conference on Next Generation Communication Software (NCS 2006), KICS, Pyoungchang, Republic of Korea, November, 2006

[7] W3C XML specification, http://www.w3.org/TR/REC-xml