International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2020): 7.803

A General Programmable Fluid Interface for a Smart Device

Ganesh M. Bhat[1], Mahesh Gosi[2]

[1]Department of Information Science and Engineering, PESIT – BSC, Bengaluru, India
ganeshmbhat96[at]gmail.com

[2]Department of Information Science and Engineering, PESIT – BSC, Bengaluru, India
maheshgosi53[at]gmail.com

Abstract: ***As we approach the dawn of the "Smart" world era, a revolution to control all devices is essential. Thus, a single device that integrates with a smart device and generates an interface that is unique for that device is highly beneficial. Imagine the concept of controlling all your smart devices at your fingertips with an application which has an interface to handle all your devices. With this application, we minimise the number of mandatory devices to control each product individually, by converting normal devices to smart devices and make a common augmented interface to interact with them. We identify them by their unique id (called a Target) and segregate basic functions, called IO Points, using Open Hybrid platform for each individual device. These basic functions work together with other IO Points on different devices to perform extra functions, as well as their factory-made tasks. Thus, enabling users to seamlessly interact with their devices, as the interface is mapped to them independently.***

Keywords: Fluid interfaces, Augmented Reality, Arduino, Direct Mapping, Reality Editor, Hybrid objects, IO-points, Object tracking

1. Introduction

Humans operate the physical world with hands. Since the dawn of human race, humans have formed shapes and affordances of objects that can simply operate with muscle memory, without using anyone's mind to even think about it. It is due to this scenario that operating the world with a touch screen promises an advantage over the paradigms previously used to operate physical things in the past. Having multiple devices for each device becomes a tedious task and sometimes difficult to keep track off. To this problem, the proposed solution is to use a fluid interface for providing fast action requirements to the users.

Users will use their daily drivers, such as their phones, to immerse themselves into an augmented reality, using them to perform actions that would usually take a bit of exercise to follow through. The Reality Editor helps us in this process by utilising Ubiquitous Computing (Ubicomp). The vision of Ubiquitous Computing is that computers will be interwoven into everyday objects to support everyday interaction with these objects [6]. To do so, the answer would be to convert common devices to smart devices called as Hybrid Objects. With that, the first thing is to figure out the basic functions which enable the object to function as its intended when working together. By accessing and utilising each function of a device individually, they enable us to control them at its elementary level. A phone or tablet's camera captures the real-time hybrid object for us to control it remotely.

This is possible by making day to day devices as Hybrid Objects. Each device will have a unique tag to identify itself from the lot. By scanning this unique identifier, it is possible to fetch its functions and controlling them with your handheld. By providing extra features to an ordinary device, indirectly there are new ways to handle the same device.

2. Current System

Human Beings are constantly surrounded by devices with which to interact frequently. The technology has developed exponentially these past years and the recent trend of IoT has helped us automate a person's work considerably with the help of smart devices. A human's interaction with objects can be classified into two types:

1) Physical Interaction – Where a person physically enters the domain space of the device and interact with it manually. Such as light switches, cooking stove, locks, etc.
2) Interaction using an external device – Where there is utilisation of an external device to interact with a device remotely. Such as television remotes, displays, speakers, etc.

The way humans interact with their devices has also been reformed with the introduction of various apps to control them, but the difficulty in using them seems to be elevated. As a matter of fact, with new technologies in the market such as Amazon Echo [8] and Google Home [7], which makes user's feel like living in a smart house, and uses voice recognition to perform various tasks, such as monitoring music, ordering online, booking a cab, and all those mundane tasks which can be finished with ease. Philips Hue [9] is another noteworthy development where users can alter the colour of the lights in their rooms, as well as provide scenic colours by connecting multiple Huelights together. Additional features for the devices can be found using their respective apps, which might be a hassle to find them individually on a person's phone. The most general way to interact is by using plug and play devices, which is directed connected to the system.

It may sometimes be difficult to open and use respective apps on every occasion, since each app will have a unique interface to match its functionalities with various menus and drop downs. So, the proposed system enables the user to seamlessly control all different devices, all with a single interface which provides an augmented menu on the screen of the user's device, with which to operate all the functionalities.

3. Proposed System

The principle being used is based on is the MIT Fluid interface design principles. It uses a platform called as Open Hybrid for interaction with everyday objects. It combines physical objects with the benefits of a flexible augmented user interface. Any object that is built with this open hybrid platform can be connected with any other object built using the same platform. A basic representation of the system is illustrated below in figure 1.

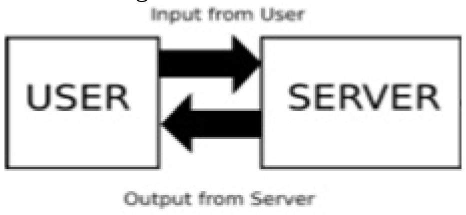

Figure 2

The entire platform is built on web standards. The user device contains a system called as the Reality Editor [1], which uses Ubicomp to interact with physical objects virtually. The Reality Editor associates a virtual object with each physical object and enables the reprogramming of the physical object's behavior using an intuitive, visual interface [6].

The device that is to be connected to the user becomes the server. This device is encased in a target (Figure 2). Once the user using any device, such as a phone or any embedded system, scans this target, it creates a HTTP GET query with a query string to the server. The server responds with a HTTP CONNECT query and thus establishes a connection between the two devices. All the data about the interface and connections are stored in the object. Once connected, the user's device will now receive data from the object about the interface to be generated for the object. This functionality can also be extended for devices with dissimilar functionality. Here, the devices that need to be connected to each other and in turn to the user, first generate GET requests to the user as per normal functionality and then initiate CONNECT requests with each other thus sharing their bandwidth. Thus, both their functionalities can now be linked and both the objects share their data to the user. But the interface generated will now be an amalgamation of both the functionalities. Any input given to any one of the device will affect both the devices. This mechanism is based on a direct mapping principle. [1]
Reality Editor: The Reality Editor is a system that supports editing the behavior and interfaces of the so called "smarter objects", i.e. objects or devices that have an embedded processor and communication capability. Using augmented reality techniques, the Reality Editor maps graphical elements directly on top of the tangible interfaces found on physical objects, such as push buttons or knobs. It allows flexible reprogramming of the interfaces and behavior of the objects, as well as defining relationships between smarter objects to easily create new functionalities.

Figure 2

4. Setup and Configuration

For this system, an Arduino Yun is used which can store the data and the interface of each device that is provided as an input for connection. The Arduino is connected to the device through a port or some other form of connection that allows the Yun to access the data stored in the embedded system of a device. Then a printout is taken of a valid target and the Yun is covered with it. This process is to convert the common object in to a Hybrid Smart Object. The target is used to cover the Yun so that when the user scans the target using the Reality Editor, a connection is setup with the Yun via a HTTP GET operation. Once the connection is setup, the data and the interfaces that are stored in the Yun are now sent via a HTTP RESPONSE connection. On the user's device, the interface is generated.

Once the initial interface has been generated then adding of IO-points [1] to the object can begin. First, it is ensured that the user's device and the Yun are on the same local network and then run Reality Editor. There are three steps involving the process:

1) Define Basic Functions of a Device
2) Add IO Points Based on Functions
3) Mapping the IO Points

1) Define Basic Functions of a Device:

The different modules of a compound device are distinguished and noted. Each module represents a basic function that the device can perform. Only if all the modules work together, the original device works as per the predefined role. These different modules are essential in the defining IO Points, as each of the modules represent an IO Point for the device.

2) Adding IO Points

A new IO Point is added for each functionality of the device. These points are visible through the Reality Editor and defines meaning to each of the points.

3) Mapping IO Points

IO points can be linked to the device to be connected via the interface. The target that is used initially for the setup on the Yun is the first IO point. The other IO point will be present on the device to be connected. The user simply uses a drag-and-drop mechanism to connect the IO point. This allows a connection between the user's device and the other device. By establishing this connection, both the points are interlinked and the user can define how the linked states should react when either of them are triggered.

If the user wishes to link two or more devices to his device, the user connects all the points required using the drag-and-drop mechanism from the other devices to link it to the main device the user wants to program. This yields all three devices being able to use the functionalities of the other device in addition to their own.

The hybrid object can be altered using an instance of itself with the Java Script [4]

varobj = new HybridObject(); (1)

4) Interface with IO-Points

.addReadListener([IO Point], callback) (2)

If a user interface wants to read data from a Hybrid Object, it first needs to send a read request.

.write([IO Point], [Value 0 ~ 1]) (3)

To write to the Hybrid Object use write(). The scale of the values should be between 0.0 and 1.0.

5) Interface with Reality Editor

.sendGlobalMessage([message]) (4)

Send broadcast messages to all other objects currently visible in the Reality Editor.

.addGlobalMessageListener(callback[message]) (5)

Used to listen to messages sent to all other objects currently visible in the Reality Editor.

.addVisibilityListener(callback[e]) (6)

Used for reading if the interface is visible or not. The interface stays active for 3 seconds after it becomes invisible.

.getPossitionX(), .getPossitionY(), .getPossitionZ() (7)

Returns a number for translation distance and position between the iOS device and the marker.

6) Developer Functions

.developer() [5] (8)

The developer() function allows access to all developer functionality. It is used to move and scale user interfaces within the Reality Editor and it gives access to the Developer Web-Page.

.add([Hybrid Object], [IO Point]) (9)

With the .add() function it can be used to add new IO Points to the Hybrid Object.

.read([Hybrid Object], [IO Point (String)]) (10)

Within the Arduino loop() function the .read is used to read data from an IO Point.

.write([Hybrid Object], [IO Point], [Value 0 ~ 1]) (11)

The Arduino loop() function can be used to write to an IO Point. All values are floating point for both the .read and .write functions are in the range between 0 and 1.

[1] IO-point: This is a marker on the object target that connects the device input and output functionalities to make it visible when scanned by the user

5. Applications

This system will be able to bind two or more independent objects to work together to finish a job, which would be not be plausible with them individually. This would be done without the hassle of finding an application for each such object in use. This can be used for home automation by converting all the devices as Hybrid Objects. When travelling in vehicles, it becomes easier to operate the vehicles by utilizing functions embedded in them. E.g. to bring down the windows of a car can be accomplished with a single button. As there is no need to have separate apps to control these functions, they can be directly mapped to the functions beforehand to control them.

6. Real World Example

Consider the example of boiling food for 10 min as per the recipe. Traditionally, one could make a note of the time the food has started to cook and manually keep checking the clock till it reaches a point where it looks like it's about 10 min from boiling or maybe even set a timer for 10 min. But with the proposed system, the user can utilize the components in our surroundings, like a toaster in the kitchen, to accomplish the same objective.

There will be the need to divide the toaster into its basic functions, functions which would enable the toaster to operate as its whole if used together. The different components of a toaster can be listed as:
1) Trigger to start the toaster
2) Timer Knob to set the time
3) The grill inside the toaster (Heating unit)

These different components are viewed as different functions in the Reality Editor and can be operated individually. To convert the toaster and stove to a hybrid object (1) is used and then add IO Points using (9) and define them with (11).

If the trigger is mapped to the timer and then to the stove using the Reality Editor, then it would be able to stop the stove after a set amount of time. The connection is made by calling (4) and (5) repeatedly. If the timer is set to 10 min and the trigger is pulled on the toaster, it will send a signal to the timer to start the count down. Once the timer has ended, the stove will get a response from the timer to stop the flame.

This same timer can be mapped to a different device for another use as well. This allows for a change of perspective completely on how to manipulate objects to perform chores that were never thought to be possible.

7. Advantages of Fluid Interfaces

Some of the benefits of fluid interfaces are:

1) Remote usage and operation of devices in the locale of the user.
2) Having a single device for connection instead of multiple devices for setting up individual connections.
3) Remote monitoring of user's device.
4) A clutter free interface which provides only the major functionalities of each device.
5) Extension and linking of one devices functionality to another.
6) Doesn't require sophisticated hardware and time consuming setup.
7) Fast connection to devices.
8) Memorisation of existing connected devices so that repeated setup of device is not necessary.
9) Non-Smart-Devices can be made smart use the provided hardware and thus their functionality can also be linked.

8. Limitations

1) Need of carrying a device to map hybrid objects.
2) Need constant Wi-Fi connection to all the hybrid devices, this with the development of the industry, this drawback is not as severe.
3) Range is limited to the user's Wi-Fi range.
4) Bandwidth is limited when used with two or more devices.

9. Future Scope and Enhancements

Since wearables, such as smart watches, VR headsets, etc. hit the market, it is possible to integrate them using this system. Wearables use the same layout techniques as handheld devices but need to be designed with specific constraints. To port any significant UI to a wearable, first thing is to customise the UI to tailor to the different shapes and appearance of a wearable, such as circular or flat screens (which form most the shapes in case of a smart watch). The UI on the wearable would primarily consist of a layout containing an array of buttons each mapped to a different IO point based on the respective device. The user can dynamically change the functions of the buttons based on his location, such as a conference hall, kitchen, etc.

By implementing the Reality Editor on an AR headset, it would be possible to see the IO Points of all the devices through the user's eyes directly, without having to drag around the device in hand. Linking of IO Points can be enabled with buttons, to provide a handsfree interaction.

10. Conclusion

Thus, in this paper, the proposed system can be used to implement an interface that can interact with our everyday objects, thus enabling the users, to maximise the utilisation of their devices and virtually make all devices work together as a single common device, which can perform activities by amalgamating them with different combinations.

References

[1] https://www.wired.com/2015/12/mits-reality-editor-app-lets-you-reprogram-the-world-with-augmented-reality/
[2] http://fluid.media.mit.edu
[3] http://www.realityeditor.org
[4] http://openhybrid.org/reference-javascript.html
[5] http://openhybrid.org/reference-arduino.html
[6] Reality Editor: Programming Smarter Objectshttp://ubicomp.org/ubicomp2013/adjunct/adjunct/p307.pdf
[7] Google Home: https://madeby.google.com/home/
[8] Amazon Echo: http://alexa.amazon.com/spa/index.html
[9] Philips Hue: http://www2.meethue.com/en-in/