A Demonstration of ScenicPeex Through a Digital Diary Application

Nodira Khoussainova Evan Welbourne Magdalena Balazinska
Gaetano Borriello Julie Letchner Christopher Ré Dan Suciu
Jordan Walke

Department of Computer Science and Engineering
University of Washington, Seattle
E-mail: {nodira,evan,magda,gaetano,letchner,chrisre,suciu,jwalke}@cs.washington.edu

Abstract

ScenicPeex is a system that provides RFID-based pervasive computing applications with an infrastructure for specifying, extracting and managing meaningful high-level events from raw RFID data. Scenic allows users to specify events of interest using a graphical interface and an intuitive visual language whereas PEEX effectively extracts these events from data in spite of the unreliability of RFID technology and the inherent ambiguity in event extraction.

We demonstrate ScenicPeex's technique through a digital diary application in the form of a calendar. ScenicPeex automatically populates the calendar with meaningful events for the user. We use data collected in a building-wide RFID deployment.

1. Introduction

Many mobile and pervasive applications rely on Radio Frequency Identification (RFID) infrastructure to discover real world events (*e.g.* Elise started a meeting with Bill at 1.06pm in room 435). RFID is an attractive technology for this purpose due to the low cost of RFID tags [6], thus allowing applications to track large numbers of objects and people. However, developing such an application is challenging for two main reasons. First, RFID technology is unreliable due to duplicate readings [3] and missed readings [1, 3]. Second, events in the real world are inherently ambiguous. For example, detecting Elise and Bill at the same location can correspond to one of many events (*e.g.* a meeting, coffee or lunch event.)

In this paper, we present a demonstration of ScenicPeex. ScenicPeex consists of two components. The first, Scenic [5], is a a tool that allows end users to graphically specify spatio-temporal

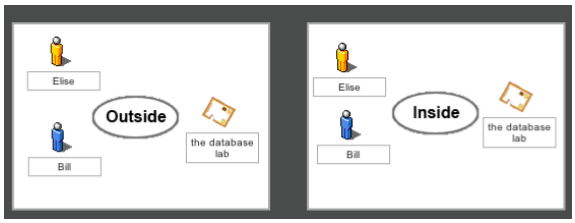

Figure 1. **A `Meeting` event specification in Scenic**

event definitions using an intuitive iconic language and storyboard metaphor. Figure 1 illustrates an event specified in Scenic. Scenic translates these graphical event definitions into a SQL-like language, called PeexL. The second component is the **P**robabilistic **E**vent **EX**traction system [4] (PEEX). PEEX is an RFID data management system that enables applications to **(1)** declaratively specify events in PeexL, **(2)** continuously and automatically extract these events from RFID data streams, and **(3)** store these events persistently and manage them. PEEX is based on a probabilistic approach to event extraction that allows it to effectively detect complex events in spite of RFID data unreliability and ambiguity.

We demonstrate ScenicPeex using Digital Diary, an automatic-calendar application. The demonstration is designed to give users a good understanding of the challenges of detecting real-world events from RFID data and the details of the ScenicPeex approach. In the demonstrations, user specify events in Scenic (*e.g.* entering rooms, meeting with other people, etc.) or view and edit previously specified events. Scenic generates the matching PeexL definitions and passes them to PEEX. The latter runs these event definitions over real traces obtained from a 150-antenna deployment in the CS Dept. at the University of Washington [5]. Using a large set of real traces allows the specification and detection of a larger set of interesting events. The detected events automatically populate a Web calendar built using the Google API [2]. Fig 2 shows an instance of the automatically populated calendar.

To explain the internals of ScenicPeex, the demonstration also includes an animated view of how Scenic events are translated into PeexL, how PEEX detects events as it processes the traces, and how it calculates the appropriate probabilities. To help the user understand the impact of various parameters on event extraction (e.g. probabilistic vs. deterministic event extraction), the application provides different calendars of events, one for each of different system configurations.

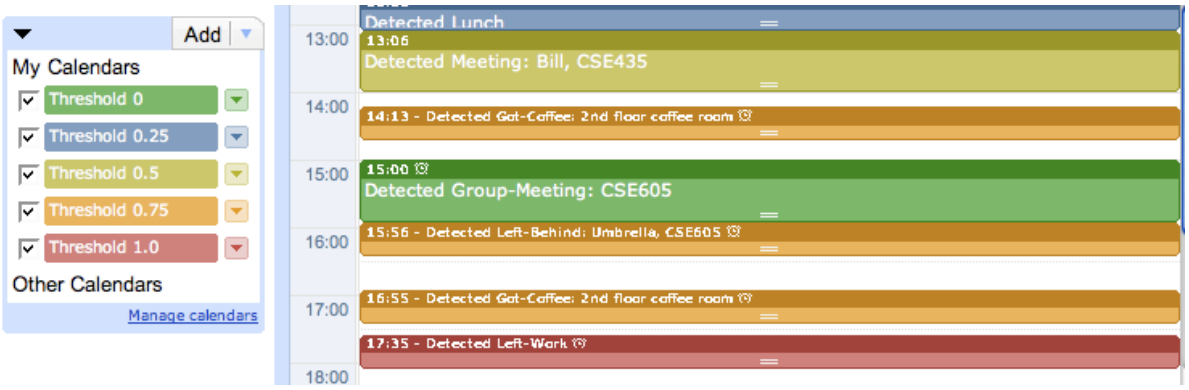

Figure 2. **Screenshot of the digital diary application.**

2. ScenicPeex System Overview

We give a brief overview of Scenic and PEEX, the two main components of ScenicPeex.

2.1 Scenic

Scenic provides the channel through which non-expert users can create and submit new event definitions to PEEX. It does so using an iconic visual language. The language supports three basic constructs: **(1) Scenes** represent point events in a sequence and are displayed as white panels over the grey sequence panel. **(2) Actors** represent five types of entities in an event each represented with a separate icon: person, group of people, object, group of objects, and place. **(3) Primitives** directly represent primitive events including with, without, inside, outside, near, far, and lasts. Figure 1 illustrates the specification of a meeting event in the database lab between Elise and Bill. From these graphical specifications, Scenic generates PeexL event definitions described next.

2.2 Events

PEEX supports probabilistic events represented as tuples and stored in relations named for each event type. The most important relation is `SIGHTING`, which has the following schema: `SIGHTING`(time, tagID, antID, prob). An example tuple in `SIGHTING` is (101, 10, 23, 1.0), which represents that at time 101, the tag with id 10 was seen by antenna 23. All tuples in `SIGHTING` are deterministic (*i.e.* have probability 1), because a tuple records the fact that the system is aware of this sensor reading.

An example of a higher-level event is a `MEETING` event with schema: `MEETING`(time, person1, person2, room, prob). An example tuple in `MEETING` is (103, 'Elise', 'Bill', 435, 0.4), which represents that at time 103, PEEX believes that Elise and Bill are having a meeting in 435 with probability 0.4.

Event Definition Language. To build high-level events from primitive `SIGHTINGS` events, PEEX uses, PeexL, a SQL-based *event definition language*. As an example, the event from Figure 1 is written in PeexL as:

```
FORALL Sightings S1, Sightings S2, Sightings S3,
Sightings S4, AntOutside A1, AntInside A2
CTABLE CMeeting C
WHERE  SEQ(AND(S1, S2), AND(S3, S4)) AND
 S1.ant = A1.ant AND S3.ant = A2.ant AND
 S2.ant = S1.ant AND S4.ant = S3.ant AND
 A1.room = A2.room AND
 S1.tag = 'Elise's tag' AND S2.tag = 'Bill's tag' AND
 S3.tag = S1.tag AND S4.tag = S2.tag AND
 S1.time is within 10 seconds of S2.time AND
 S3.time is within 10 seconds of S4.time AND
 C.person1 = S1.tag AND C.person2 = S2.tag AND
 C.room = A1.room
CREATE EVENT EliseBobMeeting E
SET E.room = C.room
```

Intuitively, the event definition says that if we see Elise and Bill at an antenna *outside* the database lab and then at an antenna *inside* the lab, there is some probability that Elise and Bill are having a meeting in the lab. `CMeeting` is a *confidence table* which stores the probability that two people are having a meeting given that they enter the same room. Importantly, confidence tables allow the probabilities assigned to an event to depend on the values used to trigger an event. For example, whether two people have a meeting when entering the same room can depend on who the two people are and which room they are entering. Perhaps the probability is higher if the two people are both database students and are entering the database lab. PEEX is able to populate confidence tables with probabilities that are learned from historical data.

2.3 PEEX System Architecture

We have designed PEEX as a layer on top of a traditional RDBMS (Microsoft SQL Server in our prototype). Fig. 3 illustrates the three major components of PEEX: the event detector, the confidence learner and the partial events generator.

Event detector. All events are stored as relations in the RDBMS. When a primitive event arrives, *e.g.* Bill is sighted at antenna 10, a corresponding tuple is appended to the `SIGHTINGS` relation. The *event detector* component periodically checks if any events have occurred. The event detector performs this check by rewriting event definitions into standard SQL queries and then executing these queries on the database. The result of these queries is a set of tuples (that represent newly detected events) which are then inserted into their corresponding tables in the database.

Confidence learner. The inherent ambiguity in complex event detection makes detecting events with certainty a difficult and sometimes even impossible task. The ambiguity arises because a combination of low-level tag reads may not correspond uniquely to a single high-level event. *e.g.* if Elise and Bill are sighted in the same room, they could be doing one of many activities such as having a meeting, leaving for lunch or even just running into one another. To handle the ambiguity, PEEX uses *confidence tables* to capture the historical probability that different combinations of tag sightings correspond to a high-level event

The *confidence learner* automatically populates the confidence tables, from annotated historical data that includes input primitive events and output composite events. We refer the reader to [4] for more details on the confidence learner.

Partial events generator. RFID errors dramatically impact applications that rely on complex events, because error rates are

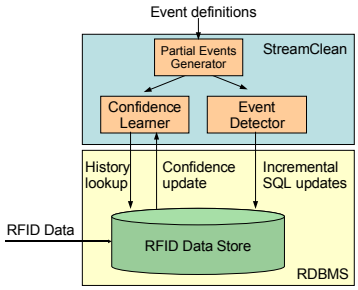

Figure 3. PEEX Architecture.

amplified at each level in the event hierarchy. To illustrate, consider the earlier `EliseBobMeeting` event, which depends on four `SIGHTINGS` events: `S1, S2, S3` and `S4`. If each `SIGHTINGS` has an error rate of 15%, the meeting event has an error rate of 48%. If three meeting events are now needed to detect an even higher-level event, that event has an error rate of 86%! Deterministic event detection is thus unworkable in an error-prone environment.

We observe, however, that it is often possible to detect composite events even when some errors occur. PEEX captures this intuition through the use of *partial events*. Given a definition of a composite event consisting of n lower-level events, PEEX detects the composite event as soon as some non-empty subset of the n events occur. For example, if only `S2, S3` and `S4` are detected, PEEX still detects the `EliseBobMeeting` event although with lower confidence. The *partial events generator* helps implement this technique. Given a composite event definition, it generates the event definitions for all non-empty subsets of the n lower-level events. From there on, these events are handled like regular events, each with its own confidence table.

3. Demonstration Content

We demonstrate a digital diary application that uses ScenicPeex to detect higher level events. This application allows a user to specify events of interest using Scenic and then uses PEEX to automatically populate the user's calendar with extracted events. Such a calendar serves as a digital diary that automatically records the events that occurred during a user's day. The user can then examine these events.

The goal of the demonstration is twofold: **(1)** We'd like to provide the user with an insight into how ScenicPeex works. **(2)** We'd like to show the challenges associated with extracting events from RFID data and the impact of various PEEX parameters on the event extraction process. We now outline the user experience step-by-step and describe how each step addresses our goals.

1. Event Specification. First, we allow the user to specify events using the Scenic tool. This step is fairly straightforward and will allow the user to define an event of interest to them. An example of an event specification in Scenic is shown in Fig 1. Users will also be able to view and edit pre-defined events.

2. Translation into PeexL. In this step, the user sees the event they specified in Scenic translated into PEEX's declarative event language, PeexL. Seeing the translation will help the user gain a better understanding of PeexL and evaluate the difficulty of writing an event definition in PeexL in comparison to specifying it graphically using Scenic. A PeexL definition for the event from Figure 1 is shown in Section 2. To clarify the translation process, the demonstration will animate the different steps of the translation.

3. Translation into SQL. This step is crucial to understanding how PEEX works. PEEX generates two pieces of SQL code for each event definition: **(1) Event detection SQL.** This SQL code is executed by the RDBMS and actually extracts events. The primary benefit of seeing the SQL statements, is that it helps in understanding how PEEX computes the probabilities of detected events. **(2) Confidence SQL.** This is for automatically populating the confidence table. It mostly helps the user appreciate that PEEX handles the probabilistic aspects of event extraction and thus manages ambiguity, without exposing the difficulties to the user.

4. Automatic Diary Population. In this step, the user can evaluate the overall event detection process by seeing how ScenicPeex populates the diary with events that the user defined in step 1. Fig 2 shows a screenshot of the application with some detected events. To further illustrate the event detection process, the demonstration will animate the steps taken by PEEX as new SIGHTINGS are read from the different traces.

5. Miscellaneous Modifications. At this stage, the user may decide to specify more events, run ScenicPeex with different parameters (*e.g.* allowing or disallowing partial events), or even alter the traces. The user can alter a trace by increasing the missed readings rate, allowing for more spurious readings of an object by nearby antennas or even by disabling an antenna indefinitely.

6. Comparing different calendars. Since our application supports multiple calendars, the user can run traces with different parameters and get a side-by-side visual comparison of the effect of each choice. For example, in Fig 2 the user has a separate calendar for different probability thresholds. The calendar labeled 'Threshold 0.75' shows only detected events with probability higher than 0.75. This set of calendars allows the user to compare the PEEX approach to a purely deterministic one (i.e. calendar with 'Threshold 1.0'). An alternative partitioning is to have two calendars: one for events that were detected by PEEX with partial events *allowed* and another for events detected with partial events *disallowed*. Such a setup would portray to the user the impact of partial events on the efficacy of ScenicPeex.

The input to our demonstration is a set of sensor reading traces from ten volunteers and their objects moving through a building. These traces are obtained from a real-world 150-antenna deployment at the University of Washington [5]. Using real-world traces enables the demonstration to run over a larger data volume than a toy deployment. Additionally, real-world data has some unexpected insights. For example, in the lab we can read laptop cords with approximately 90% accuracy, which drops to $\approx$ 10% in the real world. Real-world data thus shows the performance of ScenicPeex in practice.

4. References

[1] C. Floerkemeier and M. Lampe. Issues with RFID usage in ubiquitous computing applications. In *Proc. of the 2nd Pervasive Conf.*, Apr. 2004.
[2] Google Inc. . Google code: Developer home. `http://code.google.com/`, 2007.
[3] S. Jeffery, G. Alonso, M. J. Franklin, W. Hong, , and J. Widom. Declarative support for sensor data cleaning. In *Proc. of the 4th Pervasive Conf.*, Mar. 2006.
[4] N. Khoussainova, M. Balazinska, and D. Suciu. Peex: Extracting probabilistic events from rfid data. Technical Report 2007-11-02, Department of Computer Science and Engineering, University of Washington, 2007.
[5] The RFID Ecosystem. `http://rfid.cs.washington.edu/`, 2007.
[6] RFID journal. `http://www.rfidjournal.com`, 2006.