Bulletin of the *Transilvania* University of Braşov • Vol. 13 (62) No. 1 - 2020
Series I: Engineering Sciences
https://doi.org/10.31926/but.ens.2020.13.62.1.5

HEADLESS INTERNET OF THINGS DEVICES THAT PROVIDE USER INTERFACE MODELS

Alexandru-Constantin IONIȚĂ[1]

Abstract: *Internet of Things devices are becoming more frequent and used in the lives of everyone. Starting with sensor networks that are monitoring certain parts of an experiment in a laboratory and continuing with simple appliances almost everyone has such a device. With the increasing usage of the devices the need for quick implementation of such a device became obvious. Besides the core functionality of a device, the user interface is the most important part, the simpler it is to use the higher the chance for a user to try the device. In this paper we present the concept and implementation of a framework that provides the tooling for creating user interface models that can be stored on the device itself and rendered on request in a prepared environment.*

Key words: *Internet of Things, user interface, headless devices.*

1. Introduction

An Internet of Things device is defined as a system that has a unique identifier and is connected to the Internet [1]. There are many ways to provide user interfaces for connected devices such as Cloud services of mobile phone applications but the common point of those methods is the fact that de user interface is always created on the outside of the device so every modification in the device functionality creates the need to update the user interface part as well. There is also the possibility to program the device to be a webserver and store the user interface as a website directly on the device and provide the user a way to access it using a web browser, but this method is only usable if the device has the capabilities to store and run a webserver.

This paper attempts to provide a concept and an implementation of a framework and environment designed to allow user interface design, creation with ease and also the integration with the device by uploading the user interface data on the device itself and transmitting it on demand to an environment capable of interpreting and displaying it. The purpose of the concept is to make it possible to provide user interfaces even for small devices with limited capabilities.

[1] Dept. of Electronics and Computers, *Transilvania* University of Braşov, Romania.

2. Concept

The idea of the project is simple, provide a framework and an environment capable to create, interpret, render and run a user interface destined to be used to let the user interact with a small headless microcontroller. A microcontroller is defined as a programmable device that has input/output capabilities and is able to take decisions based on the program that is currently running [3]. The framework should be able to use any communication protocol, being it wired or wireless, but the preferred protocol will be wireless, also used in the implementation of the proof of concept. A high level concept block diagram is presented in Figure 1. Each block presented in the diagram presents possible components needed to get to the desired results. The blocks marked as device specific (blue colour) are dependent on the device that the UI is designed for. The blocks marked as framework specific are device agnostic and designed to provide the necessary tools to design, create and run user interface models.

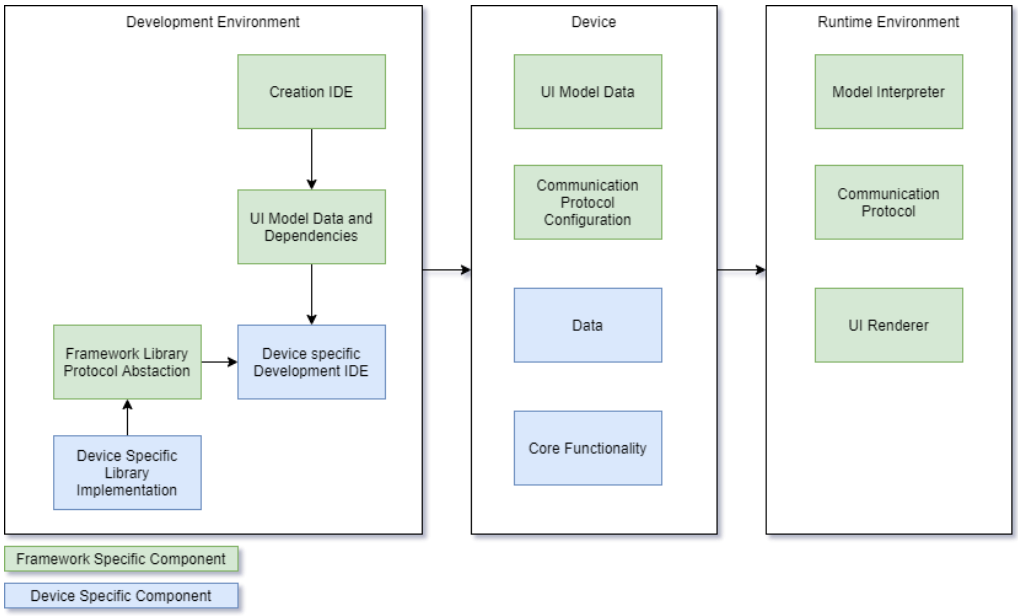

Fig. 1. *Concept block diagram*

2.1. Development Environment

The development environment is represented by a workstation that is able to communicate with the specific device and upload programs on it. Also the development environment should be able to run the creation IDE (Integrated Development Environment) provided by the framework. As presented in Figure 2 the development environment is separated in two distinct components: the Framework Development Environment that provides the necessary tools to design, create, test and export a model and the Device Development Environment that provides the necessary tools to interact with the desired device, based on the features provided by the manufacturer. The two components have

the exported model data as a common component to be able to integrate the model with the device specific functionalities.

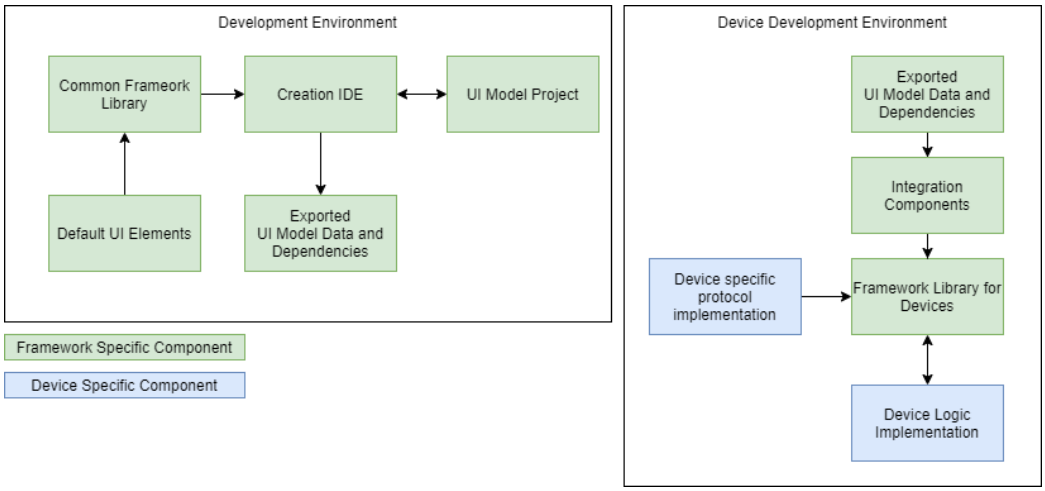

Fig. 2. *Framework and Device Development Environments*

The Common Framework Library is a collection of components that are common for the Development Environment and the Runtime Environment and allows component sharing between the two. It also contains UI element primitives (rectangles, circles) to allow the creation of custom elements and an UI element interpreter that allows the loading of the Default UI Elements library that contains some default UI elements such as a label, or a button, and also the loading of custom UI elements created with the model.

The custom elements of the model can be defined in the Creation IDE and loaded at runtime by the framework, so if a new device is connected to the network, all the provided custom elements used in the UI model will be loaded without the need to restart the Runtime Environment.

Following the Common Framework Library is the Creation IDE is a software component provided by the framework to make the creation of a model easier, providing visual tools to design the model using the drag-and-drop to move the UI elements and position them on the screen. The UI Model Project is a UI model in a format that the Creation IDE is able to interpret and open for editing. Figure 3 presents a possible representation of an UI Model Project, in JSON language that defines a simple control to display text at a certain position on the rendering area.

After the design and creation process is over the result is represented by Exported UI Model Data and Dependencies component and is destined to be used with the Device Development Environment (Figure 2) to be integrated with the desired device. This result should be made available for the Framework Library for Devices via the Integration Component to allow the device logic to use the defined variables and actions to update the user interface.

Fig. 3. *UI Model file in JSON format*

The Device specific protocol implementation, in the Device Development Environment, is a component that depends on the communication capabilities of the device and provides the implementation for the data transfer protocol to the Framework Library for Devices to allow it to communicate with the Runtime Environment and transmit the necessary data for the user interface rendering and update.

2.2. Device

The device is defined as the controller and the sensor or actuators connected to it. The framework only requires from the device to provide a wired or wireless communication mechanism. Figure 4 presents a possible structure of a device form the software implementation perspective.

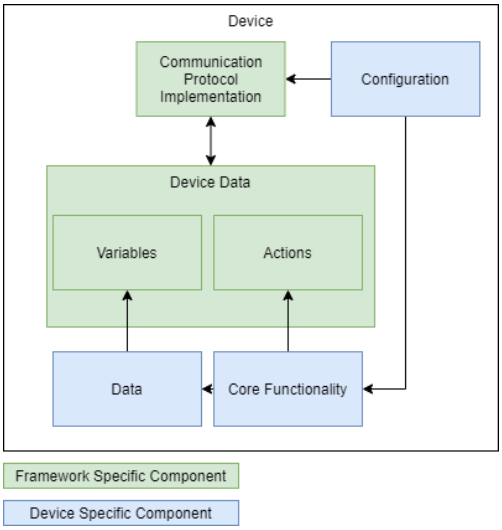

Fig. 4. *Device Component Concept block diagram*

The Device Data component represents a structure that the framework can interpret in order to be able to display the data (Variables) and also present the user all the possible interactions with the device (Actions). The implementation for each action and also the values for each variable needs to be provided by the device from the Core Functionality component and the Data component.

Everything is done according to the data provided by the user in the Configuration component, also the Communication protocol is configured that way, depending on the communication capabilities of the device.

The Core Functionality represents the min component of the device and provides the implementation for data collection and actuators control, depending on the purpose of the device.

2.3. Runtime Environment

The Runtime Environment represents the most important component of the project because it provides the means to interpret, render and interact with UI models created with the Development Environment. As presented in Figure 5 the environment consists of four big components: Communication Protocol Implementation, Common Framework Library, Default UI Elements and UI Model Runner.

UI Model Runner is the most complex component because it has the responsibility to interpret (Model Interpreter component), manage (Data Manager component) and display (UI Renderer component) all the data received from the device. Also all the actions that result from the interactions of the user with the UI are captured and interpreted by the Model Runner and then sent back to the device using the Communication Protocol.

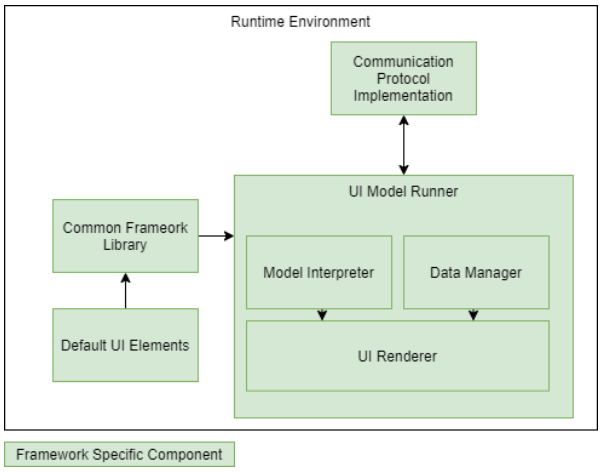

Fig. 5. *Runtime Environment block diagram*

3. Proof of Concept Implementation

The purpose of the proof of concept is to create a simple user interface model and based on concept render it, update the displayed data when new data is available from the device and create a simple UI element that allows the user to interact with the device (a button). Given that the purpose of the project is not data collection all the data transmitted by the device are hardcoded.

Following the diagram presented in Figure 1 the proof of concept was split into three components: Development Environment, Device and Runtime Environment.

3.1. Development Environment Setup and Framework Implementation

A workstation running Windows 10 was chosen as the development environment for the proof of concept and also for the framework itself.

As the framework is split into multiple components multiple programming languages were needed to implement each component. For the most part the framework is implemented in the C# programming language, and also the Runtime Environment components. The parts that interact with the device (Framework Library for Devices) are implemented in C/C++ depending on the level of each component.

To optimize the memory usage on the device a new format for the model representation was defined along with the one presented in Figure 3. All the components were simplified and only given a limited set of properties (position, size) and depending on the purpose a display text property in case of the label or an action identifier property in case of the button. As presented in Figure 6, if we want to represent a simple rectangle that has no text or action associated we only need 24 bytes, and this is the absolute minimum size for a control representation. The ID represents the type of the control, X is the positon on the X axis, Y on the Y axis (represented in screen coordinates, with the origin on the top-left corner). A modified version of the decorator design pattern was used for the implementation of the UI elements because the decorator provides a flexible alternative for subclassing for extending functionality [2].

ID (8 bytes)	X (4 bytes)	Y (4 bytes)	W (4 bytes)	H (4 bytes)

Fig. 6. *Binary data format for a simple UI Element*

3.2. Device Specific Implementation

As a test device an ESP8266 was chosen because of the wireless communication capabilities, and the protocol was chosen to be TCP/IP, with the device acting as a client and the Runtime Environment acting as a server. As the data and the actions performed on the device were hardcoded the implementation was only focused on the data transmission and reception and also data serialization/deserialization and interpretation on the device part.

3.3. Runtime Environment Implementation

The Runtime Environment consists in two main components (Figure 7) the server component that manages all the data received from the device and facilitates the communication between the device and the client component. The client component that interprets, renders and updates the model.

The two components are separated executables as such they can be run on different devices if configured properly.

In this proof of concept the Server Component was executed on a Raspberry Pi 4 Model B that was also configured to act as wireless access point for the device. The Client component was executed on a Windows.

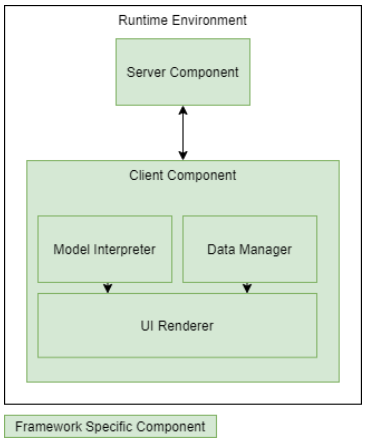

Fig. 7. *Runtime Environment implementation components*

4. Testing and Results

The first step was to create a simple model that only contains a rectangle, configure a simulated device to connect to the Raspberry Pi and send the model data to the server application and render it on the Runtime Environment client application. Figure 8 shows a rendered model from a simulated device (The Runtime Environment and the server application are the same for this simulation).

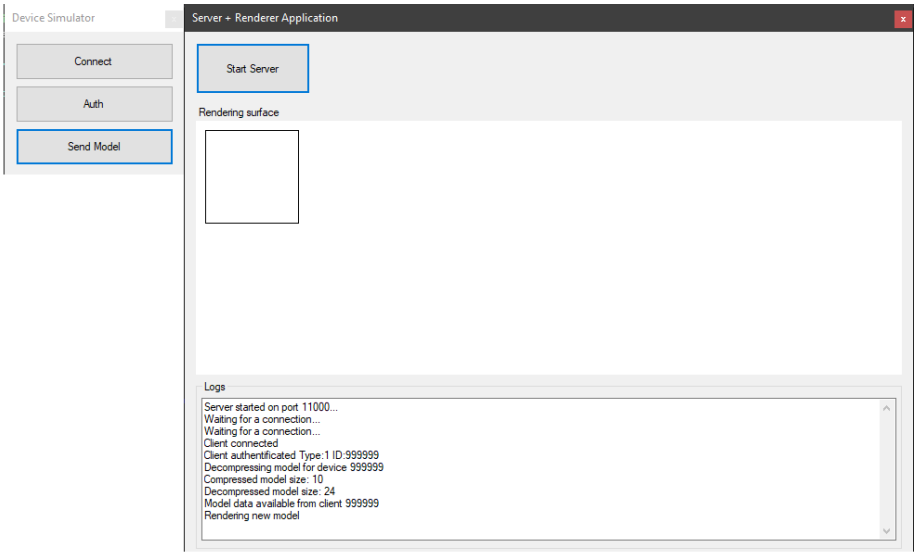

Fig. 8. *Screen capture of the first rendered model using a simulated device*

After that a model containing a label was created (Figure 9), following the model presented in Figure 3 the label was configured to display some data from the device. The other variables, besides the position, type and static text data present in Figure 9 was put in the device's Values map for the system to be able to recognize them ($0

represents the identifier for the counter). The text is parsed using regular expressions and the identifier is replaced with the actual value (10, as presented in Figure 9). Figure 9 also presents some helper functionality needed in the testing process.

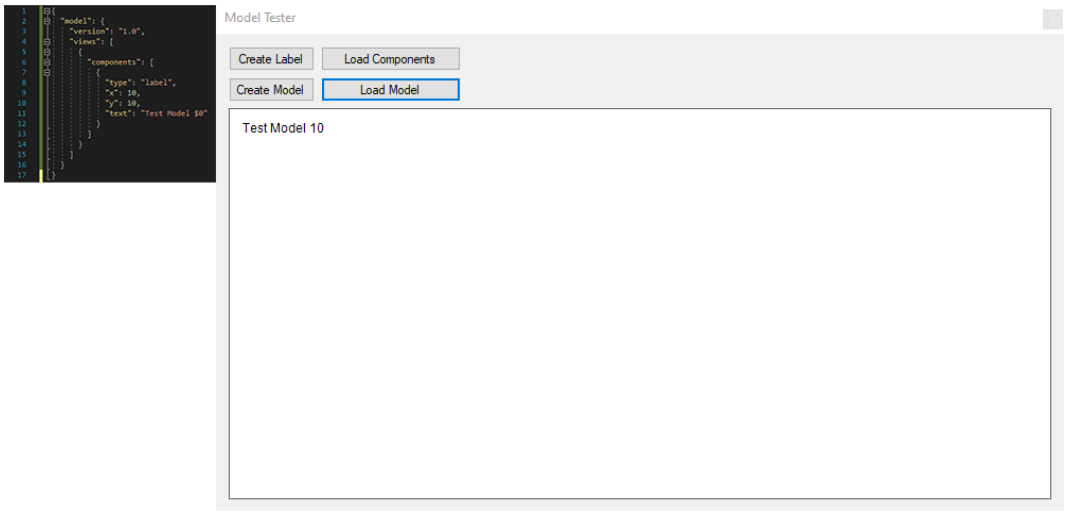

Fig. 9. *Label JSON representation and screen capture of the rendered result*

5. Conclusions

The purpose of the project was to provide the concept and implementation of a framework that provides tools for creating user interfaces for small headless devices. After the implementation testing of the proof of concept can be concluded the following:

- Given the framework and runtime is already implemented, the design and usage of an user interface for the test device was made with ease.
- The main advantage of such a framework is that it fully decouples user interface logic from the functional logic.
- The same concept implemented in the proof of concept can be applied using multiple devices.

References

1. Bahga, A., Madisetti, V.: *Internet of Things: A Hands-On Approach.* Vijay Madisetti, 2014.
2. Gamma, E.: *M Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, 2012.
3. Romanca, M.: *Microprocesoare și microcontrolere* (*Microprocessors and Microcontrollers*). Transilvania University of Brașov Publishing House, 2015.