Lessons Learned from Developing Prototypes for Customer Complaint Validation

JERE GRÖNMAN, PETRI RANTANEN, MIKA SAARI, PEKKA SILLBERG AND HANNU JAAKKOLA, Tampere University of Technology

This research introduces two prototypes installed in vehicles and a cloud service for autonomous collection of data. The prototypes utilize camera, location data, and timestamps to help those responsible for managing customer complaints, and to improve the overall quality of the provided customer service. The use of the system is illustrated by two cases: tracking and photographing bus stops, and tracking and photographing recycling areas. The first prototype is implemented for the Android mobile platform and the second one for the Raspberry Pi single-board computer. This paper discusses the differences and challenges faced in designing and implementing the two prototypes for different platforms.

1. INTRODUCTION

The Internet of Things (IoT) is the expansion of Internet services, which connects everyday physical objects to a network. This connection between network and physical world objects makes it possible to access remote sensor data and to control the physical world devices from a distance. One study addressing the IoT, which is cited quite often, is "The Internet of Things: A survey" [Atzori et al. 2010].

In this research, the focus has been redirected toward the Wireless Sensor Network (WSN) type of solution. The basic features of sensor networks were compiled in a survey by Akyildiz et al. in 2002. In this research, we present two different prototypes for collecting data. These prototypes were designed as nodes of WSN. The design processes were iterative and the main goal was to improve the prototype in every iteration round. This study is a "lessons learned" type of research on software quality and prototype testing, where we present the problems encountered and the solutions to them. This research is a continuation of our research into different areas of IoT [Saari et al. 2016; Saari et al. 2017; Grönman et al. 2018]. Often, the Agile method is used more than the traditional plan-driven methods (such as the Waterfall method) when developing prototype systems. The authors of this paper have discussed the challenges of modeling in an earlier study [Jaakkola et al. 2016].

The motivation for this study came from two transportation companies. Their customers often complain that the service is not at an acceptable level (e.g. the bus was not on time, or did not stop; trash was not collected on time). For companies, it can be difficult to ascertain the validity of the complaints, possibly causing unnecessary expenses when repeated complaints occur. This study and the two use cases presented in this paper illustrate configurable conditions for area observations (based on location, speed of the vehicle, cameras, and other sensors). In the past, the drivers photographed locations and made observation reports manually, but this process turned out to be tedious and error-prone. Thus, it was decided to design a system that could work autonomously without input from the driver. The companies can use the collected data to validate customer service requests/complaints, and to improve the overall quality of the provided customer service.

Author's address: J. Grönman, Tampere University of Technology, Pori, P.O. Box 300, FI-28101 Pori, Finland; email: jere.gronman@tut.fi.

In: Z. Budimac (ed.): Proceedings of the SQAMIA 2018: 7th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Novi Sad, Serbia, 27–30.8.2018. Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

There has been a lot of research on position systems such as vehicle tracking systems. For example [Lee et al. 2014; Jisha et al. 2017] introduced vehicle tracking systems, where the location data are stored to the database or cloud and the data could be shown with an Android mobile application. Jisha et al. deal with bus tracking systems. In these studies, the focus was on real-time tracking using the Global Positioning System (GPS). The main idea was a tracking service for customers, and the quality assurance of the transportation service was not discussed.

In our use cases the sensor nodes (mobile phones and Raspberry Pi 3 computers) send the data to the cloud service. The idea and the model of the data gathering node system were introduced in [Saari et al. 2015]. The rest of this paper is structured as follows. In Section 2, we outline the research environment and its components. In Section 3, we describe the "bus stop" case and in Section 4 the "garbage truck" case. Section 5 includes a discussion where the findings of this prototype development process are handled. The study is summarized in Section 6.

2. HIGH-LEVEL ARCHITECTURE

The goal of the system is to provide a tool for those processing customer complaints. In our case, two use cases acted as pilot studies for testing the functionalities of the system. The first case, "bus stops," consists of tracking a bus traveling on the route by collecting images, location data, and timestamps. The bus company participating in our pilot study reported that common complaints reported by customers are about the bus not arriving on schedule (too early, too late or not arriving at all) or the bus not stopping even though there were people waiting at the bus stop. The latter issue especially can be difficult to validate, and the bus company was interested in improving the quality of their bus service by finding out if and when the complaints reported a real problem.

The second use case, "garbage truck," collected the same data (images, location, timestamps). The purpose was to keep track of when the garbage truck visited the recycling area and if the bins were not emptied, and whether there was something in the area that prevented the truck from doing its work. In some cases, pictures were already being taken by the garbage truck drivers in the area around Pori, but this is, in general, manual work, and capturing and managing the pictures can be tedious and error-prone. Similarly to the bus company, the company running the garbage collection receives complaints about the quality of the work, and the company was interested in an automatic system for collecting data around the recycling sites to validate the complaints.

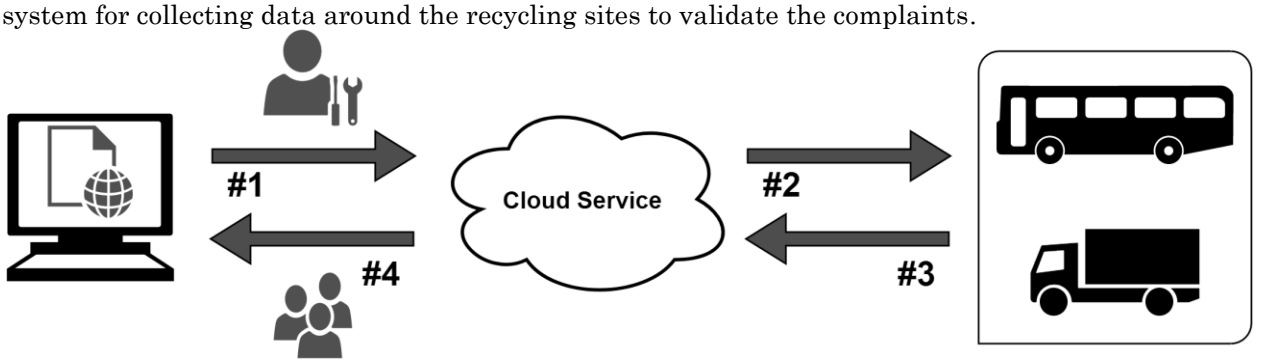

Fig. 1. High-level diagram of the system.

Figure 1 illustrates the high-level architecture. The system consists of a central service, which provides representational state transfer (REST) application programming interface (API) for client-side interaction and remote procedure call (RPC) functionality for delivering tasks to (back end) devices or for submitting task results. A simple web portal is implemented using Hypertext Markup Language (HTML) and JavaScript, which interacts with client-side methods. The web user interface

allows the user to create tasks (#1, Figure 1), which contain the pre-conditions for device operation, the selection of devices that participate in the tasks, and the desired output parameters. The tasks are delivered to the target devices (#2, Figure 1).

The pre-conditions contain options such as coordinates (or areas), time intervals (e.g., only take pictures during working hours), and velocities (e.g., should the device be moving at a certain speed or stationary). The output parameters notify the devices as to what information should be returned in the result responses (#3, Figure 1), such as pictures or location data – by default, all responses must contain timestamps.

The results can be returned in near real-time or in batches. In our use cases, there is no need for immediate responses and in general, the results can be returned at a time most convenient for the device as long as the results arrive within a reasonable time period (for example, within 24 hours). In general, the tasks do not contain information on the expected amount of output data. In our use case, most of the transmitted data consists of captured images. The amount of images is highly dependent on the speed at which the vehicle is moving and how long the vehicle stays within the designated area. The prototype application will attempt to compensate for the vehicle velocity (e.g., by capturing more pictures when the device is moving faster), but the tasks themselves do not contain any guidance for this functionality. The primary reason for this is that the device is better equipped to estimate its own capabilities and features than the service and providing overly detailed parameters would only complicate the tasks by requiring individual customization for each device.

The submitted results (#3, Figure 1) are indexed in the service and can be freely browsed by the user (#4, Figure 1), for example, by selecting a bus stop (coordinate) and the time reported in the complaint. The core service uses a platform developed in a previous project [Iftikhar et al. 2018], and for historical reasons messages based on the Extensible Markup Language (XML) are used, though other formats (e.g., JavaScript Object Notation) could be used in our use case as well.

In our current implementation no further image analysis is performed either on-device or in the service. In principle, it would be beneficial if image processing could be utilized to detect problems reported in customer complaints. In practice, the variations in environments (location, time of day, snow, rain, etc.) and different use cases (detection of people, undesired objects) make it very challenging to develop a reliable algorithm. As long as a reasonable amount of pictures is shown to the person responsible for processing the complaints, a human observer can detect problems by looking at the pictures

3. USE CASE: BUS STOPS

In the first use case, a prototype was installed in a bus traveling along a bus route within the City of Pori. The prototype is fully autonomous and does not require any input from the bus driver. The stops on the route were assigned to the prototype as GPS coordinate targets. The program code reads GPS coordinates continuously and compares them to the assigned targets. When the coordinates match, a picture is taken. The program code utilizes an implementation of the Haversine Formula, which determines the great-circle distance between two locations and is relatively simple to implement yet accurate enough for our use cases. The application can be installed on any reasonably new Android device and takes advantage of the built-in sensors and camera of the device.

The prototype keeps taking pictures in a predefined interval as long as it remains within the range of the target. The prototype scales both the time interval and the range from the target based on the speed of the bus to enable taking pictures at varying speeds, and also to compensate for the delay in waking up the camera. The idea was to take pictures of the approach to the bus stop to find out whether customers were present at the bus stop and also to obtain evidence (photos, timestamps, and location data) that the bus had passed – or stopped at – the bus stop on a certain date. The approach to one bus stop is illustrated in Figure 2.

Fig. 2. Approach to a bus stop showing pictures taken at varying distances from the target.

A background process was created in the application for sending the pictures after capture, although, depending on the network connection speed, the upload may not be real-time. A route with fewer bus stops and a smaller number of customers was chosen for the first prototype. In our case, the route had a few dozen stops, but within the city limits some routes may have several hundred stops (especially if the route is traveled in both directions). Furthermore, the local bus company was instructed to provide us routes with higher than average amount of complaints.

4. USE CASE: GARBAGE TRUCK

In this use case, a prototype was installed in a garbage truck. The truck has a predefined route where there are certain recycling areas nearby shopping centers. The locations were assigned to the prototype as GPS coordinate targets. A route with frequently visited targets was provided by the garbage truck company participating in our project. The goal was to select targets of varying size (a gas station, a supermarket, and a larger shopping center) located around the City of Pori. The location tracking was performed in an identical fashion to the "bus stop" case, with the applicable code re-written in Python. Similarly, the prototype is fully autonomous and does not require any input from the driver.

The prototype consists of a combination of a Raspberry Pi 3 single-board computer and commonly available sensor components (Adafruit Ultimate GPS HAT and Raspberry Camera Module V2 NoIR). Its operating system is Raspbian Stretch and the program code was made in Python, which is one of the commonly used languages for prototyping with Raspberry Pi. The prototype requires a 3G/4G -wireless modem to establish an Internet connection via Wi-Fi.

Fig. 3. Three pictures taken from the recycling area. On the left: in daylight; in the center: at night; on the right: a blocking obstacle.

In this use case, a connection to the cloud service was established once a day. During the test period of three months, more than 6500 pictures were taken of the targets. Pictures were taken both in daylight and at night. Figure 3 presents a comparison between day and night. Figure 3 also shows

a situation where an obstacle, in this case, a car, is blocking the truck's access to the recycling bins. The first two pictures in Figure 3 present normal daily operation, but in the case of the third picture, the car could have prevented the truck from emptying the garbage bins, possibly causing later complaints from customers about full containers.

5. LESSONS LEARNED

The development process for the use case prototypes was iterative in nature. Our goals were to both validate our ideas and to ascertain in a short period of time which technical solutions would work in realizing the prototypes.

One approach would have been to install cameras at each location, but in practice this was not feasible. In both cases all locations were outdoor locations and it would have required a considerable effort to guarantee the availability of electricity, and that the devices would not get wet, vandalized, or broken in the cold weather. The assumption was that installation in-vehicle would be easier. A minor concern was the operating temperature as the vehicles would be stored outside in Finnish winter when not in use. The Android implementation was not tested in wintertime, but there were no problems with the Raspberry Pi during the three-month trial run. The device itself did not contain ambient air temperature sensors, but the average temperature during the December-February period was slightly below zero Celsius with the coldest nighttime temperature reaching -21 °C in February [Foreca 2018]. The device was always on during the trial, running on the continuous power provided by the vehicle batteries. This approach also reduced the risk of the device failing to boot up due to cold weather. In the case of the garbage truck, obtaining constant power was a simple matter of using the cigarette lighter plugs, but, in the bus, re-wiring was necessary as the connectors inside the bus did not provide electricity when the main power was turned off.

A bigger problem than electricity was the attachment of the devices (both Android and Raspberry Pi) to the vehicle. In our case we selected a garbage truck that loaded the trash using a lift located in the front of the vehicle. The company also had vehicles that were loaded from the back, but this would have meant that the truck would have approached the recycling area in reverse, requiring camera installation at the back of the vehicle – possibly on the outside of the vehicle. For simplicity, it was decided to only use a camera to take photos through the windscreen. This left the rare case when the vehicle would be approaching the location from an unusual angle, e.g., around the corner of a building, from the side or from an otherwise bad direction for taking pictures through the windscreen. Especially in the bus stop case, there were no pictures available of every bus stop, and even if there had been, we did not want to make individual setups for hundreds of locations. Thus, it was impossible to know how the vehicle would approach each location. Regardless, it was decided to choose the windscreen approach to get the testing underway.

In practice, this approach provided more problems than expected. Initially, there was slight concern about reflections on the glass surface. In practice, this turned out not to be a big problem, because the camera would take several pictures when the vehicle was approaching the location and major reflections did not occur often enough to pose a real problem. However, a more serious problem was how to install the devices in the vehicles.

The curved windscreen of the truck and bus made the traditional suction cup-based attachments unusable. The vibration and movement of the vehicle caused the device to fall off of the window. Additionally, a permanent installation of the prototype was not desirable as there might have been a need to remove the device during testing. It would also have been impractical to make extensive modifications to the vehicles because the vehicles were in regular use by the companies. The installation of the Raspberry Pi was slightly easier as the camera as well as the GPS antenna could be detached and installed in a different place to the device itself. This meant that the components that needed to be installed near the windscreen were more lightweight than a smartphone, which

contains all the components in one package. The ideas for a final prototype installation ranged from ordering various attachment holders from the Internet to using a 3D printer to create a custom casing. In the end, the installation consisted of a lot of two-sided tape. The solution was not pretty and was passable at best. All in all, our primary concern was the device itself, the software, and testing our idea, but perhaps a little more thought should have been put into how to setup the device in a real environment regardless of the trial nature of the tests.

The members of our research team had previous experience in programming applications for the Android platform and also in using the Java programming language. Furthermore, our research team had readily available Android devices - both personal devices and devices provided by the university – that could be used in prototype development and testing. This meant that the prototype development process could be started without the need to learn the basics for a new platform. The two most popular mobile platforms (Android and iOS) provide similar starting points for our use case requirements: APIs for controlling the camera, accessing the Internet, and tracking the device location, making the choice mostly about developer preference.

Creation of a simple application for tracking the location, firing the camera in pre-designated coordinates, and uploading the pictures to a remote service was relatively simple – the APIs are well documented and a simple web search provides plentiful examples for common use cases. In general, only two problems were met related to the programming.

Firstly, we used a pre-existing service developed for previous research projects, which utilized the XML-based data format. Unfortunately, by default, the Android platform does not support standard annotation-based class definitions (e.g., Java Architecture for XML Binding), which meant that we could not directly use the same Java code as we had used in previous Java applications. This is an example of one of the generally minor problems caused by the fact that Android does not provide full API compatibility with Oracle's Java. The problem was fixed by creating an XML parser using the Android's XML pull parser and serializer, which are relatively simple to use though perhaps are slightly more error-prone by requiring modifications to the parser code when the format is modified as opposed to annotation-based solutions, which only require modifications to the class declarations.

The second programming related issue was with our implementation of the camera use. For some unknown reason, especially on older Android devices (Nexus 7 tablet and Samsung A5), using the camera repeatedly and sometimes in quick fashion caused application crashes or the camera got "stuck," capturing only a black screen. Fixing the issue required several attempts with various programming solutions and the implementation was never as stable as we had hoped for.

The issues with stability caused an additional problem. In our initial trials, a member of our research team was present in the bus, and could make corrections or restart the application when problems occurred, but in the future this would not be the case. In the next phase, the device would be installed in the vehicle for a period of three months, during which there could be a need for fixing problems and to further improve the prototype software. Repeatedly visiting the company for prototype maintenance would be a tedious process for the research team and also problematic for the company as their vehicles were in use on a daily basis.

Remotely accessing the mobile device, for example, for restarting an application or installing a new application version was a real challenge. As our application was not available in the Google application store, we could not take advantage of the remote installation options provided by the store, and directly accessing the device over the public Internet – i.e. accessing the dynamically assigned Internet protocol (IP) address – would have been difficult without developing extensive support mechanisms. This was one of the primary reasons (in addition to the unstable camera implementation and problems with installing the device to the vehicle) for dropping the Android implementation and looking for alternative options.

The Raspberry Pi-based solution provided much needed help for the remote access problem. The device is, in practice, a Linux-enabled computer, which means that many of the methods available

for desktop application development are available. In our previous projects we have had some experience with Raspberry Pi in particular, making it a logical choice, though many of the other single-board computer solutions available on the market should work as well. In any case, by using a dynamic Domain Name System (DNS) update it was possible to keep track of the public IP address assigned by the Internet provider. It was also possible to directly use version control (in our case, sub-version) as a "cheap alternative" for deploying new application versions on the device, and access to the device can be achieved using Secure Shell (SSH). On Linux, the applications can also be easily set up to start up on boot or at designated intervals either as services or by utilizing crontab. Crontab was also utilized to run scripts that periodically checked whether our application was still alive, logging the application status, and restarting the application if it had crashed.

OpenJDK is readily available for Linux and can also be used with the Raspbian operating system, enabling the use of Java applications. The advantage was that most of our previously written utility code (accessing the Internet, XML parsing from our server-side implementation, etc.) could be directly used on Raspberry. The disadvantage was that all code that accessed the device sensors and the camera would be re-written as no compatible APIs existed, and the preferred programming language was different (Java vs. Python). Accessing the camera (using the raspistill command line tool) or GPS data (using the gpsd service daemon) with Python is not difficult, though the level of API documentation is not on a par with the Android documentation. It can also be more challenging to find pre-made examples. Many of the example projects found online are, for the lack of a better word, "hacks", and the re-usability of code is more difficult than on the commonly used mobile platforms. This is also one issue that one should keep in mind when deciding which platform to use for rapid prototyping. On the positive side, a more low level API access is available on Raspberry Pi, if such functionality is required.

An important note is that remote access also creates a potential security vulnerability, which should be taken into account, especially when using potentially unstable or vulnerable prototype or development versions of applications. Using a dynamic DNS service also seems to create a hot spot for attempts at breaking into the device using dictionary and brute-force attacks. As a minimal configuration, the default SSH port and passwords should be changed and remote root access disabled. In our case, we used a separate 4G modem because Raspberry Pi does not provide a 3G/4G connection, and the modem was also set up to work as a firewall.

This remote access approach worked fairly well even though there were a few minor problems. Around the heavy industry area and the power plant located in Pori there were problems with cell reception and data transfer. The modem initially chosen also had issues with energy management. Regardless of the configuration options, after a longer period of inactivity the modem would go into a power-safe state, cutting remote access to the Raspberry Pi and sometimes the modem would "hang" requiring a physical restart. Periodically pinging the remote server seemed to fix both issues, though it would have been better if the modem had been configured to function as intended. Another minor issue with the modem was that if power were lost for whatever reason, the modem would not automatically connect to the Internet, and would instead require the user to press a button on the device. In our case a continuous power supply was made available both in the bus and in the truck to fix the issue. Nevertheless, it became clear that it can be challenging to figure out without testing how well a specific modem will work in various conditions and what configuration options are available, especially if cheaper devices targeted to end-user customers are utilized.

6. CONCLUSIONS

This paper presented a high-level diagram for a service designed to help those responsible for managing customer complaints, and to improve the overall quality of the provided customer service. The use of the system was illustrated by two cases: tracking and photographing bus stops, and

tracking and photographing recycling areas. Both cases utilized cameras installed in vehicles and location data. Furthermore, this paper discussed the issues faced in the design and implementation of the use cases. Based on a brief discussion with the companies, the initial reaction towards the prototype applications was positive, and the system was seen as an improvement over the previously utilized manual data collection. Still, to fully assess how the system contributed on the improvement of the customer complaint validation process, a more in-depth study would be required.

In any case, the use cases show that mobile platforms can work as a quick starting point for rapid prototyping – documentation and examples are easily found and the devices contain a number of built-in sensors. The disadvantage of mobile platforms is the lack of options for remote management, and single-board computers (e.g., Raspberry Pi) could provide a better platform if remote access is required. Unfortunately, it can be more challenging to find applicable examples and documentation when compared to commonly used mobile platforms. Additionally, both cases highlighted the importance of environmental factors – such as the availability of electricity, telecommunications, and installation of the prototype – even in cases when the primary goal of prototyping is in software testing or running short trials. The importance is seen especially when the testing is done in a real environment and should not disrupt the daily operation of the participating companies.

REFERENCES

Ian F. Akyildiz, Su Wy, Yogesh Sankarasubramaniam, and Erdal Cayirci. 2002. Wireless sensor networks: a survey. Computer Networks 38, 4 (Mar 2002), 393–422.

Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. Computer Networks 54, 15 (Oct 2010), 2787–2805.

Foreca. 2018. Havaintohistoria. Website. Retrieved May 14, 2018 from https://www.foreca.fi/Finland/Pori/havaintohistoria

Jere Grönman, Petri Rantanen, Mika Saari, Pekka Sillberg, and Juha Vihervaara. 2018. Low-cost ultrasound measurement system for accurate detection of container utilization rate. In 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE.

Ahmad Iftikhar, Petri Rantanen, Pekka Sillberg, Jorma Laaksonen, Shuhua Liu, Thomas Forss, Aqdas Malik, Marko Nieminen, Rakshith Shetty, Satoru Ishikawa, Jarno Kallio, Jukka P. Saarinen, Moncef Gabbouj, and Jari Soini. 2018. VisualLabel Integrated Multimedia Content Management and Access Framework. Information Modelling and Knowledge Bases XXIX (2018), 321 – 342.

Hannu Jaakkola, Jaak Henno, Tatjana Welzer Družovec, Bernhard Thalheim, and Jukka Mäkelä. 2016. Why information systems modelling is difficult. CEUR Workshop Proceedings 1677 (2016), 29–39.

R.C. Jisha, Aiswarya Jyothindranath, and L Sajitha Kumary. 2017. IoT based school bus tracking and arrival time prediction. In 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE, 509–514.

SeokJu Lee, Girma Tewolde, and Jaerock Kwon. 2014. Design and implementation of vehicle tracking system using GPS/GSM/GPRS technology and smartphone application. In 2014 IEEE World Forum on Internet of Things (WF-IoT). IEEE, 353–358.

Mika Saari, Ahmad Muzaffar bin Baharudin, and Sami Hyrynsalmi. 2017. Survey of prototyping solutions utilizing Raspberry Pi. In 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, 991–994.

Mika Saari, Ahmad Muzaffar bin Baharudin, Pekka Sillberg, Petri Rantanen, and Jari Soini. 2016. Embedded Linux controlled sensor network. In 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, 1185–1189.

Mika Saari, Pekka Sillberg, Petri Rantanen, Jari Soini, and Haruka Fukai. 2015. Data collector service - practical approach with embedded Linux. In 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015, 25-29 May 2015, Opatija, Croatia (International convention on information and communication technology, electronics and microelectronics). IEEE, 1037–1041.