Technical Report

for

RIT Mobile

Version 1.0

Prepared by

Team 5G
Steve DeCaires
Lance Fasciano
Eric Gardan
Alberto Pareja-Lecaros
Tom Schneider

Project Sponsors
Co-op & Career Services and
Development & Alumni Relations

Faculty Coach
Raghu Reddy

05/15/2010
1. **Project Overview**

   The RIT mobile application will be a modular web-based enterprise application that will be designed to fit the mobile platform of modern smart phones. Its purpose will be to ease the communication between thousands of alumni and current students, while providing a centralized location for RIT’s main services. The system will focus on integrating the multitude of web services provided by the Co-op & Career Service and the Development and Alumni Relations departments. This application will act as a central hub that incorporates both new and pre-existing services, which includes the integration of current social networks such as Twitter, Youtube, Facebook, and RSS news feeds.

   This system will be architected to support future RIT departments as modules, such as SIS and the multiple database technologies that they use. The main goal of this application will be to provide an easily extendable framework and a usable interface that students do not find cumbersome to use on a mobile device. The system must have an engaging and creative user interface, while meeting RIT’s web standards. It will also allow for a mixture of both public and user specific services. User services will require the application to meet RIT’s security standards for authentication.
2. **Basic Requirements**

**Operating Environment**

The core system will be deployed on an Application Server initially housed by Co-op & Career Services. Each subsequent department that is part of the system will house and deploy their respective web-service. The Application Server shall be a Windows-based Server running Microsoft .NET 3.5 and Microsoft Internet Information Services. The system was built using C# ASP.NET with Visual Studios 2008.

Our system has been tested to work on the following mobile devices:
- iPhone & iPod Touch
- Blackberry 9000 and greater
- HTC Incredible
- Motorola Droid

**External Interfaces**

The web-service will conform to Windows Communication Foundation's SOAP remote calls. Client mobile browsers must support latest HTML and CSS standards. JavaScript must also be available and enabled by the client. The XML responses returned by a department web-service must conform to its respective View Schema as defined on the Application Server. CAS Authentication will be used for authenticating Alumni & Student users. This is a pre-existing web application provided by RIT. The web-service back-end currently supports connections to SQL Servers and RSS Feeds. Hooks are also in place for Oracle Database Servers.

**Quality Attributes (in order of priority)**

**Maintainability** - The application was developed in accordance with good software engineering practices to help other future developers in maintaining the application. This includes the existence of proper documentation such as the inclusion of user documentation, which include the steps to deploy the project, introduction of a new department web-service, and creating new presentation views. The team also follow the C# coding standards specified by the sponsors throughout every class file. To verify that this quality is achieved, the development team will hand off the code to the sponsors and ensure that the sponsors can perform the necessary actions to continue maintaining this software and will be subject to their approval.
Scalability - The application was architected in a means to ease the growth of the system as much as possible. With the addition of a new department, only very minimal changes need to occur on the Application server. The department shall only need to create there corresponding web-service to connect to the Application Server. Creation of a new web-service is specified in the User Documentation.

Usability - The application shall be as usable as possible. To obtain a measurable usability score, the application was subjected to a SUMI review at the end of the last two spirals. The application was also tested in terms of usability during each of the development spirals by the development team and the sponsors.

Functional Requirements
3. **Constraints**

One of the biggest constraints in the system is the screen size on mobile devices. This too, however, was variable. In most cases, the screen size constraint was a significant factor on the user interface, driving decisions such as limiting user input as much as possible and making menus button-based.

Currently, the implementation seems to work quite well on newer Blackberry versions, the Android, and the IPhone. While the application is usable by other devices, they are not supported nor could we guarantee a smooth user experience. We imposed constraints as to which devices we would support for this very reason, and we have concluded that only modern smart phones would be supported, and in particular the IPhone. Other phones may still be used for this application, but the development was focused on the IPhone platform.

There are some technical limitations that exist due to the nature of the code. Because the framework is written in C#, we require the use of IIS and a Windows Server for deploying the mobile application. Also, we use CAS authentication currently to authenticate users to both of the web services. We require authentication regardless of whether or not the data must be accessed securely or not in order to simplify complications with the Alumni web service.

One of the biggest constraints in the system is the screen size on mobile devices. This too, however, was variable. In most cases, the screen size constraint was a significant factor on the user interface, driving decisions such as limiting user input as much as possible and making menus button-based.

Currently, the implementation seems to work quite well on newer Blackberry versions, the Android, and the IPhone. While the application is usable by other devices, they are not supported nor could we guarantee a smooth user experience. We imposed constraints as to which devices we would support for this very reason, and we have concluded that only modern smart phones would be supported, and in particular the IPhone. Other phones may still be used for this application, but the development was focused on the IPhone platform.

There are some technical limitations that exist due to the nature of the code. Because the framework is written in C#, we require the use of IIS and a Windows Server for deploying the mobile application. Also, we use CAS authentication currently to authenticate users to both of the web services.
4. Development Process

The development process that the team chose to use for senior project was the Spiral Model. At the outset of this project, the team identified a good number of risks associated with the development of a mobile web application. In addition, this project would inevitably require a large number of graphical user interface elements. Consequently, the team decided that the spiral model best fit the needs of the project and allowed the team to receive early iterative feedback on the development progress of the project at key points throughout the development. Since there were no requirements to use any particular process, this process was presented to sponsors and the sponsors conferred their approval.

The process helped the team communicate with the sponsor by specifying iterative spirals, which at the end of each spiral inherently implied that further approval from the sponsors would be necessary for the work completed during that spiral. While the spiral model does not explicitly state that roles are necessary for the success of the process, the team assigned a variety of roles to the team members to help assign responsibility throughout the project.

The roles were as follows:

- Tom Schneider – Team Leader/Primary Customer Contact
- Steve DeCaires – Trac Administrator/GUI Headmaster
- Alberto Pareja-Lecaros – Metrics Analyst/Test Coordinator
- Lance Fasciano – Requirements Manager
- Eric Gardan – Team Scribe/Development Manager.
5. Project Schedule: Planned and Actual

The schedule that was utilized was created early in the project life cycle, around the same time that we chose to implement our system with the spiral methodology. Going through and prioritizing the requirements for our system initially developed our schedule. We created a specific order, with the most important, most dangerous, and most desired functionality early in the cycle. Once those items were completed, we would move onto the functionality that was not as desired and easy to implement. By prioritizing functionality in this manner, we would work on the items that were required with future features and implement areas that we would encounter the most problems. This way, if we encountered unseen problems, those features could be extended in the schedule, but will still get implemented. The features that were prioritized toward the bottom of the list would risk not being implemented. However, since these items were the least desired or easiest to implement, they could be skipped or implemented quickly.

Our schedule was organized into a series of five iterations. Each iteration was designed to complete a specific set of deliverables for the project. The initial iteration was set to begin right after winter break. We gathered some of the requirements for the system from the sponsors and felt that this would be a good time to start delivering the items. The first iteration (iteration 0) was created to complete the documentation for our system. For example, at the end of the first iteration we wanted to have the SRS document, test plan, and other relevant documentation completed. This would provide a one-week period for our faculty coach to approve the documents before they were presented to the sponsors in week eight, which was during the second iteration.

The second iteration was designed to update the documentation for any scope changes that may occur and to begin implementation of key system features. For example, logging into the system was to be implemented during this iteration. System verification was a major aspect of our system, so that only students and alumni would be able to access the information. Exactly how that feature was going to be implemented was unknown to the team, so it was a high-risk feature. By implementing it early, the probability of getting it implemented correctly will be increased and decrease more problems in the future. Another feature within this iteration was to deploy a test version of our application to verify that we could publish content and view it on a mobile phone.

The third and fourth iterations were designated to implement the bulk of our systems functionality. For example, each of the widgets to display information would be implemented and all of the converters in the server would also be created. The fifth
iteration was created to test our application on a variety of mobile devices and to deploy our application to the Career Services server. Overall, our schedule was clearly defined what would be implemented when early on in the products development.

Although our schedule was defined with all of the required functionality, we ran into problems during the third iteration. The problem was not caused by incorrect planning, but was caused by a requirement change from our sponsor. This change required that our architecture change from basic client-server architecture to a client-server architecture that used a series of web services to retrieve the data. The new system would also pass XML files into a specific page and components would be generated based on the XML created. This was a major change for our system and required us to go back and reevaluate our schedule in order to properly complete the project on time. Since we were already inside of our third iteration and it was nearing an end, we decided that the bulk of the work would be within the fourth iteration. Therefore, we decided to extend the amount of time within that iteration to allow all of our features to get implemented, and then use the fifth iteration for testing, as it was previously scheduled. The fourth iteration was then moved to a total of four weeks, from a three-week iteration, and the fifth iteration was moved to a total of two weeks, from a three-week iteration.

Based on the changes to the requirements, we designated the set of features that would be implemented in the fourth iteration. This included the creation of four dynamic views that would interpret the XML files created, the web services that would contain the converters to create the XML files, and a series of links from the client side of the application to the web application. We mapped out week by week how much time would be put into the implementation of the features, as well as when they would be completed. This allowed our team to properly chart the changes and how we were going to execute the new schedule. Each week we would check the progress of the mapped tasks and verify that our schedule was still on track. If a feature was steering off the schedule, the remaining time to complete was retrieved and compared to the existing schedule to see how much of a difference it would make. This process helped greatly and allowed our team to get through a difficult development period.

Overall, our teams planned and actual schedules were different, but we were still able to complete the project on time and it met the needs of our sponsors. The requirements change that was presented to us in iteration three caught us off guard, but we accounted for the changes and developed a product that was implemented and tested based on the desired specifications.
6. System Design

Our system consisted of two separate designs, one before and one after our major design change. The original version of our design was focused on performance and extensibility. It was designed as client-server architecture. The client aspect of the application would contain a series of pages that would be statically defined. Each page would reference one or more “widgets.” A widget is a component that would be embedded on a page and execute a specific set of functionality. The widgets would be designed to follow a specific XML schema, which would be used to make sure the appropriate XML was being passed into the component. The XML would then be used to display the proper information inside of the widget.

The server-side of our original design would contain a set of readers and converters, utilizing the strategy design pattern. A reader would be used to connect to an existing data source and iterate through the contents of it. The contents retrieved would be based on a query or looking at an entire file. Originally, we were planning to have three different readers, providing the ability to connect to an SQL database, Oracle Database, and read RSS feeds. Once information was retrieved from one of the readers, a specific converter would be used to convert the raw data into a standardized XML file that could be read by one of the widgets.

In order to retrieve the proper information and execute the correct classes in the server side of the application, a method name was defined within the widget that would be used to retrieve the correct XML document. The server-side contained a single façade class, which would be used to separate the functionality that was needed away from the functionality that was not needed for the view. The façade contained information for executing database queries, as well as connection information and RSS feed URLs. When the method was executed, the appropriate reader and converter would be used to generate the XML file. This XML file would then be passed into the widget that originally called the method and display the content appropriately.

By providing the design in this manner, the application could define a static layout, provide separate color and format schemes for the widgets that were added to the pages, and allow a single type of widget be used with multiple departments. For example, a single “News Widget” could be used between the pages to display information in a list view, which contains a series of details. This allowed the system to be easily maintainable, since a single widget type could be added anywhere, as long as it took in the appropriate XML, and only allowed a few areas of the application to be
modified in order to work. This design satisfied the original needs of our customers, so we began implementation of the system using this strategy.

During our third iteration, about the second week of the second quarter, our sponsors wanted a large change to the systems design. They wanted our system to use a series of web services, specific to each department, which would be used to retrieve the standardized XML files for our system. The XML would then be passed into a specific view in the client-side of the application and would be used to dynamically create the page based on the content of the XML. This set of changes was much different from our original architecture that was created and required our team to think about how the changes could be implemented into our existing design.

After much brainstorming, we reached a conclusion that the changes were not as drastic as originally thought. The client-side of our application would remain exactly the same, except a façade method would have to be implemented to contact the web services. Similarly, our server-side design could also stay the same, except it would have to be moved over the web service. The changes of our design resulted in a system that worked appropriate to our sponsor’s needs.

The new design is split-up into two sections, client-server architecture and a series of web services. Each department contains its own set of implementation of the system as well as their own web service. When a user initially navigates to the web application, they are greeted by a home page. This home page is completely static and provides links to the navigation menu of available features for the department being
viewed. The navigation menu lists each feature vertically. Each item contains a reference within it for the type of view that will be accessed and which method in the web service that should be called to display the information in the view.

Our design incorporates a total of four views that will display the page dynamically. The XML file that is passed into it determines the information displayed in the views. Each view requires a XML file that conforms to a predefined schema, in order to interpret the results correctly. Depending on the tags and information within the XML, the page will be constructed appropriately. The file is parsed in a top-down fashion, with the components and information being displayed in the same type of manner. Each view provides a different set of components and formatting for the system to display in order to provide the most useable system for a user with a mobile device. The following are descriptions of the views that have been developed for our application:

**DataListView** - Used to display name-value pairs and lists of data. This view is primarily focused on displaying large amounts of content to the user in separate sections. Each section contains a header to separate content into their appropriate areas and a horizontal line at the end of it. Pictures can also be added to this page, which have been referenced in the XML passed into it. This page is intended to be used by itself, without any references to other views. Some pages that this may be used to display: about me information and contacts list.

**SearchView** - Used to search on multiple criteria. The view will display a series of components, such as text boxes, drop down lists, and calendars, which can be used to narrow the results of a query for another view. This view is intended to be used prior to a FeedView.

**FeedView** - Used to display data, which contains details. The view will display a series of headers separated into their own sections. Each header of information is intended to provide the user with enough understanding of what the following page will contain if it is selected. Any number of results can be displayed on the page at one time and provides the functionality for the user to page between the pages (next and previous). Only a single header, typically the title of the following page, will be displayed in each header section. This view is intended to be used prior to a DetailView and after a SearchView. Some pages that this may be used to display: search results and news items.

**DetailView** - Used to display details of a feed item. The view will display information for a feed item separated in different sections. Each section contains a header and can only contain one piece of information. Pictures can also be added to
this page, which have been referenced in the XML passed into it. This view is intended to be used after a FeedView.

A user is transported to the appropriate view based on the item that was selected inside of the navigation menu of the specific department at RIT. As previously stated, each navigation item contains a reference to the specific view and the method in the department web service that should be called in order to display information in the view appropriately. When the link is pressed, the page is navigated to the specific view and the method that was referenced in the navigation item is passed into the page and executed. When the method is executed, a call is made to the web service to retrieve the appropriate XML. Similar to our previous design, a reader will be executed to retrieve information and then passed into a converter to standardize it. Currently, we have implemented two types of readers. One is to query an SQL database and retrieve the results and the other is to retrieve an RSS feed and iterate through it.

Once information is retrieved from the specific reader, it is passed into a converter to be standardized. The converter that will be used is dependent on the method that is executed within the web service, which was appropriately referenced in the client side of the application. The converters are dependent on the information that will be viewed and in what page the information will be viewed on. For example, a user wants to see a list of news items in a FeedView. To do this, a converter built to display those news items within a FeedView is called to return the proper XML document.

Each converter built will be used to convert the raw data into a format that can be interpreted by the view. The converter needs to follow a schema for the view that it will be displayed in and needs to place information into the XML in the manner that it will be viewed on the page. After the converter function has been executed, a standardized XML has been built. The XML document is then passed to the client-side of the application and displayed in the view that it was executed on. If the XML document does not match the schema that was already established, then an error message will be displayed to the user. Otherwise, if it was built specifically for the page, the information will be displayed properly and provide the appropriate results to the user. If the view is going to be referenced to another page, then the method that will be executed for the next page will be referenced in the XML that has been passed into the view. The order of the pages that can be traversed is SearchView to FeedView to DetailView.

The transition from the old architecture to the new architecture was a major change, but one that better fulfills the needs of our sponsors. The use of a web service allows the use of the XML that is generated from the converter to be used with other
applications, such as a native smart phone application or with another web application. This provides our system the ability to be quite scalable in the future. Similarly, a new RIT department could begin to use our system with minimal effort to set up their initial application. The client side can be set up in a short period, but the establishment of the converters to display the information appropriately will take a bit longer to create and connect with the application.

Our system is also extremely extensible; specifically since the client side of the application does not have to be changed in large amounts. Since the views that are going to be used have already been implemented, the only changes that need to be added to navigate to a new page would be to add a new link in the navigation menu that contains the type of view and the method to execute within the web service. Most of the changes that have to be accomplished would be in the web service for the department, since a converter will need to be constructed to display the information within the view correctly. This also provides for maintainability, since there are few places that the code has to be modified in order to get it to work. Overall, the new version of the application provides a large number of benefits over our original design.

Although there are many benefits to the use of this architecture, there is one slight cost. Since we have to contact an external web service in order to retrieve the correct XML document, there will be a performance decrease due to the extra work involved. In our old design, all of the work to develop the correct XML document was completed on the same server that the application was being ran on. This provided enhanced performance because there was minimal travel between the application and the client side of the application. However, since the web service is now outside the application, there will be an increased time to retrieve the proper XML that will be displayed. Although the effects will be minimal if the web service is on the same server, if the web service was located in an alternate location, the application may run much slower.

After the requirements changes and the implementation of our web application, the new design has provided a system that meets the needs of our sponsors and provides the possibility to be expanded to other departments with relative ease. Development of the system went without many difficulties and resulted in an application that was exactly how we envisioned it. The design of the system provides many benefits and will allow the application to be used for many years to come.
7. **Process and Product Metrics**

The main goals for our measurements and metrics effort were to keep the project on schedule, to help identify potential problems as early as possible, and to continuously update our spiral iterations so that they are more realistic and exact. To that effect, we determined the following metrics as vital to the success of our project:

- Software Usability Measurement Inventor (SUMI)
- Number of Trac tickets closed per milestone
- Effort per person
- Performance Benchmarks

As a mobile web application, we determined usability to be the most fundamental quality attribute of our system. By establishing a metric in order to track the quality of use, we detected potential flaws with the user interface. Since our end-user demographic includes all students and alumni at RIT, distribution of a SUMI questionnaire was not difficult, although getting people to actually take the survey was. We used both the Alumni & Development and Co-op & Career offices to help distribute the questionnaire to our potential demographic. We distributed our survey initially at the end of week 17 and continued to obtain results until after the end of spiral 4 due to lack of user response.

Trac has a tool that allowed us to set specific milestones for our application and could keep statistics on the number of tickets closed during a particular milestone. We utilized this tool not only as a way to keep track of bug fixing progress but also to verify that our bug/defect finding efforts were adequate. To that effect, any particular milestone that varied greatly in the number of tickets produced could be looked at as a warning flag for incomplete or haphazard testing. It could also be an indication of far fewer features being integrated into the product.

Spiral 3 ended up having the most tickets closed, although in all milestones we closed 100% of tickets. This is mainly due to our process of either moving a ticket to the next milestone or scaling down scope.
Effort estimations for each individual on the team were required by the senior project parameters but were also a way of making estimates more accurate for the team by showing the amount of time past activities took for the person. This ultimately allowed us to divide up the work in a manner such that no one person had significantly more work than anyone else and also allowed us to assign appropriate tasks to individuals who took the least time to accomplish those tasks.
During our second term our effort estimation got significantly better, although in the first term we actually overestimated how long everything would take which was not bad. In general we can see that we had a good idea of how long tasks would take and we worked diligently to make sure that everything stayed on track.

Performance testing was done in order to get some benchmarks as to how fast the application would load on the mobile phones on both WIFI and 3G connections. We wanted to make sure that the pages would load quickly enough in order to ensure people would use the application.
In general WIFI was always faster, and screens with more data took longer to load (as expected). All of the phones performed similarly in tests, with the Incredible outperforming the Motorola Droid and the IPhone. The statistical variation between all of the datasets was small, and performing the Student T-Test showed that the variations between the datasets were insignificant, meaning their means were close to zero.

Unfortunately we still have fairly insignificant data due to the sample size, however it was made abundantly clear that using anything but the more modern smart phones makes the web application largely unusable due to the poor user experience of web applications on such phones.

In the end, the metrics show that our application is fairly usable and the project itself went smoothly even with hiccups such as the redesign. The best thing we got out of the testing is a general sense of which phones performed poorly with our site and which worked well. We now have a sense of which phones could use better support and what features affect other phone platforms the most. We feel the next group that uses our framework will best serve our metrics.
8. **Product State at Time of Delivery**

In terms as to what was agreed upon with our sponsors, our application is feature complete. Only a few features were missing from what was initially planned in the SRS. These features were agreed to be dropped from both parties in the second term. This was due to either budget constraints (external API), or lack of access to specific student data. Despite going through a re-design early in the second term, we did not have to sacrifice a single customer feature.

The application is currently deploy on a test server owned and housed by Co-op and Career Services. The Co-op and Career office are currently in the process of transferring the core application and their web service to commercial server with a better infrastructure to handle the increased stress load. The Development and Alumni Relations office are currently planning on deploying their web-service at the end of the summer, after transferring to LDAP authentication.

The Spiral process provided us with the concept of prototyping, which had greatly help communicate to our sponsors what we had envisioned the system to be. Overall, the Spiral process provided a little too much overhead for the scale of this particular project.
9. **Reflection**

Overall, we had a very positive experience. Although the design change created a difficult situation for the group, we felt it provided a more “real-world” scenario and helped us design a better overall system. We had great sponsors, and they provided us with feedback as quickly as possible. The group had a wide variety of skills, which allowed us to work well together as a team, both in the sense of process management as well as development. In addition, our group dynamic worked well and there were no personal problems between members of the team. For future teams, we would advise that they create prototypes early and often, as these can be invaluable to helping the sponsors understand the team’s current vision.