SpecTalk: Conforming IoT Implementations to Sensor Specifications

Yi-Bing Lin1,3,*, Sheng-Lin Chou2, Ted C.-Y. Chang5 and Yun-Wei Lin6

1 School of Computing, National Cheng Kung University, Taiwan;
2 Information and Communication Research Laboratories, Industrial Technology Research Institute, Taiwan;
3 College of Humanities and Sciences, China Medical School, Taiwan;
4 College of Artificial Intelligence, National Yang Ming Chiao Tung University, Taiwan; liny@nctu.edu.tw
5 Quanta Computer, Taiwan; ted.chang@quantatw.com
6 College of Artificial Intelligence, National Yang Ming Chiao Tung University, Taiwan; jyneda@gmail.com

* Correspondence: liny@nctu.edu.tw;

Received: date; Accepted: date; Published: date

Abstract: Due to the fast evolution of Sensor and Internet of Things (IoT) technologies, several large-scale smart city applications have been commercially developed in the recent years. In these developments, the contracts are often disputed in the acceptance due to the fact that the contract specification is not clear, resulting in a great deal of discussion of the gray area. Such disputes often occur in the acceptance processes of smart buildings, mainly because most intelligent building systems are expensive and the operations of the sub-systems are very complex. This paper proposes SpecTalk, a platform that automatically generates the code to conform IoT applications to the Taiwan Association of Information and Communication Standards (TAICS) specifications. SpecTalk generates a program to accommodate the application programming interface of the IoT devices under test (DUTs). Then the devices can be tested by SpecTalk following the TAICS data formats. We describe three types of tests: self-test, mutual-test and visual test. A self-test involves the sensors and the actuators of the same DUT. A mutual-test involves the sensors and the actuators of different DUTs. A visual-test uses a monitoring camera to investigate the actuators of multiple DUTs. We conducted these types of tests in commercially deployed applications of smart campus constructions. Our experiments in the tests proved that SpecTalk is feasible and can effectively conform IoT implementations to TAICS specifications. We also propose a simple analytic model to select the frequency of the control signals for the input patterns in a SpecTalk test. Our study indicates that it is appropriate to select the control signal frequency such that the inter-arrival time between two control signals is larger than 10 times of the activation delay of the DUT.

Keywords: Acceptance test, Internet of Things (IoT), IoT specification, smart building, sensor, Taiwan Association of Information and Communication Standards (TAICS)

1. Introduction

Many smart applications have been developed with Internet of Things (IoT) technology [1] [2] in large-scale smart city constructions. However, contracts of intelligent application constructions are often disputed in the acceptance due to the fact that contract specifications are not clear. The higher the level of intelligence, the greater the controversy especially when the customers and the vendors that provide the solutions have different imaginations about the functionality of smart products. Such disputes often occur during the acceptance processes of smart buildings, mainly
because intelligent building systems are expensive and the operations of the sub-systems are very complex due to the fast evolution of IoT technology.

According to Allied Market Research [3], the smart building market will reach 35 billion USD by 2020, with a compound annual growth rate of about 30 per cent, driven by advances in the IoT and sensor networks [4]. A smart building typically includes different subsystems in the same building [5]. To effectively integrate and connect the subsystems, a major challenge is the deployment of the Information and Communication Technology (ICT) common trench to satisfy the needs for intelligent building system integration. These needs make it difficult for traditional contracts to standardize specific acceptance methods.

The classical method in contract law established a formula for quotation and acceptance in the 19th century, allowing buyers and sellers to reach a consensus on contracts. Formal tests of customer requirements and business processes are typically carried out at the time of acceptance to determine whether the system meets the acceptance criteria and to enable the customer to decide whether or not to accept the system [6]. Flexible intelligent systems contracted in the traditional way are likely to be controversial for misleading conduct and misrepresenting the right to accept. This phenomenon becomes more obvious as the intelligence elasticity of the products increases in the era of rapidly evolving ICT.

Interestingly enough, acceptance tests for complicated mobile systems have been well conducted. In mobile telecommunications systems, data formats and communication protocols are defined by standard organizations such as European Telecommunications Standards Institute (ETSI) and 3GPP. Following their specifications, developed systems can be systematically tested. For example, Testing and Test Control Notation (TTCN) has been used to transplant the layer-3 functions in 5G and LTE protocol stacks to carry out test instruments. In [24] an implementation scheme of terminal Radio Resource Management conformance test system was implemented with TTCN-3, and the feasibility of this scheme was verified by running the designed test cases. To speed up TTCN-3 test system testing, a tool was created to visualize large test systems [25][26]. This tool was use to analyze all standardized test suites available at www.ttcn3.org, finding circular dependency and unnecessary source files in almost every test suite.

Unfortunately, there are no comparable globally accepted specifications for smart applications. In [27], Cucumber and Selenium were used to automate testing activity for a system-level test on well-known platforms such as SOASTA (Android) and Ranorex (iOS). The authors pointed out the difficulties in establishing compatibility with all smart applications due to inconsistent data formats, noting that it is challenging to ensure compatibility with much variability of the application platforms.

Similarly, the study in [28] investigated model-based software tests by using the Selenium library with the Graph Walker tool, which outperforms traditional software tests where JUnit and TestNG libraries were used. Comparisons of these two methods were performed and reported on parameters of code line number, actual error detection rate, and test run time parameters. At the smart application level, model-based software tests cannot be conducted due to inconsistent data
Therefore, acceptance tests of intelligent systems (smart applications) are difficult because there are no globally accepted standards to specify these systems.

At present, intelligent system related standards, including building automation and energy-based air conditioning [7][19], have gradually involved advanced networking and IoT technologies. Since multiple systems with different communication protocols [8] [9] [10] and data formats [11] [12] are often developed in a building, the matter of how to integrate the various systems and collect the required information is a key factor in managing building intelligence. At this stage, the market has a variety of intelligent energy management systems, independent intelligent building subsystems and data format interfaces. It is difficult to incorporate various independent subsystems for overall monitoring and management, which affects the open and common development for future intelligent networking to heterogeneous services in smart buildings.

In light of the application situation and demand for intelligent building energy management system, Taiwan Association of Information and Communication Standards (TAICS) specifies the data format standard of intelligent building [19], which is applicable to the data exchange between the subsystems and the devices under the integrated system monitoring platform or the integrated operation center (IOC). However, most vendors are unable to study TAICS specifications carefully and provide Application Programming Interface (API) software that conforms to TAICS specifications. When customers request to follow TAICS standard, vendors may significantly increase the price, causing more controversy. Therefore, we design SpecTalk, an interface code generation system that makes it easy for the vendors to automatically interwork their intelligent systems to the TAICS specification-compliant platform for acceptance tests. SpecTalk does not involve data exchange within subsystems or devices. Vendors can easily integrate the APIs of their systems to the TAICS test system by using SpecTalk’s Graphical User Interface (GUI). The paper is organized as follows. Section 2 introduces TAICS; Section 3 proposes SpecTalk; Section 4 describes how SpecTalk is used to conduct three types of acceptance tests; Section 5 proposes an analytic model to select the frequency of the control signals in the input pattern of a IoT device test.

2. Taiwan Association of Information and Communication Standards

To highlight the state-of-the-art sensor technology in Taiwan, it is important to introduce the sensor standard efforts promoted by the Ministry of Economic Affairs (MOEA). In Taiwan, MOEA is in charge of the 5G/IoT development. For 5G/IoT specification promotion, the mission is carried out by Taiwan Association of Information and Communication Standards (TAICS) sponsored by MOEA.

TAICS is an industry organization founded in June 2015 with the members from industry, research and academia organizations in Taiwan. TAICS bridges the local industry with global standard initiatives/organizations by contributing the study results or consolidated consensus, and also develop the local standard or study report per request. TAICS is open to participation from all the companies/organizations with divisions in Taiwan. Seven technical committees (TCs) have been chartered focusing on

- Advanced Mobile Communication
In Taiwan, most solution providers deliver quick and cheap IoT application constructions (such as smart building, smart agriculture, smart manufacture, and so on) to the customers without conforming to any standards. To resolve this issue, This paper proposes SpecTalk, a tool that can automatically generate an interface for any IoT application to conform to the TAICS specifications. SpecTalk has been developed by Accton (one of the largest ICT companies in Taiwan, Revenue USD 1.85 Billion in 2020), and aims to become the standard interworking mechanism of TAICS. SpecTalk is being used to produce receiving reports for the following smart park constructions:

- Hsinchu International AI Smart Park (Fig. 11)
- Smart Campus of China Medical University in Shuinan Trade and Economic Park, Taichung (Fig. 12)

SpecTalk also aims to regulate agriculture related data for the Council of Agriculture, Taiwan. We will use SpecTalk to interwork with the specifications of the following international partners:

- The Telecommunication Technology Committee (TTC), Japan
- The Telecommunications Industry Association (TIA), USA
- Institute of Electrical and Electronics Engineers (IEEE), USA
- China Communications Standards Association (CCSA), China
- European Telecommunications Standards Institute (ETSI), EU
- The Association of Radio Industries and Businesses (ARIB), Japan
- Telecommunications Standards Development Society (TSDSI), India
- Malaysian Technical Standards Forum Bhd (MTSFB), Malaysia
- Telecommunications Technology Association (TTA), Korea

3. The SpecTalk Architecture and Procedures

All IoT-related specifications characterize an IoT device with the following fields as illustrated in Fig. 1: (1) device type or device model (DM; e.g., Generator); (2) device ID (e.g., GEN-{NNNN}) for a generator and the description for the device features (DFs) including (3) DF name in Chinese; (4) DF name (e.g., tankOilLevelStatus); (5) DF abbreviation (e.g., oilSts); (6) I/O (input or output); (7) DF parameter (e.g., status with the states {NORMAL, LOW}).
Fig. 1. The TAICS specification for “Generator”, the Chinese descriptions are translated to English as follows: 裝置 ID (device ID), 裝置類別 (device type), 短名稱 (DF name, see (1)), 監控標示 (DF type; see (2)), 預設單位/列舉值集 (parameter type; see (4)), 中文名稱 (Chinese name; see (5)), 長名稱 (Long Name; see (6))

The SpecTalk platform consists of five components: Device Feature Management (Fig. 2 (2)), SpecTalk Database (Fig. 2 (3)), Device Model Management (Fig. 2 (4)), Device Application Generation (Fig. 2 (5)) and Test Engine (Fig. 2 (8)).

Fig. 2. The SpecTalk approach to test an IoT device with TAICS specification.

From the specification of an IoT device (Fig. 2 (1)), we store its software configuration in the SpecTalk Database (Fig. 2 (3)). The configuration is used to automatically create the Sensor & Actuator application (SA) interface (Fig. 2 (6)) that binds the IoT device under test (DUT; Fig. 2
3.1. Construction of the SpecTalk Database

SpecTalk is a web-based platform, which can be operated in any computing device with a browser (for example, a smartphone). In this platform, the configurations of IoT devices are stored in the SpecTalk Database through a web-based Device Feature window. We first create and store all DFs of an IoT device using the Device Feature Management (Fig. 2 (2)). Consider the tank oil level status as an example. We first fill the DF name “oilSts-I” (Fig. 3 (1)), then we select the DF type as “IDF” (Fig. 3 (2)). Note that the DF name is appended with “-I” if it is an IDF, and is appended with “-O” if it is an output device feature (ODF). The category field is filled with the number of the specification TS-0022 (Fig. 3 (3)). There is one parameter of type “Boolean” to represent the status of the oil level (Fig. 3 (4)). We put extra information about the DF in the description window (Fig. 3 (5) and (6)). After the DF information filling is complete, the oilSts DF configuration is saved in the SpecTalk Database.

We create the DM of the IoT device using the Device Model Management by clicking “Device Model” (Fig. 4 (2)). Then we jump from Device Feature window (Fig. 4 (1)) to Device Model window (Fig. 4 (3)).
In the Device Model window, we select the DFs from the Input/Output Device Feature lists (Fig. 5 (1)). The Device Model Management automatically includes the selected DFs in the device model (Fig. 5 (2)). When we click the “Save” button, a dialog box pops up for filling the DM name (Fig. 5 (3)). Then the DM “Generator” is saved in the DM list of the SpecTalk Database.

3.2. Generation of the SA Interface

SpecTalk automatically generates the test procedures for acceptance of a contract in two steps. At Step 1, SpecTalk generates a Sensor & Actuator (SA) interface (Fig. 2 (6)). The vendor fills the API of the DUT (Fig. 2 (7)) into the SA interface. At Step 2, the vendor executes the SA interface code generated by SpecTalk, and the IoT DUT is connected to the Test Engine (Fig. 2 (8)). Then SpecTalk conducts acceptance test and automatically generate the test report (Fig. 2 (9)). The Test Engine is modified from an IoT application development platform called IotTalk [13].

By switching from Device Feature/Model window (Fig. 6 (1)) to the IoTtalk project window (Fig. 6 (2)), we create the Generator-Test project (Fig. 6 (3)). The Model list of the project window (Fig. 6 (4)) contains the model of the DUT, and the Generator provides test procedures for the acceptance testing.
6 (4)) provides the device model indexes stored in the SpecTalk Database. When we select “Generator” (Fig. 6 (5)), the Generator window pops up. The features of vendor’s generator are a subset of the Generator device model. Therefore, SpecTalk allows selection of the DF number to set up the actual configuration of the generator under test (Fig. 6 (6)). After we save the setups (Fig. 6 (7)), the graphical representation of the generator under test is shown in the project window (Fig. 6 (8)). We can click the DF icon (Fig. 6 (9)) and further set up the parameters of the DF, e.g., the max and the min values of the DF (Fig. 6 (10)).

![Fig. 6. Specifying the generator under test.](image)

![Fig. 7. Automatic generation of the codes for the DA and the SA interface.](image)
When we complete the IoTtalk project (Fig. 7 (1)), SpecTalk automatically generates the code for Device Application (DA; Fig. 7 (2)) and the incomplete code for Sensor & Actuator Application (SA) interface (Fig. 7 (3)). Then the vendor fills the API of the IoT DUT to complete the code for the SA interface.

The DA is responsible for communication between the DUT and the Test Engine. The protocols include HTTPS and MQTT. The lower-layer communications can be Ethernet, WiFi, LTE or 5G.

The code for the SA interface is listed below:

```python
01. import time, random, requests
02. import DA
03. import Your-API
04. ServerURL = 'https://DomainName'
05. Reg_addr = 'GEN-0012'
06. DA.profile['dm_name'] = 'Generator'
07. DA.profile['df_list'] = ['oilSts-I', 'battSts-I', 'ventSts-I']
08. DA.profile['d_name'] = 'Your-Device-Name'
09. DA.register(ServerURL, Reg_addr)
10. while True:
11.    try:
12.        oilSts_data = Your-oilSts-function
13.        DA.push ('oilSts-I', oilSts_data)
14.        battSts_data = Your-battSts-function
15.        DA.push ('battSts-I', battSts_data)
16.        ventSts_data = Your-ventSts-function
17.        DA.push ('ventSts-I', ventSts_data)
18.    except Exception as e:
19.        print(e)
20.    if str(e).find('mac_addr not found:') != -1:
21.        print('Reg_addr is not found. Try to re-register.')
22.        DA.register (ServerURL, Reg_addr)
23.    else:
24.        print('Connection fails.')
25.        time.sleep(1)
26.        time.sleep(0.2)
```

In the above program, the black lines without yellow marks are a general template for any IoT device to follow the TAICS specification. The codes with yellow marks are generated through the steps described in Fig. 6. The red parts are the codes to be typed in by the vendor (Fig. 7 (4)) through the GUI for the SA interface (Fig. 8). This GUI tailored for the IoT DUT is automatically generated by SpecTalk. Specifically, the values of the fields “Name”, “API”, “oilSts”, “battSts”, and “ventSts” are inserted into the code for the SA interface by the Device Application Generation (Fig. 8 (1)). At Line 5, “your-API” provided by the vendor is used to access the functions for the IoT DUT.
The complete code for the SA interface (Fig. 8 (2)) is saved in the file “File-for-SA-code” specified in Fig. 8 (3). Note that when a DF is created in Fig. 3, the DF is used to build the device model in TAICS. Specifically, a DM and its DFs are created for TAICS specification, and every “real” vendor device is a subset of the TAICS DM and may not have all device features specified in the TAICS DM. The actual features of the DUT are actually mapped to SpecTalk through the procedure in Fig. 8. In the current implementation, Python, Java, Javascript, and C versions the SA interface codes are available in SpecTalk.

Fig. 8. Binding the DUT and SpecTalk by creating the SA interface of “Generator”.

At the side of the IoT DUT, the vendor executes the SA interface code (Fig. 9 (1)) that registers the device to the Test Engine (Fig. 9 (2)). The Test Engine retrieves the configuration of the IoT DUT from SpecTalk Database (Fig. 9 (3)), and executes the test procedures to generate the test report.

Fig. 9. The SpecTalk test process.

A test procedure can be easily created as an IoTtalk project. A simple example is described as follows. From the Model list (Fig. 10 (1)), we select “Generator” and “Display” (Fig. 10 (2)). Display prints out whatever data it receives. By dragging a line between “ventSts-I” (Fig. 10 (3))
to “Controller-O”, the data will be delivered by this link (Join 2). To make sure this test procedure behaves correctly, we click the “Simulation” button (Fig. 10 (4)). Following random number generation simulation [15][16] or trace-driven emulation [14], the test results are shown in the Monitor window (Fig. 10 (5)).

**Fig. 10.** A simple SpecTalk test procedure.

### 3. The SpecTalk Acceptance Tests

Three types of SpecTalk tests can be developed following the test procedure creation process similar to the one in Fig. 10.

- Self-test involves the IoT DUT only
- Mutual-test involves two or more IoT DUTs
- Visual-test involves the IoT DUTs and a monitoring camera

In this section, we provide several examples to show how SpecTalk works.

#### 3.1. Self-Test

Hsinchu County is developing the Hsinchu International AI Smart Park (Fig. 11), which is managed by Accton. SpecTalk will be used by Accton to verify all smart applications in the park. Most of them will be achieved through self-tests. Another example is the smart campus of China Medical University (CMU) in Shuinan Trade and Economic Park, Taichung (Fig. 12). In this subsection, we use the generator and the water-cooled chiller of the CMU smart campus as examples to show how to conduct self-test.
In the IoTtalk window, we create a project “Self-Test”. Besides the three IDF s we created in Fig. 6 (see Fig. 13 (5), (7) and (9)), we also create two output device features (ODFs): the power switch of the generator “Gen-Switch-O” (Fig. 13 (2)) and the power switch of the smoke exhaust ventilator “Vent-Switch-O” (Fig. 13 (4)). In the emulation mode, SpecTalk automatically generates the control signals through IDF s Switch-I1 and Switch-I2 (Fig. 13 (1) and (3)) to turn Gen-Switch-O and Vent-Switch-O on and off. Switch-I1 turns on the generator to see if both the battery level and oil consumption increase. The validation is performed by Test Output through battSts-O, oilSts-O and Switch-O1 (Fig. 13 (6), (8) and (11)). Switch-I2 tests if ventSts-I correctly indicates the ventilator operation. The validation is performed by Test Output through ventSts-O and Switch-O2 (Fig. 13 (10) and (12)). In the SpecTalk emulation, “Test Inputs” automatically generates sequences of test signals through random number generators. We can also use pre-stored test vectors as inputs sent to the Test Engine. The test procedure is
developed manually through the IoTtalk GUI (which minimizes the programming effort) at the first time. Then the procedure is saved in SpecTalk to be reused for similar test cases in the future. Suppose that the oil consumption rate is given in the vendor’s data sheet. If the tank is full, then it will be consumed below the threshold in, for example, 50 minutes. Then the test procedure will check if oilSts-I gives an alert after 50 minutes.

![Self-Test Configuration](image)

*Fig. 13. Self-test configuration for the generator.*

Another example is the self-test configuration for water-cooled chiller. In the TAICS specification, there is a two-parameter IDF “chWF-I” to indicate the water flow of the chiller. The IDF chWF-I is created through the Device Feature Management like what we did in Fig. 3 except that we set the parameter number to be 2 (Fig. 14 (1)). The first parameter is chWQ (Fig. 14 (2)), a sensor to measure the water flow rate in liter per minute (LPM). The second parameter is chSVPos (Fig. 14 (3)), a sensor to measure the position of water supply valve in percentage. The ODF is chSVCtrl-O (Fig. 14 (4)), the position control switch of the water supply valve. In our test, the values of chSVPos and chSVCtrl should be the same in the normal operation.
Fig. 14. The IDF and the ODF for the water-cooled chiller.

The test procedure for the water-cooled chiller is configured through the WCchillerTest project. In this project, SpecTalk automatically generates the control signals through the test input TESTchSVCtrl-I (Fig. 15 (1)) to turn chSVCtrl-O (Fig. 15 (2)) on and off. In Fig. 15 (4) and (5), we check if the position of the water level and the water flow rate measured by chWF-I (Fig. 15 (3)) are consistent.

Fig. 15. Self-test configuration for the water-cooled chiller in the smart building.

3.2. Mutual-Test

A mutual-test consists of at least one sensor and one actuator under test [17]. An example is the window control where the CO2 sensor (Fig. 16 (a)) controls an electric window (Fig. 16 (b)) to enhance the air quality in a room [18].
The test configuration is illustrated in Fig. 17. When the CO2 level is too high, the window is triggered to open through the control path (3)→(2). Therefore, through the test output (Fig. 17 (4)) we observe if the CO2 level can be limited below a value. Also, in the emulation mode, SpecTalk can generate a designed sequence to turn on and off the window through Switch-I (Fig. 17 (1)). Then when Switch-O is “1”, Test Output checks if the CO2-O value decreases. Fig. 17 also shows that a simple test procedure can be easily created through a smartphone.

Another example is the mutual test for a greenhouse in the Bao Mountain, Hsinchu. In the greenhouse, the fan and the drippers (Fig. 18 (1) and (2)) are controlled by the farming sensors (Fig. 18 (3)).
We create a mutual-test configuration to investigate if the farming sensors and the actuators interact correctly. The farming sensors include those for soil humidity, soil temperature, and wind speed (Fig. 19 (5), (7) and (9)). The actuators include the fans and the drippers (Fig. 19 (2) and (4)). The test inputs (Fig. 19 (1) and (3)) control the actuators and serve as ground truth to the test cases.

**Fig. 18.** The sensors and actuators of the greenhouse in the Bao Mountain, Hsinchu.

**Fig. 19.** Mutual Test Configuration for the greenhouse.
When Switch-I1 triggered the drippers at 12am in Feb. 4, 2019 (Fig. 20 (1)), the relative humidity should increase and the temperature should decrease. The time series charts indicated that both the humidity and the temperature decreased. Therefore, we conclude that the dripper and the temperature sensor were normal and the humidity sensor failed. Similarly, when Switch-I2 turned on the fan at 11:25am and turned it off at 2:24pm (Fig. 20 (2) and (3)), the wind speed meter measured the correct wind speed, and therefore, both the fan and the wind speed meter were normal.

![Time series charts of the greenhouse sensors](image)

**Fig. 20.** Time series charts of the greenhouse sensors.

### 3.3. Visual-Test

When an actuator cannot be evaluated by a self-test or a mutual-test, we can observe its behavior visually. For example, when we use smartphone to generate a sequence of test pattern (Fig. 21 (1)) to lock and unlock a door shown in Fig. 21 (2), we do not know if the door is actually locked or not. Therefore, during the test period, we may use the monitoring camera (Fig. 21 (3); typically installed in the ceiling) to detect the lock status. SpecTalk uses YOLO (Fig. 21 (4)).
21 (4)) for detection [20], and the results are compared with the test pattern at the test procedure (Fig. 21 (5)).

Another visual test was conducted in a fountain of the Ming Der High School, Taichung. In this example, the monitoring camera is used to test multiple actuators such as sprinklers (Fig. 22 (1)) and street lamps (Fig. 22 (3)). The test inputs can be automatically generated or manually triggered through the control board (Fig. 22 (2) and (4)). When we turn on the sprinkler control (Fig. 22 (5)), the YOLO Recognition is expected to detect four water columns (Fig. 22 (6)).

The SpecTalk test configuration is illustrated in Fig. 23 (a), which involves (3)→(4), (3)→(8), (3)→(10), (3)→(12), (3)→(14), (7)→(8), (9)→(10), (11)→(12), and (13)→(14). The lamp test is conducted at night (Fig. 23 (b)), where five lamps are expected to be detected by YOLO. The SpecTalk test configuration involves (1)→(2), (1)→(6) and (5)→(6). In the sprinkler test, YOLO reports the statuses of individual sprinklers. Alternatively, in the lamp test, YOLO does not report the status of a specific street lamp.

Visual-tests are particularly useful for retrieving data from analog meters as illustrated in Fig. 24. In visual-tests, YOLO already knows the specific spots under test from the monitoring camera, and therefore can accurately detect the changes on these spots. We can conduct multiple tests simultaneously in one video screen, for example, the sprinkler test and the lamp test in the Ming Der High School (Figs. 22 and 23).
Fig. 22. The fountain in the Ming Der High School, Taichung.

(a) SpecTalk test configuration    (b) Lamp test
Fig. 23. Visual-test configuration for a fountain in the Ming Der High School, Taichung.

Fig. 24. Visual-tests for analog meters.

4. Selection of the Frequency of the Control Signals in the IoT Device Test

While one may attempt to quickly complete an IoT device test by selecting a high frequency of the control signal generation for the input pattern. Such selection may result in incorrect test results. Consider the test in Fig. 19 where the dripper actions are triggered by the control signals issued from Switch-I1. Suppose that the dripper motor receives the “on” signal at $\tau_0$, and the motor is turned on at $\tau_*$, then the activation delay of the motor is $t = \tau_* - \tau_0$. The characteristics (the mean and the variance) of $t$ for the actuator may be provided in the vendor’s data sheet. Otherwise we need to measure the operation delays. Such measurement can be simply conducted using the Network Time Protocol. If Switch-I1 continues to send the control signals at $\tau_1, \tau_2, \ldots, \tau_n, \tau_{n+1}$, where $\tau_n \leq \tau_* \leq \tau_{n+1}$, then the signals arrive at $\tau_2, \ldots, \tau_n$ will be ignored and the motor will execute the instruction arrives at $\tau_{n+1}$. Therefore, the inter-arrival times of the control signals should be longer than $t$ to avoid producing wrong test results.

Fig. 25. The timing diagram.
Let $t_i = \tau_i - \tau_0$. We typically generate the Poisson input patterns. That is, the inter-arrival times of the control signals have an Erlang distribution with the expected value $E[t_i] = i/\lambda$. Then the density function $g(t_i)$ has an Erlang density function

$$g(t_i) = \frac{\lambda^i t_i^{i-1}e^{-\lambda t_i}}{(i-1)!}$$

Let $t$ have a general density function $f(t)$, then

$$\text{Pr}[t > t_n] = \int_{t=0}^{\infty} f(t) \int_{t_n}^{\infty} \left[ \frac{\lambda^n t_n^{n-1}e^{-\lambda t_n}}{(n-1)!} \right] dt_n dt$$

$$= \int_{t=0}^{\infty} f(t) \sum_{i=0}^{n-1} \left[ \frac{\lambda^i t_i^{i-1}e^{-\lambda t_i}}{i!} \right] dt$$

$$= \sum_{i=0}^{n-1} \left( \frac{\lambda^i}{i!} \right) \int_{t=0}^{\infty} t^i f(t) e^{-\lambda t} dt$$

Let $f^*(s) = \int_{t=0}^{\infty} f(t) e^{-\lambda t} dt$ be the Laplace Transform of $f(t)$. Then from frequency-domain derivative of the Laplace Transform, Eq. (1) is re-written as

$$\text{Pr}[t > t_n] = \sum_{i=0}^{n-1} \left( -\frac{\lambda^i}{i!} \right) \left[ \frac{f^*(s)}{s} \right] \bigg|_{s=\lambda}$$

From Eq. (2), we have

$$\text{Pr}[t_n < t < t_{n+1}] = \left( -\frac{\lambda^n}{n!} \right) \left[ \frac{f^*(s)}{s} \right] \bigg|_{s=\lambda}$$

Eq. (3) gives the probability that $n$-1 consecutive control signals are ignored by the IoT DUT. The probability that no control signal is lost is $\text{Pr}[t < t_2]$, which is expressed as

$$\text{Pr}[t > t_2] = 1 - \text{Pr}[t < t_2]$$

$$= 1 - f^*(\lambda) + \lambda \left[ \frac{f^*(s)}{ds} \right] \bigg|_{s=\lambda}$$

If $f(t)$ is a Gamma density function with the shape parameter $\alpha$ and the scale parameter $\beta$, then its Laplace transform is

$$f^*(s) = \frac{\beta^\alpha}{(s + \beta)^\alpha}$$
The Gamma distribution is considered because this distribution is often used in computer and communication modeling [22][23]. Substitute Eq. (5) to Eq. (3) to yield

\[
\Pr[\tau_n < t < \tau_{n+1}] = \left(\frac{\alpha + n}{n}\right) \left(\frac{\lambda}{\lambda + \beta}\right)^n \left(\frac{\beta}{\lambda + \beta}\right)^a 
\]  

(6)

Substitute Eq. (5) to Eq. (4) to yield

\[
\Pr[t > t_2] = 1 - \frac{\beta^a}{(\lambda + \beta)^a} - \frac{a\lambda\beta^a}{(\lambda + \beta)^{a+1}} 
\]  

(7)

In the Gamma distribution, a small \(\alpha\) implies a large variance of \(t\), and the operation of the DUT is not stable. We have

\[
\lim_{\alpha \to 0} \frac{\beta^a}{(\lambda + \beta)^a} = 1 \quad \text{and} \quad \lim_{\alpha \to 0} \frac{a\lambda\beta^a}{(\lambda + \beta)^{a+1}} = 0
\]

Therefore, from Eq. (7),

\[
\lim_{\alpha \to 0} \Pr[t > t_2] = 0 
\]  

(8)

Eq. (8) says that if the activation time of the DUT has high variance, then the control signals are likely to be lost during the test. For the mean value analysis [21], we assume that \(\alpha = 1\), then \(\text{E}[t] = 1/\beta\) and Eq. (6) is re-written as

\[
\Pr[t > t_2] = 1 - \frac{\beta}{\lambda + \beta} - \frac{\lambda\beta}{(\lambda + \beta)^2}
\]

\[
= \left(\frac{\lambda}{\lambda + \beta}\right)^2 = \left(\frac{\text{E}[t]}{\text{E}[t_1] + \text{E}[t]}\right)^2 
\]  

(9)

If we select \(\text{E}[t_1] = 10\text{E}[t]\), then \(\Pr[t > t_2] > 0.99\), and almost no control signals will be lost. Note that in Eq. (6), if \(\alpha \to 0\),

\[
\lim_{\alpha \to 0} \Pr[\tau_n < t < \tau_{n+1}] = \left(\frac{\lambda}{\lambda + \beta}\right)^n
\]

which is a general form of Eq. (8). When \(\alpha \to \infty\), Eq. (6) says that \(\lim_{\alpha \to \infty} \Pr[\tau_n < t < \tau_{n+1}] = 0\), and \(\lim_{\alpha \to \infty} \Pr[t > t_2] = 1\). The above analysis indicates that it is appropriate to select the control signal frequency such that the inter-arrival time between two control signals is larger than 10 times of the activation delay of the DUT. Noted that the residual probability of wrong IoT device communication sequence still exists. Such probability is affected by the variance of operation (switching) time of the controlled actuator.
5. Conclusions

This paper proposed SpecTalk, a platform that automatically generates the code to conform IoT applications to TAICS specifications. Specifically, SpecTalk generates a program to accommodate the API of the IoT DUTs. Then the device can be tested by SpecTalk following the TAICS data formats. We described three types of tests: self-test, mutual-test and visual test. A self-test involves the sensors and the actuators of the same DUT. A mutual-test involves the sensors and the actuators of different DUTs. A visual-test uses a monitoring camera to investigate the actuators of multiple DUTs. We conducted these types of tests in Hsinchu International AI Smart Park and the greenhouse of the Bao Mountain, Hsinchu. We also exercise acceptance tests of the smart buildings China Medical University in Shuinan Trade and Economic Park and the fountain of the Ming Der High School, Taichung. Our experiments in the tests proved that SpecTalk is feasible and can effectively conform IoT application implementation to TACIS specifications. We also proposed a simple analytic model to select the frequency of control signals for the input patterns in a SpecTalk test. Our study indicated that it is appropriate to select the control signal frequency such that the inter-arrival time between two control signals is larger than 10 times of the activation delay of the DUT. In this paper, we conducted primary derivation of the probability aiming to produce simple close form. Right now, we do not have enough practical examples to investigate a secondary detailed derivation (which may result in tedious non-close form equations). The secondary derivation will be considered as our future work.

Based on SpecTalk we can interwork TACIS with all smart application conforming to the specifications developed by TTC (Japan), TIA (USA), IEEE (USA), CCSA (China), ETSI (EU), ARIB (Japan), TSDSI (India), MTSFB (Malaysia) and TTA (Korea). For example, we can use the procedure in Fig. 8 to connect to all devices that conform to ETSI TS 103 410-4 specification. Through this simple process, TACIS data formats can be transformed to the data formats specified by other standard organizations.

References

1. oneM2M, Functional Architecture, TS-0001 http://www.onem2m.org/technical/published-drafts
2. oneM2M, Base Ontology, TS-0012, http://www.onem2m.org/technical/published-drafts
3. Allied Market Research. Smart Building Market, Allied Market Research 
https://www.alliedmarketresearch.com/smart-building-market
7. BACnet - A Data Communication Protocol for Building Automation and Control Networks, 
8. MODBUS, MODBUS Application Protocol Specification V1.1b3, 
http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf
9. IETF, Hypertext Transfer Protocol – HTTP/1.1, IETF RFC 2616, Internet Engineering Task Force,
19. TACIS. Data format standard and test specification for intelligent building energy management system v2, TACIS TS-0022, 2021.
24. Chen; F. and Zhou; G., RRM conformance testing in TD-LTE system based on TTCN-3, International Conference on Information and Automation (ICIA), 2013
27. Santos; I., Filho; J. C. C., Souza; S. R. S., A survey on the practices of mobile application testing, Conferencia Latinoamericana En Informatica (CLEI), 2020

© 2021 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).