Real-time Annotation Tool (RAT)

Kyle D. Feuz and Diane J. Cook
School of Electrical Engineering and Computer Science
Washington State University

Abstract

A variety of tools have been developed to bring about the realization of pervasive context-aware systems. However, there are still many challenges faced by such applications when trying to determine the context of who, what, when, where, why and how activities are being performed. One such challenge is obtaining the necessary data to train such context-aware systems. In this paper we present a real-time annotation tool which has been developed to streamline the process of collecting and annotating data in an experimental setting. By decreasing the amount of time needed to collect and annotate data, we can increase the rate at which applications can be developed and trained to recognize the current context and react accordingly. These applications can then deliver relevant information in a timely manner supporting a broad range of tasks.

Introduction

One of the challenges in building an activity recognition system is obtaining labeled data which the system can use for training the activity recognition algorithms. A typical workflow involves first collecting the data and then in a separate phase having the collected data labeled by an expert. The data labeling phase represents a significant bottleneck in processing the data. It takes our trained annotators approximately one hour to process a day's worth of smarthome sensor data when the data is from an unscripted environment (Szewcyzk et al. 2009). When the data is from a scripted environment the annotation task is much more difficult due to the concentrated nature of the tasks and the increased level of detailed in the annotation that is often associated with scripted tasks. Thus annotating pre-scripted data typically takes a trained annotator 30 minutes to annotate one hour's worth of data.

If we can eliminate the additional overhead of annotating the data in a separate phase, the total time required to collect and annotate data would be reduced by as much as 33% for scripted data, based on the above estimates for annotation time. To this end, we have developed the Real-time Annotation Tool (RAT) which allows data to be annotated as it is collected. In this paper we provide a detailed description of the RAT and provide some preliminary results on its effectiveness as an annotation tool.

The RAT is an cross-platform python/pyGTK GUI application which provides the experimenter with a set of buttons to press as specified activities occur in the smart home. In our setup, the experimenter is in a room separate from the experiment but is able to observe the participant through live video feeds. The experimenter can also communicate with the participant through an intercom system. As the experimenter sees activities occurring they only need to click the corresponding button and the activity is automatically annotated in the dataset. This significantly reduces the amount of time and effort required to annotate data by allowing the data to be annotated at the time of collection.

Background

The RAT is designed to operate with the CASAS architecture (See Figure 1). We provide a brief introduction to the architecture here and refer readers to Kusznir's work for more details (Kusznir and Cook 2010; Kusznir 2009). The CASAS architecture is divided into three main layers: the physical layer, the middleware layer, and the application layer. At the physical layer are the hardware components such as motion sensors, door sensors, light switches and other devices. The middleware layer is governed by a publish/subscribe manager and a set of component bridges. The manager provides named channels on which component bridges can publish and receive messages. The middleware also provides time-stamping of events and assigns UUID to each event. Communication flows from the individual components to the manager and from the manager to the individual components via customized XMPP bridges. The application layer consists of application built on top of the middleware. This may include activity recognition algorithms, activity discovery algorithms, prompting systems, and others. The RAT operates in the application layer.

The primary function of the RAT is to provide real-time annotation of the incoming data. In order to accomplish this, the RAT needs to interact with the raw data in the physical layer and the archive storage in the middleware layer. These interactions are done through the middleware and we will discuss the pertinent components here. Figure 2 shows the sensor layout in the testbed. Events from the physical layer are reported on a raw_events channel to the middle-

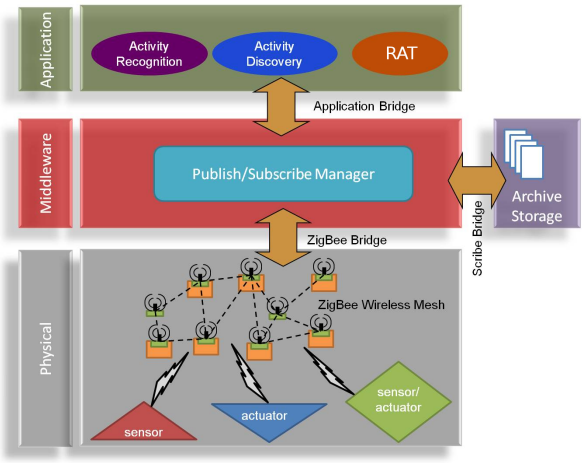

Figure 1: CASAS smart home architecture

ware. These events are time-stamped and assigned a UUID and then uploaded into a storage database. Events can be annotated in two ways and the RAT does both. First, events can be annotated as belonging to a particular experiment and dataset. This allows all relevant events to be associated with an experiment and dataset for easy retrieval. Second, events can be tagged with arbitrary activity labels (e.g., cooking, sleeping, and eating). Real-time event tagging is provided by the middleware through the tag channel. Events can be tagged as belonging to multiple experiments/datasets and can have multiple labels.

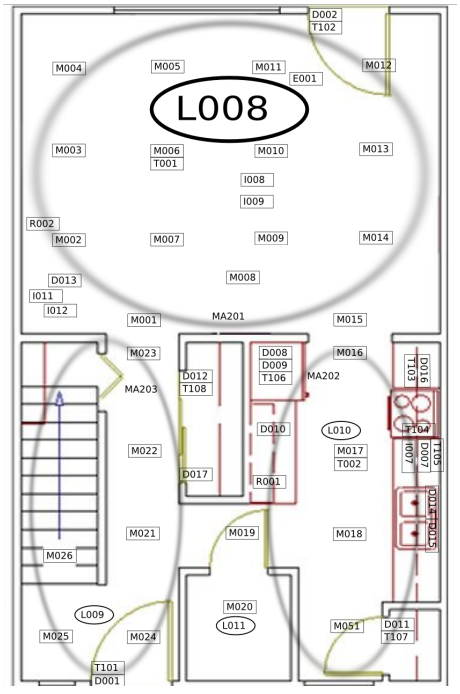

Figure 2: Sensor layout in the smart apartment

Implementation Details

The design of the RAT was heavily influenced by the end users (i.e. the people overseeing the data collection process). An initial prototype was developed and put into the end users' hands. The requirements for this initial prototype are listed in Table 1.

Integrate with CASAS middleware architecture
Record start/stop times of activities
Record unexpected events
Deliver recorded prompts
All relevant time-sensitive buttons accessible without scrolling
Operate in Record Mode or Test Mode
Reusable for multiple experiments
Configurable by non-programmers
Datasets can be annotated multiple times

Table 1: Initial requirements for the RAT

In several sessions, the experimenters then provided feedback on features they felt were missing or should be changed. These features were then integrated into the system. The additional features and requirements which were added to the system are listed in Table 2.

Record specific values or measurements
Record specific values from multiple experts
Record arbitrary additional text information
Provide feedback indicating what has been clicked
Correct or re-annotate mistakes
Allow for multiple experimental conditions/treatments
Allow for randomization in conditions/treatments

Table 2: Additional requirements for the RAT from user feedback

The first requirement, **Integration with the CASAS middleware**, involves a few basic steps. First, the RAT must connect to the middleware using the authentication information provided by the user. Second, the RAT subscribes to incoming events on the raw_events channel. Every incoming event is then tagged as belonging to a specific Experiment and Dataset through the tag channel. Additionally, the RAT publishes events for certain button presses (which then get tagged and labeled) and also publishes commands to deliver prompts. In this way, the RAT is able to tag all incoming events as belonging to a particular experiment and dataset, annotate the dataset using the user button presses, and deliver prompts. We have also implemented an option which eliminates the reliance on the middleware and instead produces a textfile containing all of the events and tags generated by the RAT with a timestamp and identifier. This option naturally does not include the ability to play prompts remotely or to record data external to the RAT itself.

Activity start and stop times are recorded through button presses on the RAT by the experimenter. Each time the button is pressed an event is published on the raw_events channel. This event then gets tagged by the RAT as the start or stop time for that activity. Smaller steps of the activity can also be annotated. However, instead of having the user

press both a start and a stop button for these sub-steps, they just press a single button when the event occurs. This helps relieve the burden on the experimenter by reducing the number of clicks that must be performed.

The RAT can also **record specific values and measurements**. A textfield or drop-down box is displayed to the experimenter which can be used to enter the desired value. Pressing the submit button causes an event to be published on the raw_events channel and to be tagged with the specified value through the tag channel. The RAT also allows for **specific values from multiple experts** to be recorded. To accomplish this multiple fields are shown with the same submit button. Similarly **arbitrary text information** can be entered through a text field and a submit button. This is useful to note **unexpected events** or other important information.

Prompts can be delivered by a button press or a key press. This publishes a prompt message on the control channel with the prompt file to be played. A specific prompt device can be specified or the default prompt device can be used. The prompt device can be specified by the experimenter or the default prompt device can be used.

With so many possible buttons, one issue that must be addressed is how to allow for all of the **time-sensitive buttons to be clicked without scrolling**. This is achieved by having a few standard but versatile layouts built-in to the RAT. For example, a list layout can be used to show activities with many sub-steps (See Figure 3). If not all the activities fit in the list we can dynamically rearrange the order of activities so that the currently relevant activity (or activities) is shown first in the list. Alternatively, a grid layout can be used to show a larger number of activities with fewer sub-steps (see Figure 4).

The RAT **provides feedback on multiple levels** to indicate which buttons have been clicked and what data has been recorded. First, when a button is clicked its color changes to provide immediate feedback that the button has been clicked. Second, all of the events published by the RAT (i.e. every button event) gets recorded and can be viewed in the history window. This window shows the time of the event, the dataset name, the event name and the annotation tag. This history window can be viewed at any time and the data can be saved to a csv file.

The problem with **correcting or re-annotating data** in a real-time system is that once data is published it cannot be unpublished. One possible solution is to publish additional events with the corrections or new annotations. This is the approach taken by the RAT. All of the events and tags published by the RAT are also saved by the RAT and are displayed in the history window (see Figure 5). Double-clicking on any event in the history window opens up a dialog display which allows the experimenter to select the type of correction and enter any additional notes. Currently there are four types of corrections available: Ignore, Timing, New Value, or Other. The "ignore" correction is used to mark an event as invalid. The reason this event is to be ignored can be specified in the notes section. The "timing" correction is used to specify a new timestamp for the event. The new timestamp is entered in the notes section. The "new value" correction is used to specify a new value to record which is given priority over the old value. The new value is entered in the notes section. Last, the "other" correction is used for any other correction that needs to be noted.

Time	PID	Event	Tag	Value
2013-04-02 14:33:27.17	PSTEST	[6 Activities Prompting:3][Cooking:5]	Cooking	START
2013-04-02 14:33:27.87	PSTEST	[6 Activities Prompting:3][Cooking:5][Retrie	Retrieve BOWL Subs	0
2013-04-02 14:33:28.88	PSTEST	[6 Activities Prompting:3][Cooking:5][Retrie	Retrieve GLASS Subs	0
2013-04-02 14:33:29.53	PSTEST	[6 Activities Prompting:3][Cooking:5][Retrie	Retrieve MEASURING (	0
2013-04-02 14:33:30.97	PSTEST	[6 Activities Prompting:3][Cooking:5][Retrie	Retrieve UTENSIL	
2013-04-02 14:33:32.36	PSTEST	[6 Activities Prompting:3][Cooking:5][Fills M	Fills MEASURING CUP	0
2013-04-02 14:33:33.32	PSTEST	[6 Activities Prompting:3][Cooking:5][Microv	Microwaves water in M	0
2013-04-02 14:33:33.92	PSTEST	[6 Activities Prompting:3][Cooking:5][Add w	Add water to BOWL	
2013-04-02 14:33:34.67	PSTEST	[6 Activities Prompting:3][Cooking:5][Returr	Return MEASURING CU	0
2013-04-02 14:33:37.84	PSTEST	[6 Activities Prompting:3][Cooking:5][Retrie	Retrieve Britta filter	
2013-04-02 14:33:38.54	PSTEST	[6 Activities Prompting:3][Cooking:5][Fills gl	Fills glass with filtered	0
2013-04-02 14:33:39.43	PSTEST	[6 Activities Prompting:3][Cooking:5][Returr	Return filtered water	0
2013-04-02 14:33:39.94	PSTEST	[6 Activities Prompting:3][Cooking:5][Bring t	Bring bowl to dining ro	0
2013-04-02 14:33:40.83	PSTEST	[6 Activities Prompting:3][Cooking:5][Bring g	Bring glass to dining ro	0
2013-04-02 14:33:41.33	PSTEST	[6 Activities Prompting:3][Cooking:5][Succes	Successful completion	Yes

Figure 5: RAT example history screenshot

The RAT is designed to have **two different modes of operation: an actual record mode and a test mode**. In the record mode events are tagged as belonging to the specified experiment and dataset. In the test mode events are tagged as belonging to the specified experiment and the test dataset. This allows the functionality of the RAT to be verified without requiring the creation of new datasets. The RAT can be set to enter test mode explicitly but we also put the RAT in test mode any time the current dataset is not started. When the RAT is in this implicit test mode we provide strong visual feedback (bright red buttons, no running RAT icon, and a big green start button) to the experimenter to remind him/her to start the dataset before actual data collection begins. This allows the experimenter to have a new dataset ready to go but still verify the functionality of RAT without recording irrelevant data to the dataset.

The RAT allows a **dataset to be annotated multiple times**. This feature is important for at least two different reasons. First, an experiment could be unexpectedly interrupted. For example, the hardware or software could fail or something could go wrong with the experimental procedure. In these cases it is important that the same dataset can be re-opened for recording and annotation. The second reason for allowing a dataset to be opened multiple times for annotation is to allow the experimenter to add additional time-insensitive information to a dataset. An experimenter may want to go back and record additional notes or values from an experiment. In order to do so, the experimenter must be able to reopen a dataset for annotation.

Annotating an existing dataset requires a few additional constraints. A dataset is not simply a locally stored file that can be created or saved. Instead it is a set of events that have been tagged in the database with a dataset name. We need to distinguish between new and existing datasets and provide reasonable constraints to prevent accidental opening of existing datasets. First, the RAT should only be used on a single computer to prevent name clashes resulting from race conditions (i.e. both RATs check that the name has not been used, the name is free so both RATs use it. Now two separate datasets are being recorded as a single dataset. This

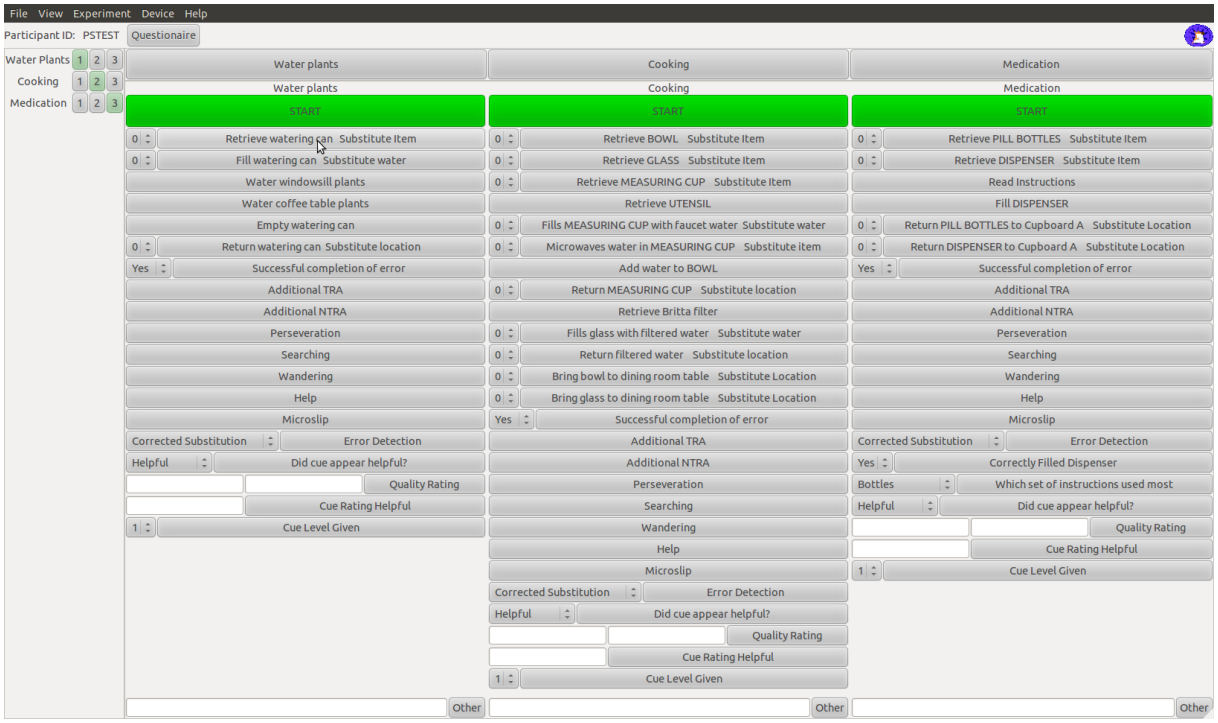

Figure 3: RAT example list view screenshot

constraint could be resolved through naming conventions). Second, the RAT tests to ensure that no other instance of the RAT is running on this computer (again to prevent race conditions). Third, we separate creating a new dataset from opening an existing dataset. If an experimenter attempts to create a new dataset that already exists they are shown an error message and must enter a new dataset name.

When opening a dataset it is important that the RAT be configured properly for that dataset. The RAT allows for **multiple experimental conditions** (e.g. prompting vs no prompting). Additionally, the **conditions may have a random component** (e.g. randomize the order of activities performed). The layout and functionality of the RAT is affected by the experimental condition which is applied to a dataset. For example, one condition might do activities 1,2 and 3, while the other condition only does activity 3. Only the relevant activities should be shown for that condition. Therefore, when re-opening an existing dataset, the condition must be identical to the original condition for that dataset. When conditions are predetermined this is a simple matter of recording what condition has been applied. However, when a condition includes a randomized component a little more effort is required. We solve the randomization problem by using two random number generators. The first generator is seeded using the standard seed (i.e. system time). This random number generator then generates a random number which is used as the seed value for the second random number generator. All of the randomization for a condition is then done using this second random number generator with a known seed value. We save this seed so that the same randomized sequence can then be recreated when needed.

Lastly, the RAT must be **reusable for multiple experiments** and should be **configurable by non-programmers**. This is achieved through an xml configuration file which can be understood and edited without extensive programming experience. The details of the xml specification are given in the next section.

XML specification

The content displayed in the RAT needs to be configurable even by non-programmers. To achieve this, almost all of the functionality of the RAT is configured via an xml file. Table 3 lists the defined xml elements and attributes. Table 4 lists the allowed sub-elements for each element. A study is composed of Devices, Prompts, Experiments, and Conditions. Devices are either atomic or composed of other devices if the root device is a meta-device. Prompts are also atomic. Experiments are composed of Prompts, Questionnaires, and Tasks. Both Questionnaires and Tasks are composed of Events and Prompts. An event is either atomic or composed of a prompt. Conditions are composed of Samples and Experiments. Rather than defining a full experiment in condition, the id of an existing experiment can be used. Similarly, the tasks and prompts composing that experiment can be specified with an id and ordering. The level of prompt is also specified in the conditions. Listing 1 shows a simple xml configuration example. Using this customizable xml configuration file the RAT can be used for multiple experiments and can be configured by the experiment with-

Element	Attribute	Default Value	Description
Study			The root tag
	name	“”	Displayed in the title bar, Experiment name in the database.
Prompt			Defines a prompt which can be delivered by the RAT
	name	“”	Displayed by the prompt button
	id	-1	Unique identifier
	file	“”	Filename of the prompt
	default_device		Default device id on which the prompt should be played
	level	0	Set the prompt level (Should be set as part of a “Condition”)
Device			Defines a device on which a prompt can be played
	name	“”	Displayed in the device menu
	id	-1	Unique identifier
	serial	“”	Serial number used for the middleware
	location	“”	Location used for the middleware
	meta_device	False	Allows a device to represent multiple devices
Condition			Defines a condition/treatment to be applied
	name	“”	Displayed in the condition menu
	id	-1	Unique identifier
Sample			Defines a sampling method
	id	-1	Unique identifier
	choice_list	“”	List of possible samples
	replace	True	Specify with or without replacement
Experiment			Defines the experiment to be conducted
	name	“”	Displayed in the experiment menu
	id	-1	Unique identifier
	display_mode	list	Specifies the layout of the tasks (list/grid/grouped).
	columns	1	the number of columns in the grid
	start	False	Specify display of experiment start/stop button
Task			Defines a task to be performed
	name	“”	Displayed as the task label
	id	-1	Unique identifier
	desc	“”	More detailed description shown as a tooltip
	group	1	Specify the task group (Used for grouped display_mode)
	start	True	Specify display of task start/stop button
	order	“”	Specify the display order (Should be set as part of a “Condition”)
	level	“”	Specify the prompt level for this task (Should be set as part of a “Condition”)
Event			Defines a specific event
	name	“”	Displayed as the button label
	id	-1	Unique identifier
	desc	“”	More detailed description shown as a tooltip
	type	button	Specify the display type (button/field/choice)
	choice_list	“”	Specify the possible choices (used with type=choice)
	inputs	1	Specify the number of inputs (used with type=choice or field)
Questionnaire			Defines a questionnaire (special type of task)
	name	“”	Displayed in the questionnaire title bar
	id	-1	Unique identifier

Table 3: RAT XML Elements and Attributes

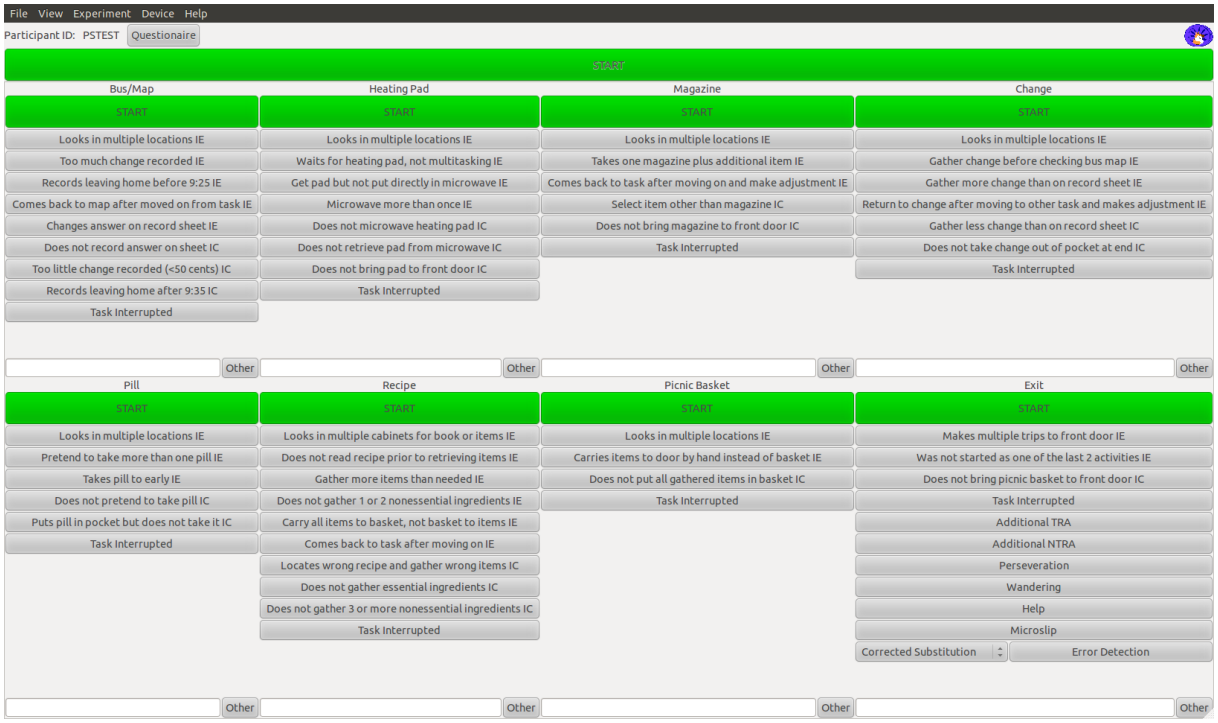

Figure 4: RAT example grid view screenshot

out requiring any programming changes to the RAT itself.

Element	Sub-elements
Study	Device Prompt Experiment Condition
Device	Device
Prompt	
Experiments	Prompts Questionnaire Task
Questionnaire	Prompt Event
Task	Prompt Event
Condition	Experiment Sample
Event	Prompt

Table 4: RAT XML Elements and Attributes

Usability Survey

The RAT has been used to collect and annotate data from multiple sets of experiments in the smart apartment. Describing the full experimental setup for each situation is outside of the scope of this paper, however, we do provide a brief description of one of the experiments to better illustrate the purpose and use of the RAT.

The smart apartment is equipped with a variety of motion sensors, door sensors, item sensors, and other sensor. Participants are brought into the apartment and asked to perform three household tasks out of six possible task including, water plants, take medication, wash countertop, sweeping and dusting, cooking, and handwashing. Which three task are selected and the order of the tasks are selected randomly. During the task, a prompt may be played to help the participant complete the task. This prompt can be an indirect audio prompt, a direct audio prompt or a direct audio/video prompt. The type of prompt that is given for each activity is dependent upon the experimental condition being applied to that participant. The RAT only displays the three activities which are to be performed under the given experimental condition and highlights the prompt button which should be played for this condition. The experimenter remains in a separate room from the participant and observes the participant via live video feeds. At the same time they use the RAT to annotate the data and to deliver the prompts. Figure 6 shows an experiment in progress. The psychologists are interested in this experiment to analyze how different clinical conditions affect the performance of everyday activities while the computer scientists are interested in gathering data for activity recognition, detecting prompt situations, and predicting clinical conditions from behavioral data.

To quantify the usability of the RAT we asked the experimenters conducting the above experiment to complete a short survey about the RAT. Five experimenters completed the survey. With one exception, experimenters completing the survey were Psychology graduate and undergraduate students with little to no programming experience and no previous experience using the RAT prior to conducting the ex-

Listing 1: Example XML Configuration File

```
<?xml version="1.0"?>
<Study name="Transition Prompting" >
  <Condition name="A" id="0">
    <Sample id="1" choice_list="2,3,4,5,6,7,8,9,10,11,12,13"
     replace="False" />
    <Experiment id="2">
      <Prompt id="1" order="1" level="1" />
      <Task id="1" order="*1" />
      <Task id="5" order="*1" />
      <Task id="8" order="*1" />
      <Task id="13" order="1" />
    </Experiment>
  </Condition>
  <Condition name="B" id="1">
    <Experiment id="2">
      <Prompt id="1" order="1" level="1" />
      <Task id="1" order="2" />
      <Task id="5" order="3" />
      <Task id="8" order="4" />
      <Task id="13" order="1" />
    </Experiment>
  </Condition>
  <Device name="_DMN" serial="" location="dmn" id="1"/>
  <Device name="_Tablet" id="2" meta_device="True">
    <Device name="_Default" id="20" />
    <Device name="_Kitchen-tablet" serial=""
    location="kitchen-tablet" id="22" />
    <Device name="_Livingroom-tablet" serial=""
    location="livingroom-tablet" id="23" />
  </Device>
  <Prompt name="Prompt" id="1" file="7" default_device="1"/>
  <Experiment name="Transition" id="1" >
    <Task name="Magazine" id="1" />
    <Task name="T.V." id="2" />
    <Task name="Puzzle" id="3" />
  </Experiment>
  <Experiment name="Prompting" id="2" >
    <Task name="Magazine" id="1" >
      <Prompt id="1" />
    </Task>
    <Task name="Copy Recipe" id="5" />
    <Task name="Closet Items" id="8" />
    <Task name="Memory Notebook" id="13" >
      <Event name="Used Voice" id="1" />
      <Event name="Used Pen" id="2" />
      <Event name="Typed" id="3" />
    </Task>
  </Experiment>
</Study>
```

periments in the smart apartment. The one exception is the lab manager who had experience using the RAT because she helped write the xml configuration file used in the experiment described above.

The survey consists of several questions requiring a response on a ten-point scale. A few of the questions were more open-ended (e.g., If you were designing this software, how would you change it so that it worked better for you?). All of the quantitative questions and the average responses are shown in Table 5.

Overall the results are positive. The question receiving the highest response is "How often did system errors occur?" indicating the low occurrence of system errors. Other questions receiving high responses include, "Does the arrangement of information ... make sense?" and "How quickly did you learn to use the system?" This is also reflected in the responses to the open ended questions. For example, when asked "What did you like about the software you used? Were there any features that you found particularly useful? Why?", multiple responses indicated the ease-of-use and the organization with one respondent saying "The interface is easy to use. Just click a button and go."

Quantitatively, users generally indicated that the RAT is

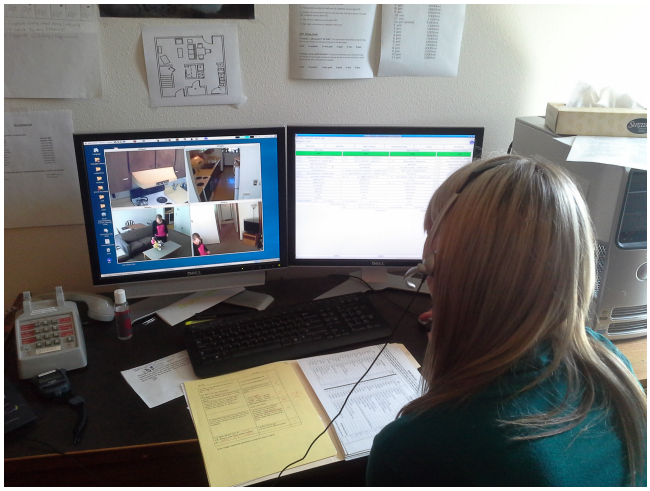

Figure 6: Experimenter using the RAT

easier to use than other methods of data collection they had used in the past. When asked "How does the RAT compare to previous methods of data collection and annotation that you have used?" the average rating is an 8.6. More detail about the RAT's ease-of-use as compared to other data collection methods is seen in the open-ended responses to the following question: "In what ways (if any) is using the RAT easier than previous data collection and annotation methods you have used?" Users indicate such features as "No transcribing involved" and the elimination of repetitive data entry, "Saves time because we don't have to re-enter this information into the database."

Of course, the survey also helped bring to light some issues encountered by the experimenters while using the RAT. The question receiving the lowest response is "How easy was it to explore features of the application by trial and error?" This may indicate that the test mode is not properly understood or is not being adequately utilized. Alternatively, it may be due to the nature of working with real-time data and the inability to easily correct mistakes.

The inability to easily correct mistakes also stood out in the qualitative responses. When asked "What did you not like about the software you used? Were there any features that gave you particular grief? Why?" Several respondents indicated difficulty fixing errors. As one respondent stated, "It's difficult to fix errors or change incorrect items, especially when time-stamped." Similarly, when asked, "In what ways (if any) is using the RAT harder than previous data collection and annotation methods you have used?" one respondent said, "Writing things down is a bit more difficult, but it allows you to fix errors, change time-stamps, and specify unusual events with more ease." Before issuing this survey we were unaware that correcting errors with timestamps was so problematic for the experimenters. We have since started looking at possible solutions to decrease the burden required to correct an error.

Question	Response (Average)
On a scale of 1-10 (1 terrible, 10 wonderful) what was your overall impression of the application?	8.4
On a scale of 1-10 (1 difficult, 10 easy) how easy was it to successfully use the application?	9.2
On a scale of 1-10 (1 frustrating, 10 satisfying) how enjoyable was it to use the application?	8.8
On a scale of 1-10 (1 never, 10 always) how helpful were the screen layouts?	8.6
On a scale of 1-10 (1 harder, 10 easier) how does the RAT compare to previous methods of data collection and annotation that you have used?	8.6
On a scale of 1-10 (1 inadequate, 10 adequate) was there a sufficient amount of information on each screen?	9.2
On a scale of 1-10 (1 illogical, 10 logical) Does the arrangement of information on each screen make sense?	9.4
On a scale of 1-10 (1 difficult, 10 easy) how easy was it to learn how to use the system?	8.8
On a scale of 1-10 (1 slow, 10 fast) how quickly did you learn how to use the system?	9.4
On a scale of 1-10 (1 discouraging, 10 encouraging) how easy was it to explore features of the application by trial and error?	7.4
On a scale of 1-10 (1 undependable, 10 dependable) how did you feel about the reliability of the system?	8.5
On a scale of 1-10 (1 frequently, 10 seldom) how often did system errors occur?	9.6
On a scale of 1-10 (1 with difficulty, 10 easily) how well do you think you can accomplish tasks?	9.4

Table 5: Quantitative survey questions and results

Discussion

We measure the effectiveness of the RAT strictly in terms of the amount of time saved during the data collection and annotation process. In the future we will also measure the effectiveness of the RAT in terms of the inter-rater reliability (Armstrong et al. 1997) and the accuracy of the online annotations obtained via the RAT vs. offline annotations made via the video recordings.

Ideally, the RAT would completely eliminate the annotation processing time required after the data has been collected leading to a 33% reduction in the total amount of time required to collect and annotate the data. In practice, the RAT does not achieve this goal due to operator errors which occur during the collection phase. These errors are noted by the experimenter using the RAT, but cannot typically be corrected automatically. Instead, the annotations must be scanned for errors and unexpected events before the final annotations are generated. Anecdotally, the number of errors has been low, usually less than four per hour of data collection. Processing these errors is a matter of a simple string search to find the error and then editing the file with the manual correction. On average, the total post-processing time is less than five minutes per hour of collected data. This represents a significant reduction in post-processing time as compared to the previous post-processing time of 30 minutes per hour of collected data.

Although the initial goal of the RAT was to provide annotated behavioral data in real-time, the RAT can also be used to created coded data. For example, most of the data we collect from experiments run in the smart apartment is used both by computer scientists for activity recognition and discovery, and also by clinical psychologists for studying certain clinical conditions such as Alzheimer's dementia or Parkinson's disease. In the past, the coding of the data for the psychologists was done separately from the annotating of data for the computer scientists. The RAT can actually be used to unify these two tasks. The RAT allows for all the necessary information for coding the data to be collected while simultaneously annotating the data for the computer scientists. Note that this eliminates the need of having an additional experimenter present just to run the RAT.

Collecting data, annotating data and simultaneously coding data leads to additional time-savings. In the past, coding scripted data took an estimated 5-10 min per hour of collected data. Each segment of the experiment had to be assigned to an expert trained in coding that specific data as it is too much to expect a single expert to learn all of the coding rules for the entire experiment. With the RAT all of this coding can be done automatically.

Coding data automatically is a multi-step process. First, the storage database is queried to extract the relevant events and tags. Then this data is processed by a script which extracts the desired features and outputs them in a csv file which can then be imported into excel or a standard statistical package such as SPSS. Examples of features which are extracted include counts of events, the order of event sequences, time between events, and ratings entered by the experimenter. All of these features are easily calculated from the timestamped and annotated data provided by the RAT. We are currently designing methods to automate this extraction and conversion process.

Conclusion

Knowing the who, what why, where, when and how of a situation is vital to providing relevant context-based applications. Unfortunately, developing such systems is a resource intensive process both in terms of time and money. Collecting and annotating data which can later be used when training context-aware applications is one area that consumes these resources. We have developed a real-time annotation tool which facilitates the data collection and annotation process by allowing data to be collected and simultaneously annotated by an expert.

The RAT meets many key requirements which make it an ideal tool for use in the data collection process. It is versatile and can be used for a variety of experimental setups and can be configured without extensive programming experience. The interface has been designed and influenced by user feedback and is well-tailored to the experimenter's needs. Survey results indicate that the RAT is user-friendly, intuitive to

learn and use and represents a significant improvement over previous data collection and annotation methods.

Using the RAT has resulted in a significant decrease in the amount of time required to collect and annotate scripted data. Previously, for every hour of spent collecting scripted data approximately thirty minutes of additional time was necessary to annotate the data. The RAT reduces that time significantly by allowing most of the annotation process to occur during data collection and then only a small amount of time (usually five minutes or less per hour of scripted data) is required to clean up any errors which have been noted by the experimenter.

We plan to continue to improve the RAT to better meet experimenters needs and plan to release an open source version in the near future. This will allow the wider community to enjoy the benefits of the RAT as well as contribute additional improvements. The RAT will be hosted at casas.wsu.edu and will be available for non-commercial use. Building reliable, pervasive context-aware applications requires a broad range of support tools and system architectures. The RAT contributes to this larger goal and will assist in making context-aware applications an everyday convenience to users everywhere.

References

Armstrong, D.; Gosling, A.; Weinman, J.; and Marteau, T. 1997. The place of inter-rater reliability in qualitative research: an empirical study. *Sociology* 31(3):597–606.

Kusznir, J., and Cook, D. J. 2010. Designing lightweight software architectures for smart environments. In *Intelligent Environments (IE), 2010 Sixth International Conference on*, 220–224. IEEE.

Kusznir, J. 2009. Clm as a smart home middleware. Master's thesis, Washington State University.

Szewcyzk, S.; Dwan, K.; Minor, B.; Swedlove, B.; and Cook, D. 2009. Annotating smart environment sensor data for activity learning. *Technology and Health Care* 17(3):161–169.