CyRIS (May 14-04)
Final Report

Team Members:

Nathan Clague
Michael Krantz
Zachary Patzwald
Maxwell Philips
Micah Stevenson
David Vriezen

Advisor:

Dr. Manimaran Govindarasu

Client:

Brock Ascher

Team Website:

http://seniord.ece.iastate.edu/may1404/

I. Table of Contents

II. Introduction

When our team began the CyRIS project, our options were wide open. The Iowa State University Department of Electrical and Computer Engineering had just installed a massive 16 foot wide by 6.25 foot tall touch screen wall. This wall was composed of twelve individual 55-inch LED touch screen monitors that are connected together to form one huge screen. At that time, the video wall could only display basic slideshow presentations using a program called Intuiface. Since then, presentations on the video wall have improved through the addition of more content and better navigation between presentation areas. One notable breakthrough came when the video card in the master computer was upgraded to allow the system to display lag-free full screen video at resolution high enough for a crisp image. However, the video wall is still limited to displaying relatively static content without the opportunity for much user interaction. Our main goal in this project was to improve the video wall by adding more interactive content including applications and games.

To realize this goal, we created the following design objectives:

1. Integrate our content with the existing Intuiface presentation software
2. Implement several new interactive applications
    a. Twitter, Facebook, and RSS feeds from the Electrical and Computer Engineering Department
    b. Campus map with real-time CyRide bus system information
    c. Staff directory
    d. Campus webcam viewer
3. Create or utilize a platform that can easily be used in future senior design projects
4. Make the video wall more interesting to passers-by to encourage them to interact with it

Our strategy for completing the project included several steps. The first step was to familiarize ourselves with the existing Intuiface presentation software. The next step was to determine the best way to create applications that would integrate well with the existing system. We were then able to determine which applications were feasible and determine requirements for each of these applications.

During the second half of the project, our team focused on implementation, testing, and deployment of our project. We continued implementation of our applications and refined the integration of our chosen platform with the existing environment. Our team also tested our product throughout the project to ensure that it would work well after deployment of our system. We then completed documentation for our system. Finally, our team has deployed the project for our client.

III. Revised Project Design

Modular Design Descriptions

Module	Design Description
CyRIS Content Manager	The CyRIS content manager provides an interface for launching sub applications. This behavior is realized via a scrollable horizontal array of icons. Selecting an icon spawns an instance of the corresponding sub application. Multiple instances of each sub application can be spawned a once. The content manager controls all sub application instances, allowing them to be expanded to full screen mode if desired. Instances of sub applications can be dragged, scaled, and rotated within the content manager. Up to ten simultaneous touch inputs can be detected by the content manager and propagated to the sub applications as control events. The content manager will be launched from an Intuiface presentation and will return to the Intuiface presentation when it is exited. At the client's request, an alternative execution path was developed in which sub applications are immediately launchable from the Intuiface presentation. When launched this way, sub applications will automatically enter full screen mode and only one sub application will be launchable simultaneously. Each application launched in this manner will return control directly to the Intuiface presentation when the user exits the application.
Feeds Application	The feeds application displays information from social media affiliated with Iowa State University's Electrical and Computer Engineering Department. Individual tweets taken from a Twitter feed crawl across the bottom of the screen. A feed combining Facebook and RSS data is scrollable on the right side of the screen.
Maps Application	The maps application allows users to interact with a map of Ames. The map can be zoomed and panned and its view can be switched between computer-rendered map views (Microsoft Road, Microsoft Hybrid, and Open Street Maps) and aerial views (Microsoft Aerial). Additionally, the maps application leverages the NextBus API to provide CyRide route and stop information to users. Bus locations are drawn onto the map in real time using GPS coordinates so users can see which buses are currently in operation and where they are.
Staff Directory Application	The staff directory application provides an offline version of Iowa State University's Electrical and Computer Engineering Department Staff Directory to users. Images and text displayed in the staff directory application are scraped from the online directory via JSoup. The staff directory application provides information about staff members including their name, title, address, phone number, email address, and staff photo.

	Results from the staff directory application are searchable and displayed in a grid format. The user can expand any individual result to view information for that staff member.
Webcam Application	The webcam application displays a feed from various publicly available camera feeds located across campus. Users are able to choose which feed to view when multiple feeds are available.

Content Manager Requirements

Functional Requirements	Non-Functional Requirements
1. The content manager must be able to generate multiple sub application windows and return control to the Intuiface presentation 2. The content manager must allow for multiple sub applications simultaneously accessing input and output	1. The content manager shall match the look and feel of the current Intuiface presentation

Feeds Application Requirements

Functional Requirements	Non-Functional Requirements
1. The feeds application must display information from social media approved by the ECpE department 2. The feeds application must scroll text across the screen while users are interacting with other parts of the system 3. The feeds application must display photos embedded in social media updates without allowing users to continue browsing the internet	1. The feeds application shall scroll text across the screen at a readable speed 2. The feeds application shall allow for the addition or removal of specific social media accounts

Maps Application Requirements

Functional Requirements	Non-Functional Requirements
1. The maps application must display a pannable and zoomable map	1. The maps application shall be aesthetically pleasing and bus icons displayed on the map shall be the

2. The maps application must show all currently-operating CyRide routes and all buses on each route 3. The maps application must not exceed data transfer limits imposed by the NextBus API	same color as the route they represent 2. The maps application shall be resistant to service disruptions

Staff Directory Application Requirements

Functional Requirements	Non-Functional Requirements
1. The staff directory application must be populated with information taken from the ECpE department's online staff directory 2. The staff directory application must display expandable entries that, once expanded, contain detailed information about each staff member 3. The staff directory application must be searchable by text input	1. The staff directory application shall display staff names, titles, addresses, phone numbers, email addresses, and photos for each staff member

Webcam Application Requirements

Functional Requirements	Non-Functional Requirements
1. The webcam application must stream video from webcam feeds on campus	1. The webcam application shall be resistant to service disruptions

Standards

Extensible Markup Language (XML) 1.0

http://www.w3.org/TR/REC-xml/

The CyRIS project didn't directly create data in the XML format, but we did interact with a fair amount of data stored in the XML format. All of the data provided by the NextBus XML Feed was provided in XML format.

NextBus XML Feed 1.22

https://www.nextbus.com/xmlFeedDocs/NextBusXMLFeed.pdf

The NextBus XML Feed was instrumental for accessing and using information about the CyRide bus system. Among other things, this specification allowed us to access detailed information about CyRide bus routes and continually-updated CyRide bus location information.

XML Schema Part 0: Primer Second Edition

http://www.w3.org/TR/xmlschema-0/

The XML Schema Part 0 was the primary source used to create XSD files used to validate NextBus XML data. This document provides a summary of the more detailed XML Schema Part 1: Structures (http://www.w3.org/TR/xmlschema-1/) and XML Schema Part 2: Datatypes (http://www.w3.org/TR/xmlschema-2/) documents. Information needed that was not included in the XML Schema Part 0: Primer was found through additional research.

MotionJPEG Video Format

http://www.jpeg.org/public/fcd15444-3.pdf

Displaying video from the cameras on campus required accessing streams in the MotionJPEG format.

Oauth 2.0

http://tools.ietf.org/html/rfc6749

The Facebook and Twitter APIs require OAuth authentication when using their web services.

RSS 2.0

http://validator.w3.org/feed/docs/rss2.html

The feeds application interacts directly with data in the RSS 2.0 format. This format is backwards compatible with the RSS 1.0 specification.

Atom

http://tools.ietf.org/html/rfc5023

The feeds application also interacts with data in the Atom format, an RSS alternative.

Twitter REST API v1.1

https://dev.twitter.com/docs/api/1.1

The feeds application consumes data from this web service.

Facebook Graph API

https://developers.facebook.com/docs/graph-api/reference/

The feeds application consumes data from this web service.

JSON

http://www.ietf.org/rfc/rfc4627.txt

The data returned from the Facebook and Twitter web services is in this format.

WHATWG HTML5

http://www.whatwg.org/specs/web-apps/current-work/multipage/

Data parsed into the Staff Directory application is present in this HTML format.

IV. Implementation Details

CyRIS Content Manager

The Content Management system was designed with two design principles in mind, the observer design pattern and the decorator design pattern. These patterns were inherited from the Multi Touch for Java (MT4J) framework we built our content manager with. Consistency with the legacy code allowed for simplification of tasks such as launching programs and making action listeners on the objects within the content manager.

The content management scene itself is the observer class and all of the launched sub applications within it are observees. This design choice was made when determining the best way to handle launching more than one application at a time within the content manager. The entire goal of the content manager was to allow for multiple children as it would demonstrate the capabilities of the touch screen wall by allowing a multiple users to interact with the wall simultaneously. When an application is launched from within the manager it generates a child process. This parent child relationship is best demonstrated when a child application in windowed mode is listening to a scaling gesture event. The scaling gesture is processed in local space for child window as the child window is the component actually receiving the touch input. This gesture data is then transformed to the global space for the content manager through a transformation matrix in the parent process. Once this transformation is complete, the parent process handles the redrawing of its child window at the newly scaled size.

The decorator design pattern is shown in the content manager by the addition of our feeds application as an MT4J scene to the background of the main window. The addition of the feeds application in this manner is similar to adding a decorative component to any other graphical interface.

Feeds Application

The Feeds app is implemented using data provided by Facebook, Twitter, and RSS. The data is retrieved using Facebook4j and Twitter4j, libraries that allow for the manipulation of data from their respective service in Java. These frameworks abstract interactions with Oauth 2.0 and web services, allowing us to focus on developing to meet the client's needs.

The Feeds app uses model-view-controller architecture. The model is the data retrieved from web services. The view is the object appearing on the screen in a crawling list for Twitter information or a scrollable list for Facebook and RSS information. The controller works behind the scenes putting model data into the view objects, as well as responding to touch gestures and transforming the view accordingly.

Maps Application

Much of the map drawing functionality already existed in the MT4J framework. Our team set the default starting point of the map to Coover Hall (the building on campus housing the video wall) and restricted from zooming out past a certain point so that the primary focus of the map would be Ames and the Iowa State University campus.

Integration with the CyRide bus system is realized using data provided by NextBus, a company that provides bus tracking services for bus systems across the country. All data provided by NextBus is retrieved from RESTful web endpoints and delivered via XML files. There are several different endpoints available from NextBus depending on which type of information is being requested but their XML structure is nearly identical. NextBus data is processed in four distinct layers: an XML schema definition layer, a data request layer, a data storage layer, and a data update handler layer.

The first layer is the XML schema definition layer. This component is used to verify that the XML file returned is structured as expected. We can specify which elements and attributes we are expecting from each endpoint and specify default values for any missing information through an XML schema definition. The information required to create our XML schema definitions was taken from the NextBus Public XML Feed documentation.

The second layer is the data request layer. This layer is responsible for requesting data from NextBus and saving it in the data storage layer. Each component in this layer is implemented as a thread that requests new data periodically (typically every 10-12 seconds) in the background. This allows recently-updated data to be available continuously without stopping program execution to update data. This layer is also responsible for providing the most recently updated information to each active instance of the maps application. By allowing multiple application instances to share the same data request layer through a singleton design pattern, we can eliminate on redundant data requests and ensure our data consumption remains below the NextBus API data limits.

The third layer is the data storage layer. After data is read from an XML document, it is stored in an appropriate class or hierarchy of classes. This allows the data to be accessed in a Java-native format and eliminates the need to traverse the XML document directly when the data is being used. Storing data in this way also allows manipulation using Java's primitive data types, rather than manipulating a string representing each value. This is particularly helpful when dealing with data such as GPS coordinates and bus route colors from NextBus.

The fourth layer is the data update handler layer. After the data request layer requests and receives data from NextBus, the data request layer notifies all of its listeners of the updated data. These listeners are primarily components from this data update handler layer. There can be many listeners for each data request component. When the data update layer receives a

notification that there is new data, it completes some action based on the new data (typically removing old information from the map and redrawing updated information.)

Staff Directory Application

The staff directory application was designed to restrict access to an internet browser on the video wall. The Electrical and Computer Engineering department already has a staff directory hosted on its website and our client wanted to see the information from that directory on the video wall without allowing users to visit non-sanctioned websites. Our staff directory application is a recreation of the data contained in the online staff directory. This data is retrieved through a web scraper utilizing JSoup, an open source HTML parsing library designed for extracting and manipulating data. The application extracts directory entries for each staff member each time it is launched, ensuring local data is as current as the information online.

The second layer of this application is the user interface created within the MT4J framework. The data retrieved by the web scraper is stored locally and then organized and displayed in the correct format on screen. Each staff member's photo and name is shown as a clickable menu item that leads to a popover with detailed information from their profile. This menu can be searched with text input to filter results for only staff members with names that match the input.

Webcam Application

The webcam viewer uses two layers to access and display video feeds. The first layer, the feed layer, accesses the MJPEG video feed present as an already hosted web endpoint from various webcams across campus. It uses the IPCapture library to decode each frame from the feed to a JPEG image.

The second layer is the application controller. It handles the display of the video feeds as well as user input. Whenever the feed layer is updated with new information and a new JPEG is ready to be drawn in the controller layer, the controller leverages the MT4J framework to display the new image to the screen. This layer also handles input events and can switch between video feeds when the user chooses to do so.

V. Testing

User-Level Testing

Using a test environment in the senior design lab, our team has tested continuously throughout the development process during the second semester of our project. This testing has been focused on ensuring correct application behavior during expected use cases and edge cases. Another major component of our user-level testing is regression testing. As new features were added to each of our applications our team performed user-level testing to ensure the system still behaved as expected and all features were backwards compatible.

Performance Testing

Another important aspect of our system is displaying information parsed from web endpoints. When our system is deployed it should be able to handle this in real time. To ensure this behavior will occur, our team used performance testing to measure the response of various applications in our system.

Specific performance related threats were posed by our maps application, our staff directory application, and our webcam application. These applications all rely on displaying information parsed from web endpoints so their performance could suffer at any time if they are not designed in a robust manner that accounts for service disruptions and error handling.

Performance testing results for those applications are as follows.

Test Case	**Tester's Name**	**Results**	**Description**
Maps application performance test 1	Nathan Clague	Pass, no fatal errors	The maps application was left running for one hour. The console log was monitored for errors resulting from service interruptions or incorrect program states.
Maps application performance test 2	Nathan Clague	Pass, no fatal errors	The maps application was left running for one hour. The console log was monitored for errors resulting from service interruptions or incorrect program states.
Staff directory performance test 1	Michael Krantz	Pass, no fatal errors	The staff directory application was tested with multiple input sequences over various selectable components. The console log was monitored for

			errors, and the UI was checked for proper output.
Webcam performance test 1	David Vriezen	Pass, no fatal errors	The webcam application was run for two hours while observed for non-network-induced lag and crashes in the video.

Security Testing

Any system deployed for public consumption must undergo security testing to keep flaws to an absolute minimum. Due to the fact that our system will be available for public interaction in a public area, our team had to limit the capabilities of the user and withhold certain privileges to maintain a secure system. We wanted to ensure that the inputs a user can deliver remain limited to valid inputs for our system and are not malicious or exploitative. Many exploitative inputs are removed from the possible input space because users will not have access to a keyboard. However, we still want to test for possible edge cases.

Appendix A

Operation Manual

1. Setup
    1.1 Requirements
        - Internet access or compiled project already downloaded
        - Windows 7 touch environment for full functionality, Windows 7 environment for mouse control
        - 32-bit Eclipse
    1.2 Basic Flow: execute content manager
        - Download the compiled project from the team website
        - Extract downloaded folder
        - Open a terminal window and navigate to the extracted folder. Once there, navigate to jre6x32/bin
        - Execute the command “java -jar ContentManager.jar”
    1.3 Alternate Flow: execute standalone applications
        - Instead execute the command “java -jar APPLICATION-NAME-HERE.jar”
    1.4 Alternate Flow: execute from source code
        - Download the project source code from the team website
        - In Eclipse, import the source code as a new project
        - In package mt4j-desktop/src/cyris/contentManager run LaunchContentManager.java as a Java Application
    1.5 Alternate Flow: update jar from source code
        - After importing the project into Eclipse, export the project as a runnable .jar using LaunchContentManager.java as the main method
        - Name the exported jar file ContentManager.jar
        - Copy ContentManger.jar into C:/Program Files/Java/JAVA-VERSION/bin
        - Open C:/Program Files/Java/JAVA-VERSION/bin in a terminal window
        - Execute the command “java -jar ContentManger.jar”
    1.6 Alternate Flow using Intuiface Presentation
        - Download the compiled project from the team website
        - Extract downloaded folder
        - Open the Intuiface presentation
        - Play the Intuiface presentation
        - Click the button to launch our project
2. Demo
    1.1 Requirements
        - Windows 7 touch environment
        - Setup section completed
    1.2 Basic Flow
        - After launching our project a black loading screen will appear

- The content manager will be executed
- Individual sub applications can be interacted with

1.3 Application Showcase

1.3.1 Content Manager

- Drag input events in the background will generate particle effects mimicking the drag event
- The horizontal array of application icons can be scrolled back and forth via drag input events
- The exit button can be tapped and held to return control to whichever method was used to execute our project
- Tapping an application icon will launch the corresponding application in windowed mode
- Instances of multiple applications can be launched at one time
- Multiple instances of a single application can be launched at one time
- Application instances can be dragged, scaled, rotated, sent to full screen, and returned to windowed mode
- Application instances persist until they are closed by the user

1.3.2 Feeds Application

- The Facebook and RSS feed can be scrolled through on the right side of the screen
- The Twitter feed automatically scrolls across the bottom of the page

1.3.3 Maps Application

- The maps application allows users drag and scale a map of Ames
- CyRide bus locations are drawn onto the map in real time using route colors and bus headings so the buses appear to ‘drive’ around the map

1.3.4 Staff Directory Application

- The staff directory is displayed in a text-searchable two-dimensional array of clickable staff profiles
- Detailed information for each staff member is available as an expanded popover on click

1.3.5 Webcam Application - “Eye on ISU”

- The feeds from the primary webcam will automatically be displayed, if available
- The displayed feed can be changed using the buttons along the right side of the screen

1.3.6 Touch Tails Application

- Not designed by our team, this application allows users to drag a short pattern on the screen and see it transformed into a snake-like object that slithers about

1.3.5 Air Hockey Application

- Not designed by our team, this application allows users to play air hockey in two player mode

1.3.6 Physics Playground Application

- Not designed by our team, this application allows users to slide 2d objects across the screen that are affected by gravity and collisions with other 2d objects

Current versions of the compiled project and the source code are downloadable from our team website, http://seniord.ece.iastate.edu/may1404/.

Appendix B

Alternative Design

Design	**Reason for Change/Omission**
Feeds application: native to Intuiface	Originally we planned to display data feeds within an Intuiface presentation only to take advantage of Intuiface's Interface Asset component. In this design, the feeds would not be visible once a user launched the content manager. Our client wanted the feeds to be integrated with everything else in our project, so we changed our design to meet this specification. Also, Intuiface wasn't capable of providing a marquee-like scroll effect, and MT4J could.
Maps application: pathfinding	Our initial scope for the maps application was quite ambitious. We considered designing it to allow for pathfinding across campus such that a user could enter a start point and an end point and would be shown an optimal path to walk or ride a bus. We eventually omitted this feature from our project because it was not part of the client's specification and it would have required a significant amount of developmental effort.
Maps application: building maps	Another feature we planned to add to our maps application was the display of building maps when users clicked on the buildings. Over the course of the project, our client asked us to focus more on integration with CyRide and the NextBus API so we decided to omit maps features that weren't related to this part of the specification.
Webcam application: video effects	One early design for the webcam application included access to video filters that could be applied to the live stream. This design was omitted from the final project because processing live video requires large amounts of processing power and we were unable to find external libraries that handled this in an efficient manner.

Appendix C

Other Considerations

Our team would like to recognize Jason Boyd for his help in deploying versions of our project to the Union Pacific Multimedia Wall. We would also like to acknowledge the ECpE department for providing a development and testing environment for our team in the senior design lab that we could use throughout the second semester of our project.

Working on this project was a great way to cap off our time at Iowa State University. We have all gained valuable experience working in a team, working for a client, and developing a large-scale system over an extensive timeframe.