Developing Device-independent Applications for Active and Collaborative Learning with the SLICE Framework

Wade Fagen
Department of Computer Science
University of Illinois
Urbana, IL 61801 USA
wfagen2@illinois.edu

Sam Kamin
Department of Computer Science
University of Illinois
Urbana, IL 61801 USA
wfagen2@illinois.edu

Abstract: This paper presents a technical overview of the development of applications for the SLICE framework, a multi-platform and device-independent solution for creating applications focused on active and collaborative learning. SLICE applications are written using XML to describe the user interface and JavaScript to program any application-specific logic. By writing an application once, a SLICE application developer may deploy their application on Android devices, Windows tablet PCs, or PCs running Windows, Mac, or Linux. To demonstrate the development of SLICE applications, this paper walks the reader through programming a simple clicker SLICE application. In doing so, the reader in introduced to some of the key features of SLICE including the Model-View-Controller design pattern and the SLICE Cloud networking model. Finally, an overview some some existing SLICE applications that are used daily is presented.

Introduction

In recent years, technology has played an increasing role in classroom environments (Nakakuni, Okumura, & Fujimura, 2011) (Kennedy & Cutts, 2005). Many studies and research papers have been published using laptop computers, tablet PCs, iOS or Android-based cell phones and tablets, and electronic voting systems or EVSs (Nahrstedt, Angrave, Caccamo, & Campbell, 2010). Since 2006, professors and lecturers at The University of Illinois have utilized the SLICE framework to deploy classroom technology as part of their day-to-day class to meet their teaching objectives (Kamin & Fagen, 2012).

The SLICE framework provides several abstracts to application developers to allow for rapid development, deployment, and evaluation of software particularly related to active and collaborative learning. One of these tools, an **`InkPanel`** control, provides application developers the ability to share a "digital whiteboard" as any part of an application. For example, we have used the InkPanel as part of a PowerPoint-like lecturer application (Fagen & Kamin, 2012) that allows for both a lecturer and students to annotate the lecture slides during lecture. Additionally, the SLICE framework provides the SLICE Cloud, allowing every SLICE application to join the same virtual classroom without any classroom-specific setup.

Before 2012, SLICE has existed exclusively on Windows tablet PCs using the .NET framework and the Microsoft.Ink API. Realizing that the modern classroom is filled with cell phones, tablets, laptop computers, and other intelligent devices, we have recently extended the SLICE framework to allow for every application developed using the SLICE framework to run not only on Windows, but also on Android-based tablets and Android-based cell phones as well as Mac and Linux-based computers.

In our previous work, we have introduced the SLICE framework through many of the applications that we have developed using SLICE. In those publications, we present only a broad overview of the framework itself (Fagen & Kamin, 2012). In this paper, we will present a detailed look at how a user would develop their own SLICE applications, walk through the development of some SLICE applications, and provide an in-depth look at the infrastructural components that the SLICE framework provides. We finally will conclude with an overview of some

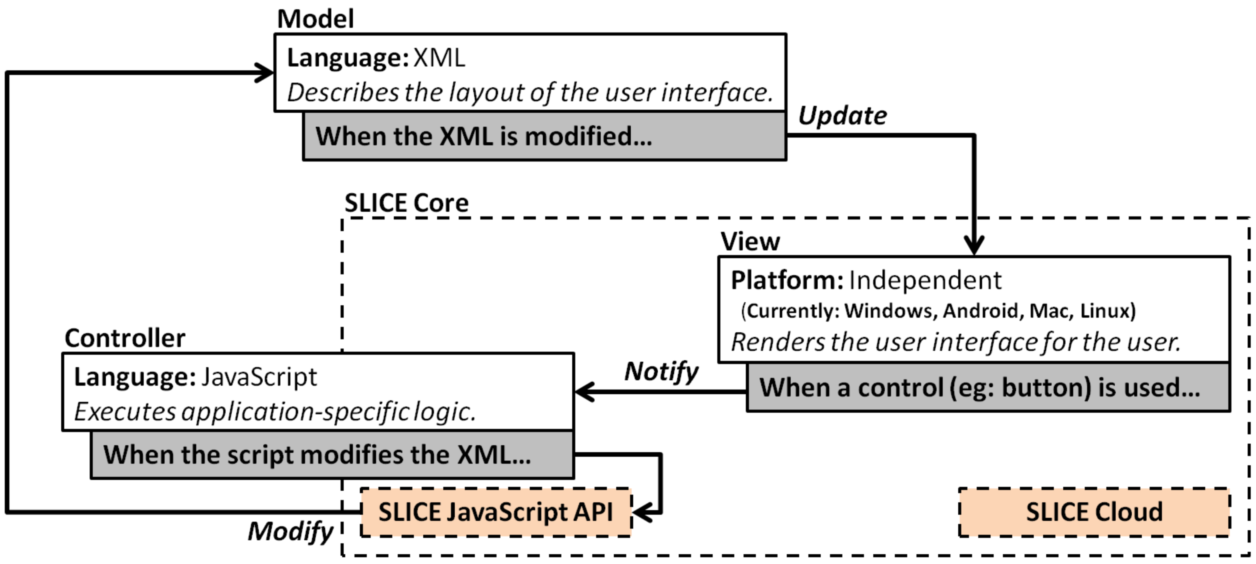

Model
Language: XML
Describes the layout of the user interface.
When the XML is modified…
Update
SLICE Core
View
Platform: Independent
(Currently: Windows, Android, Mac, Linux)
Renders the user interface for the user.
Notify
When a control (eg: button) is used…
Controller
Language: JavaScript
Executes application-specific logic.
When the script modifies the XML…
Modify
SLICE JavaScript API
SLICE Cloud

Figure 1: The Model-View Controller design pattern of SLICE framework applications.

of the existing applications that are already used on a daily basis at The University of Illinois and provide the full source code to all of these applications on the SLICE website.

SLICE Applications

The SLICE framework utilizes a common Model-View-Controller (MVC) design pattern (Krasner & Pope, 1988), allowing for a logical and programmatic separation between the graphical layout ("Model"), the application logic ("Controller"), and the user interface seen by the user ("View"). Figure 1 shows how the SLICE framework provides interactions between each component of the MVC design pattern.

In developing a SLICE application, only one file is required: an XML file describing the initial layout of the user interface. For very trivial applications, where no application logic is required, the "Controller" is unnecessary and is not required by the SLICE framework. We will discuss the XML file and how to layout applications in Section 3 of this paper.

If application logic is needed, the SLICE framework allows any number of JavaScript files to describe that logic. The JavaScript files are never ran directly, but JavaScript functions are called when certain events occur on the SLICE framework. These events may be user interface interactions (such as a button click) or external messages (such as a network message from another instance of your SLICE app). We discuss when JavaScript functions are called and how to write JavaScript for SLICE in Section 4.

The XML and JavaScript files alone make up a complete SLICE application. However, these files alone are not an executable application. To run any SLICE applications, the application must be launched by the SLICE program itself – the platform-specific executable that interprets the XML, renders the view, connects SLICE to the SLICE Cloud, and executes the user's scripts. Since only the SLICE Core is native to a platform, application developers can program a single application and their SLICE application will run on all platforms supported by SLICE. At the time of publication, SLICE supports Windows (.NET Framework 3.5+ using the Microsoft.Ink API), Android (native Android application with support for Android 2.2+ devices), Mac, and Linux.

As shown in Figure 1, one of the primary roles of the SLICE Core is to render the user interface that is described by the XML and to execute the JavaScript scripts when actions are preformed to the user interface. However, the SLICE Core also provides for access to various features of the underlying operating system including networking, loading and saving files, and various global SLICE settings. Instead of requiring each user to handle their own networking server and client, the SLICE framework provides a cloud service called SLICE Cloud for applications that need to communicate with other instances of SLICE in a given classroom. We will discuss the full features of the SLICE Cloud and how to use the service in Section 4 when we develop some sample applications.

User Interface Attributes ***(Applies to all user interface controls)***
`X`, **`Y`**, **`Width`**, **`Height`**: Integer-valued attributes related to the display of a user interface component
`BackColor`, **`ForeColor`**: Standard 16-bit or 32-bit color syntax (#ff0000) or a system color ("Black")
Specific Component Attributes
`Text`, **`Font`**, **`FontSize`**: Specifies the text and its appearance. *(***`Button`***,* **`TextBox`***, and* **`Label`***)*
`TextAlign`, **`Center`**: Allows alignment of text. *(Applies to* **`Button`** *and* **`Label`***)*
`Image`: Specifies the file that contains the image to be displayed. *(Applies to* **`Button`** *and* **`Label`***)*
Scriptable Attribute Events
`OnMouseEnter`, **`OnMouseLeave`**: JavaScript function to call when the mouse movies over a component.
`OnClick`, **`OnRightClick`**, **`OnMiddleClick`**, **`OnDoubleClick`**: Function to call on a mouse action.
`OnKeyPress`: JavaScript function to call on a keyboard action.
`OnTextChanged`: JavaScript function to call when text changes in a **`TextBox`**.
Ink-Specific Attributes and Scriptable Attribute Events
`PenColor`, **`PenWidth`**, **`Transparency`**: Look-and-feel of an ink stroke on an **`InkPanel`**.
`InkStrokeHandler`: JavaScript function to call when an ink stroke is added to an **`InkPanel`**.

Table 1: Partial list of attributes used in the XML to describe and interact with the user interface.

Developing the SLICE User Interface Using XML

As part of every SLICE application, a user must describe the initial layout of the user interface for their application with an XML document. Using XML to describe the user interface has been used in SLICE since the initial version of SLICE in 2006, and the design choice of describing a user interface with XML has also been chosen for frameworks including Microsoft .NET's Windows Presentation Foundation (WPF) (Microsoft, 2010), Adobe AIR (Adobe Systems Incorporated, 2012), and Mozilla's XULRunner (Mozilla Foundation, 2012).

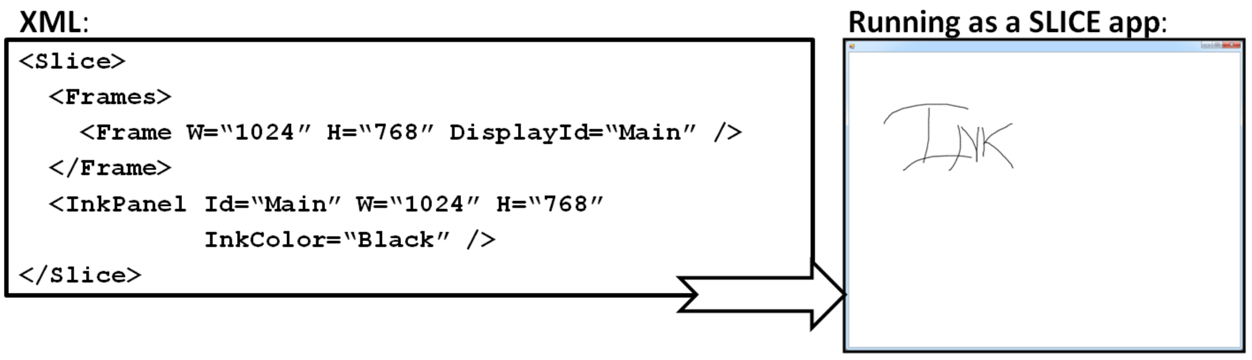
Much like other languages and frameworks that use XML to describe the user interface, the root element of the XML document must be specific to the framework itself. For SLICE, the root of every SLICE application's XML file must be a **`<Slice>`** element. Optionally, the **`<Slice>`** element may contain a variety of attributes to specify certain attributes about the application as a whole. The three attributes specific only to the **`<Slice>`** element are:

- **`JSDefs`**: A list of JavaScript files that contains the functions to be ran when specific actions occur.
- **`Init`**: A list of JavaScript functions to be executed as soon as SLICE application has been fully initialized. That is, once the XML has completely read and shown to the user.
- **`OnExit`**: A list of JavaScript functions to be executed after the user interface has been hidden and the SLICE framework is exiting.

The **`<Slice>`** element's children nodes will describe the application-specific user interface. The only requirement is that one child element must contain data about the application frames, the highest order user interface component. A frame is synonymous to an application window in Windows, Mac, or Linux, or a full-screen application on an Android-based device.

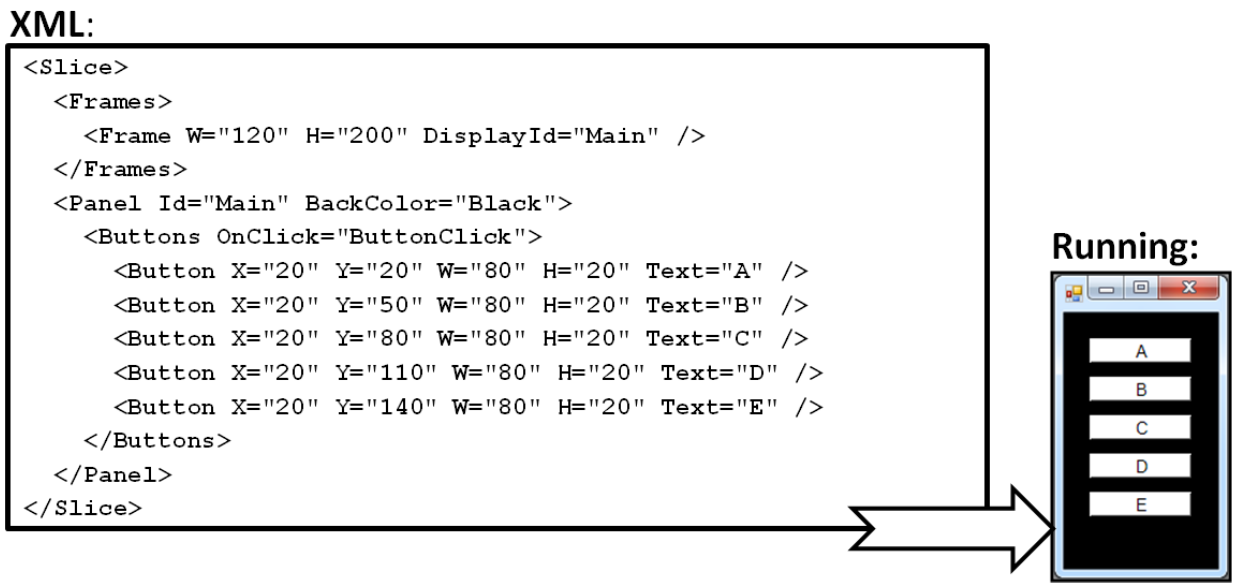
Information about all of the frames that are used in by a SLICE application must be contained in a **`<Frames>`** element. Each individual frame is a **`<Frame>`** element as a child of **`<Frames>`**. Most SLICE applications use only a single frame, which requires only a single **`<Frame>`** element. Since a **`<Frame>`** is a user interface element, the large number of attribute that can be applied to the **`<Frame>`** and other user interface nodes are described in Table 1.

Since the frame is the highest order user interface control in SLICE, every other control must be contained in a frame. To allow maximum flexibility, the user interface components that are displayed inside of a frame cannot be a child of a frame. Instead, the top-level element to include in the frame needs a special attribute **`Id`** that matches



```
<Slice>
  <Frames>
    <Frame W=“1024” H=“768” DisplayId=“Main” />
  </Frame>
  <InkPanel Id=“Main” W=“1024” H=“768”
            InkColor=“Black” />
</Slice>
```

Figure 2: The complete source code to a SLICE application, alongside the application running on Windows 7 with several strokes drawn on the **`InkPanel`**.



```
<Slice>
  <Frames>
    <Frame W="120" H="200" DisplayId="Main" />
  </Frames>
  <Panel Id="Main" BackColor="Black">
    <Buttons OnClick="ButtonClick">
      <Button X="20" Y="20" W="80" H="20" Text="A" />
      <Button X="20" Y="50" W="80" H="20" Text="B" />
      <Button X="20" Y="80" W="80" H="20" Text="C" />
      <Button X="20" Y="110" W="80" H="20" Text="D" />
      <Button X="20" Y="140" W="80" H="20" Text="E" />
    </Buttons>
  </Panel>
</Slice>
```

Figure 3: A simple clicker application using the SLICE framework.

the **`DisplayId`** of the **`<Frame>`** element. Figure 2 shows a simple SLICE application that contains an **`InkPanel`** inside of a single frame.

In the sample application that was just introduced, the user interface control used was a control that is central to many of the existing SLICE applications that are used in classrooms daily: an **`InkPanel`**. The **`InkPanel`** allows for users to write on that user interface component with a tablet pen, a mouse, or their finger (on touch-based devices). In our previous work, we have written about the success of a “Lecturer Application” and a “Code Review Application” that both heavily use tablet based Windows machines and the **`InkPanel`** control. In the lecturer application, professors of large classes use a SLICE app to annotate their slide set during lecture, interactively work with students on solving in-class problems, and may display the work of other students in the class to the classroom as a whole. In the code review application, students use a SLICE app that displays a “digital whiteboard” of the student’s code.

Additional user interface controls supported by the SLICE framework include controls more familiar to many users: **`Label`**, **`TextBox`**, **`Button`**, and a **`Panel`**. For example, using a series of buttons, we could create a simple classroom-clicker application similar to the i>Clicker, eInstruction Clickers, or other Electronic Voting Systems (EVSs). Figure 3 shows a SLICE app running a simple clicker application. We will develop this application further in the next section.

Method	Description	Example
`this[String name]`	Returns the value of the XML attribute with the name **`name`** as a String.	`var value = Node["Name"];`
`GetChildren()`	Returns a JavaScript array of the **`TreeNodes`** that are the children of the **`TreeNode`**.	`var children = Node.GetChildren();`
`GetParent()`	Returns the TreeNode that is the parent of the **`TreeNode`** or null if the **`TreeNode`** does not have a parent.	`var parent = Node.GetParent();`
`AppendChild (TreeNode node)`	Appends the node **`node`** as a child to the **`TreeNode`**.	`var child = ...; Node.AppendChild(child);`
`static FindNodeById (String id)`	Searches the entire XML tree for an element with the ID attribute equal to the specified **`id`**.	`TreeNode.FindNodeById ("Display");`
Other Methods		
`GetName(), NumChildren(), GetSiblings(), Position(), CreateChild(), RemoveChild(), Remove(), RemoveChidren(), InsertAt(), InsertAfter(), InsertBefore(), PreviousChild(), NextChild(), Clone(), HasChild(), GetChild(), FindNodeByAttribute(), FindChildByAttribute(),GetX(), GetY(), FindAllNodesByName(), HasAttribute(),GetW(), GetH(), ToString(), ToXML(), GetZoomLevel(), ...`		

Table 2: A subset of the available methods in the **`TreeNode`** class. Complete documentation is provided on the SLICE website.

Developing Application Logic Using JavaScript

In this paper, we have discussed the XML used to describe the user interface and the SLICE Core that renders that user interface. Up to now, users can draw ink strokes on an **`InkPanel`** or press a button, but no actions are preformed as a result of the user's interaction. Table 1, introduced earlier, presents some of the available actions (eg: **`OnClick`**) that we will examine in this section.

When developing application logic, you must inform SLICE which functions that you wish to perform when a certain event takes place. Much like setting a component's **`Text`** attribute, scriptable attributes are an XML attribute in the XML document. However, scriptable attributes accepts a list of functions that should be invoked when the action is preformed. This list may contain a single function, where **`OnClick="ButtonClicked"`** will run the JavaScript function **`ButtonClicked()`**, or a list of functions separated by a pipe character. The XML attribute **`OnClick="A | B"`** would run the JavaScript function **`A()`** followed immediately by the JavaScript function **`B()`**.

The JavaScript written by a developer of a SLICE application is standard JavaScript with the addition of a set of functions to interact with SLICE (the "SLICE JavaScript API") and one special JavaScript class to manipulate the XML called a **`TreeNode`**.

The `TreeNode` Class

As part of the SLICE JavaScript API, a new class is available for your use in JavaScript called a **`TreeNode`**. This class is the representation of any and all of the XML elements in the XML document, from the top-level **`<Slice>`** element all the way down to an individual **`<Button>`**. An app developer may create new **`TreeNode`**s, append them to the XML in any location, and navigate through the XML via **`TreeNode`** methods. Four of the primary methods used in the **`TreeNode`** class are listed in Table 2. A full set of methods are available as part of the full documentation on the SLICE website.

When any JavaScript function is called by SLICE, a global variable called **`Sender`** is always set to the **`TreeNode`** of the component that invoked the call to the JavaScript function. This is particularly important since many attributes in XML are inherited from their parent. For example, consider the following source code snippet for the application shown in Figure 3:

```
<Buttons OnClick="ButtonClick">
  <Button X="20" Y="20" W="80" H="20" Text="A" />
  ...
</Buttons>
```

***Figure 4**: A snippet of the XML source code for the SLICE application shown in Figure 3.*

When a user clicks the **`Button`** that contains the text "A", the JavaScript global variable **`Source`** is set to the **`TreeNode`** that represents the **`Button`** shown in the source. After the **`Source`** is set, the **`ButtonClick()`** function is called.

Modifying the XML via JavaScript

When developing a clicker application, one of the first things a user may expect is the ability to view the selection they have selected as a confirmation of their vote. To begin to offer this, we could simply display the button that they click at the bottom of the application. To do this, we would need to add only two things: a **`Label`** to display their response and a function to write their answer to the **`Label`**. We can add the Label to the appropriate place of the user interface by the following XML:

```
<Label X="20" Y="170" H="20" W="80" Id="Display"
       BackColor="Black" ForeColor="White" Center="True" />
```

Figure 5: Additional XML added to the "Clicker Application" to show the user their response.

The second addition is the application logic in JavaScript to display their selected response in the Label we just created. To do this, we define the **`ButtonClick()`** function that was defined as part of Figure 5:

```
function ButtonClick()
{
  var display = TreeNode.FindNodeById("Display");
  display["Text"] = Source["Text"];
}
```

Figure 6: The JavaScript **`ButtonClick()`** function called in response of a user clicking a button on the clicker application.

When a user clicks on the Button labeled "A", the result of the JavaScript is that the attribute **`Text`** of the **`Label`** is modified to "A". As soon as this attribute is changed, the user interface is updated; the user sees that they choose "A", as shown in student clicker application screenshots shown in Figure 9.

Networking in SLICE: The SLICE Cloud

With the focus of SLICE development on classroom applications, one of the key features of the SLICE framework is the ability to seamlessly connect and interact with other machines in the classroom. The SLICE Cloud provides the ability for any SLICE application to join the classroom that the user is currently in. To do this, a SLICE application runs one command in JavaScript:

```
SliceCloud.Connect(classroom)
```

Immediately, and without any other code, the SLICE application is connected with all of the other instances of their SLICE application in a given classroom. The **`Connect()`** command may optionally connect to the SLICE Cloud with parameters to identify themselves as an individual or as part of a group.

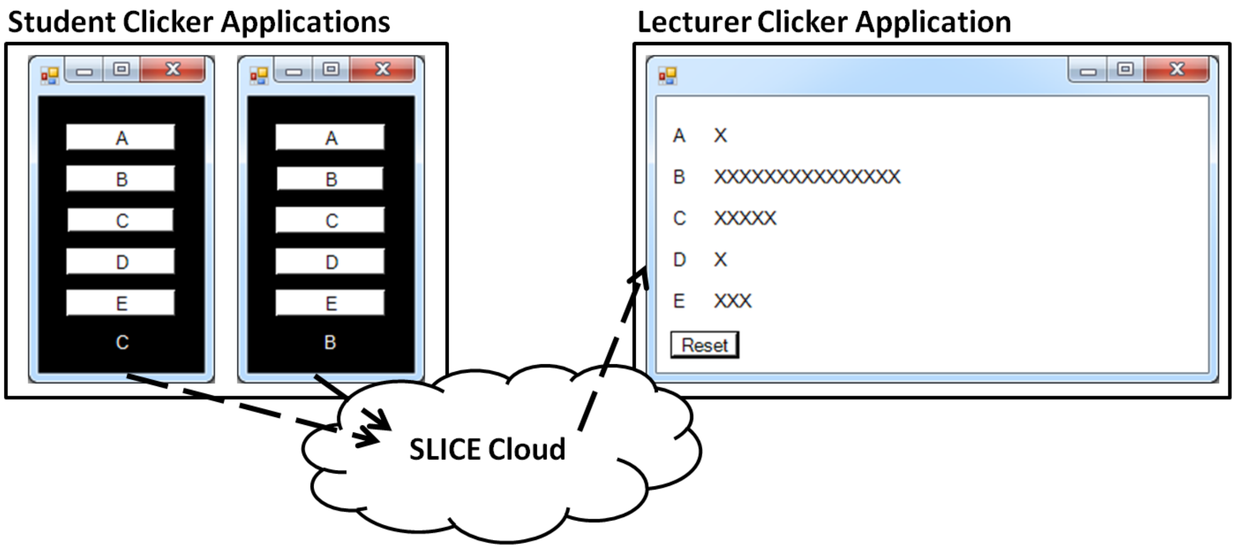


Figure 9: A sample clicker application with a student clicker application and a lecturer answer viewer developed with less than 80 total combined lines of code.

As we introduced in Section 4.1, the only class that is a part of the SLICE JavaScript API is the **`TreeNode`** class. Just like in our XML, every message sent as part of SLICE must be a **`TreeNode`**. Continuing with the "Clicker Application", a complete clicker application would also need to inform the lecturer of the choice that each student has made. We can complete this application by doing two things:

- Creating a **`TreeNode`** class with our clicker answer in it.
- Sending that answer to the Lecturer.

We will accomplish both of these tasks in three lines of JavaScript, using a couple of functions described in the full documentation available on the SLICE website:

```
 var message = TreeNode.MakeTreeNode("Answer");
message["Answer"] = Source["Text"];
SliceCloud.SendByAttributes(answer, "Lecturer");
```

Figure 7: Preparing and sending a message on the SLICE Cloud.

At this point, we have a complete student application for a simple clicker application. However, no application is receiving any of the messages directed to the Lecturer. A separate application, or the same application initialized with different parameters, would need to be created to receive messages sent to the Lecturer.

The lecturer application needs to do only two things to receive the messages directed to it by the students: connect to the SLICE Cloud with the "Lecturer" attribute and register a JavaScript function to be called when a message is received. The code to complete both of these tasks, along with processing the message, is shown in Figure 8 below:

```
function OnStart() {
  SliceCloud.Connect(classroom, "Lecturer");
  SliceCloud.AddMessageRecievedCallback("OnMessage");
}

function OnMessage(message, senderId) {
  var answerBox = TreeNode.FindNodeById   (message["Answer"]);
  answerBox["Text"] = answerBox["Text"] + "X";
}
```

Figure 8: JavaScript code connecting to the SLICE Cloud with the attribute "Lecturer" and processing all incoming network messages in the OnMessage() JavaScript function.

In all, the complete "Clicker Application" that was developed for the reader consists of a student and lecturer application. The student application may be ran any number of supported devices and will communicate the students' answers with the lecturer, which graphically displays an aggregation of the current responses to the question. The entire application, with all application logic including networking, involved only 34 lines of XML between the two applications and 37 lines of JavaScript. The entire source code for both applications is available on the SLICE website and a screenshot of both apps running side-by-side is shown in Figure 9.

Conclusion

In this paper, we have presented a technical overview of the development of SLICE applications. In doing so, we have developed a complete and functional clicker application that allows for students to submit responses to a lecturer interface that allows the lecturer to view the aggregated answers of all the students in a classroom. While the clicker application is not novel, the simplicity of the XML, JavaScript, and networking model allows for more complicated applications to be built with ease and deployed across a heterogeneous set of platforms including Android-based tablets and smart-phones, Windows-based tablets, and PCs running any modern operating system (Windows, Linux, or Mac).

Some of the key features of SLICE, including the ability to accept touch- and pen-based input and the SLICE Cloud that allows SLICE apps to connect to a virtual instance of a given classroom, makes SLICE a particularly effective platform to develop applications designed for classroom settings with an emphasis on active and/or collaborative learning goals. In our previous publications, we have explored the usefulness of several applications developed with the SLICE framework including a PowerPoint-like "Lecturer Application" and a group "Code Review Application" (Fagen & Kamin, 2012). These applications, along with the examples shown in this paper, are all available for download on the SLICE website (both as part of the SLICE application and in source code form).

As we work on more applications ourselves and view their effectiveness in the classroom, we will continue to report on the most successful uses of in-class technologies and reflect on cases where the technology did not meet the stated education goals. We encourage the reader to try out one of the many existing SLICE applications in their learning environments or to develop their own applications using the SLICE framework, all of which is available on the SLICE website at **`http://slice.cs.illinois.edu/`**.

References

Adobe Systems Incorporated. (2012, 4). *Adobe AIR Developer Center*. Retrieved from Adobe Developer Connection: http://www.adobe.com/devnet/air.html

Fagen, W., & Kamin, S. (2012). A Cross-Platform Framework for Educational Application Development on Phones, Tablets, and Tablet PCs. *Proceedings of the 2012 International Conference on Frontiers in Education: Computer Science & Computer Engineering (FECS 2012).* Las Vegas, Nevada.

Kamin, S., & Fagen, W. (2012). Supporting active learning in large classrooms using pen-enabled computers. *Proceedings of the 2012 International Conference on Frontiers in Education: Computer Science & Computer Engineering (FECS 2012).* Las Vegas, Nevada. Retrieved from slice.cs.illinois.edu; submitted for publication

Kennedy, G. E., & Cutts, Q. I. (2005). Construction of a Collaborative Learning Environment through Sharing of a Single Desktop Screen. *Journal of Computer Assisted Learning, 21*(4), 260-268.

Krasner, G. E., & Pope, S. T. (1988). A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. *System.*

Microsoft. (2010, 12). *Windows Presentation Foundation*. Retrieved from Microsoft Developer Network (MSDN): http://msdn.microsoft.com/en-us/library/ms754130.aspx

Mozilla Foundation. (2012, 4). *XULRunner*. Retrieved from Mozilla Developer Network: https://developer.mozilla.org/en/XULRunner

Nahrstedt, K., Angrave, L., Caccamo, M., & Campbell, R. (2010). *Mobile Learning Communities – Are We There Yet?* Information Trust Institute, University of Illinois at Urbana-Champaign.

Nakakuni, M., Okumura, M., & Fujimura, S. (2011). Construction of a Collaborative Learning Environment through Sharing of a Single Desktop Screen. *International Conference on Frontiers in Education: Computer Science and Computer Engineering.*