IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON NOVEL LEARNING APPLICATIONS AND SERVICES FOR SMART CAMPUS

Received March 26, 2018, accepted April 23, 2018, date of publication May 2, 2018, date of current version June 5, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2832222

CampusTalk: IoT Devices and Their Interesting Features on Campus Applications

YI-BING LIN[1], (Fellow, IEEE), LI-KUAN CHEN[1], MIN-ZHENG SHIEH[2], (Member, IEEE), YUN-WEI LIN[1], (Member, IEEE), AND TAI-HSIANG YEN[1]

[1]Department of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan
[2]Information Technology Service Center, National Chiao Tung University, Hsinchu 30010, Taiwan

Corresponding author: Yi-Bing Lin (liny@cs.nctu.edu.tw)

This work was supported in part by the Ministry of Science and Technology under Grant 106-2221-E-009 -006 and Grant 106-2221- E-009 -049 -MY2, in part by the Ministry of Education through the SPROUT Project-Center for Open Intelligent Connectivity, National Chiao Tung University, Taiwan, in part by Academia Sinica under Grant AS-105-TP-A07, and in part by the Ministry of Economic Affairs under Grant 106-EC-17-A-24-0619.

ABSTRACT Internet of Things (IoT) allows interactive learning of students by inspiring their innovation everywhere on the campus. This paper proposes CampusTalk that provides convenient access to cyber and physical devices through the web technology. In this way, the campus applications can be accessed anywhere in the world through any computing device with a display and a browser without installing any mobile app. CampusTalk unifies the concepts of automatic creation of ''mirror device features'' (simulated counterpart) and talkPal device features, which are nicely demonstrated through several applications on campus. The core application of CampusTalk is SmartPhoneTalk that allows other applications to be accessed by students through their smartphones without installing any mobile apps. We describe three CampusTalk applications, including MusicTalk, PingPongTalk, and SkeletonTalk, to demonstrate cyber and physical interaction with SmartPhoneTalk. After the IoT devices of these applications have been developed and are accommodated in CampusTalk, the students can use the CampusTalk GUI to connect them with various combinations for innovative applications without extra programming effort.

INDEX TERMS Art design, campus applications, cyber and physical interaction, Internet of Things.

I. INTRODUCTION

Internet of Things (IoT) has become very popular because it provides interconnection of uniquely identifiable computing devices within the existing Internet infrastructure [12]–[14]. IoT applications have been successful for B-to-B services, but have not been widely deployed for sustainable commercial B-to-C services. We found that university campus provides excellent sustainable service trial environment for both B-to-B and B-to-C services. Particularly, IoT allows interactive learning of students by expiring their innovation everywhere on the campus. To integrate IoT with learning environments is not trivial, especially if we want to create the environments outside the classrooms. To promote smart campus, National Chiao Tung University (NCTU) is deploying several IoT-based smart campus applications including temperature/PM2.5 detection (Figure 1), parking (Figure 2), emergency button, and dog tracking. These applications are created based on IoTtalk [7]–[10], an IoT application management platform that can be installed on top of IoT protocols such as OpenMTC [5], AllJoyn [6], OM2M [11] and an arbitrary proprietary protocol.

We propose CampusTalk by tailoring IoTtalk for campus applications, which is designed such that the IoT devices and network applications are modularized and can be conveniently reused through the graphical user interface (GUI). Therefore, the students with programming ability can easily create IoT innovation with new applications.

CampusTalk is also designed with a friendly GUI that allows students of non-computer science major to create their innovations without programming. Furthermore, the students can use their smartphones to access IoT-based smart campus services without installing any mobile apps. To achieve the above goals, the design and implementation of CampusTalk is not trivial. The remainder of this section describes how these goals are achieved.

A. CampusTalk ARCHITECTURE

Figure 3 illustrates the CampusTalk functional blocks, which consists of the IoTtalk server (Figure 3 (a)) [9], [10], the IoT devices (Figure 3 (b) and (c)), and possibly a smartphone (Figure 3 (d)) that serves as the gateway for the IoT devices [7]. The IoTtalk engine (Figure 3 (3))

FIGURE 1. PM2.5 detection in NCTU campus. (a) The NCTU campus map with 11 PM2.5 sensors and 2 LoRa gateways. (b) An outdoor PM2.5 sensor device.

FIGURE 2. Smart parking in NCTU campus.

interacts with the Device Application (DA; see Figure 3 (2)) to deliver/retrieve the IoT data. The IoTtalk GUI (Figure 3 (1)) provides a friendly web-based user interface to quickly establish connections and meaningful interactions among the IoT devices (to be elaborated). Through this GUI, a user instructs the IoTtalk engine to create or set up device features, functions, and connection configurations.

The DA is responsible for connecting IoT devices to the IoTtalk server, which consists of two software components. The Device Application to Network (DAN; see Figure 3 (4)) and the Device Application to IoT Device (DAI; see Figure 3 (5)). The DAI is device dependent, which communicates with the IoT device following the message format specified by the the IoT device application (IDA; Figure 3 (6)). The IDA implements the sensor and/or actuator software to be executed in the IoT device hardware. For a sensor IDA, the DAI may also implement sensor fusion algorithms (such as Kalman Filter and Extended Kalman Filter, sensor data storage and sensor power management) to manipulate the new data obtained from the IDA. The DAN communicates with the IoTtalk server for IDA registration and data exchange. The connection is established with wireline or wireless technologies (LTE, NB-IoT or WLAN). When an IoT device attaches to CampusTalk, the DAN initiates the registration procedure to inform the IoTtalk server of this connection. After the registration, a corresponding network application assigned to the IoT device is executed by the IoTtalk server. In the current implementation, the IoTtalk server provides HTTP-based RESTful application programming interfaces (APIs) or MQTT APIs for the DA to deliver the IoT data. To connect the DA to an IoTtalk server with other protocols, we only need to modify the DAN. The DAI and IDA are not affected.

The DAN and the DAI of a DA are always co-located in the same hardware. There are two scenarios to place the DA and the IDA. In both scenarios, the DANs are the same and

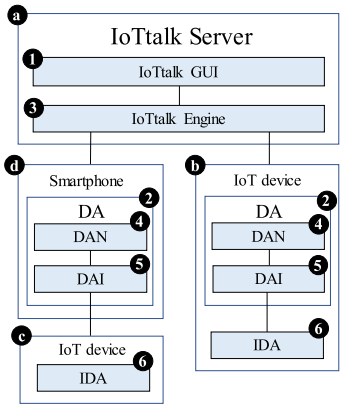


FIGURE 3. The CampusTalk functional diagram.

can be reused. In the internal DA scenario, both the DA and the IDA are installed in the IoT device (Figure 3 (b)). In the external DA scenario, the DA resides in, e.g., a smartphone (Figure 3 (d)), and the IDA is located at a separate IoT device (Figure 3 (c)). In this scenario, the DAI typically communicates with the IDA by using Bluetooth. We note that a smartphone itself can be an IoT device following the internal DA scenario. With the DAN, DAI and IDA structure, we effectively modularize the software components of the IoT devices, and can easily reuse these components to speed up the creation of IoT devices for IoTtalk applications.

B. DEVICE FEATURES

To allow students to easily create their applications, there must be a simple way to manipulate sensors and actuators. Also, some IoT devices must be automatically created. To achieve above goals in CampusTalk, every IoT device is characterized by its functionalities or ''device features''. A device feature (DF) is a specific input or output ''capability'' of the IoT device. The input device features (IDFs) can be sensors (such as a temperature sensor, an accelerator or a PM2.5 sensor) or control mechanisms (such as a keyboard, a button or a switch). The output device features (ODFs) are actuators such as a display screen, a speaker and so on. CampusTalk automatically generates a network application for every IoT device connected to the IoTtalk server. When the values of the IDFs are updated, an IoT device informs the network application to take some actions, and the network application sends the result to the ODF of the same or another IoT device to affect that output device. In other words, the network application describes how IoT devices interact with each other through their device features.

A DF can be physical or cyber (virtual). A cyber DF is implemented as an animation displayed in the standard screen of a computer or a mobile device. A cyber ODF can be shown in an arbitrary display hardware. On the other hand, a cyber IDF must be shown in a touch-screen display so that the user can give inputs to the IDF. An IoT device may have both IDFs and ODFs. For each IoT device, we group its IDFs in a subset called the input device. Similarly, the ODF subset is

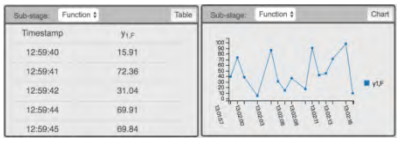


FIGURE 4. Two formats of display talkPal for Acceleration IDF (the y axis).

called the output device. If all DFs of a device are physical (cyber), then it is called a physical (cyber) device. For every IDF, CampusTalk automatically creates a ''simulated'' IDF (a mirror IDF) that uses a random number generator to simulate the values produced by the IDF.

For every DF, CampusTalk automatically creates a ''talkPal'' DF. The talkPal DF for an IDF is a display ODF. This talkPal ODF illustrates the data produced by the IDF in the table (text) format or in the chart (curve) format (Figure 4). The talkPal DF for an ODF is a cyber remote control IDF (two examples will be illustrated in Figures 9 and 25 (a)). The talkPal DF concept guarantees that every IoT device connect to the IoTtalk sever can talk to ''someone''. That is, an IDF talks to the user through its display talkPal ODF. The user talks to an ODF through its remote control talkPal IDF. The display talkPal ODF is a convenient mechanism that allows the students to test and debug the IDFs with little or without any programming effort. Therefore, CampusTalk supports the IoT applications with cyber and physical interaction.

CampusTalk provides convenient access to cyber devices through the web technology. In this way, the cyber devices can be accessed anywhere in the world through any computing device with a display and a browser without installing any application software. In [9], we have shown that all cyber remote controllers (to actuators) can be automatically created as web pages. Most IoT solutions have focused on connecting physical devices [4]–[6], and some interactive designs focused on cyber devices (but do not consider them from the IoT viewpoints) [3]. None of them consider the concepts of automatic creation of ''mirror device features'' (simulated counterpart) and talkPal device features. CampusTalk unifies the above concepts and implementations, which will be demonstrated through several applications on campus. Development of the IoT devices for these applications requires skillful programing effort. After these IoT devices have been created and are accommodated in CampusTalk, the students can use the GUI to connect them in different combinations for various innovative applications. This paper is organized as follows. Section 2 introduces SmartPhoneTalk that turns a smartphone into an IoT device without installing any mobile app. With SmartPhoneTalk, every student and any visitor in NCTU can access smart campus applications at anytime and anywhere. Section 3 describes MusicTalk that allows a large number of students to participate in big campus events, where their smartphones serve as musical glow sticks. Section 4 uses PingpongTalk as an example for sport activities as well as Physics lecturing. Section 5 demonstrates SkeletonTalk for interactive design to inspire students in the art class.

FIGURE 5. The web-based device application for smartphone.

II. SmartPhoneTalk

Most students have smartphones that can serve as input and/or output devices to access campus applications. Many IoT approaches utilizing smartphones require installation of mobile apps in the smartphones. On the other hand, campus applications using CampusTalk do not need to install any app software, even if the sensors of the smartphones are involved. To our knowledge, SmartPhoneTalk is the first approach that utilizes the sensors of smartphones without the need of mobile app installation. With our approach, the students with any smartphones can access NCTU campus applications at anytime and anywhere. In CampusTalk, ''Smartphone'' is an input device with three IDFs Acceleration, Gyroscope and Orientation, where these IDFs are sent to the IoTtalk server through a web-based DA. As shown in Figure 5 (1), the user opens the browser in the smartphone, e.g., Chrome, to access the web page and directly execute this DA without extra application installation. The values of the three IDFs are shown in the screen of the smartphone. The Acceleration IDF (Figure 5 (2)) provides the acceleration values in three axes (x, y, z) measured by the smartphone. The Gyroscope IDF (Figure 5 (3)) provides the rates (α, β, γ) at which the smartphone rotates around three axes (x, y, z). The Orientation IDF (Figure 5 (4)) provides the motion (α, β, γ) from the physical orientation in three axes (x, y, z) of the smartphone, and is expressed in degrees ranging from 0 to 360.

In our implementations the values of these IDFs are obtained from the built-in inertial measurement unit of the smartphone through the JavaScript Web APIs ''DeviceMotionEvent'' and ''DeviceOrientationEvent'' [2]. The JavaScript code of the web-based Smartphone DA is listed below.

```
Line 1   window.addEventListener('devicemotion', function(event) {
Line 2       var Ax = event.accelerationIncludingGravity.x;
Line 3       var Ay = event.accelerationIncludingGravity.y;
Line 4       var Az = event.accelerationIncludingGravity.z;
Line 5       var Ga = event.rotationRate.alpha;
Line 6       var Gb = event.rotationRate.beta;
Line 7       var Gg = event.rotationRate.gamma; }
Line 8   window.addEventListener('deviceorientation', function(event) {
Line 9       var Oa = event.alpha;
Line 10      var Ob = event.beta;
Line 11      var Og = event.gamma; }
Line 12  function Update() {
Line 13      Push('Acceleration', [Ax, Ay, Az]);
Line 14      Push('Gyroscope', [Ga, Gb, Gg]);
Line 15      Push('Orientation', [Oa, Ob, Og]);}
Line 16  setTimeout(Update, 250);
```

To obtain acceleration and gyroscope values, the IDA registers an event listener (Line 1) to start receiving the changes

FIGURE 6. VR experience with smartphones at the student activity center.

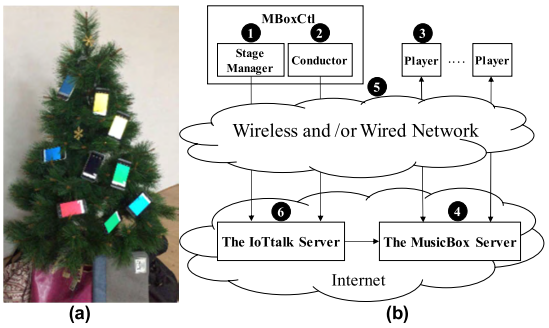


FIGURE 7. MusicTalk in a student dorm. (a) Singing tree. (b) The network architecture.

of Acceleration and Gyroscope. Lines 2-4 read the values for Acceleration. Lines 5-7 read the values for Gyroscope. Line 8 registers an event listener to receive the Orientation changes. Lines 9-11 obtain the current orientation of the smartphone.

After the values of Acceleration, Gyroscope, and Orientation are obtained, the DA periodically (e.g., for every 250ms) pushes those values to the IoTtalk server (Lines 12-16). Then the student can use these three IDFs to arbitrarily control other ODFs.

In CampusTalk, the sensors of smartphones have been intensively used for games and physics experiments carried out on NCTU campus. For example, NCTU has developed an VR game called "A Boy Fighting Devil Party" (which allows the students to shake their mobile phones (to produce the changes for the *Acceleration* IDF) to enhance, e.g., the sandy wind effect in the game (Figure 6). The audience experiencing the game loved the effects.

Note that some sensors or actuators such as flashlight and sounds are not standard, and some handset manufacturers do not allow access to these features through web pages. The rest of this paper will show how a smartphone is used in the campus IoT applications without installing any mobile app.

III. MusicTalk

A CampusTalk device itself can be a system that controls multiple IoT devices. An example is MusicTalk, a musical glow stick application used in big events on campus (e.g., the commencement ceremony), where the students use their smartphones as glow sticks that can be coordinated to sing and flash to generate the chorus and the Mambo dance flash light effects. We also use MusicTalk in the campus Christmas party where students hang their smartphones in a Christmas tree to automatically sing the songs and flash color lights for season's greetings (Figure 7 (a)).

Figure 7 (b) illustrates the MusicTalk network architecture. To control MusicTalk, CampusTalk automatically generates a cyber (input) device as MusicTalk's talkPal called "MBoxCtl" (Music Box Controller) when the user selects the DFs and the input device in the GUI (to be elaborated). This talkPal device consists of two parts, i.e., the stage manager (Figure 7 (1)) and the conductor (Figure 7 (2)), which control several players (music instruments; Figure 7 (3)) and coordinate them to play a song. In this application, a player is an output IoT device (such as a smartphone) that can play music and/or flash light. The players are connected to a server (Figure 7 (4)) through a wired/wireless network (Figure 7 (5)). This server and the connected players together are treated as an output IoT device called MusicBox.

Similarly, the stage manager and the conductor are implemented as an input IoT device that connects to MusicBox through the IoTtalk server (Figure 7 (6)). The wireless technologies for the IoT devices can be BLE, Wi-Fi, LTE, or NB-IoT. Both the IoTtalk server and the MusicBox server can be installed in a wireless Access Point connected directly to the devices (Figure 7 (1), (2) and (3)) without involving public Internet.

The conductor determines which song to play and how to play the song (such as selection of the key, the mode, the period and the volume). The stage manager determines the roles of the players and the light design. The stage manager first determines the maximal number of players involved, and then groups these players into several clusters. The players in a cluster will have the same behavior, i.e., play the same music instruments. For example, all of them play violins.

CampusTalk implements MusicTalk by automatically configuring the connections between the stage manager/conductor and MusicBox in the GUI window illustrated in Figure 8. In this GUI, an input device is represented by an icon placed at the left of the window, which consists of smaller icons that represent IDFs, and an output device is represented by an icon placed at the right-hand side of the window, which includes ODF icons. For example, the icon in Figure 8 (1) represents the MBoxCtl input device, which includes two parts. The stage manager part consists of three IDFs: an integer *C* (Figure 8 (3)) that represents the number of clusters, an integer *N* (Figure 8 (4)) that represents the number of the players in a cluster, and a floating-point number *L* (Figure 8 (5)) that represents the luminance intensity of the light (for a player with the flash capability). For the purpose of readability, if both an IDF and an ODF have the same name, the GUI will append the IDF name with "-I" and the ODF name with "-O". In Figure 8, for example, we have *C-I* IDF and *C-O* ODF.

The conductor part includes the following IDFs: a JSON-format object *Song* (Figure 8 (6)) representing a MIDI (Musical Instrument Digital Interface) file [1],

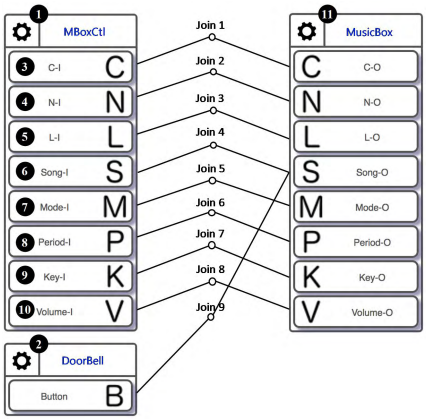


FIGURE 8. The GUI for connecting the IoT devices in MusicBox.

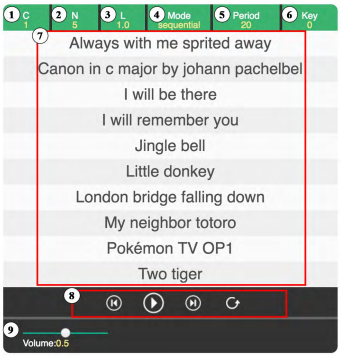


FIGURE 9. The MBoxCtl web page for the stage manager/conductor.

an integer Period P (Figure 8 (8)) representing the length of a music segment, an integer *Key*(Figure 8 (9)) representing the altered scale of the played song, a floating-point number *Volume* (Figure 8 (10)) in decibels (dB), which represents the volume of the played song and an integer *Mode* (Figure 8 (7)) representing the playing mode. In the sequential mode, the server cuts a song into several P-second segments, and the players sequentially perform the music segment in rotation. Assume that the song is partitioned into I segments and the players are grouped into C clusters. Every player in cluster c performs the i-th segment where $c \equiv i(\text{mod}C)$ for $1 \leq c \leq C$ and $1 \leq i \leq I$. In the parallel mode, the server sends a song to all players simultaneously then the players perform the song in parallel.

Based on Figure 8 (1), a web page called MBoxCtl (Music Box Controller) is developed for stage manager/conductor as illustrated in Figure 9. In this figure, (1) - (3) are pull-down lists controlled by the stage manager. Through these lists, one can set C, N and L. In Figure 9, $C = 1$, $N = 5$ and $L = 1.0$. Figure 9 (4) - (8) are controlled by the conductor. Figure 9 (4) - (6) are pull-down lists that only show the selected values just like (1) - (3). Figure 9 (7) is the song playlist and Figure 9 (8) are play-control buttons including play/pause, previous, next, repeat-once functions. Figure 9 (9) is a sliding bar for volume control.

FIGURE 10. Layout of the MusicBox web page. (a) Initial layout (C=6, N=5). (b) After the player joins the red cluster.

With the web-page implementation in Figure 9, multiple smartphones can serve as the stage manager/conductor by typing the IP address of the web page through their browsers. Therefore multiple parties (people who hold the smartphones) can simultaneously control the behavior of MusicBox.

In the GUI, the MusicBox output device is represented by an icon with 8 ODFs (Figure 8 (11)) to be controlled by the stage manager/conductor. Through C, N and L. given by the stage manager, a MusicBox web page is created and can be browsed by multiple smartphones as illustrated in Figure 10, where $C = 6$ and $N = 5$. The layout of this web page consists of C color boxes with a number no larger than N. A color box represents a cluster, and the number inside the box represents the number of players who can join the cluster. A player (e.g., a smartphone) joins MusicBox by accessing the MusicBox web page through its browser. Then the layout shown in Figure 10 (a) is displayed in the smartphone screen. When a color box is clicked, the screen of the player will become the color of the selected box with the texts indicating the parameter values for N, *Key*, *Volume* and *Note* (Figure 10 (b)).

When the next player browses MusicBox, the number in the red box in Figure 10 is decremented by 1 (i.e., 4 in our example). After N players, have selected the same cluster, the number in the box of the cluster becomes 0, and no new player is allowed to join the cluster. When the play button of the MBoxCtl web page (Figure 9 (8)) is pressed, the players will play the song and flash the light with the color representing their clusters.

The DA and the IDA of the MusicBox are illustrated in Figure 11, which follows the external DA scenario. The MusicBox DAI is a web server. When MusicBox is activated, its DAI registers to the IoTtalk server through the DAN, and creates $C \times N$ IDAs. An IDA is a web-page browser to be executed at, e.g., a smartphone, while the DA resides in a separate computer. Each of the IDAs establishes a connection to the DAI via the WebSocket protocol.

The DAI consists of four components. Through the DAN (Figure 11 (1)), the DAI MsgHandler (Figure 11 (2)) receives the messages from the stage manager and the conductor. The messages include the parameters C, N and L. According to these parameter values, the WebPage Generator (Figure 11 (3)) creates $C \times N$ IDAs (the web pages) by using HTML, CSS and JavaScript to manipulate the MIDI files

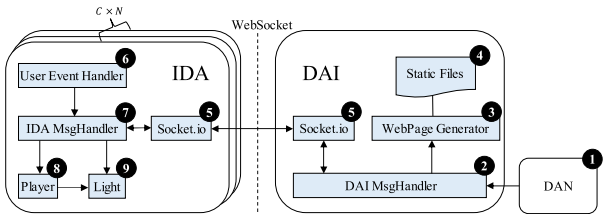


FIGURE 11. The MusicBox DA and IDA.

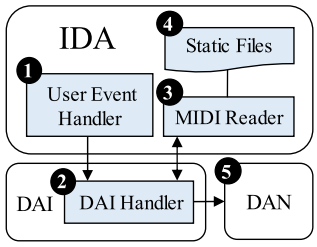


FIGURE 12. The MBoxCtl DA and IDA.

FIGURE 13. The ping pong paddle with the Koala device.

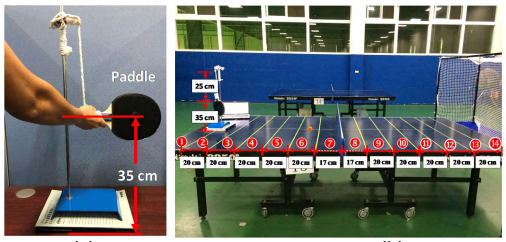


FIGURE 14. PingPongTalk setup. (a) Paddle height. (b) Table marks.

stored in the Static Files (Figure 11 (4)). In a typical web page design, all browsers can access and interact with the web page. For example, the MBoxCtl web page can be accessed by several smartphones and can be controlled by them simultaneously. On the other hand, the MusicBox web page can only be accessed by a limited number of smartphones for every cluster. We implement a simple mechanism in the DAI MsgHandler of MusicBox to achieve this feature. The details are given in Appendix A.

Based on the WebSocket library, Socket.io (Figure 11 (5)) provides communications between the DAI and the IDAs. An IDA consists of 5 components. The User Event Handler (Figure 11 (6)) detects the color box selected by the user, displays the chosen color with the decremented number, and sends the corresponding cluster number to the DAI through the IDA MsgHandler (Figure 11 (7)). The DAI updates the number of players joining that cluster. The IDA MsgHandler receives the messages sent from the DAI and dispatches them to the corresponding task modules to control the background luminance of the display for the player. The Player module (Figure 11 (8)) plays the chosen song, and instructs the Light module (Figure 12 (9)) when to flash.

The DA and the IDA for MBoxCtl are illustrated in Figure 12, which follows the internal DA scenario. The IDA consists of three components (Figure 12 (1), (3) and (4)). When the conductor selects a song, the User Event Handler (Figure12 (1)) sends this request to the DAI Handler (Figure 12 (2)). The DAI Handler then instructs the MIDI Reader (Figure 12 (3)) to read the MIDI file of the song in the Static Files (Figure 12 (4)) and parse it into a JSON object. Then the object is sent to MusicBox through the DAN (Figure 12 (5)). Note that MBoxCtl is a talkPal of MusicBox whose IDFs can be automatically generated by CampusTalk except for the song playlist (Figure 9 (7)). The developer only needs to provide the names by creating the static files to produce this list.

In Figure 8, after the MusicBox DA/IDA has been developed, the students can re-configure it to create various alert applications without extra programming efforts. For example, the students can control the Christmas tree in Figure 7 (a) by a door bell button (Figure 8 (2)).

IV. PingPongTalk

Some IoT applications require tight coordination between the physical input devices and the cyber output devices. That is, the behavior of the output device must be adjusted based on the experiments conducted on the physical input device. A good example is the college Physics experiments such as the pendulum swing and spring harmonic motion. The details can be found in [8]. We have implemented 46 interactive physics experiments from the first-year college Physics textbooks. This section elaborates on how to implement this type of cyber output devices by an example called PingPongTalk.

As an extension of the animation for parabola experiments in the Physics class, PingPongTalk animates the interaction between a ping pong paddle and a ping pong ball, which is used in ping pong ball practice in NCTU. Specifically, this application measures the paddle behavior by utilizing the Koala IoT device (Figure 13) that is an inertial measurement unit including an acceleration sensor. We attach this device to the back of a ping pong paddle. The acceleration sensor collects the measured data, and transmits them to the IoTtalk server.

The paddle is fixed at the height of, e.g., 35cm above the table (Figure 14 (a)). The area of the table is partitioned into fourteen lanes marked by the yellow tape, so that we can observe where the ball hits the table. In this way, the students can replay the real ball motion in the PingPongTalk animation and to make sure that the animated ball behavior is consistent with the real scenario. The widths of lanes 1-6 and 9-14 are 20 cm ((1) - (6) and (9) - (14) in Figure 14 (b)). The widths of lanes 7 and 8 are 17 cm ((7) and (8) in Figure 14 (b)).

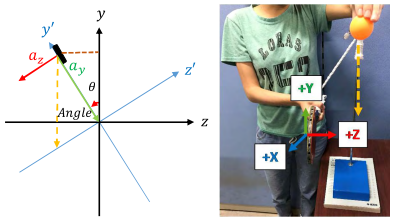


FIGURE 15. Calculation of the angle.

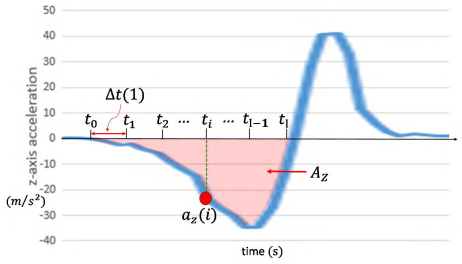


FIGURE 16. The $a_z(i)$ values against $\Delta t(i)$.

To obtain the acceleration and the initial speed of the ball, we conduct experiments with a real ball hitting in a ping pong table. For $j \geq 0$, at time t_j let $a_y(j)$ and $a_z(j)$ represent the y-axis and the z-axis accelerations, respectively, where t_0 is the time when the ball is dropped. For simplicity, denote $a_y = a_y(1)$ and $a_z = a_z(1)$, where t_1 is the time when the paddle hits the ball. Figure 15 illustrates how to use the acceleration data (a_y, a_z) to calculate the angle θ between the y'-axis and the y-axis in Figure 15 (which is the ODF *Angle* of PingPongTalk). According to the law of cosines, we use the (a_y, a_z) to calculate *Angle* by Equation (1):

$$Angle = \begin{cases} 90 - \cos^{-1}\left(\dfrac{a_y}{\sqrt{a_y^2 + a_z^2}}\right)\left(\dfrac{180}{\pi}\right), & \text{if } a_z > 0 \\ 90 + \cos^{-1}\left(\dfrac{a_y}{\sqrt{a_y^2 + a_z^2}}\right)\left(\dfrac{180}{\pi}\right), & \text{otherwise} \end{cases} \tag{1}$$

The initial speed is computed as follows. When the player swings the paddle along with negative z-axis acceleration direction, the Koala device records the acceleration data $(a_x(j), a_y(j), a_z(j))$ and the time interval $\Delta t(j)$. Figure 16 illustrates the $a_z(j)$ curve against time points t_j in the observation period. We use Riemann sum to compute the gray area A_z. In Figure 16, *for* $0 < i \leq I$, $a_z(i) < 0$ is the z-axis acceleration values, and $\Delta \mathbf{t}(i) = t_i - t_{i-1}$ is the time interval between t_i and t_{i-1}. Then

$$A_z = \sum_{i=1}^{I} a_z(i)\Delta t(i) \tag{2}$$

Let S be the horizontal distance between where the ball is hit by the paddle and where the ball hits the table (see Figure 17). We obtain $a_z(i)$ through experiments and use

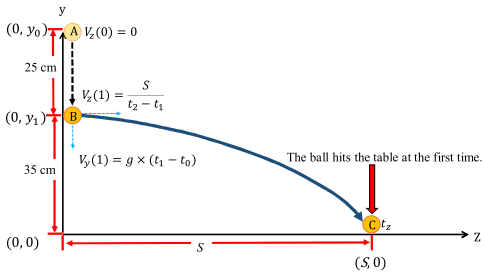


FIGURE 17. The locus of the ping pong ball.

TABLE 1. $A_{z,k}$, S_k and $V_{z,k}$ for $1 \leq k \leq 10$.

Experiment	1	2	3	4	5	6	7	8	9	10
$A_{z,k}$ (cm²)	257	280	293	301	302	306	314	318	330	360
S_k (cm)	70	82	85	85	85	90	90	90	92	100
$V_{z,k}$ (cm/s)	564	661	685	685	685	725	725	725	741	806

Equation (2) to calculate area A_z through the velocities at t_j for $0 \leq j \leq I$, where $a_z(0) = a_z(I) = 0$. The results are listed in the first raw of Table 1 where $A_{z,k}$, S_k and $V_{z,k}$ are the A_z, the S and the V_z values measured in the k-th experiment for $1 \leq k \leq 10$. We measure S by repeatedly observing the loci of a real ping pong ball. The first two rows of Table 1 list $A_{z,k}$ against S_k.

The initial speed of the ball is computed by using the horizontal displacement S. In Figure 17, the ball is positioned at point A $= (0, y_0)$, B $= (0, y_1)$, and C $= (z_0, 0)$ at times t_0, t_1 and t_Z, respectively. Denote the velocity $V(i)$ of the ball at time t_i as $(V_z(i), V_y(i))$. The player holds the ball at position $(0, y_0)$ at time t_0, where y_0 is the height of the ball. The player drops the ball at $V(0) = (0, g(t_1 - t_0))$ and the paddle hits the ball at position $(0, y_1)$ at time t_1. Then the ball hits the table at position $(S, 0)$ at time t_Z. At time t_1 the horizontal velocity of the ball is $V_z = V_z(1)$ computed as follows:

$$S = V_z \times (t_Z - t_1) \tag{3}$$

Based on Equation (3), the third row of Table 1 lists $V_{z,k}$ for S_k at the second row. Using $A_{z,k}$ and $V_{z,k}$ in Table 1, we build a linear regression model to compute the locus of the ball. The model is established by the least squares method with two regression coefficients ω and δ. In Equation (4), $\hat{\omega}$ and $\hat{\delta}$ are the least squares estimates of ω and δ:

$$V_z = A_z \times \hat{\omega} + \hat{\delta} \tag{4}$$

we compute

$$\hat{\omega} = \frac{\sum_{k=1}^{10}(A_{z,k} - \overline{A_z})(V_{z,k}\overline{V_z})}{\sum_{k=1}^{10}(A_{z,k} - \overline{A_z})^2} \text{ and } \hat{\delta} = \overline{V_z} - \hat{\omega} \times \overline{A_z} \tag{5}$$

where

$$\overline{A_z} = \frac{\sum_{k=1}^{10} A_{z,k}}{n} \text{ and } \overline{V_z} = \frac{\sum_{k=1}^{10} V_{z,k}}{n} \tag{6}$$

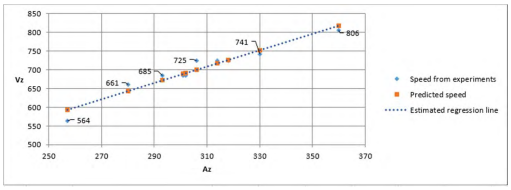


FIGURE 18. The estimated regression line.

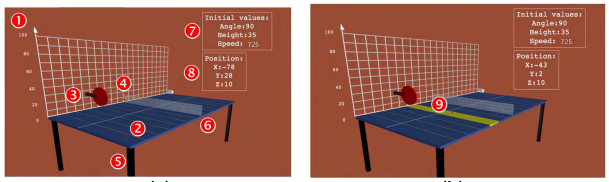


FIGURE 19. The ping pong game animation. (a). Before the ball touches the table. (b). When the ball touches the table.

$\overline{A_z}$ and $\overline{V_z}$ can be obtained by using Equation (6) and $A_{z,k}$ and $V_{z,k}$ in Table 1, and then we substitute the computed results into Equation (5) to yield $\hat{\omega} = 2.1778$ and $\hat{\delta} = 33.587$. Figure 18 illustrates the estimated regression line, where the diamonds represent $\overline{V_z}$ obtained in Equation (6), and the squares represent the speeds computed by using the regression model.

The regression statistics are 0.966359432 for the multiple R, 0.933850552 for the R-squared, 0.925581871 for the adjusted R-squared, and 17.12899786 for the standard deviation. Therefore, our calculation is accurate.

The $A_{z,k}$, S_k and $V_{z,k}$ values obtained from the experiments are used to create a cyber device for ping pong game animation. When a student swings the Koala-attached paddle without actually hitting a ping pong ball, the acceleration values are sent to the IoTtalk server, and we can investigate the locus of the ball in a 3-D animation shown in Figure 19. This animation provides a virtual grid in the Y-Z plane (Figure 19 (1)) and the ping pong table serves as the X-Z plane (Figure 19 (2)). The virtual grid helps to observe the locus of a ping pong ball (Figure 19 (4)) hit by the paddle (Figure 19 (3)). We also draw the table lags (Figure 19 (5)) and the table net (Figure 19 (6)) for visual effect. The statistics are shown in Figure 19 (7) and (8). When the ball hits the table, the hit lane turns from blue to yellow (Figure 19 (9)).

In the GUI (Figure 20 (a)), the PingPongTable animation is an output device with two ODFs: *Angle* and *Speed*. This cyber output device follows the internal DA scenario. The Koala is a physical input device with one IDF Acceleration. Koala follows the external DA scenario where the IDA is installed in the Koala device and the DA is installed in a smartphone. The *Acceleration* IDF is connected to the *Angle* ODF through Join 1, and is connected to the *Speed* ODF through Join 2. As mentioned in [9], the GUI allows one to use a function to manipulate the data delivered in a connection. By clicking the join circle of the connection, the GUI pops up a window for creation of the function. In PingPongTalk, we click the Join 1 circle to write a Python function to calculate the

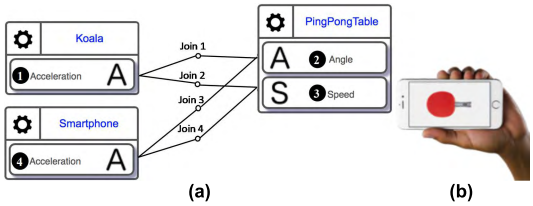


FIGURE 20. The connection of PingPongTalk through Koala and a smartphone. (a) PingPongTalk configuration in the GUI. (b) A smartphone serving as ping pong paddle.

angle of the paddle using Equation (1). Similarly, in Join 2, we write the initial speed function using Equation (4). After PingPongTable and the functions in Joins 1 and 2 have been implemented, a student can also use a smartphone as a ping pong paddle (Figure 20 (b)), and connect its *Acceleration* IDF to PingPongTable through Joins 3 and 4. The function in Join 3 (Join 4) is the same as that in Join 1 (Join 2). In this way, a student can easily connect his/her smartphone to this application without any extra programming effort.

V. SkeletonTalk

The Institute of Applied Art in NCTU has created many cyber-physical interactive designs. It is particularly interesting to produce the physical device and its cyber counterpart in an interactive design. This section shows the scenario where an output device has both the cyber and the physical implementations that share the same DA. We use an artwork called SkeletonTalk as an example, which is an CampusTalk application allowing various input devices to control the shape change (i.e., compression) of a skeleton. We have implemented both physical Skeleton (Figure 21 (a)) and cyber Skeleton (Figure 21 (b)). Skeleton is a special hexagonal prism in which the top and the bottom are the regular hexagonal pyramids. Besides the top and the bottom pyramids, a S-size Skeleton has S layers, and every layer has the same graphical pattern constructed by three types of stalks: stalk s0 (Figure 22 (a)), stalk s1 (Figure 22 (b)) and stalk s2 (Figure 22 (c)). A layer consists of three sub-layers called left (Figure 22 (d)), middle (Figure 22 (e)) and right (Figure 22 (f)). The sub-layers are connected by stalks s0. When the skeleton is not compressed, every stalk s0 is placed horizontally. When the skeleton is compressed, the end point of the stalk s0 connected to the left (or the right) sub-layer moves toward the middle sub-layer.

With a color lightbulb inside the skeleton body, the physical Skeleton is an interactive ceiling lamp. As we pointed out before, Skeleton can grow and compress. Through growing and compressing, Skeleton's shadow in the floor shows beautiful geometric patterns (Figure 23). This physical output device follows the internal DA scenario, where the DA and the IDA are implemented in an Arduino board [9], and the IDA controls the mechanical skeleton stalks through the output pins of the board.

The cyber Skelton reuses the internal DA of its physical counterpart, where the IDA implements the animation to

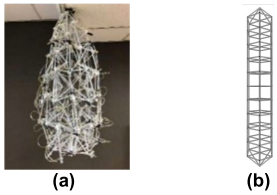


FIGURE 21. Physical and cyber Skeletons. (a) Physical (Size = 2). (b) Cyber (*Size* = 5, *Angle* = 0°).

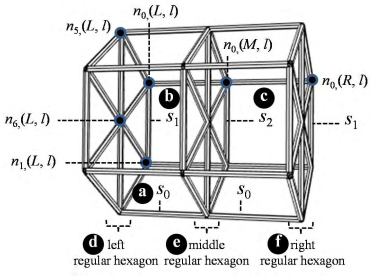


FIGURE 22. A skeleton graphical pattern.

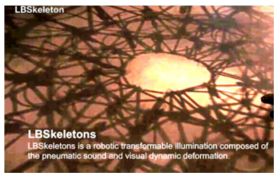


FIGURE 23. The Skeleton shadow on the floor.

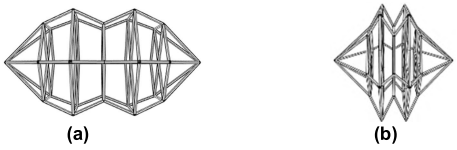


FIGURE 24. Skeletons with different angles (*Size* = 2). (a) *Angle* = 30°. (b) *Angle* = 60°.

illustrate compression of the stalks of the skeleton with color change. Therefore, the output IoT device for Skeleton has three ODFs: *Size*, *Angle* and *Color*. The *Size* ODF represents the "layer" of Skeleton to be grown, which ranges from 0 to 10. For example, *Size* = 2 in Figure 21 (a) and Figure 24, and *Size* = 5 in Figure 21 (b). The *Color-O* ODF represents the color of Skeleton. The *Angle* ODF represents the compressed degree of Skeleton, which ranges from 0 to 90 degrees. For example, *Angle* = 0° in Figure 21 (b), *Angle* = 30° in Figure 24 (a), and *Angle* = 60° in Figure 24 (b).

We implement the cyber Skeleton in Java, which continuously draws the graphical skeleton patterns with the specified ODF values. We use a web-based keypad to control the color of Skeleton (see Figure 25 (a)), which can be accessed through a web browser. We also encourage the students to use other input devices to control the Skeleton's color, e.g., the color sensor [8].

Figure 26 illustrates an example for remote controlling of Skeleton. The remote controller follows the internal DA scenario, which is a browser in a smartphone or a computer. Keypad 1 (K_1; the keypad of the first smartphone;

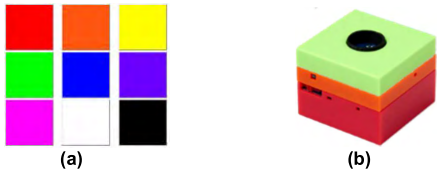


FIGURE 25. The Color input device feature. (a) a color keypad. (b) a color sensor.

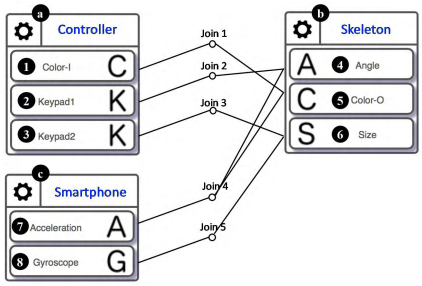


FIGURE 26. Remote Control connection to Skeleton.

see Figure 26 (2)) connects to *Angle* (A), where K_1 ranges from 0 to 9, and A ranges from 0 to 90 degrees. The IoTtalk sever automatically makes the following conversion: $A = \left(\frac{90}{9}\right) K_1 = 10K_1$. Keypad 2 ($K_2$; the second smartphone's keypad; see Figure 26 (3)) connects to *Size* (S), where S ranges from 0 to 10. That is, $S = \left\lfloor \left(\frac{10}{9}\right) K_2 \right\rfloor = \lfloor 1.1K_2 \rfloor$. *Color-I* (the third smartphone keypad that represents the color map; see Figure 26 (1)) connects to *Color-O* (Figure 26 (5)) and the triple RGB values of *Color-I* are directly assigned to *Color-O*.

A student can connect the smartphone to Skeleton, and use, for example, the *Acceleration* (Figure 26 (7)) and the *Gyroscope* IDFs (Figure 26 (8)) to control Skeleton without any programming effort. Note that the current version of the remote control is Skeleton's talkPal automatically created by IoTalk. For a physical Skeleton, the size is fixed, and any input from the Keypad2 IDF to the Size ODF will not have any effect.

VI. CONCLUSION

This paper proposed CampusTalk, a series of IoT applications developed in NCTU campus based on the IoTtalk platform. Through this platform all cyber and physical IoT devices created in CampusTalk can be quickly reconfigured and reused to create new applications without or with little programming efforts. CampusTalk unifies the concepts of automatic creation of "mirror device features" (simulated counterpart) and talkPal device features, which are nicely demonstrated through several applications on campus. The core application of CampusTalk is SmartPhoneTalk that allows other applications to be accessed by students through their smartphones without installing any mobile apps. We described three CampusTalk applications for cyber-physical interaction with SmartPhoneTalk. MusicTalk serves as an example for a big campus event, where a large number of students can use their smartphones to play flashing glow sticks. PingPongTalk

showed how animation of experiments in the Physics class can be integrated with the IoT technology to provide interaction with the students, and then turned into sport exercise such as a ping pong game. SekeletonTalk is an example of student innovation in inactive art. The IoT devices developed in these applications can be reused to interact with each other. In particular, the students can create new applications (for example, use the singing Christmas tree to control the compression and lighting of Skeleton) without extra programming effort. Video demos for SmartPhoneTalk, MusicTalk and SkeletonTalk can be found in *https://youtu.be/ZLoXieCG9_g*, *https://youtu.be/p8usRSIoaok and https://youtu.be/scfeKSLjJmA*. These videos show how the real GUI interface works and how the IoT devices behave.

APPENDIX

This appendix illustrates the code for DAI MsgHandler. Figure 27 lists the stage manager part. Line 2 specifies seven clusters in the variable clusterArray. Line 3 gives the default values for $C = 7$, $N = 5$ and$L = 1.0$ (full-intensity of the light). At Line 4, numberArray stores the number of members who join each of the clusters. Lines 5-7 define the functions to reset the values for C, N and L. Line 9 defines the pull function to be called by the DAN (Figure11 (1)). This function obtains the new values for C, N and L from the stage manager (Figure 12) through the IoTtalk server.

In Figure 11 (5), the web socket is used for communication between the DAI (i.e., serverSocket) and the IDA (i.e., clientSocket). Four event types are defined. The event type ''connect'' is handled by serverSocket to connects the IDA with the DA (Line 16).

Three types of the events are handled by clientSocket:

- getNumbers: Lines 17 and 18 handle the IDA request (Figure 11 (3)) to obtain the number of the clusters.
- join: Lines 19-27 handle the IDA request to join a cluster c. If less than N players have joined this cluster, then the IDA's request is accepted (Lines 21-25). Otherwise, the request is rejected (Line 27).
- close: When a player closes his/her IDA webpage, Line 28 detects this event, and the IDA leaves cluster c (Line 29). The number numberArray[c] is decremented by one at Line 30. Lines 31 and 32 update the member number of cluster c shown in other players' webpages.

Figure 28 lists the code of the conductor part for DAI MsgHandler. Line 2 gives the default values for *Song* = *null* (no song is selected), *Mode* = *0* (to play sequentially), *Period* = *20* (20 second per music segment), *Key* = *0* (stay in the same scale) and *Volume* = *0.5* (play with the volume in half). Line 3 defines a flag *playState* to indicate if a song is played. Line 4 defines a flag *repeatSong* to determine if a song will be repeated or not. Line 5 defines a variable *progress* to indicate the progress of the *Song*. Line 6 defines a variable *nextC* to indicate the next cluster to play in the sequential mode. Lines 7-16 define the functions to reset the values for *Song, Mode, Period, Key* and *Volume*. Line 18 defines the pull function to be called by the DAN (Figure11 (1)). This function obtains

```
Line 1  var msgHandler4StageManager = (function () {
Line 2  var clusterArray = ['#dc143c', '#00bfff', '#ffd700', '#3cb371', '#1e90ff', '#ffa500', '#9932cc'];
Line 3  var C = clusterArray.length, N = 5, L = 1.0,
Line 4          numberArray = new Array(C).fill(0);
Line 5  function resetC(data){ C = parseInt(data); }
Line 6  function resetN(data){ N = parseInt(data); }
Line 7  function resetL(data){ L = parseFloat(data); }
Line 8  return {
Line 9      pull: function (odfName, data) {
Line 10         var obj = data[0];
Line 11         switch (odfName){
Line 12             case "C-O": resetC(obj); break;
Line 13             case "N-O": resetN(obj); break;
Line 14             case "L-O": resetL(obj); break; }},
Line 15   setSocketIo: function (serverSocket) {
Line 16         serverSocket.sockets.on('connect', function (clientSocket) {
Line 17             clientSocket.on('getNumbers',function () {
Line 18             serverSocket.sockets.emit('getNumbersACK', numberArray); });
Line 19             clientSocket.on('join', function (c) {
Line 20               if(numberArray[c] < N) {
Line 21                 clientSocket.join(c);
Line 22                 numberArray[c]++;
Line 23                 serverSocket.sockets.emit("getNumbersACK", numberArray);
Line 24                 serverSocket.to(clientSocket.id).emit("join", {message: "ACK", cluster: c});
Line 25                 serverSocket.sockets.in(c). emit("numberOfJoin", numberArray[c]); }
Line 26               else {
Line 27                 serverSocket.to(clientSocket.id).emit("join", {message: "NAK"}); } });
Line 28                 clientSocket.on('close, function (c) {
Line 29                   clientSocket.leave(c);
Line 30                   numberArray[c]--;
Line 31                   serverSocket.sockets.emit("getNumbersACK", numberArray);
Line 32                   serverSocket.sockets.in(c).emit("numberOfJoin", numberArray[c]);  }); }); } }; })();
Line 33  exports.msgHandler4StageManager = msgHandler4StageManager;
```

FIGURE 27. DAI MsgHandler: stage manager.

```
Line 1  var msgHandler4Conductor = (function () {
Line 2  var Song = null, Mode = 0, Period = 20, Key = 0, Volume = 0.5;
Line 3  var playState = false,
Line 4      repeatSong = false,
Line 5      progress = 0,
Line 6      nextC = 0;
Line 7  function resetSong(data){
Line 8      Song = JSON.parse(data);
Line 9      if (playState) serverSocket.sockets.emit("interruptSong");
Line 10       progress = 0; nextC = 0;
Line 11       if (Mode == 0) snedPartialSong2NextCluster();
Line 12       else if (Mode == 1) snedSong2Players(); }
Line 13 function resetMode(data){ Mode = parseInt(data); }
Line 14 function resetPeriod(data){ Period = parseInt(data); }
Line 15 function resetKey(data){ Key = parseInt(data); snedKey2Players(); }
Line 16 function resetVolume(data){ Volume = parseFloat(data); snedVolume2Players(); }
Line 17 return {
Line 18     pull: function (odfName, data) {
Line 19             var obj = data[0];
Line 20             switch (odfName){
Line 21                 case "Song-O":   resetSong(obj); break;
Line 22                 case "Mode-O":  resetMode(obj); break;
Line 23                 case "Period-O":  resetPeriod(obj); break;
Line 24                 case "Key-O":    resetKey(obj); break;
Line 25                 case "Volume-O": resetVolume(obj); break; }},
Line 26     setSocketIo: function (socket) {
Line 27                 serverSocket = socket;
Line 28                 serverSocket.sockets.on('connect', function (clientSocket) {
Line 29                 clientSocket.on("playEndACK", function(){
Line 30                 if (repeatSong && Mode == 0 && progress == Song.length) {
Line 31                   progress = 0; snedPartialSong2NextCluster();
Line 32                 else if (repeatSong && Mode && progress == Song.length){
Line 33                   progress = 0; snedSong2Players();
Line 34                 else if (Mode == 0) snedPartialSong2NextCluster(); });
Line 35                 clientSocket.on('close', function(){
Line 36                   currentC = ((nextC+C)-1)%C;
Line 37                   if (numberArray[currentC] == 0 && Mode == 0)
                                    snedPartialSong2NextCluster(); }); }
Line 38  exports.msgHandler4Conductor = msgHandler4Conductor;
```

FIGURE 28. DAI MsgHandler: conductor.

the new values for *Song, Mode, Period, Key* and *Volume* from MBoxCtl (the conductor in Figure 12) through the IoTtalk server. At Lines 28-36, three event types are defined. The event type ''connect'' is handled by serverSocket to connects the IDA with the DA (Line 28).

Two types of the events are handled by clientSocket:

- playEndACK: When a song ends, this event is triggered. The DAI takes one of three actions. The first action (Lines 30 and 31) repeats the *Song* in the sequential mode. The second action (Lines 32 and 33) repeats the *Song* in the parallel mode. For the third action (Lines 34), the DAI sends the next part of the *Song* to the next cluster to play.
- close: When a player closes his/her IDA webpage, Line 35 detects this event. If the closed player is the only member in the current cluster (Line 37) in the sequential mode, then the DAI sends the next part of the *Song* to the next cluster to play.

ACKNOWLEDGMENT

The physical skeleton was created by Chih-Chieh Huang. The cyber skeleton was implemented by Qing Liu. PingPongTalk was implemented by Ya-Lan Chang.

REFERENCES

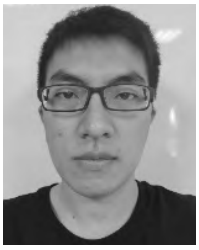
[1] J. Chadabe, "Part IV: The seeds of the future," *Electron. Musician*, vol. 16, no. 5, May 2000.
[2] Mozilla Developer Network. (2016). *DeviceMotionEvent*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API
[3] S. Huang, *Transparent Organ*. Genèva, Switzerland: Salon Int. Des Invention, 2014.
[4] Arduino. (2018). *Arduino Yun*. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardYun
[5] *Machine-to-Machine Communications (M2M): MIa, DIa and MId Interfaces*. document ETSI TS 102 921 V1.3.1, 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_ts/102900_102999/102921/01.03.01_60/ts_102921v010301p.pdf
[6] AllJoyn. (2018). *Open Connectivity Foundation*. [Online]. Available: https://openconnectivity.org
[7] Y.-B. Lin, *et al.*, "EasyConnect: A management system for IoT devices and its applications for interactive design and art," *IEEE Internet Things J.*, vol. 2, no. 6, pp. 551–561, Dec. 2015.
[8] Y.-B. Lin, Y.-W. Lin, C.-M. Huan, C.-Y. Chih, and P. Lin, "IoTtalk: A management platform for reconfigurable sensor devices," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1552–1562, Oct. 2017, doi: 10.1109/JIOT.2017.2682100.2017.
[9] Y.-W. Lin, Y.-B. Lin, M.-T. Yang, and J.-H. Lin, "ArduTalk: An Arduino network application development platform based on IoTtalk," *IEEE Syst. J.*, pp. 1–9, Nov. 2017.
[10] Y.-W. Lin, Y.-B. Lin, C.-Y. Hsiao, and Y.-Y. Wang, "IoTtalk-RC: Sensors as universal remote control for aftermarket home appliances," *IEEE Internet Things J.*, vol. 4, no. 4, pp. 1104–1112, Aug. 2017.
[11] oneM2M. (2016). *Standards for M2M and the Internet of Things*. [Online]. Available: http://www.onem2m.org/
[12] Y.-Y. Shih, A.-C. Pang, and P.-C. Hsiu, "A storage-free data parasitizing scheme for wireless body area networks," in *Proc. IFIP Netw.*, Trondheim, Norway, Jun. 2014, pp. 1–9.
[13] X. Li, R. Lu, X. Liang, X. Shen, J. Chen, and X. Lin, "Smart community: An Internet of Things application," *IEEE Commun. Mag.*, vol. 49, no. 11, pp. 68–75, Nov. 2011.
[14] H.-W. Kao, Y.-H. Ju, and M.-H. Tsai, "Two-stage radio access for group-based machine type communication in LTE-A," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, U.K., Jun. 2015, pp. 3825–3830.

YI-BING LIN (M'96–SM'96–F'03) received the bachelor's degree from National Cheng Kung University, Taiwan, in 1983, and the Ph.D. degree from the University of Washington, USA, in 1990. From 1990 to 1995, he was a Research Scientist with Bellcore. He then joined National Chiao Tung University (NCTU), Taiwan. In 2010, he became a lifetime Chair Professor of NCTU, and the Vice President of NCTU in 2011. From 2014 2016, he was a Deputy Minister with the Ministry of Science and Technology, Taiwan. Since 2016, he has been appointed as the Vice Chancellor with the University System of Taiwan (for NCTU, NTHU, NCU, and NYM). He is currently an Adjunct Research Fellow with the Institute of Information Science, Academia Sinica, and the Research Center for Information Technology Innovation, Academia Sinica, and a member of the Board of Directors with Chunghwa Telecom. He was co-authored the books *Wireless and Mobile Network Architecture* (Wiley, 2001), *Wireless and Mobile All-IP Networks* (John Wiley, 2005), and *Charging for Mobile All-IP Telecommunications* (Wiley, 2008). He is AAAS Fellow, ACM Fellow, and IET Fellow. He received numerous research awards, including the 2005 NSC Distinguished Researcher, the 2006 Academic Award of Ministry of Education, the 2008 Award for Outstanding Contributions in Science and Technology, the Executive Yuen, the 2011 National Chair Award, and the TWAS Prize in Engineering Sciences in 2011 (the Academy of Sciences for the Developing World). He is the Chair of the IEEE Taipei Section.

LI-KUAN CHEN received the B.S. degree from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2014, and the M.S. degree with the Department of Computer Science, Nation Chiao Tung University, Hsinchu, in 2018. His research interests include operating system and Interent of Things.

MIN-ZHENG SHIEH received the B.S. and M.S. degrees in computer science and information engineering and the Ph.D. degree in computer science and engineering from National Chiao Tung University, Taiwan, in 2003, 2004, and 2011, respectively. From 2012 to 2016, he served as an Assistant Research Fellow with the Information and Communication Technology Laboratories, National Chiao Tung University. Since 2016, he has been an Assistant Professor with the Information Technology Service Center, National Chiao Tung University. His main research interests include computational complexity, algorithms, coding theory, and discrete mathematics.

YUN-WEI LIN received the B.S. degree in computer and information science from Aletheia University, Taipei, Taiwan, in 2003, and the M.S. and Ph.D. degrees in computer science and information engineering from National Chung Cheng University, Chiayi, Taiwan, in 2005 and 2011, respectively. He has been an Assistant Research Fellow with National Chiao Tung University since 2013. His current research interests include mobile ad hoc network, wireless sensor network, vehicular ad hoc networks, and IoT/M2M communication.



TAI-HSIANG YEN received the bachelor's degree in application mathematics from National Chiao Tung University, Taiwan, in 2011, where he is currently pursuing the M.S. degree with the Department of Computer Science. His current research interests include Internet of Things, machine learning, and artificial intelligence.