Architecture Design to Support a Smartphone-based Student Response System

Aaron Mangel, Alexander Preston, Stephen Hughes, Ben Schafer
Department of Computer Science
University of Northern Iowa
305 ITTC Cedar Falls, IA 50614
amangel@uni.edu, preston@uni.edu

Abstract

The number of smartphones and smart devices in students' possession is ever-growing. It is now seemingly more common to see a smartphone in a student's hand versus the alternatives. Rather than simply allowing these devices to only be defined as distractions in the class room, this project aims to take a proactive approach in harnessing the capabilities of these devices, replacing and enhancing what we currently know as the standard Student Response System, sometimes referred to as clickers, in the classroom. This paper will discuss the existing architecture of Student Response Systems, why a replacement would be useful in a classroom environment, and the development of a new Student Response System which offers more to the educator.

Introduction

Students with smartphones are equipped with an always-connected-to-the-internet device. Often, these devices can be distractions in class. However, with the ability to develop applications for these devices, not much of a gap exists between distractions in the classroom and networked-based collaboration and instant feedback from students. This project aimed to build a smartphone-based student response system which would be flexible and expandable, yet simple to use in the classroom.

The traditional Student Response System (SRS) consists of several major parts; a proprietary input devices for the student, a receiving unit which is typically installed in the classroom, and a software package that allows for the responses to be aggregated. In typical use, the teacher asks a question, showing a list of multiple possible answers from which the students can choose. As the students use their input devices to respond, the answers are collected and combined into a visual representation, most commonly as bar graph.

While this system is proven to work and has been adopted in classrooms across the country, it is limited in functionality and carriers several disadvantages. First, the proprietary devices for students to input their answers are an additional cost to either the institution or student, and are not reusable in the situation where multiple different response systems are in use. Second, the types of questions are limited. Traditional response systems offer multiple choice questions which limits the ways that students can interact with a system.. Also, student responses are typically kept anonymous. While this may be beneficial in terms of student responses being anonymous to each other, instructors may benefit from being able to identify participants for additional assistance and feedback.

Designing a New Architecture

As we approached building a building a more advanced SRS, we wanted to address some of the weaknesses of the current model, as well as add new functionality to the system. One of the most important issues we wanted to tackle was accessibility of the system for students and instructors. The system should be able to run on existing hardware. We chose the Android OS as a beginning point for development because of its accessibility and how pervasive these devices are in the market. This allows both students and instructors to use a variety of hardware to utilize the SRS.

Originally, we considered a system that was more in line with a traditional response system. We began with specific styles of question such as multiple choice and multiple correct. With these types of homogeneous questions, we were expanding on what a traditional system was capable of doing. After working with this for a time, we came to the decision that being able to create questions that were a mixture of question types would provide a great deal of flexibility while still allowing us to perform like a traditional response system.

To begin exploring the implications of mobile input on a shared display system, a prototype infrastructure has been designed and implemented. This system was originally designed from the perspective of implementing a Classroom Response System that could be extended to a) solicit and collect a wide spectrum of student input and b) allow the input put to be processed and interpreted generically. The architecture for this system (illustrated in Figure 1) consists of the following components:

Question Management Package – The software in this component allows an instructor to pose questions or invite students to interact with additional software. This component allows teachers to prepare questions in a variety of types and formats both during and prior to class. Such a system should "ask" questions of the class, or a subset of participants in the class, and manage the potential interaction, sequencing and branching of multiple questions.

Student Input Clients – These clients are software packages that run on the student devices. Upon receiving a question, the software in these components generates an appropriate graphical interface to collect input from the student. It is important to note that the composition of the interface is dependent on the type of question asked. In its most basic form, the client can display a set of multiple-choice buttons, emulating a traditional classroom response system. However, a wide array of additional control widgets are also supported, including, sliders, dials, and dropdowns. As described earlier, these devices can also provide keyboard (either virtual or physical) and mouse or touch-screen functionality.

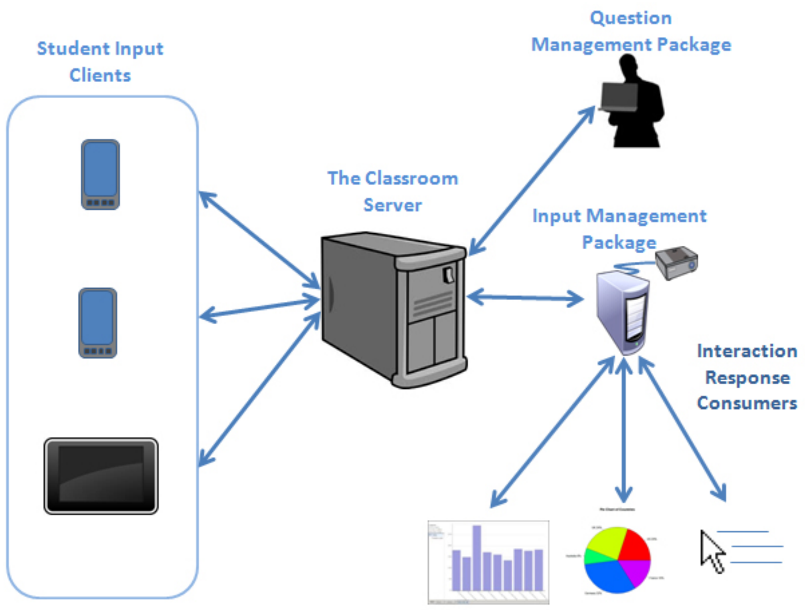

Figure 1 : System Architecture

Input Management Package – The software in this component gathers and pre-processes the responses from the Student Input Clients. It performs basic data aggregation services such as filtering, tallying and categorizing responses or providing simple statistical calculations such as mean, mode, minimum and maximum operations. The data is packaged into a standard API for use by additional components.

The Classroom Server – The components above need to be able to communicate with each other. The server provides a mechanism to allow each of the components listed to focus on its specific job and limit communication to be between the component and the server. The server coordinates message passing and data sharing between the separate components.

Interaction Response Consumers – These components allow for the processed responses to be displayed to the class. These consist of both stock visualization tools – such as distribution graphs/tables and word clouds – as well as custom visualization programs. Consumers can also allow student input to be interpreted as commands, acting as a bridge to specialized OS or API based events to trigger interaction in existing software.

The usage of consumers in the architecture allows for the system to be installed with the inherent functionality of a SRS built in and ready to use, but allows for new consumers to be installed which will provide a currently working system with new functionality. We believe this is where the true potential of this architecture lies. With the ability of extension, multiple Input Management Packages may be communicating with one server while having different consumers available to them. To the server, this makes no difference. Utilizing this feature allows the SRS to be as general or as specialized as the user wishes.

Use Case Scenarios

We believe this student response system to satisfy all of the current duties of a more common response system, yet it is more powerful and versatile to support a variety of more specialized scenarios.

- *Groups*
  The system allows for students to be categorized into groups. Utilizing this feature, an instructor may split students into two (or more) groups and ask separate questions to each group.

- *Collaboration (Line Graph)*
  The system is capable of promoting collaboration between students. Currently the IMP holds a consumer which is a line graph. This is meant to be paired with a question where the students are asked for a slope and a y-intercept. What the consumer displays is a target line and the current line. The students then must work together to make the current line match the target line. The consumer will average the students answers and shift the line as necessary. Here is an instance where no single student can answer a question, but rather they must work together to achieve the desired outcome.

- *Mouse Control:*
  We currently have a question type which turns a student's device into a touchpad for the display. This is done via a mouse control consumer, and a mouse question

type. This could allow a teacher to give a student control of the pointer on his/her computer for all the classroom to see. We believe this can add a new level of interaction with students. Allowing them to demonstrate something to the entire class without have to leave their seat.

Future work

We would like to expand the system to be multi-platform. Currently, the teacher and student clients run on the Android OS, and the server and IMP run on the desktop using the Java Runtime Environment (making them multi-platform). First, we would like to make desktop versions of the student and teacher clients allowing them to be run from most laptops. Additionally, we would also like to support the iOS platform. We believe these actions will provide coverage over most technologies utilized. Also, we would like to expand the display consumers to provide a larger variety of choices in how to present the student responses in a more meaningful way. We have discussed such options as utilizing bluetooth controls to control LEGO Mindstorm robots. Something such as this would promote increased collaboration between students. We have also looked into creating consumers which act as bridges to other processes which could utilize data coming in from clients. This could take form in several ways. One such manner would be “plugging into” existing online services utilizing HTTP requests. Another option is if another product wanted to be able to integrate with the SRS. They could utilize the power of a consumer to send feed information directly into their separate program. This allows the structure of the SRS to stay untouched and everything is handled via a consumer.