moZo – A Concept for Gastronomic Service Improvement and Optimization

Eric Gabriel*, Michael Ram†, Henner Nielsen‡, Helmut Dispert§

Kiel University of Applied Sciences
Faculty of Computer Science and Electrical Engineering

Abstract

Customer satisfaction is of utmost importance for all types of business. This is especially crucial when direct contact to the customer is required, as for example in restaurants where a good customerâĂŞwaiter relation can make the difference. Waiting for service is common and often depends on an unspecified communication attempt, to get attention for placing order, paying bills, etc. Obviously an optimization of this work flow is desirable and will have a direct impact on client satisfaction and sales.

In this paper we discuss how this problem can be solved using state-of-the-art wireless microcontroller technology. The project employs a wireless sensor network as a table-based bidirectional communication and alert system, allowing the guests to ask for service and to receive a short feedback message. The presented solution covers the table (desk) devices and an Android application to handle requests, manage orders, and optimize service. It also includes a server for the system configuration, the communication between the devices and for showing the restaurant performance.

For maximum flexibility, the desk devices are based on Arduino microcontroller boards and the ZigBee wireless standard. The server runs on an ARM architecture Raspberry Pi using HTML5, Ajax and WebSockets for system configuration and service management. The client application supports the Android 4.0+ platform, using a clear and intuitive visual interface.

I. Introduction and Idea

Due to the lack of existing options to call a waiter/waitress except eye contact and raising the hand while someone is in the line of sight, we think that a solution for this problem is definitely needed. Customers will save a lot of time and nerves at the lunch-break and the restaurant might be able to serve more customers per day, while the customers will be satisfied with the service. In addition we want to create an innovative, easy to use, adaptable and scalable solution for the restaurant side, which makes the daily working procedures more comfortable.

The main idea of this project is to create a user friendly solution for restaurants and their customers. From the customer side we would like to extinguish the following problem: The customer wants to call upon a waiter/waitress for services, but there is no according person in the line of sight. Sometimes one has to wait several minutes for taking an order or paying the restaurant bill. To eliminate this problem, we want to implement a communication device with a button at every table within the according restaurant so that the customer might call a waiter/waitress by clicking the button and getting a visual response that the service has been informed and will serve the customer soon. From the restaurant side we would like to develop a complete and modern solution for the billing and the service according to normal restaurant activities using portable tablets and a server that acts like a cash register with several options for example to adapt the system

*Eric Gabriel, eric.gabriel@student.fh-kiel.de
†Michael Ram, michael.ram@student.fh-kiel.de
‡Henner Nielsen, henner.nielsen@student.fh-kiel.de
§Helmut Dispert, helmut.dispert@fh-kiel.de

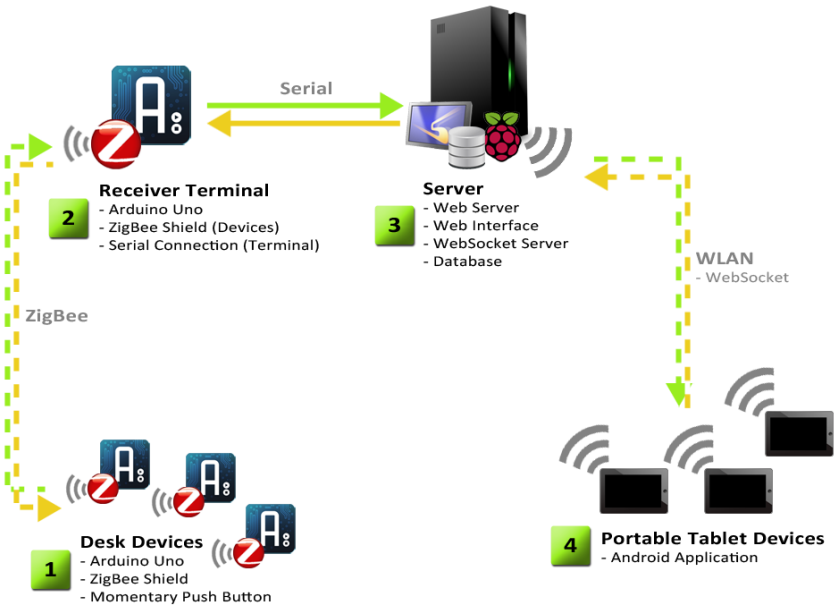

Figure 1: *Communication Flow of moZo*

to different restaurants and viewing what has been ordered at any existing table.

II. Implementation

II.1 Concept and Communication Flow

A desk device's button (1, see Figure 1) is pressed by a customer when service is desired. The desk device sends a message with the device's serial number, a command and the push button state. This information is sent via a wireless ZigBee connection to the receiver terminal (2), which consists of an Arduino Uno with an attached ZigBee shield. The receiver terminal splits the received data (serial number, command and push button state) and sends this via a serial connection to the server (3). The server consists of a Raspberry Pi, which runs a web server with a web interface, a WebSocket server and a MySQL database. The server processes the data, looks up the according table number of the requesting device's serial number in the database and sends the information via TCP/IP to every portable tablet device (4). Every tablet device receives this information and the according table starts to pulsate in a red color in the Android application.

The tablet device of the employee, who has confirmed to serve the table, sends an acknowledgement back to the server via TCP/IP. The server sends a message to all tablets that the table will be served. The table is now highlighted with a green color in every application. After that the server looks up the ID (MAC-Address) of the according table in the database and sends the acknowledgement and the ID via the serial connection to the receiver terminal. The terminal sends the acknowledgement via ZigBee to the according desk device and the customer gets a visual feedback that this table is served soon. The momentary push button stops pulsing and is constantly illuminated.

While taking the orders using the Android service application, the button illumination of the served table is automatically switched off.

II.2 Hardware

Two independent programs had to be developed to realize the desk device functions as well as the terminal functions for these two individual devices (see Figure 2).

II.2.1 Desk Device

The desk device consists of a microcontroller, a ZigBee-module (digi xBee) for wireless communication, a battery, a power switch and the momentary push button for the user input.

In the main loop the program requests the state of the input button and switches a variable according to its state. To make the system as flexible as possible we're using the xBee modules in API-mode (API=2). This mode enables the modules to use a protocol to transmit data, meaning, a checksum of the packet is calculated to avoid junk data and every packet received by the terminal module is confirmed. It offers several other benefits such as requesting several properties and configuration details of the modules. The system makes use of these features by requesting its own MAC address (which is used to address the modules) and generates a string containing this address as well as the current state of the system. From the system's point of view there are three possible states: 0 (the initial state, button has not been pressed), 1 (state when the button is pressed and pulses the illumination) and 2 (the request has been acknowledged by the waiter). If the state of the button is changed by pressing the button, the system sends the payload containing the own address and the state of the button via xBee to the receiver device until it receives a response that the command is understood correctly. Once a second the microcontroller checks whether new packets have arrived or not. If so, the program analyzes the payload and reacts by switching the illuminated button and sending out the corresponding acknowledgement to the server via xBee.

The delivery of each packet is made sure by the measures mentioned above; firstly, a random number between 1 and 1000 microseconds is used to delay the transmission artificially to avoid simultaneously arriving packets on the receiver side. Secondly, this process is repeated until the xBee module confirms the successful delivery by receiving the information from the receiving site that the checksum of the received packet is correct. And in a third step the system controls, if a DEVACK is received within the next three seconds, which indicates that the whole transmission was successful and well understood from the counterpart, i.e. the receiving site.

The device is also able to convert the input/output strings into hexadecimal numbers (necessary for addressing) and back to string. As mentioned in the description of the schematic, it is also possible to output detailed status information of the current connection, the transmission and several other useful information, if it is connected to serial port.

II.2.2 Receiver Device

The counterpart of the desk device is represented by the server device. Although both devices share some of the source code (like for sending/receiving packets using the Zig-

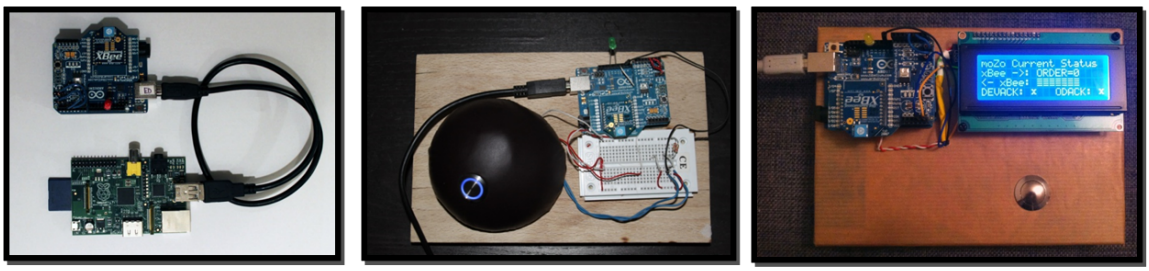

Figure 2: *Different types of devices: receiver device (left), desk device (middle) and its debug information (right)*

Bee module, converting string to hex values or change parameters of the ZigBee module), they differ in many ways from each other. The reason for that is naturally the difference in the task at hand.

Like the desk device the server device has available an interrupt function that is executed every second and listens for new packets. If new packets are available, the payload containing the string with the address, command and state of the requesting device is processed and sub parted. For future applications that implicate the automated answer of the server device it is also capable of responding to this request immediately using the received parameters; nevertheless, this feature isn't used yet. After processing the string and checking for correct content it is output to the serial port and from there further processed to generate the respective actions and answers.

The server device also listens permanently on the serial port for incoming strings. If the length, it's syntax and the contained command fulfill the required form (i.e., is whether an ODACK or a DEVACK), the device constructs the address from that string, generates a new xBee-Request and transmits it to the requesting desk device, again expecting the respective answer from it. In default mode the reinitialization of transmitting a command that is not responded is the duty of the script accessing the serial port of the server device – but as well as the "manual" answers to incoming requests described above the device is capable of automating this process as well with just a few adaptations.

As well as the desk device this device is also equipped with a second (virtual) serial port for the wireless communication over the ZigBee module. In contrary to the desk device (where the physical port is just used to output additional information and not necessarily required) this second port at this device is inevitable since it must communicate as well to the software module on the server via the physical connection as to the desk devices wirelessly using ZigBee

II.3 Software

II.3.1 Terminal Software

Our terminal is a Rapsberry Pi, which runs Debian Linux as well as Apache 2.2.22 with PHP 5.4.4 and PDO support, MySQL 5.5.28 and a node.js server.

Serial Communication

Since the receiving Arduino device is serially connected to the Raspberry Pi, a script is needed in order to manage bidirectional serial communication and to connect with the web interface as well as to relay in- and outcoming requests to/from the Android devices. This script ensures protocol compliance throughout the attached components.

node.js Server

A node.js server has been created using the WebSocket technology, that came up with HTML5. It is possible to create full-duplex, persistent TCP connections between the clients and the server. Therefore this technology is the best solution to offer near-zero latency push services like it is needed in our application.

The WebSocket server listens for new connections and possibly this connection is appended to the connection array. Then a message event listener is bound to every connection. In case a message is received the type is determined. Type 0 messages are broadcasted. If a message can't be sent – because of client connection closures etc. – this connection is removed from the connection array. Type 1 messages include requests for script executions. A client can ask the node.js server via a WebSocket connection to execute a script at a given path and with given parameters. After the node.js server has finished this execution asynchronously the result is delivered to the requesting client. This means the client can get any kind of responses such as database queries or similar operations without the need to open additional HTTP connections.

Web Interface

The web interface is secured by a username

and password login procedure. After successful login one is able to see the actual floor plan of the restaurant with the accordingly placed tables. The appearance resembles the Android application's view. The web interface has some similarities to the application like viewing the actual states of the tables and displaying the orders by clicking on a table, but also some other administrative settings like changing menu categories and items in the database. Moreover it is possible to add picture icons to the different categories. The admin role is able to change/add a different floor plan (image) to the interface. This new floor plan is automatically transmitted and displayed at all tablet devices. In addition one is able to add new tables, place tables and delete tables from the floor plan. The new table alignment is transmitted to all tablets and displayed as well. The web interface can be used to display statistics tables, which has been fetched out of the database showing interesting data like:

- What has been sold when and how often?
- How many requests of customers have been submitted?
- Which waiter/waitress has served the most?
- How fast have the requests been answered?

II.3.2 Android Application

At the start screen (Restaurant View, Figure 3) of the application one is able to see the restaurant map as a background image, which is dynamically fetched from the server. Therefore this map is not fixed and can be changed using the web interface according to the restaurant's needs. The different tables are also set to their positions using the web interface, they appear immediately at every used waiter device, which is connected to the server. The table icons are displayed with their respective table number, their balance and their request state (no current request, requesting service or being served). The Android application offers general service functionality such as placing/deleting orders, which can be managed by simply touching the table icons. All data is organized in a central database on the terminal and can be retrieved by all Andoird devices as well as the web interface on demand.

III. Discussion

III.1 Strengths

- We designed the whole system to be very user friendly. Everything is easy to use and self-explanatory.
- The system is adaptable to almost every kind of restaurants without self service. The owner just has to add a specific floor plan, add the menu items and place the tables using the web interface.
- The system is an advantage for both sides. The customer will be satisfied with the service and the waiter/waitress does not need to write everything down, remember the orders, be highly alert if somebody has a wish, etc. The restaurant manager might as well increase the efficiency of his service personal.
- The system is very compatible. One might use any current Android based tablet or smart phone. It is not necessary to use a Raspberry Pi as terminal. One might just use a regular computer instead.
- The system is complete. You do not need other solutions in the whole cash register and serving process.
- The system is highly scalable. You can add as many tablet devices or desk devices as needed.
- If an item runs out of stock it can be easily disabled in the menu management. After that it won't be shown on any of the devices while it is not available.
- The administrator is able to analyze the restaurant statistics.

III.2 Weaknesses

- The application display is not able to zoom at this point of time. Larger restaurants which consist of different levels or

several huge rooms may have a problem with the display. As mentioned in the next chapter "Prospects" this is considered in the future plan.
- The coordinating script is not an optimal agent between the Node.js-Server and the serial connection regarding the communication timing. The connection between the tablet device and the Node.js-Server is not as good as expected. The concept works theoretically perfect but the tablet device does not manage this as planned. This seems to be a problem because the testing tablet device needs to run an android modification, which is currently in an early state and has a few abnormalities.

IV. Conclusion

IV.1 Prospects

- Optimize the software communication by using a native Android device such as the Nexus 7, Nexus 10
- Enhancing the project by upgrading each customer table with an additional tablet, enabling the customer to access the food / beverage database and order from the tablet without consulting the waiter
- Add functionalities to the web interface and optimize the interface for the waiters
- Adding a swipe or zoom ability to the screen view for covering different rooms or levels in larger restaurants.
- Adding different language support to the application and the web interface. That should not be a big deal, because there are basically only a few single words and error messages. The menu items can be added in every possible language by the admin.
- Extend the functions of the desk devices by installing non-volatile memory to enable the to change configuration details dynamically
- Redesign the Desk devices, shrink the overall size and optimize the power consumption which enables them to be integrated in a larger variety of decoration items.

IV.2 Conclusion

In conclusion we have to mention that our uncertainty regarding the interaction between the large amount of different components was in general overcome relatively fast. All of the components had no complex problems in the beginning, only in the detail we had a few difficulties regarding the communication, whose are now successfully eliminated.

Since the microcontrollers reach their computational limit our programs had to be optimized in several hindsights for managing everything according to our needs. Same goes for the server unit: We had to increase the performance of the device using several measurements. The core frequency, the SDRAM frequency and the GPU frequency of the Raspberry Pi was very important for the whole interaction of the system, which is why these components are massively overclocked. To further reduce the access times we also put an extreme performant SD-Card instead of the normal class 10 card which is hosting the operating system.

Nevertheless we managed those difficulties and developed a system that is complete and makes successfully use of several different technologies that are weaved together in a way that is pretty close to a real world application. Although there are some prospects we would like to include in the future the whole system is nearly ready to use in a restaurant.

References

[1] Arduino Software (28.01.2013): *Arduino Uno* http://arduino.cc/en/Main/ArduinoBoardUno, last access: 07.02.2013

[2] Arduino Software (27.02.2012): *Xbee Shield* http://arduino.cc/en/Main/ArduinoXbeeShield, last access: 07.02.2013

[3] The Pi Hut (Mann Enterprises Ltd): *The Raspberry Pi* http://thepihut.com/pages/the-raspberry-pi, last access: 07.02.2013

[4] WebSocket.org: *What is WebSocket?* http://www.websocket.org/, last access: 07.02.2013

[5] Android Developers: *Get the Android SDK* http://developer.android.com/sdk/index.html, last access: 07.02.2013

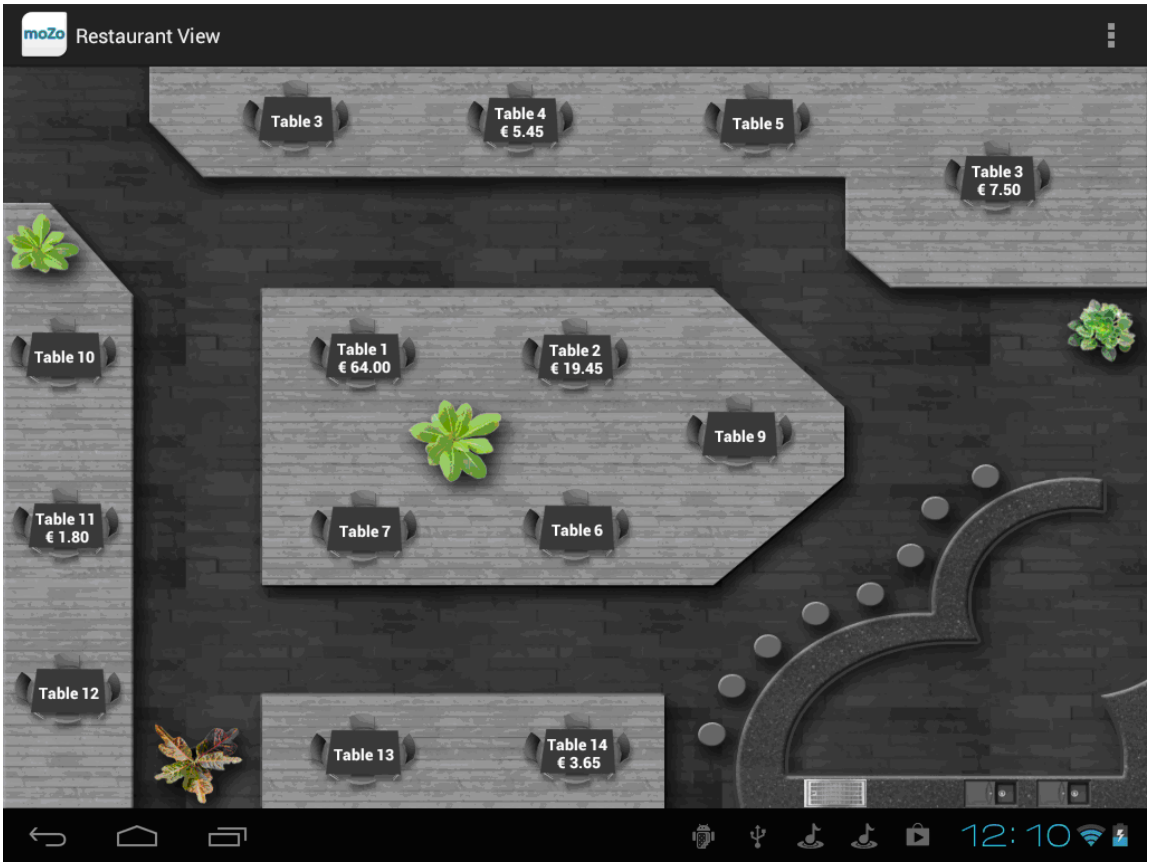

Figure 3: *Screenshot of the Android Application*