A Demonstration of Cascadia Through a Digital Diary Application

Nodira Khoussainova, Evan Welbourne, Magdalena Balazinska, Gaetano Borriello, Garrett Cole, Julie Letchner, Yang Li, Christopher Ré, Dan Suciu, Jordan Walke
Department of Computer Science and Engineering, University of Washington
Seattle, WA, USA
nodira, evan, magda, gaetano, gbc3, letchner, yangli, chrisre, suciu, jwalke@cs.washington.edu

ABSTRACT

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

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

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management Systems ; H.5.2 [**Information Systems**]: Information Interfaces and Presentation User Interfaces

General Terms

algorithms, design, experimental

Keywords

probabilistic event extraction, user interfaces, RFID

1. INTRODUCTION

Many mobile and pervasive applications rely on Radio Frequency Identification (RFID) infrastructures 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 [8], which allows applications to track large numbers of objects and people. However, developing RFID applications is challenging for two main reasons. First, RFID technology is unreliable. It often produces duplicate readings [5] or missed readings [2, 5]. 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.)

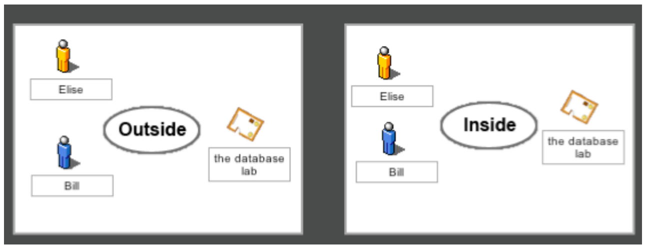
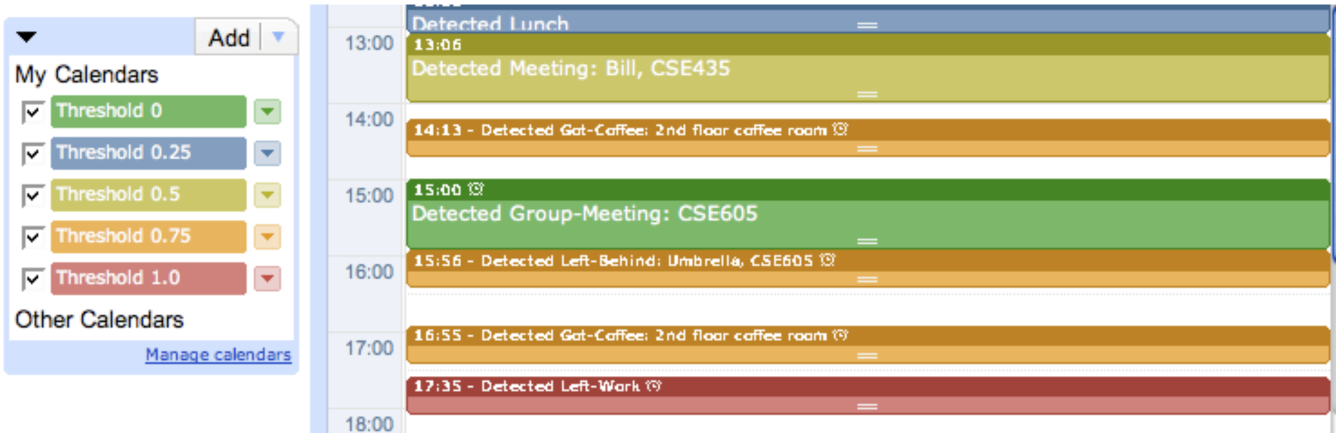


Figure 1: A `Meeting` **event specification in Scenic**

We present a demonstration of Cascadia. Cascadia is a new infrastructure aimed at simplifying the development of user-centric RFID applications. It is a system for specifying, detecting and managing RFID events. The two key components of Cascadia are Scenic and PEEX. Scenic [7], is a tool that allows developers and even end users to graphically specify spatio-temporal 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 PeexL, a SQL-like language for defining high-level events. The **P**robabilistic **E**vent **EX**traction system (PEEX) [6] is an RFID data management system that enables applications to declaratively specify events in PeexL. It then continuously and automatically extracts these events from RFID data streams and stores them persistently to simplify their subsequent management. PEEX uses a probabilistic approach to event extraction that allows it to effectively detect complex events in spite of RFID data unreliability and ambiguity. Overall, Cascadia thus provides an infrastructure for extracting and managing RFID events and a graphical interface for specifying them.

There are several potential applications for Cascadia. IT support teams can use Cascadia to track their equipment and detect events. A support team can, for example, detect when someone moves a projector from one conference room to another, a user returns a laptop, or someone takes a video camera out of the building. Cascadia can be used in hospitals to monitor patients. Cascadia can also be used in social networking applications. An example of a social networking application is one which provides a user's social network with an ambient awareness of that user's status.

We demonstrate Cascadia using Digital Diary, an automatic-calendar application. We choose this application for its simplicity and more importantly for its transparency that can provide users with a good understanding of the challenges of detecting real-world events from RFID data. It can also help understand the details of



Detected Lunch

13:06
Detected Meeting: Bill, CSE435

14:13 - Detected Got-Coffee: 2nd floor coffee room

15:00
Detected Group-Meeting: CSE605

15:56 - Detected Left-Behind: Umbrella, CSE605

16:55 - Detected Got-Coffee: 2nd floor coffee room

17:35 - Detected Left-Work

Figure 2: Screenshot of the digital diary application.

the Cascadia approach. In the demonstration, users specify events in Scenic (*e.g.* entering rooms, meeting with other people, etc.) 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 [7]. 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 [4]. Fig 2 shows an instance of the automatically populated calendar.

To explain the internals of Cascadia, 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.

2. CASCADIA SYSTEM OVERVIEW

In this section, we give a brief overview of Cascadia. First, we present the system architecture in Figure 3. Next, we describe PEEX and Scenic.

2.1 Overall Architecture

At the lowest level, Cascadia receives and stores raw RFID data from a hierarchy of distributed Reader Gateway processes that merge and forward streams of *tag read events* (TREs) from RFID readers.

The TREs are processed with a particle filter [3] to populate the `AT` relation with smoothed, probabilistic location events. The `AT` relation has schema `(time, tagID, locID, prob)`. Particle filters are a commonly used sampling-based inference technique. In our case, we use a particle filte to infer a tag location (hidden state) from TREs (observations). More precisely, we use the particle filter to infer the tag's distribution over discrete locations (hallways, rooms, etc).

The `AT` relation is then processed by PEEX (Probabilistic Event EXtractor), which continuously extracts and stores higher-level events.

Finally, Cascadia further simplifies the process of event specification with Scenic, a user-level tool that assists non-experts in specifying common higher-level events. We further describe PEEX and Scenic below.

2.2 PEEX

PEEX [6] is Cascadia ś event detection subsystem. PEEX supports probabilistic events represented as tuples and stored in relations named for each event type. The most basic event and thus relation is `AT`. An example tuple is `AT(101, 10, 23, 0.6)`, which represents that at time 101, the tag with ID 10 was seen in location 23 with probability 0.6. Every tuple in the `AT` relation is an output of the particle filter.

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

2.2.1 Event Definition Language

To build high-level events from primitive `AT` events, PEEX uses, PeexL, a SQL-based *event definition language*. Event specifications in PeexL have the form:

FORALL $I_1, I_2, ..., I_n$
[CTABLE C]
WHERE Condition
CREATE EVENT E
SET Assignments

The arguments to the `FORALL` clause, $I_1, \ldots, I_n$, correspond to `AT` events, other composite events, or to regular database tables and may optionally be preceded by a negation `!`. The `CTABLE` clause specifies the confidence table, which helps in handling event ambiguity as we discuss in the following section. The confidence table is optional, and in our current implementation, only the application developer can specify it (*i.e.* , it cannot be specified through Scenic). The `WHERE` clause is as in SQL with the addition of the `SEQ` predicate which we adapt from [1, 9]. *i.e.* `SEQ(`$I_1, I_2, \ldots, I_m$`)` states that I_j`.time` $\leq$ I_{j+1}`.time` for $j \in \{1, \ldots, m\}$. One can also specify that an argument in the `SEQ` operator `LASTS` for a specified period of time. Finally, the `CREATE EVENT` and the `SET` clauses define the name and the attributes of the new event.

As an example, the `MEETING` event from Figure 1 is written in PeexL as shown in Figure 4

Intuitively, the event definition says that if we see Elise and Bill together in the hallway *outside* the database lab (line 5,6) followed by *inside* the lab (line 7,8), there is some probability that Elise and Bill are having a meeting in the lab (line 13,14). The event definition uses a helper table, `LocId2Descr`, to map location IDs to

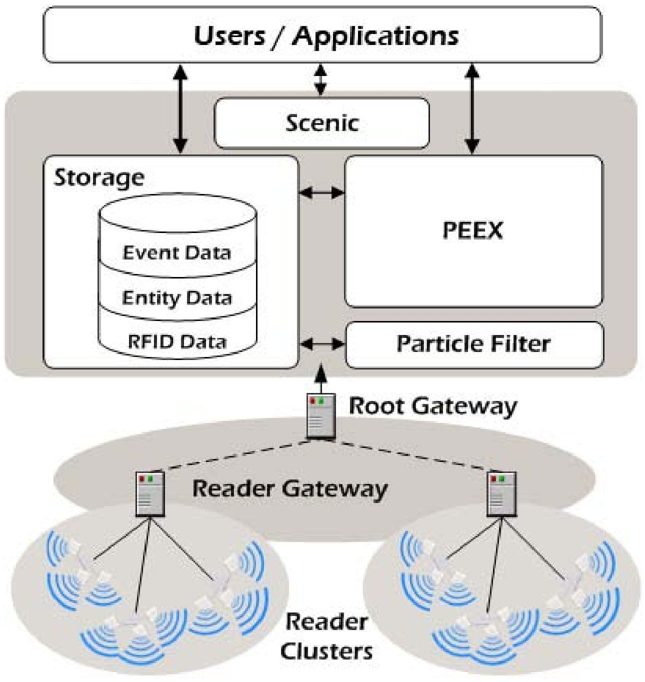


Figure 3: Cascadia system architecture.

a description of the location. `LocId2Descr` has schema `(locID, descr)`. `CMeeting` is a *confidence table* which stores the probability that two people are having a meeting given that they enter the same room. It has schema `(person1, person2, room, prob)`. Importantly, confidence tables allow the probabilities assigned to an event to depend on the values used to trigger the 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 (lines 11,12). 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.

2.2.2 PEEX Architecture

We have designed PEEX as a layer on top of a traditional RDBMS (Microsoft SQL Server in our prototype). PEEX consists of two major components: the *Event Detector* and the *Confidence Learner*.

Event Detector. At a high level, the Event Detector leverages the underlying RDBMS during event extraction. All events are stored, in the RDBMS, as relations. When a primitive event arrives from the particle filter, *e.g.* Bill is at antenna 10 with probability 0.3, a corresponding tuple is appended to the `AT` 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.

There are six key parts to the transformation from PeexL to SQL. *(1)* All `SEQ(`I_1, I_2`,..)` constructs are transformed into explicit predicates on input event timestamps. *(2)* The `LASTS` predicate for an argument in the `SEQ` (if there is one), is translated into a *count sub-query*. For example, if one specifies that an underlying event must last for 10 seconds, this is translated into SQL which specifies that the event must occur once, then again ten seconds later, and also at the eight distinct timesteps in between (thus the count is 10). *(3)* All negations are re-written into outer-joins, which join two relations but also include tuples without matches in the result. *(4)* To avoid repeatedly detecting the same events on successive runs, the Event Detector transforms event definitions into *incremental* queries. These queries only retrieve combinations of low-level events in which at least one has occurred in the most recent δ window. *(5)* Additionally, the Event Detector inserts a predicate stating that all the underlying events must occur within the larger time window (Δ seconds). *(6)* Finally, the generated SQL includes a calculation that computes the probability of the event as a function of the probabilities of the events on which it depends as well as the appropriate probability from the corresponding confidence table.

```
1)   FORALL AT S1, AT S2,  AT S3,  AT S4,
2)      LocId2Descr L1, LocId2Descr L2
3)   CTABLE CMeeting C
4)   WHERE  SEQ(AND(S1, S2), AND(S3, S4)) AND
5)      S1.locID = L1.locID AND S2.locID = L1.locID AND
6)      L1.descr = 'DB lab hallway' AND
7)      S3.locID = L2.locID  AND S4.locID = L2.locID AND
8)      L2.descr = 'DB lab' AND
9)      S1.tag='Elise tag' AND S2.tag='Bill tag' AND
10)     S3.tag = S1.tag AND S4.tag = S2.tag AND
11)     C.person1 = S1.tag AND C.person2 = S2.tag AND
12)     C.room = L2.descr
13)  CREATE EVENT EliseBobMeeting E
14)     SET E.room = C.room
```

Figure 4: Meeting Event in PeexL

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. Like the Event Detector, the Confidence Learner uses SQL to populate the confidence tables. The SQL is generated from the same event definition that is used for event detection. We refer the reader to [6] for more details on the confidence learner and PEEX.

2.3 Scenic

Defining events directly in PeexL is difficult for application developers and impossible for end users. For this reason, Scenic provides a simpler channel through which non-expert users can create and submit new event definitions to PEEX.

Scenic presents an iconic visual language that represents event primitives and entities as icons which can be dragged and dropped onto a storyboard to specify a sequence of point events, or *scenes*. Thus, to specify an event users just "tell the story" of the event, scene by scene. Figure 5 shows the Scenic interface, which consists of a toolbar, below which is a working area called the sequence panel.

Scenic's visual language consists of three basic constructs: **(1)**

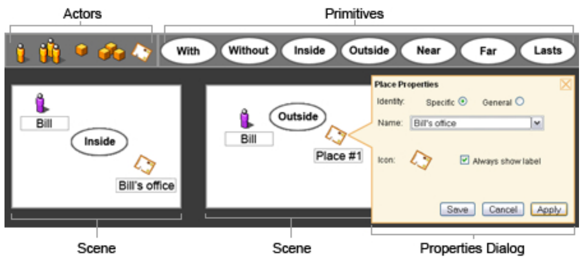


Figure 5: A screenshot of Scenic showing scenes, actors, primitives and a properties dialog for an actor.

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 the graphical specifications, Scenic generates PeexL event definitions. These definitions can then be submitted to PEEX or can be embedded inside applications.

3. DEMONSTRATION CONTENT

We demonstrate a digital diary application that uses Cascadia to detect higher level RFID 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)** The demonstration provides the user with an insight into how Cascadia works. **(2)** It shows the challenges associated with extracting events from RFID data and the impact of various PEEX parameters on the event extraction process.

The input to our demonstration is a set of sensor reading traces from ten volunteers and their objects moving through a building. The traces are obtained from a real-world 150-antenna deployment at the University of Washington [7]. Real-world traces enable 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.

The demonstration allows users to specify events with Scenic, and watch as Cascadia automatically populates the diary with detected events. If a user wishes to understand the techniques more deeply, she is then able to examine the translation from Scenic into PeexL and PeexL into SQL. Additionally, the user may also alter the data traces. She may even adjust the parameters of PEEX to effectively compare PEEX's probabilistic technique to a deterministic one. By using real-world data and allowing the user to alter input and compare techniques, the demonstration enables the user to understand the performance of Cascadia in practice.

4. CONCLUSION

This demonstration showcases an application of and the inner workings of our event extraction tool Cascadia. Cascadia is a system that provides RFID-based computing applications with an infrastructure for specifying, extracting and managing meaningful high-level events from raw RFID data.

In this demonstration, we allowed viewers to graphically specify complex events using Scenic, and detect the specified events using PEEX. We also allowed the users to view the translation of the specification into PeexL (PEEX's event language), and even view the SQL used for extracting the events from raw data. Through this demonstration, the viewer gains insights into the challenges and the workings of event extraction from RFID data.

5. ACKNOWLEDGEMENTS

This work was partially supported by NSF Grants IIS-0627585, IIS-0513877, IIS-0713576, and Gifts from Microsoft including a gift under the SensorMap RFP.

6. REFERENCES

[1] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proc. of the 20th VLDB Conf.*, Sept. 1994.

[2] C. Floerkemeier and M. Lampe. Issues with RFID usage in ubiquitous computing applications. In *Proc. of the 2nd Pervasive Conf.*, Apr. 2004.

[3] Fox, D. et al. Bayesian filtering for location estimation. *IEEE Pervasive Computing*, 2(3):24–33, July-September 2003.

[4] Google Inc. . Google code: Developer home. `http://code.google.com/`, 2007.

[5] 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.

[6] 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.

[7] The RFID Ecosystem. `http://rfid.cs.washington.edu/`, 2007.

[8] RFID journal. `http://www.rfidjournal.com`, 2006.

[9] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *Proc. of the 2006 SIGMOD Conf.*, June 2006.