RFID Event Specification Using Templates in Scenic and Event Notification

Kayla Gould, Evan Welbourne, Magdalena Balazinska

Department of Computer Science and Engineering
University of Washington, Seattle
Seattle, WA 98195

{keucken,evan,magda}@cs.washington.edu

ABSTRACT

Scenic is a program that allows the user to graphically specify an event to be detected by the RFID Ecosystem in the Paul G. Allen Center at the University of Washington. The user drags icons representing people, places, items and "event primitives" [5]. In this paper we discuss how Scenic has been extended to (1) save and recall an event definition, (2) edit previously created events, and (3) create event templates.

Notifier is a new program we created that allows the user to subscribe to notification for an event that they have created using Scenic. The user enters contact information such as an email or a phone number and specifies which method to use to be notified when the event occurs. The system monitors the readings from the RFID Ecosystem and when it has detected that the event has occurred, it sends out a notification via the specified medium.

Categories and Subject Descriptors

H.4 [**Information Storage and Retrieval**]: Miscellaneous; H.5 [**Information Interfaces and Presentation**]: Graphical user interfaces (GUI)

General Terms

Design

Keywords

RFID

1. INTRODUCTION

The project is designed to make use of the RFID Ecosystem set up in the Paul G. Allen Center at the University of Washington [4]. The ecosystem consists of a network of RFID antennas and readers that have been strategically placed on all six floors of the building and the basement. To make use of this ecosystem, users volunteer to wear RFID tags where-ever they go and also usually tag some of their personal items such as books, keys, laptops, and backpacks. In addition to the hardware and users, the RFID Ecosystem provides software and a database for recording which tags were seen by which antennas.

There are two specific goals set out by this project. The first goal is to enable users to more easily and efficiently create events by updating previously created events and by creating and updating event templates. The second goal is to build a working notification program to allow users to subscribe to and receive notifications about RFID events that occur in the project's pervasive RFID environment.

The programs developed under this project will allow users to track occurrences of interest involving themselves, their tagged items, and places that are covered by the RFID antennas. These occurrences are called "events" and may be such things as when a person and their advisor meet in the advisor's office, when a person is seen leaving the building without their keys, or when a stationary, inanimate object changes location (*i.e.* is stolen). Using our programs, the user can specify events either from scratch or from templates, edit them as much as they please and subscribe to notification for the events. This means when the event happens, the system will send out a notification to let the user know that their event actually took place.

We evaluate the usability of our system through a user study and find that the addition of templates is indeed beneficial and that both programs are quite user friendly. We also identified a few minor interface design changes to make.

1.1 Motivation

Humans are often forgetful. We use many different ways of reminding ourselves of important details, such as post-it notes, emails to ourselves, and entries on calendars. Most of these methods are created and then must be visible at the critical time for the reminder to be effective. For example, consider the case where the user writes a post-it note reminder to bring a book to a meeting to give to someone and must see the note as the user is leaving their office. The reminder may be overlooked or may not be necessary if they already has the book in their hands.

Our system will allow the user to subscribe to conditional reminders which, depending on the notification method selected, may be able to deliver the reminder right to you no matter where you are.

This project extends and integrates with the Cascadia System [5]. Cascadia is a middleware for developing applications that need to specify, detect, and manage RFID

Some Conference '09, Some Date, 2009, Some Location.

events. Cascadia allows developers to incorporate RFID events without having to deal with the inherent uncertainty and ambiguity in RFID data. Cascadia also handles event detection, thus allowing the developer to merely subscribe to an event and wait for messages saying that the event has happened. Our project extends the portion of Cascadia that handles event specification and adds the ability to have detected events reported to the users in a customizable, high-level fashion.

1.2 Related Work

Our project is similar to the SPECs project done by Mik Lamming [2] where they used small peer-to-peer computers mounted on people and objects to keep track of which items were with a person so that the person can be reminded when they leave an item behind. Their system required a very minimal number of devices to work but was only designed for keeping track of items. Our system can be told to remind you if you forget certain items but events other than forgetting things can be tracked, where the SPECs project was only scripted to keep track of items. We also use a building-wide RFID ecosystem of readers instead of peer-to-peer computers.

The RFID reminding project done by Boriello *et al.* [1], worked on keeping track of tagged inanimate objects. Their system allowed the user to write reminders about what items they should have with them at certain times. The system would watch for those items and then remind the user on a special wristwatch when they forgot an item. Their method of specifying "reminders" was text-based and might make an average user disinclined to learn the system, where our method of specifying events, which can also be used to keep track of items, is graphical and perhaps much more comfortable for an inexperienced user.

Sohn *et al.* [3] worked on a location-based mobile reminder system called Place-Its. Place-Its allows a user to go to and name places and then create reminders to be deployed when they are once again near that place. They found that location-based reminders are useful even when the accuracy of the location detection is not very good. They used GPS technology for location sensing. Our RFID location sensing is innately more accurate but is only available within the confines of the RFID reader deployment area.

1.3 Contributions

This project builds on existing research but makes the following contributions.

1. **Abstracting from technical details**. Unlike programs such as the reminder system by Boriello or the SPECs project [1, 2] users of our program are be able to specify what to be notified about without needing to learn a scripting or programming language. While a background knowledge of how the system works is helpful, as long as the user can drag and drop icons and use right-click menus, they should be able to create event specifications using Scenic. The events can also be managed very easily with the templates and tables that have been added to Scenic (Section 2).
2. **Allowing users to choose notification details**. Many of the projects previously developed allow the user to only use one method of notification such as a mobile phone, or a wristwatch [1, 2, 3]. Our program allows the user more choices of notification and even allows them to customize the notification message (Section 3)

We present the event specification ability of Scenic in Section 2 and the notification ability of Notifier in Section 3. We discuss the system architecture in Section 4. Finally, we describe our user study design and discuss the results in Section 5 and conclude in Section 6.

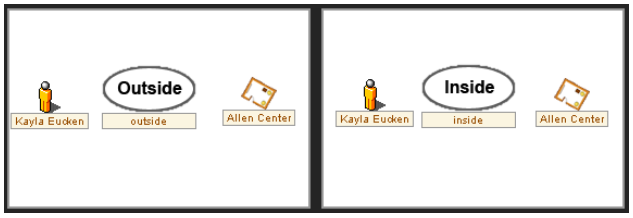

Figure 1: An event where a user enters the Allen Center

2. EVENT SPECIFICATION

On the base level, the data obtained from the RFID Ecosystem is a series of RFID tag reads by particular antennas. Specifically, tag reads consist of a tuple in the form '(tag, location, time)'. To the average user, this level of data is not very useful. Our goal is to abstract the data, attach meanings to it, and allow management of it in such a way that a non-technical user could find it helpful in their daily life.

One such management abstraction is the ability of the user to specify RFID events. An RFID event is a set or series of sets of tag reads that have some meaning associated with them. One simple event is the sighting of the person tag for Person A outside the Allen Center and then a sighting of the person tag for Person A by an antenna inside the building. Such a sightings would mean that Person A has entered the building.

In the Cascadia system and our project, events will be specified using a program called Scenic [5]. Scenic provides a visual way for the user to specify scenes of an event. The user can, for example, add a scene to the event, drag down icons that represent a person, the modifier "Outside," and an icon that represents a place. They could then right-click the icons to specify that the person is themself and the place is the Allen Center. They could then add a similar scene consisting of themself inside the Allen Center. The combined scenes form an event where the user is seen entering the building (see Figure 1).

While this event specification process is relatively easy, the accuracy is sometimes questioned because the relative positions of the icons can change the perceived meaning of the event. For example, in an event that is "Person A Near Person B Far Database Lab," the arrangement of the icons may make it unclear whether the 'Near' and 'Far' apply to Person A and Person B, Person A and the database lab, or Person B and the database lab.

This project has added to Scenic the ability to specify events using previously created templates. A template is an event with some of the details left out. These will be created by developers and the user will have the option to create their own template if they wish. We anticipate that the incorporation of templates will help increase the confidence with which people can specify events in Scenic. It may also

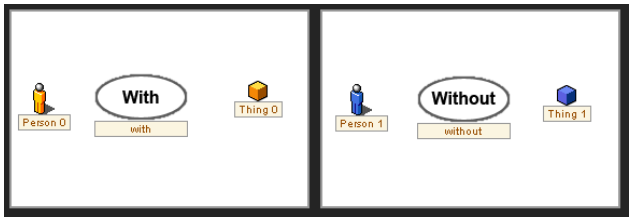

Figure 2: A Scenic template where a person loses or discards an item.

increase the accuracy of the events specified, as measured by whether or not the event actually describes what the user intended.

The GUI and functionality of Scenic have been extended quite a bit. Not only can users drag and drop icons to make events and templates, but they can actually view events and templates that they have previously made. A significant accomplishment of the project was the ability to simply reload an event or template for editing. Lastly, all but one of the property lists for event actors have ceased to be hardcoded, thus allowing for more truely personallized events.

2.1 Templates

The user can create a template which is saved as a name, a description, and an XML representation. They can also create new events either from scratch or from a template that they or a developer has previously specified. GUI components have been added to allow the display of previously specified events and templates so that they can be viewed, edited, and deleted. A template consists of an event, such as the building entering event above, but with some of the details left out. The user will supply the missing details, enter a name for the event, and save it. For an example of a template, see Figure 2, which is a template for a person losing or discarding an item.

The XML representation of a template or event is created by parsing through the grammar tree that represents the scenes and actors in an event. All grammar elements have toXML() methods which are used in the same manner as a toString() method. Depending on the type of grammar object, the name, type, ID, position on the screen and other details may be stored in the XML. To restore the template or event, the XML is parsed and the grammar tree is recreated.

2.2 Dynamic Property Lists

Scenic has also been modified to use dynamically populated lists of properties for actors such as people, people groups, items, and places. One result of this change is that items listed in events are real, tagged items instead of fake, hard-coded item names (see Figure 3). The lists of people's names, groups of people, item names, and places are now all populated dynamically. Item groups may be temporarily removed since there is currently no source of data for that category.

Dynamic property lists are implemented by replacing the hard-coded lists of actors in the ActorLibrary class with methods that will pull rows from the system database and populate the array lists in ActorLibrary with the correct type of Person, PersonGroup, Thing, or Place objects.

The names of people are populated from an existing table of RFID system users. For privacy reason, only the user

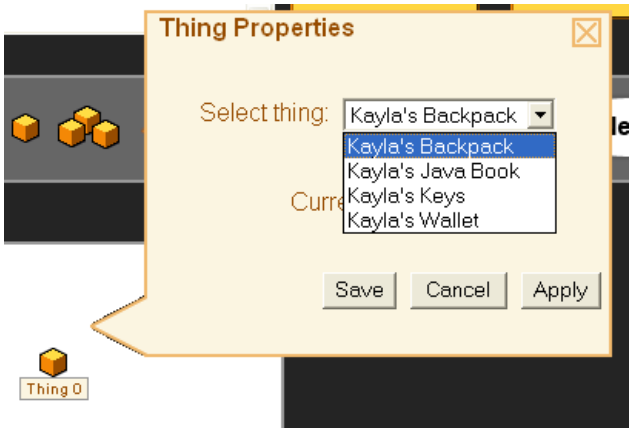

Figure 3: A dynamically populated property list in Scenic

and people who are friends of the user can be used to make events. This increases an individual's privacy as it prevents people from creating events with people they do not have permission to track. Currently only groups of people that the user owns, such as "My Friends" will be returned. This may be extended at a later date to also include groups in which the user is a participant. Similarly, only the names of items belonging to the user are visible to them and available for use in events. The Allen Center is still hard-coded as a place for all users since that is the location of the RFID Ecosystem, but all other places listed are ones owned by the user.

3. EVENT NOTIFICATION

In addition to providing Scenic for event specification, our project provides a program called Notifier that allows the user to subscribe to notification for an event that they have previously specified. Notifier consists of two programs, a GUI, referred to as Notifier, and a server-side java application, referred to as Notification Module.

3.1 Notifier

Notifier requires very little effort to set up the first time a user launches it. The program prompts first-time users for their email, phone number, and phone provider. The system is not currently hosted in such a way that users can actually log in, so the program prompts all users for contact information. Once the program is hosted in some way that permits logging in, it will differentiate between new and returning users. At that point the user's contact information will also be required and stored to the database.

Returning users will automatically be forwarded to a list of events that they have notification subscriptions for. From that list the user can choose a notification subscription to modify, activate or deactivate a subscription, and delete a subscription. When an event is selected for modification, the details of the event are loaded and the user may make changes, including changing the notification details. Activating or deactivating a subscription currently only changes the value of a single field in the database but once the program is actually connected to the event detection system it will change whether or not the event is watched for. The difference between deactivating and deleting a sub-

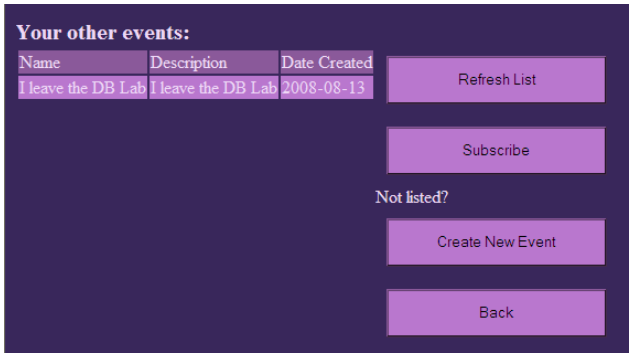

Figure 4: Notifier: The list of a user's previously specified events which have not yet been subscribed to.

scription is that a deactivated subscription remains in the database while a subscription that is deleted is deleted from the database and must be recreated if the user wants to use it again.

In addition to viewing currently subscribed to events, the user can select an event from a table of events previously created using Scenic and click a button to subscribe to notification for it (See Figure 4). The user can specify whether or not the event is recurring and can choose whether to be notified by email or SMS text message, or, some day, phone message (See Figure 5). There will be a default message which, in the case of the email message, takes the form "The RFID Notification System has detected that your event named [event name inserted here] has happened." The notification messages are customizable except in the case of recording your own phone message, which has not been implemented yet. For events that the user specifies email or SMS as the notification type, their default or customized message is stored. The user can also customize and preview a text-to-speech phone message, but the phone messages do not store yet and you can't yet record your own message.

There will also be an option to change the probability threshold for the notification. For example, if the system is only 60% certain that the event has occurred, it will not bother the user unless they have specified a certainty threshold of 60% or less. This will allow the user to specify a low threshold for events that are more critical to detect, such as an inanimate object moving from a predetermined spot (possibly being stolen), and to specify a high threshold for events that are less critical and they don't want to be bothered about unless the system is very certain.

When the user saves the notification details, the program saves the notification to the database and will submit the event for detection.

3.2 Detection and Notification

For the actual event detection, the system will make use of the Cascadia Event Manager which will take the event specified by the user and notify the project's system once it has decided that one of the user's events has occurred. Cascadia uses a particle filter to take the raw data from the antennas, smooth it, and assign probabilities to where the tag was located. The smoothed data is then sent to PEEX, a probabilistic event detector, which monitors for data that matches the specifics of an event. The Cascadia

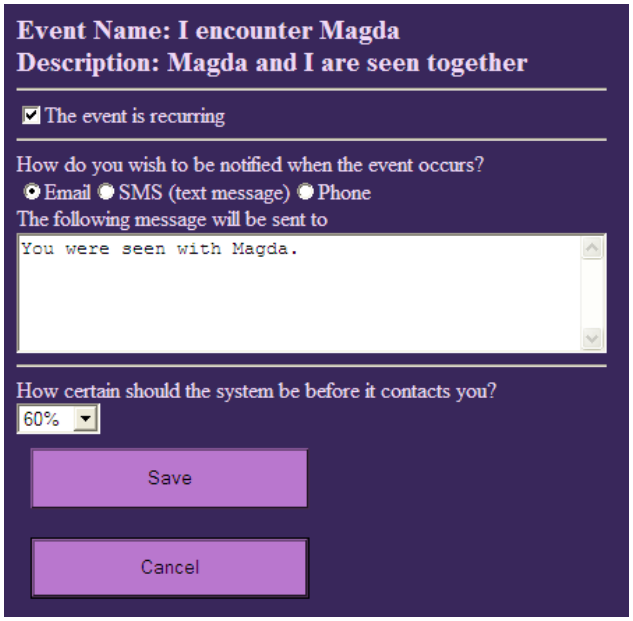

Figure 5: Notifier: The notification creation form.

event detection system does not currently run in real time so the connection with Cascadia has not been established or even attempted.

The program responsible for actually sending out the notifications is called NotificationModule. As mentioned above, it is a java application that runs on the server and listens for events to be detected. Once events are actually detected, NotificationModule will receive the detected event's Event ID and the probability with which it was detected. Once the program has ascertained that the probability that a specific event has occurred exceeds the user's thresh-hold probability, it will send the user the predefined notification. This will entail sending out an email, sending a SMS text message, or calling the user's cell phone and playing a prerecorded message when they pick up.

NotificationModule is currently capable of sending emails and SMS messages. To do so, it uses the Javamail API and a University of Washington mailserver. The SMS messages are sent using exactly the same method as the emails. To manage this, we keep track of the user's phone provider. Most popular phone providers support the ability to send text messages as emails. To send a text message as an email, the phone number is concatenated with the appropriate extension supplied by the phone provider. For example, as of the time of this writing, Verizon supplies the extension "@vtext.com" so a text message sent to a Verizon user would take the form "5554443333@vtext.com". Other phone providers an the corresponding extensions are listed in Figure 6.

4. SYSTEM ARCHITECTURE FOR SCENIC AND NOTIFIER

The system architecture for this project is fairly simple and is represented in Figure 7.

Scenic and Notifier both consist of web applications which communicate with servlet backends. The Scenic servlet stores and retrieves events and template in the Cascadia

TAYLOR.test_k...honeProviders | Object Explorer

	ProviderName		Extension
	Alltell	...	@message.alltel.com
	AT&T	...	@txt.att.net
	Cingular	...	@cingularme.com
	Metro PCS	...	@MyMetroPcs.com
	Nextel	...	@messaging.nextel.com ...
	Other	...	@Teleflip.com
	Powertel	...	@ptel.net
	Sprint	...	@messaging.sprintpcs.co...
	SunCom	...	@tms.suncom.com
	T-Mobile	...	@tmomail.net
	Unicel	...	@utext.com
	US Cellular	...	@email.uscc.net
	Verizon	...	@vtext.com
	Virgin Mobile	...	@vmobl.com

Figure 6: Known phone providers and the email-to-text-message extensions that they provide

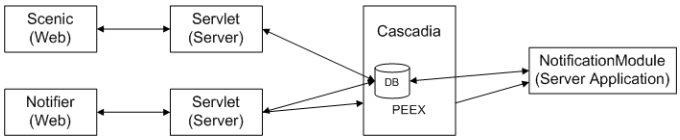

Figure 7: A diagram of the system architecture for Scenic and Notifier

system database. For more details on the tables used, see Appendix A.

The Notifier servlet retrieves the user's previously specified events from the Cascadia system database. It also stores and retrieves the notifiications and notification details that the user creates. For more details on the tables used, see Appendix A. The Notifiier servlet also communicates with the Cascadia Event Manager and thus Peex, as indicated in Figure 7.

When events are detected by Peex, the information about the event is sent to NotificationModule. NotificationModule will look up the necessary information in the Cascadia System database and will send out the notification.

5. EVALUATION

The evaluation for this project was somewhat incomplete since the required supporting RFID Ecosystem programs are not all in states of completion such that they can be used by real users. However, the initial, informal evaluation was still very useful.

5.1 User Study Design

The user study was designed to provide some evaluation for the design of the GUIs of both Scenic and Notifier. Other than just getting user feedback on whether the design was intuitive and easy to use, the study was intended to answer the following questions:

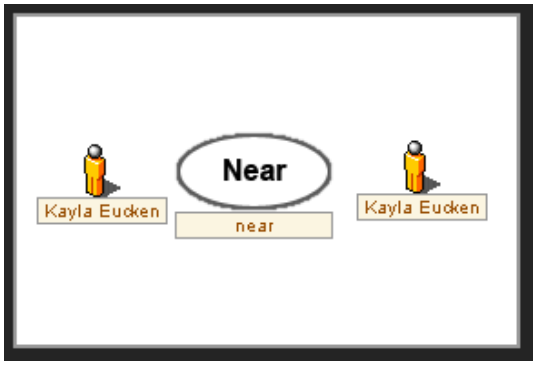

Figure 8: The correct event specification for Scenic task 1.

1. **Does using templates make it faster to specify events?** We would expect that using templates should make it faster to specify events, although maybe not by much unless the event is fairly complex.
2. **Does using templates increase the accuracy of specified events?** We would expect that using templates should increase the accuracy because the user can see what has been set out as a model of a functional event.
3. **Does using templates increase the confidence in the accuracy of the specified event?** We would expect that using templates should increase the user confidence in the result because they have some confirmation that what they are creating is valid.
4. **How does the effect of using templates change as event complexity increases?** The effect of using templates is expected to increase as the complexity of the event increases, *i.e.*the more complex the event the more beneficial it is to have a template to at least start from.
5. **How easy is it to carry out specific tasks in Notifier?** We hope that it will be fairly easy and intuitive to complete tasks in Notifier.

In order to answer the above questions, the study participants were given the following scenarios for four events to specify using Scenic. The expected answers were constructed as reference in order to assess the accuracy of the participant's events. Note that the participant substituted their own name where the scenario says 'you'. For the reference 'correct' events, the user's name was Kayla Eucken and would be replaced with the participant's name in the real events.

The complexity of the events is measured by counting the number of icons needed to represent the event. While this definition is somewhat arbitrary and does not necessarily capture the true complexity of thought represented by the event, it is a simple, non-subjective method of measuring complexity and was deemed the best available.

1. **You want to keep track of all your meetings with Kayla.**
   - Complexity: 3
   - Correct event: see Figure 8
2. **You want to record when you enter the database lab.**
   - Complexity: 6

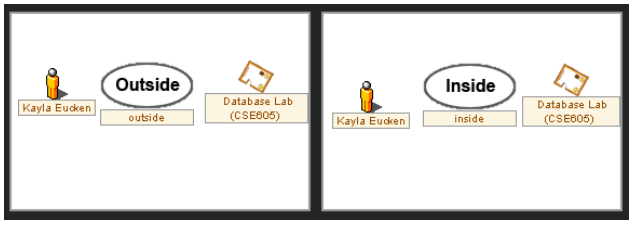

Figure 9: The correct event specification for Scenic task 2.

Figure 10: The correct event specification for Scenic task 3.

- Correct event: see Figure 9

3. **You want to know if you leave the building without your keys.**
   - Complexity: 10
   - Correct event: see Figure 10
4. **You want to keep track of your coffee breaks where your coffee breaks consist of you carrying your coffee mug out of the database lab and staying with it in the Jaech Gallery.**
   - Complexity: 15
   - Correct event: see Figure 11

The accuracy (correctness) of the participant's events was measured as the number of icons that were consistant with the reference event, divided by the number of icons in the reference event. For example, suppose that the participant does task 3 (Figure 10) and leaves out the Inside, Outside, and place icons. Their accuracy would then be 6/10.

Two sets of templates were created for the study. One set, referred to as 'good,' included templates believed by the investigator to either be templates that would create events that were exactly what the scenarios suggested or else were templates that could be easily extended to create the events from the scenarios. The list of 'good' templates can be seen in Table 1.

The other set of templates, referred to as 'bad,' consisted of templates that were similar in concept to what was being asked but not actually what the user was being asked to do. For example, the second scenario asked for a user entering a place but while a template for Person Enter Place was not provided, Person Leave Place was. The list of 'bad' templates can be seen in Table 2.

In addition to the four events to be created with Scenic, the participants were given the following four tasks to complete using Notifier.

1. **When you enter the database lab, you want to receive an email that reminds you to read over Cynthia's paper and email her your comments.**

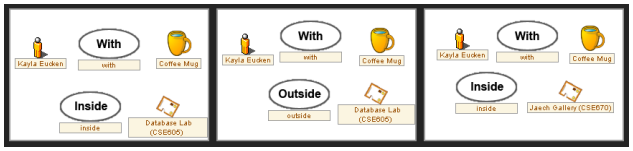

Figure 11: The correct event specification for Scenic task 4.

Table 1: 'Good' Templates

TemplateName	Template Description
Person Bring Item In	A person is seen with an item outside and then inside a place.
Person Bring Item Out	A person is seen with an item inside and then outside a place.
Person Near Person	Two people are seen together
Person Leave Place	A person seen inside a place and then outside the place
Person Enter Place	A person seen outside a place and then inside a place
Item Enter Place	An item is seen outside a place and then inside the place.
Item Leave Place	A item is seen inside a place and then seen outside the place Person
Leave Item Behind	A person is seen with an item in a place and then seen somewhere else without the item.
Person With Item	A person is seen with an item.

- Correct action: select the event from the list of unsubscribed events, subscribe, create a custom email message, save.

2. **If you forget your keys in the building, you want to receive a text message saying that you forgot your keys and you want to be notified if the system is even half certain that you forgot your keys.**
   - Correct action: select the event from the list of unsubscribed events, subscribe, create a custom text message, select 50% threshold, save
3. **You loaned your book to Kayla for a week so for now you don't want to receive notification when your book is outside the database lab.**
   - Correct action: select the event from the subscribed events list, unsubscribe.[1]
4. **All the SMS notifications generated by your event Me With Backpack are getting annoying**

[1]Prior to the user study, the Activate and Deactivate buttons were accidentally removed. This was not discovered until the user study had begun and it was too late to change the wording of the question to eliminate "for now."

Table 2: 'Bad' Templates

TemplateName	Template Description
Person With Item	A person is seen with an item.
Person Leave Place	A person seen inside a place and then outside the place
Person Lose/Discard Item	The person is seen with an item and then without it.
Item Near Item	Two Items are seen together.
Person Near People	A person is seen near one or more people.

so you want to change the notification type to Email but keep the same notification message.

- Correct action: select the event from the subscribed events list, edit, copy and paste the message from SMS to email, save as email notification.

The participants were given a brief tutorial on how to use Scenic and then given the list of events to specify and asked to begin. The time it took the participant to create each event was recorded. After each event, the participant was asked to rate how confident they were that the event was correct on a scale of one to five where one meant very not confident and five meant very confident. Once all four events were specified, the participants were given a questionnaire with four questions and the computer was set up with the Notifier program.

The participants were given a brief explanation of Notifier and were shown the different tables of events (subscribed to vs. not subscribed to) and then given the four tasks to complete. The time taken for each task was recorded and after each task the participant was asked to rate how easy the task was on a scale of one to five where one meant very hard and five meant very easy. Once all four tasks were completed, the participant was asked to fill out four questions on the questionnaire and the study was complete.

5.2 User Study Results

The study was conducted on nine different people from within the department, and as such all participants were computer literate with technical backgrounds. All but one had seen Scenic and Notifier demonstrated before, but only two had ever used it before, and then only when it was in an earlier form. Of the nine participants, four were male and five were female. Most participants were in their early to mid twenties.

Three of the participants were instructed to construct the events from scratch. The remaining six participants were instructed to construct events from templates, if possible. Of the six participants using templates, three had only the 'good' templates available and three had only the 'bad' templates available.

The average time for each group for each complexity of task is shown in Figure 12. As we had expected, at least to a small degree, having good templates can, for the most part, make it slightly faster to specify events. The results also highlight the fact that it is important to have good templates available since with bad templates it often took slightly longer for people to specify events. Another observation that can be inferred from the results is that a user's first event may take quite a bit longer than expected just because they have not used the program before, as suggested by the points for tasks of complexity 3 (done first) and of complexity 6.

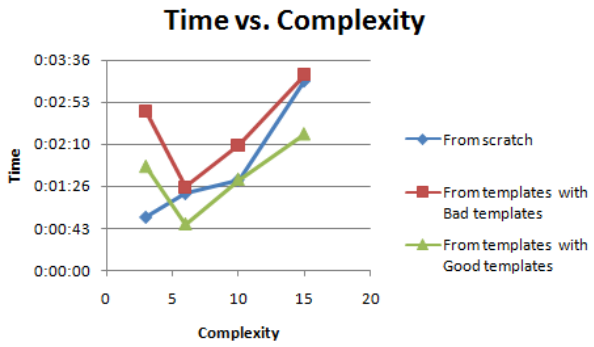

Figure 12: The average time taken by each group for each complexity of task.

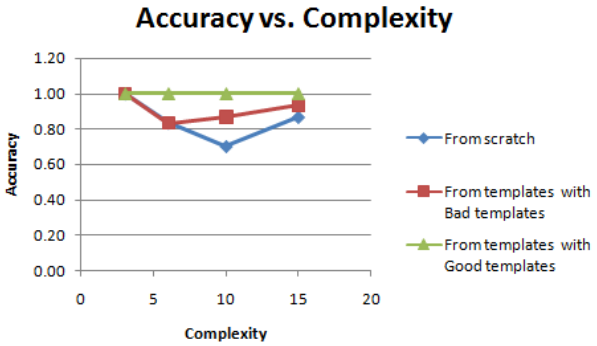

Figure 13: The average accuracy for each group for each complexity of task.

The effect of using templates as complexity increases is not entirely clear, but the trends in Figure 12 suggest that as complexity increases, having good templates even to start from can decrease the amount of time taken.

The average accuracy for each group for each complexity of task is shown in Figure 13. While most participants successfully specified events with 100% accuracy, the results support the hypothesis that having good templates can increase the accuracy of events specified. It is also interesting to note that even having bad templates as examples appears to increase the accuracy slightly.

The effect of having templates as the complexity increases is somewhat unclear, but this may be due to the fact that no templates were provided that directly matched the fourth event to be specified. It may also be due to our method of measuring accuracy where small mistakes make less of a difference as complexity increases. On the other hand, many events that users may be interested in are not terribly complex so having templates may still increase accuracy enough to be worthwhile.

The average confidence for each group for each complexity of task is shown in Figure 14. The results are somewhat

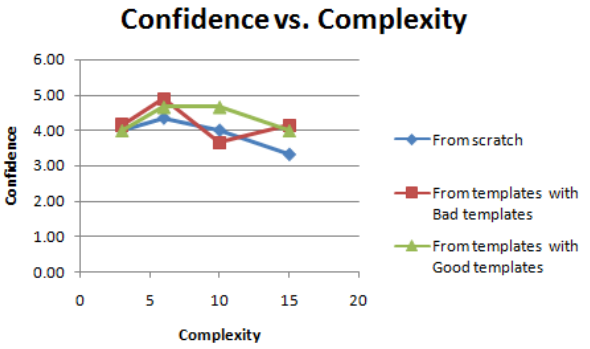

Figure 14: The average confidence for each group for each complexity of task.

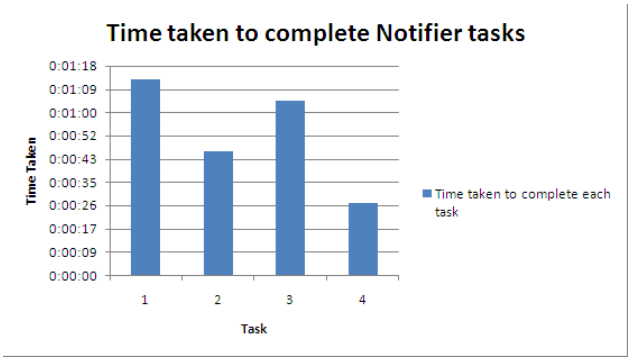

Figure 15: The average time taken to complete each task.

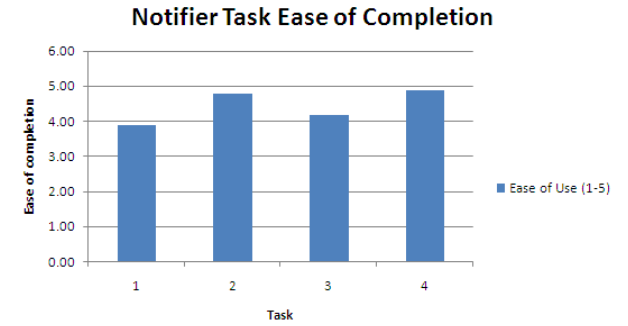

Figure 16: The average ease of completion reported for each task.

mixed, perhaps due to our small sample size. While the confidence that the person reported is also largely influenced by the perceived wording of the questions, it is interesting to note the low confidence which was reported for the first task done. This may be explained by the fact that people had never used the program before so they weren't sure if what they specified was correct. They were much more confident about their second task, but, as expected, their confidence dropped as the complexity of the event increased.

The answers to the questionnaire were also very useful. On average, people would fairly strongly agree (4.44 out of 5) that it was easy to use the iconic language to specify an event. People agreed (4.11 out of 5), on average, that they would be able to specify an event that they wanted to keep track of. Lastly, on average, people agreed (4.11 out of 5) that the design of the GUI was logical and/or intuitive.

Many people also requested the ability to copy and paste both icons and scenes and to rearrange scenes by dragging and dropping. Several expressed some concern about the ambiguity of their specifications. One also pointed out that it would be more logical to go from the table view of templates to select one template and create an event from it directly instead of having to go back to a different place and select it from a drop down list. This last proposed change has already been incorporated into Scenic following the study.

The average time taken by all participants to complete the tasks in Notifier is shown in Figure 15. It is interesting to note that even though tasks 1 and 2 (subscribing to notification for an event) are very similar in difficulty and task 2 was even slightly more complicated, people took much less time on task 2. This suggests that once they started learn the interface it became much faster to use. A number of the participants mentioned that once they finished subscribing to an event, they were confused to not see the event in the table in front of them, making them not realize that they were done.

The fairly large amount of time taken for the third task, which was meant to be fairly simple (delete a subscription) took people a surprisingly long time. A possibly explanation for this, as mentioned by one of the participants was that the wording of the question made her not want to delete the subscription, just to edit it to make it not send notifications for now. However, the participants' responses to the bad wording ended up being even more useful than if the wording had been more accurate because it suggested that if they wanted to only temporarily not recieve notification that they would automatically want to select the notification subscription and edit it. Another reason for the amount of time expended was that some participants didn't realize at first that the subscription had been created previously since they had not done it themselves.

The average amount of time taken for the fourth task suggests that it was relatively easy to edit an event to change the notification type. Again, however, some participants had a little difficulty discerning between the two tables in order to select the subscription to edit.

After each task completed in Notifier, the participants were asked how easy it had been. The averages of the responses for each task are shown in Figure 16. All tasks were reported as being fairly easy, although the first task done was rated as the hardest and the third task which had less than ideal wording was rated down somewhat as well.

The results from the questionnaire were also very useful. On average, participants said that it was fairly easy (4.28 out of 5) to subscribe to notification for an event. Participants also agreed (4.11 out of 5) that the design of the GUI was logical and/or intuitive. It was also fairly encouraging that most people strongly agreed (average of 4.78 out of 5) that they would be interested in receiving notifications for real world RFID events.

There were a number of very useful comments about the design of Notifier. People also said that the similarity in appearance between the two tables of events confused them.

Two suggested that the two tables of subscribed and unsubsrcibed events might be better on the same page. This advice has been incorporated following the study and the two tables were moved to the same page. Other confusion, as mentioned above, came from the fact that there was no way to temporarily stop receiving notification for a subscribed event.

Other observations varied from one participant who said it would have been nice to be able to double click on a subscription in order to edit it to one participant who said that Notifier had too much purple in the interface. A number of participants observed that the concept of a recurring event was not clear and one pointed out that it was more logical to have the opposite where an event happens only once. Another commented that it might be nice to have more than one notification type for the same event.

Overall, the results of the user study were very useful in evaluating the interfaces for both Scenic and Notifier. Many of the comments and observations from participants will result in changes in design in both Scenic and Notifier.

6. CONCLUSION

Scenic is a program designed to allow non-technical users to specify RFID events. It has been extended to incorporate templates in an attempt to make it even easier for users to quickly create accurate events.

Notifier is a program to allow users to subscribe to notification for events that they have specified using Scenic. Once the system has detected the event it will send out the specified messge via the user's choice of either email or SMS text message.

As a part of this project, an informal user study was conducted to assess how easily users can use the programs and to answer a number of questions concerning how the use of templates affects the time taken to specify an event, the accuracy of the event, and how confident the user is that the event is correct.

The results of the study suggest that once a user has had a small amount of practice using Scenic the use of good templates speeds up the process of specifying an event. It is important to have templates that are what the user needs because the prescence of templates that are not quite right ('bad') may actually cause event specification to take longer than if no templates were used at all.

The accuracy of events created using Scenic is highest when users start from good templates, slightly worse when they start from templates that are not quite right, and worst when the user doesn't use any templates. The effect of using templates is highest at a medium to high complexity of event but for simple or very complex events the benefits of templates may not be seen.

Users' confidence in the accuracy of the events they specify is generally slightly higher when using templates, either good or bad, but not by very much. In all cases, the user's confidence in the accuracy of their event goes down as the complexity of the event increases.

The results of the study show that Notifier is relatively easy to use once a user has had an opportunity to use the interface. The results do, however, highlight certain parts of the interface design which could be improved. One such improvement that has already been incorporated was to move the two tables of events to the same page so that users can clearly determine which table they are looking at. Another change made to the design was to replace the option for "This event is recurring" with "This event happens more than once" to be more clear.

Overall the project was considered to be successful because the programs were changed or developed successfully and it has been shows that Scenic and Notifier are relatively easy to use.

Table 3: Templates

Column Name	Data Type	Details
TemplateID	int	Primary Key
TemplateName	nchar(40)	
TemplateDescription	ntext	
Owner	nchar(32)	
XML	ntext	

7. REFERENCES

[1] G. Borriello, W. Brunette, M. Hall, C. Hartung, and C. Tangney. Reminding about tagged objects using passive rfids. In *UbiComp 2004*, 2004.

[2] M. Lamming and D. Bohm. Specs: Another approach to human context and activity sensing research, using tiny peer-to-peer wireless computers. In *UbiComp 2003*, 2003.

[3] T. Sohn, K. Li, G. Lee, I. Smith, J. Scott, and W. Griswold. Place-its: A study of location-based reminders on mobile phones. In *UbiComp 2005*, 2005.

[4] University of Washington. RFID Ecosystem. `http://rfid.cs.washington.edu/`.

[5] E. Welbourne, N. Khoussainova, J. Letchner, Y. Li, M. Balazinska, G. Borriello, and D. Suciu. Cascadia: A system for specifying, detecting, and managing rfid events. In *MobiSys 2008*, 2008.

APPENDIX

A. DATABASE DESIGN FOR SCENIC AND NOTIFIER

A.1 Scenic Tables

Scenic requires two tables for storing and retrieving event and template information as well as a number of other tables for dynamically populating the lists of primitive properties. One of these tables is Scenic specific while the others are system tables.

The Templates table (Table 3) stores the information created in Scenic by saving a template. It stores the name, description, and an intermediate XML representation. Scenic is the only program that uses this table.

A.2 RFID Ecosystem Tables

The Events table (Table 4) stores the information created by saving an event in Scenic. It includes the name, description, PeexL representation, and the intermediate XML representation. This table is populated only by Scenic but is used by other applications such as Notifier.

The rfid_system.users table (Table 5) is a system table of RFID Ecosystem users. Scenic uses it in combination with the rfid_system.friend table (Table 6) to populate the list of people dynamically.

Table 4: Events

Column Name	Data Type	Details
EventID	int	Primary Key
Username	nchar(32)	
EventName	nchar(40)	
EventDescription	ntext	
DateCreated	datetime	
XML	ntext	
PeexL	ntext	

Table 5: rfid_system.users

Column Name	Data Type	Details
csnetid	varchar(50)	Primary Key
name	varchar(50)	
img_url	varchar(256)	
use_ip	bit	

Table 6: rfid_system.friend

Column Name	Data Type	Details
owner	varchar(50)	Foreign Key from rfid_system.users
friend	varchar(50)	Foreign Key from rfid_system.users
gid	int	Foreign Key from rfid_system.grp

Table 7: rfid_system.grp

Column Name	Data Type	Details
gid	int	Primary Key
name	varchar(255)	
owner	varchar(50)	Foreign Key from rfid_system.users

The rfid_system.friend table (Table 6) is a system table which may be used by Rfidder#, a social application under development [4], at some point but is currently manually populated. The owner and friend fields come from the rfid_system.users table (Table 5) and the gid field comes from the rfid_system.grp table (Table 7). At the time of this writing, the rfid_system.friend table (Table 6) was only populated with fake data.

The rfid_system.grp table (Table 7) is a system table not currently used by any other program. Scenic uses it to dynamically populate the list of person groups that can be used to specify events. At the time of this writing, the rfid_system.grp table (Table 7) was only populated with fake data.

The rfid_system.place table (Table 8) is a system table that may eventually be populated by a program such as the one being created by Leilani Battle. Scenic uses it to dynamically populate the list of places that can be used to specify events. At the time of this writing, the rfid_system.place table (8) was only populated with fake data.

The rfid_system.entity table (Table 9) was created as another part of the RFID Ecosystem Project. It is populated by the Kiosk program that links user-created names with RFID tag numbers. Scenic pulls from this table to populate the list of Thing names that a user can specify events with.

A.3 Notifier Tables

Notifier and the corresponding notification module need a number of tables. Notifier pulls information from the Events table (Table 4) so that the user can select events they have previously specified. Both Notifier and the notificiation module use the Notification table and the specific notification method tables that store the message to be delivered.

The UserDetails table (Table 10) stores the contact information collected by Notifier the first time the user logs in. It contains the user's email and phone number. The notification module retrieves this contact information when an event has been detected.

The PhoneProviders table (Table 11) stores the extension provided by the phone provider that allows you to send a textmessage as an email. For example, the extension "@vtext.com" when concatenated with a Verizon user's phone number becomes an email address so that you can send the user a text message with an email program. The table is populated with real data found from searching online. The extensions for AT&T, Sprint, and Verizon have been verified by testing at the time of this writing.

The Notification table (Table 12) stores the information generated by subscribing to notification for an event. It references the Events table but also records the notification type, the probability thresh hold, whether the event is recurring and whether the event is currently active. The table is populated by Notifier and retrieved from by the notification module.

The EmailNotification table (Table 13) stores the message to be sent for email notifications. It references the Notification table and if a notification is deleted, the deletion cascades and deletes the entry in this table. The table is populated by Notifier and retrieved from by the notification module.

The SMSNotification table (Table 14) stores the message to be sent for SMS notifications. It references the Notification table and if a notification is deleted, the deletion cascades and deletes the entry in this table. The table is populated by Notifier and retrieved from by the notification module.

Table 8: rfid_system.place

Column Name	Data Type	Details
pid	int	Primary Key
name	varchar(255)	
owner	varchar(50)	Foreign Key from rfid_system.users

Table 9: rfid_system.entity

Column Name	Data Type	Details
eid	int	Primary Key, Foreign Key from rfid_system.tag
type_id	int	Foreign Key from rfid_system.entity_type
owner	varchar(32)	
name	varchar(128)	
img_url	varchar(256)	

Table 10: UserDetails

Column Name	Data Type	Details
Username	nchar(32)	Primary Key
PhoneNumber	nchar(11)	
Provider	nchar(40)	Foreign Key from PhoneProviders
Email	nchar(30)	

Table 11: PhoneProviders

Column Name	Data Type	Details
ProviderName	nchar(30)	Primary Key
Extension	nchar(30)	

Table 12: Notification

Column Name	Data Type	Details
EventID	int	Primary Key, Foreign Key from Events
NotificationType	nchar(10)	
ThreshHold	float	
Recurring	bit	
Active	bit	

Table 13: EmailNotification

Column Name	Data Type	Details
EventID	int	Primary Key, Foreign Key from Notification
Message	ntext	

Table 14: SMSNotification

Column Name	Data Type	Details
EventID	int	Primary Key, Foreign Key from Notification
Message	ntext	