Probabilistic RFID Data Management

Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu
Department of Computer Science and Engineering
University of Washington
Seattle, WA
{nodira,magda,suciu}@cs.washington.edu

June 20, 2007

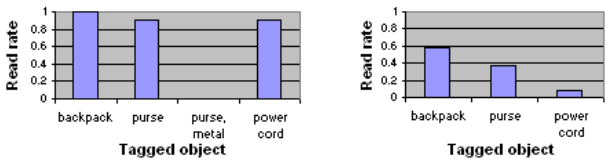
Abstract

Radio Frequency Identification (RFID) technology is increasingly being used to improve various industrial processes, such as supply-chain management. Successes of this technology in industrial settings are leading many to consider other uses of RFID, including user-oriented public deployments. However, the noisy, low-level data produced by RFID readers is almost impossible to use or comprehend in most but the simplest settings.

We present PEEX (Probabilistic Event EXtractor), a system that manages *probabilistic high-level events* from imprecise and erroneous RFID data. PEEX allows users to define probabilistic events from lower-level events. By using probabilities, the system copes with the noise in the data and the inherent ambiguity in the event extraction. We have built PEEX as a layer on top of a traditional RDBMS. We demonstrate, through experiments with real RFID traces collected on a small antenna deployment, that PEEX significantly improves event detection rates compared with deterministic techniques, and provides applications a flexible trade-off between event recall and precision.

1 Introduction

In the past several years, Radio Frequency Identification (RFID) technology has become increasingly popular as a flexible and relatively low-cost solution for tagging and wireless identification [35]. Currently, the main use of RFID technology is in the supply-chain management domain [32]. However, successes of RFID in industrial settings are leading many to consider pervasive deployments of this technology, where objects and people carry tags and RFID antennas are scattered through the environment. Such deployments can potentially enable many new types of user-oriented applications [5, 33] from simple tracking and alerting services, to sophisticated studies of social phenomena.



(a) In-laboratory read-rates (b) Practical read-rates

Figure 1: **Read-rates measured for various objects**

Managing RFID data, however, raises significant challenges [7, 21]. In particular, RFID readers produce streams of low-level observations of the form: "Tag 344 was last seen at antenna 647 at 3:20pm". This low-level data must be transformed into high-level events meaningful to applications, such as "Alice entered the conference room at 3:20pm", or "Alice's keys appear to be missing from her purse". There are two important issues in performing such extrac-

tions. The first issue is reliability. RFID antennas frequently fail to read tags in their vicinity [15, 22] and nearby antennas can detect the same tags at the same time [22]. Figure 1 illustrates some sample read-rates that we observed in laboratory experiments and in a 2-week pilot study on a small deployment [37]. As the figure shows, failures are frequent, especially on metal objects, or when participants move around freely.

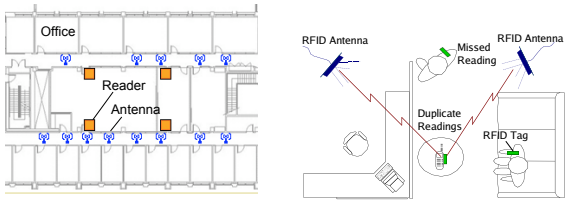
The second issue in transforming low-level observations into high-level events is ambiguity. A combination of low-level tag reads may not correspond uniquely to a single high-level event. This is especially true in pervasive deployments where detecting a person at a sequence of locations may indicate that they are performing one of several possible activities, *e.g.*, Alice is printing a paper or sending a fax.

Most previous schemes for RFID event detection ignore ambiguity and input data errors [34, 39]. Schemes that do consider data errors propose to clean the data deterministically before processing it [22, 23, 29, 34]. A deterministic model cannot handle ambiguous events, and, as we show in this paper, input data errors can cause a deterministic extraction to miss significant numbers of events. We also show that, even though it usually helps, in some cases, deterministic RFID data cleaning can in fact lower the detection rates for some events.

To address the limitations of existing techniques, we propose to use a probabilistic model for processing RFID data: we propose a new approach and present a new system for extracting high-level *probabilistic* RFID events from low-level tag reads.

In this paper, we focus on techniques for extracting high-level events from RFID data. We do not address query execution over the detected events. However, because the output that our system produces follows standard probabilistic models, existing probabilistic databases [8, 9, 38, 10] could be used to execute queries over our system's output.

We implemented PEEX as a layer on top of an RDBMS [26]. Using experiments with real RFID data traces, we show that a probabilistic approach to event detection significantly improves detection rates compared with deterministic techniques. Improved detection rates come at the cost of a lower precision.



(a) Sample deployment (b) Types of errors

Figure 2: **RFID deployment and errors**

However, because probabilities are associated with all detected events, applications can choose their desired trade-off between detection rate and precision by considering only events above some probability threshold.

Overall, we show that the uncertainty of a monitored environment cannot be cleaned away and hidden from applications. Instead, modeling and incorporating that uncertainty in the form of probabilistic rather than deterministic events is necessary, especially for pervasive RFID deployments and applications.

2 Application Scenarios

Unlike other location systems that tag objects with expensive devices [28, 36], RFID-based systems can rely on inexpensive, passive RFID tags (the typical cost of a tag is approximately 20 to 40 cents [30]). Low-cost tags make it feasible to track large numbers of objects, opening the door to many new types of useful services.

To experiment with RFID technology and applications, we have deployed RFID antennas in all the hallways in our department building. Figure 2(a) illustrates our deployment. Read ranges for RFID depend primarily on the tags — some tags are only read from about one foot while others can be read as far as 10 to 20 feet [30]. Read ranges, however, also depend on antenna positions, orientations, manufacturers, and on the environment. RFID readers located in nearby offices poll antennas continuously and send information about detected tags to a back-end database. We use our RFID deployment as the motivating scenario throughout the paper.

Given such a deployment, a simple yet useful application consists in tagging people's belongings and departmental assets and allowing owners to track the movements of their objects over time. This application enables owners to find their objects, receive alerts when they forget their objects behind, or simply recover the historical movements of an object (*e.g.*, to discover the source of damage to an object or who may have used it last). If people also carry RFID tags, more sophisticated applications can enable users to find each other at any time, or receive an alert when other users enter or leave specific locations (*e.g.*, their office or the conference room).[1] Tracking participants in a community also enables studies of various social behaviors.

Concretely, consider an application that enables users to determine the current location of co-workers. In most cases, users do not want to see low-level information such as "Chuck was last seen at antenna 37 at 11:56am" but higher level information such as `In-Business-Meeting(Chuck, Alice)`. To enable such a scenario, we need a system to transform the imprecise, low-level sensor data into high-level events meaningful to applications. In this example, the `In-Business-Meeting(Chuck, Alice)` event occurs when Chuck and Alice both walk into a room together, each with their notebooks. Such a high-level event is likely to be constructed from lower-level events that indicate that a person has entered a room, that their notebook has entered the same room around the same time and that they have not left the room before the other attendee arrives. We may be even more certain of our conclusion if this pair has a history of meeting together in specific offices.

These scenarios illustrate the main challenges we face:

Ambiguity. With pervasive applications, the same set of low-level observations can correspond to multiple distinct high-level events. For example, it is never certain that Alice and Bob are having a business meeting together, one of them may have stopped by to work with another officemate. Additionally, the two colleagues may not always meet to only work together. A fraction of the time, they may instead be going to attend a seminar. The high-level events that can be extracted from RFID data are thus probabilistic rather than deterministic in nature. This uncertainty propagates as probabilistic events are aggregated into even higher-level events. For example, if the system has only limited confidence in the underlying `Entered-Room` events, the confidence in the business meeting event will be accordingly lower.

Errors. As illustrated in Figure 1, RFID data contains significant amounts of false negatives. In general, error rates depend on the equipment used, the object material (*e.g.*, metallic objects or objects containing water have much lower detection rates than other objects), and the orientation of tags and antennas [37]. RFID data may also contain false positives where nearby antennas detect the same tag at the same time as illustrated in Figure 2(b). False positive rates depend primarily on antenna deployment. In our case, for example, antennas cover slightly overlapping spaces, causing some false positives. False positives also occur in higher level events, especially those which depend on the non-existence of low level sightings.

Cleaning such errors with certainty is not always possible. For example, if a person is seen by the elevator and then disappears, it is not certain that the person left the building. The person may have gone to get some coffee, but the RFID antenna by the coffee shop failed to detect them. Similarly, if a cell phone appears to be in two offices at the same time, it is not necessarily clear which location is correct.

In the next sections, we present PEEX, our probabilistic event extraction system, and show how it addresses the above challenges. We present PEEX's probabilistic event language in Section 3. In Section 4, we present PEEX's system architecture and the detailed event detection algorithm. We evaluate PEEX's performance in Section 5, present related work in Section 6, and conclude in Section 7.

3 Probabilistic Event Language

In this section, we present PEEX's event language.

[1] In these applications, privacy issues are paramount, but these issues are outside the scope of this paper.

3.1 Probabilistic Event Model

The probabilistic event model used in PEEX borrows elements from the event model proposed by Demers *et al.* [12] and from recent probabilistic data models [8, 20, 38].

3.1.1 Events

We start by defining a deterministic event, or, shortly, an *event*, which is a named tuple of the form:

$$\text{EventType}(\underline{\text{evID}}, A_1, \ldots, A_k, \text{time})$$

Here `EventType` is the type name of that event: PEEX stores all events of this type in a relation called `EventType`. The attributes are as follows: `evID` is a system-generated event key, $A_1, \ldots, A_k$ are the categorical attributes of the event, and `time` is a temporal attribute that capture the time when the event occurred. Unlike Demers *et al.*, in PEEX, we assume that events are instantaneous: *i.e.*, their duration is always one time-unit[2]. Examples of events are:

```
SIGHTING(evID, tagID, antID, time)
ENCOUNTER(evID, tagID1, tagID2, antID, time)
COFFEE-ENCOUNTER(evID, tagID1, tagID2, antID, time)
ENTERED-ROOM(evID, tagID, ROOMID, time),
LEFT-ROOM(evID, tagID, ROOMID, time)
IN-BUSINESS-MEETING(evID, tagID1, tagID2, ROOMID, time)
```

For example `SIGHTING(8228,tag779,ant32,202)` represents the event that antenna 32 picked up a reading of tag779 at time 202; the system has assigned the unique event identifier 8228. In PEEX, we assume that RFID readers produce continuous streams of RFID tag readings. These readings are appended to a base relation called `SIGHTING` and stored persistently before being processed. All events in `SIGHTING` are deterministic. A *primitive event* in PEEX thus corresponds to a tuple in the `SIGHTING` relation. PEEX exploits the append-only property of base relations to process primitive events incrementally. By contrast, an event like `ENTERED-ROOM(5394, tag404, room555, 300)` is a *composite event*, in the sense that it was derived from other more basic events: it represents the fact that tag 404 is believed to have entered room 555 at time 300, and may be inferred by the system from, say, successive antenna readings of the tag 404 along the hallway leading to room 555. Here, too, `ENTERED-ROOM` is a base relation where the system inserts events as soon as they are inferred.

Note that `tagID` plays no special role among an event's attributes, and some events may refer to more than one tag (*e.g.*, `ENCOUNTERED` above), or none at all. This is important for complex applications that need to manage complex composite events relating multiple people and/or objects.

3.1.2 Probabilistic Events

Unlike primitive events, most composite events in PEEX are probabilistic. Formally, a *probabilistic event* consists of a name, a key, and a joint probability distribution on all its other attributes. The distribution is specified separately for the time attribute and for the categorical attributes. For the categorical attributes it is specified by a direct enumeration of their joint probability distribution. For an illustration, consider the probabilistic event `ENTERED-ROOM(evID, tagID, room, time, prob)` (a `prob` attribute has now been added). One event 5394 is now defined as follows:

```
ENTERED-ROOM(5394, tag404, room555, 300, 0.4)
ENTERED-ROOM(5394, tag404, room505, 300, 0.3)
ENTERED-ROOM(5394, tag404, room501, 300, 0.1)
```

Event 5394 consists now of a probability distribution on the three rooms where the system believes that tag 404 may have entered. Note that their probabilities add up to less than one, because the system leaves open the possibility that the tag moved to a different location. In this illustration a single attribute (`room`) was uncertain; if more attributes are uncertain, then we simply enumerate all their combinations of values (*i.e.*, the entire joint distribution), for example if `tag` and `room` are uncertain then we have tuples of the form:

```
ENTERED-ROOM(5394,tag404,room505,300,0.4)
ENTERED-ROOM(5394,tag404,room561,300,0.2)
ENTERED-ROOM(5394,tag111,room561,300,0.3)
```

In addition to uncertainty about categorical event attributes, the exact time when an event occurs is often uncertain as well. As a simple example, we ran an experiment where a person walked by two antennas placed a few meters apart in a hallway. The travel time, as measured by the antennas tag reads, var-

[2]Extension to long-duration events is left for future work.

ied between 0.7 seconds and 2.5 seconds depending on the experimenter's speed. If the first antenna was located a few meters from an office door, and the second antenna was the door, then, in our experiment, the ENTERED-ROOM event would occur between 0.7 and 2.5 seconds after the last antenna sighting. In general, because RFID tag reads occur only when objects pass in front of antennas, and because it would be costly to cover every square foot of an environment, individual tags are detected only periodically. This periodicity is one reason for the uncertainty about the exact timing of events.

To capture this uncertainty in PEEX, the time attribute value is specified independently, by a probability distribution function. The system currently has support for the normal distribution. Thus, the time value of a probabilistic attribute is an abstract data type consisting of a mean and variance for representing the normal distribution. For example, the last event above may be specified as follows:

```
ENTERED-ROOM(5394, tag404, room508,  normal(3200,52), 0.1)
```

meaning that the time attribute has a normal distribution with a mean of 3200 and a variance of 52. In order to maintain simplicity, PEEX requires all events to use a normal distribution for its time attribute. The normal distribution is ideal because the class of normal distributions is closed under linear combinations.

Since the time attribute of a probabilistic event is an abstract data type, only a limited number of operations are supported over this attribute including creation (*e.g.*, we may create a new value normal(310,25) representing a normal distribution with mean 310 and standard deviation 25), copy (*e.g.*, in SET x.time = y.time we simply copy the pdf from event y to event x), translation by a constant (*e.g.*, in the expression x.time + 30), translation by another normal distribution (*e.g.*, in SET x.time = y.time + normal(400,30.4))and comparison (*e.g.*, the predicate x.time < y.time is interpreted as the probability that the pdf for x.time is less than the pdf for y.time).

3.2 Event Language

In PEEX, users define *composite probabilistic events* from lower level primitive events using a declarative query language with a single construct:

```
FORALL I1, I2, ..., In
[ CTABLE C ]
WHERE Condition
[ WITH m FALSE NEGATIVES ]
[ WITH q FALSE POSITIVES ]
CREATE EVENT E
SET Assignments
```

The arguments of the FORALL clause, I_1, ..., I_n, correspond to primitive events, to previously defined composite events, and/or to regular database tables. They have the same syntax as a SQL FOR clause and are possibly preceded by a negation sign ! as explained in Section 3.4.3. The condition in the WHERE clause actually consists of four components. The first component consists of conditions concerning the order of events (specified using the SEQ operator). The second component talks about how to use the confidence table. While the third and fourth components consist of conditions about only the positive relations in the FORALL clause and those about the negated relations, respectively. Each component has a syntax very similar to the body of a traditional WHERE clause. For brevity, we do not use this exact syntax in the examples that follow.

E is the type name of the composite event. The SET clause defines the attributes (categorical and temporal) of the new event. A trivial illustration is given in the example below:

```
FORALL SIGHTING I
WHERE I.antID = 'antenna036'
CREATE EVENT NEAR-ROOM E
SET   E.tagID = I.tagID,
      E.room = 'Room508',
      E.time = I.time;
```

Given a base event SIGHTING(8778, tag432, antenna036, 420) the system will generate a composite event NEAR-ROOM(238, tag432, Room508, 420). This is a deterministic event (*i.e.*, its probability is 1, and its time attribute is a point, 420), and the event id 238 is system-generated. We assume here that antenna 036 is located at the entrance to room 508. In general, applications are not interested in low-level events, but instead in such composite high-level events, which are more meaningful.

The challenge is to enable applications to express

```
FORALL SIGHTING I1, SIGHTING I2, SIGHTING I3
WHERE  SEQ(I1, I2, I3)
       AND I1.antennaID = 'antenna32'
       AND I2.antennaID = 'antenna35'
       AND I3.antennaID = 'antenna39'
       AND I1.tagID = I2.tagID
       AND I2.tagID = I3.tagID

CREATE EVENT ENTERED-ROOM E
SET  E.tagID := I1.tagID
     ( E.room := 'room555' CONFIDENCE 0.4 |
       E.room := 'room505' CONFIDENCE 0.3 |
       E.room := 'room501' CONFIDENCE 0.1)
     E.time  := I3.time
```

(a) No confidence table

```
FORALL SIGHTING I1, SIGHTING I2, SIGHTING I3
CTABLE FLOOR5-STATS C
WHERE  SEQ(I1, I2, I3)
       AND I1.antennaID = 'antenna32'
       AND I2.antennaID = 'antenna35'
       AND I3.antennaID = 'antenna39'
       AND I1.tagID = I2.tagID
       AND I2.tagID = I3.tagID
       AND I1.tagID = C.tagID

CREATE EVENT ENTERED-ROOM E
SET  E.tagID := I1.tagID
     ( E.room  := C.room CONFIDENCE C.conf )
     E.time  := I3.time
```

(b) Using a confidence table

Figure 3: **Examples of probabilistic event definitions for** `ENTERED-ROOM`

such events and for the system to detect them in spite of errors in the input readings and uncertainty in the way the composite events are defined. This is where the additional clauses come into play. We explain these clauses in sections following the next section which presents a simple taxonomy of events.

3.3 Event Taxonomy

In this section we describe a taxonomy for events in our event language. Along with definitions of the three classes of events, we also give examples of events in each class.

We begin with L_0, the class of all raw events. All events in `SIGHTING` belong to L_0.

Next is L_1, which contains all events in L_0 and also all events which can be defined using one `SEQ` operator, only conjunctions and no negations. An example of an event in L_1 is the `ENTERED-ROOM` event (Figure 3).

Finally we have L_2, the class consisting of all events in L_1 in addition to all those events defined on top of L_1 events using one `SEQ` operator with both conjunctions and negations allowed. An example of such an event, is `In-Business-Meeting` which is detected when the system sees one person `Enter-Room` with their notebook, followed by another person (also with their notebook) and the first person has not `Left-Office` before the second person arrives.

Disjunctions are captured in our language through the use of multiple event definitions. Therefore, there is no need for nested negations or conjunctions, as these can be defined with disjunctions. For example, if an event consists of a sequence operator `SEQ(A,!(AND(B,C)),D)` this could be represented by two separate event defnitions consisting of sequence operators `SEQ(A, !B, D)` and `SEQ(A, !C, D)`. This allows us to represent all events that are naturally defined using nested negations or conjunctions, by using multiple event definitions in L_2.

3.4 Ambiguous Composite Events

A given combination of RFID tag readings defines a composite event only with limited certainty. As an example, in our deployment, antennas are placed two to three offices apart. When a user stops between a pair of antennas, the user may thus be in one of several offices. Furthermore, the visit to a room may correspond to different high-level tasks (*e.g.*, printing a paper or faxing some documents).

One way to capture such ambiguity, is for PEEX's language to provide a `CONFIDENCE` modifier for the `SET` clause. This modifier lets a user state the probability that a combination of observations matches a high-level event. Consider the example in Figure 3(a), which defines the composite event `ENTERED-ROOM`. We assume that three consecutive readings of the same tag by antennas 32, 35, and 39, signal that the tag moves towards a cluster of three rooms, most likely towards room 555, but maybe towards 505 or 501. These alternatives could be captured in the event definition by assigning different values to the room attribute, with different confi-

dences. Such choices define a joint probability distribution on the `room` attribute of the new event. Similarly, users could define joint distributions on multiple attributes as:

```
SET (   A1 := v1, A2 = v2 CONFIDENCE c |
        A1 := v1', A2 = v2' CONFIDENCE c')
```

which assigns probability `c` to the values (v_1,v_2) and probability `c'` to the values (v_1',v_2').

In general, however, event confidences may not be constant; they may depend on different attributes of an event or may even be correlated with some attributes. For example, certain people are more likely to go to room 501 (*e.g.*, the owner of that office) than to room 555 (say, a conference room). The exact probabilities may further depend on other factors, such as the time of day.

To enable the specification of such correlations, we propose to use separate *confidence tables*. A confidence table is any table in the relational database, but is typically a table with a schema of the form:

```
CONF_TABLE(A1, A2, ..., An, conf)
```

Confidence tables are explicitly created by the user or application. Figure 3(b) illustrates the approach, by showing the specification of the `ENTERED-ROOM` event with a confidence value computed from a `FLOOR5-STATS` confidence table, which appears in the `WHERE` clause. In this example, the confidence is no longer defined in the event definition, but is read from the `FLOOR5-STATS` table, and depends on both the room number and the tag identifier: the schema of `FLOOR5-STATS` is simply `(tagID,room,conf)`.

The use of a confidence table has two important consequences. First, it allows the application to define different confidences for each person/room pair. Second, it allows the system to *learn* these confidence values from training data, as we explain below.

Next, we show how to define a probability distribution function on the time attribute of an event. As an example, consider the following event definition for `LEFT-THE-BUILDING`:

```
FORALL SIGHTING I
WHERE I.antID = 'floor1elevator'
CREATE EVENT LEFT-THE-BUILDING E
SET   E.tagID := I.tagID
      E.time = normal(I.time+10, 20)
      CONFIDENCE 0.6
```

This event definition specifies that if a person is seen by the elevator on the first floor at some time t, then that person is 60% likely to leave the building at some time which has a normal distribution with an expected value of $t + 10$ and a standard deviation of 20. A perhaps better approach is to use a confidence table, as illustrated below:

```
FORALL SIGHTING I
CTABLE WALKING-OUT-TIMES W
WHERE I.antID = 'floor1elevator' AND I.tagID=W.tagID
CREATE EVENT LEFT-THE-BUILDING E
SET   (E.tagID := I.tagID
      E.time = W.time + 10
      CONFIDENCE W.conf)
```

This generates the composite events shown in Fig. 4.

LEFT-THE-BUILDING

1128	tagID	time	prob
1129	AliceTag	normal(410,30)	0.5
1130	Laptop460Tag	normal(260,5)	0.8
1130	AliceTag	normal(610,8)	0.7
...			

Figure 4: **Sample `LEFT-THE-BUILDING` events**

3.4.1 Learning Confidences

A natural question is where the confidence values should come from. PEEX allows these confidences to be either specified manually in the confidence tables, or learned from training data.

In order to perform learning the system needs a set of training data, consisting of an instance of input events, the confidence table (but an empty one), and an instance of the output (composite) events. The latter can be obtained from historical data by asking users to annotate their movements with the corresponding activities for some time period. From these inputs PEEX computes the probability that a combination of low-level observations matches a higher-level event. We describe this procedure in detail in Section 4.3.

3.4.2 Assigning Probabilities to Composite Events

The probability of a composite event created from deterministic events corresponds to the value assigned to it in the confidence clause. The probability of composite events created from probabilistic events must take into account the probabilities of the underlying, lower-level events. For example,

an ENTERED-MEETING-EVENT could be defined as the combination of an ENTERED-ROOM and a HAS-LAPTOP event. Let's denote with p_1 and p_2 the probabilities of these lower-level events. By default, PEEX assumes that the events are independent and returns p_1p_2 as the probability of the composite event.

It may happen, however, that the lower-level events are *correlated* because they are defined in terms of the *same* composite (probabilistic) event. For example, suppose that the definition of ENTERED-ROOM relies on another composite event, BOB-LEFT-HIS-OFFICE which has probability p. Thus, $p_1 = pqr\ldots$ where $q, r, \ldots$ are the probabilities of other composite events used in the definition of ENTERED-ROOM.[3] Assume now that the HAS-LAPTOP event is also defined in terms of BOB-LEFT-HIS-OFFICE, hence $p_2 = pq'r'\ldots$. Now the ENTERED-ROOM and HAS-LAPTOP events are correlated, and the probability that both have happened is no longer p_1p_2.

To properly handle these correlations, PEEX rewrites the definition of each new event in terms of its lower-level events: *i.e.*, it recursively inlines the definitions of all underlying probabilistic events. In the example, such a rewrite leads to a new event definition that uses BOB-IN-HIS-OFFICE twice, which PEEX minimizes. PEEX now computes the correct probability $pqr\ldots q'r'\ldots$ (here p occurs only once), which is correct.

At the end of the process, the probability that all lower-level events occurred is multiplied by the value in the confidence clause of the new event.

Another challenge in computing the probability of composite events involves events which depend on the non-existence of simpler events. For example, the In-Business-Meeting event depends on the non-existence of Left-Room events for the time period of when the first person enters the room and the second person enters the room. However, if such Left-Room events do exist with probabilities $p_1, ..., p_k$. We may not immediately rule out the existence of the In-Business-Meeting event, unless the probability of at least one Left-Room event is exactly 1.0. In all other cases, we must multiply our original calculation for the In-Business-Event by $\prod_{i=1}^{k}(1-p_i)$, the probability that none of the violating Left-Room events occured.

[3]If an event contains choices, these choices are assumed to be exclusive and the overall probability that the event occurred is given by the sum of the probabilities of the choices.

3.4.3 Temporal Aspects of Events

Our event definition language is primarily based on SQL, which allows us to borrow (and extend) the probabilistic data model described in [8, 20, 38]. Our language also includes powerful constructs for predicates on event ordering, which we borrow from [6, 39]. We describe two such constructs.

The first is SEQ(I_1, I_2, ...). This is a predicate stating that the events I_1, I_2, ... come in this order: *i.e.*, I_1.time $< I_2$.time $\wedge$ I_2.time $< I_3$.time ....

The second construct is the bang ! in front of a variable in the FORALL clause, which specifies the non-occurrence of the given event. Consider the following event definition:

```
FORALL SIGHTING I1, SIGHTING ! I2, SIGHTING I3
WHERE   SEQ(I1, I2, I3)
        AND I1.tagID = I2.tagID AND I2.tagID = I3.tagID
        AND I1.antID = 'ant10' AND I3.antID = 'ant20'
...
```

The definition indicates that event I_2 should NOT exist: the event corresponds to two sightings I_1, I_3 of the same tagID, at antennas 10 and 20 respectively, such that there is *no* sighting of the same tagID in between them.

3.5 Coping with Sensor Errors

Sensors are brittle devices; the data they produce frequently contains errors: a *false negative* is when an RFID antenna fails to detect a tag, while a *false positive* is a wrong reading (*e.g.*, by a neighboring antenna).

False negatives are by far the most prevalent. Depending on object types, we observed between 5% and 100% false negative readings in our early deployment (*i.e.*, metal laptops were never detected and were lowering the detection rate of objects in the same bag).

False positives, where two antennas detect the same object at the exact same time, are rare. However, in our deployment, objects are frequently within read-range of two consecutive antennas. If an object

moves slowly or stops, it may appear to be jumping between the antennas, producing a sequence of tag reads of the form:

```
(100,tag404,ant1,202)
(101,tag404,ant2,203)
(102,tag404,ant1,204)
...
```

where tag 404 appears to be jumping between antenna 1 and antenna 2. Both types of errors dramatically impact applications that rely on complex events, because their rate is amplified at each level in the event hierarchy. For example, if a composite event is based on five primitive events, each of which has a rate of 10% false negatives, then the composite event has a rate of 41% false negatives. If five such events need to be combined to compute a composite event at the next level of the hierarchy, then the error rate there is 92%.

To address this challenge, when defining a composite event from `N` primitive event PEEX allows the user to specify a `WITH m FALSE NEGATIVE` clause. This clause instructs PEEX to tolerate up to `m` missing primitive events. More precisely, PEEX automatically extracts all subsets of $N-1$, $N-2$, ..., $N-M$ lower-level events and generates appropriate event definitions. PEEX adds these extra definitions to the system and learns their confidences using historical data. We call these additional event definitions *partial events*. Similarly, the user is able to specify `WITH q FALSE POSITIVE` to allow for partial events, where some events that should not have occurred were actually present.

To illustrate, consider a composite event called `COFFEE-BREAK`, which we detect by observing a person first on the hallway, then in the restroom (to clean her mug), then on the hallway, then at the coffee machine, and there are no other intermediate sightings of that person:

```
FORALL   SIGHTING I1, SIGHTING !J1, SIGHTING I2,
         SIGHTING !J2, SIGHTING I3, SIGHTING !J3,
         SIGHTING I4
WHERE    SEQ(I1,J2,I2,J2,I3,J3,I4)
         AND I1.tagID = J1.tagID ...
WITH 2 FALSE NEGATIVES
WITH 1 FALSE POSITIVES
CREATE EVENT COFFEE-BREAK ...
```

The composite event depends on four positive RFID readings I_1, I_2, I_3, I_4, and three negative readings J_1, J_2, J_3 (*i.e.*, none of these readings is allowed to exist). Clearly, given the high error rate of RFID readings, our likelihood of computing the composite event correctly is quite low. Instead we allow the system to tolerate up to 2 false negatives (*i.e.*, two of the events I_1, I_2, I_3, I_4 may be missing) and up to 1 false positives (*i.e.*, one of the events J_1, J_2, J_3 may be present). We analyze the effect of these clauses in Section 5.3.

4 PEEX Architecture

We have designed PEEX as a layer on top of a traditional RDBMS (we use Microsoft SQL Server [26] in our implementation). This design enables us to demonstrate the benefits of probabilistic RFID data management, while leveraging all the features of an existing database management system.

As illustrated in Figure 7, PEEX is comprised of three modules: an *Initializer*, an *Event Detector* and a *Confidence Learner*. The Initializer creates an environment for PEEX in the database. The Event Detector performs both event detection and generation. The Confidence Learner populates confidence tables from historical data. The current system also provides the user with a graphical user interface through which to enter event definitions, to activate events, to check the details of events and to run the system. (See Figure 5).

In this section, we give a description of the main functions of these modules.

4.1 Initializer

The initializer has two main functions. The first is setting up an environment for PEEX in the database and loading information about the existing environment from the database. The second function is that of adding events and their descriptions to the database. We describe each of these functions in more detail.

The first time one runs PEEX, the system runs a simple initialization process. The initialization process creates an environment for PEEX in the database. It creates tables which keep track of global variables, such as the window size (i.e. the length of the data window we process at each step) , the maxi-

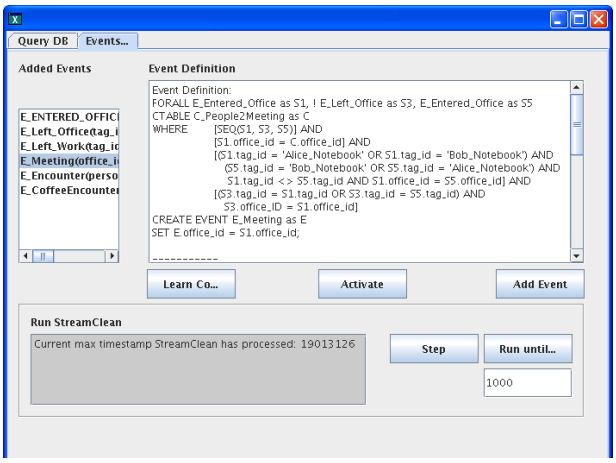


Figure 5: **PEEX's graphical user interface.** Consists of two tabs. The first for querying about the generated events. The second (shown) for learning confidences, adding events, activating events and running the system either step by step or until a specified timestamp.

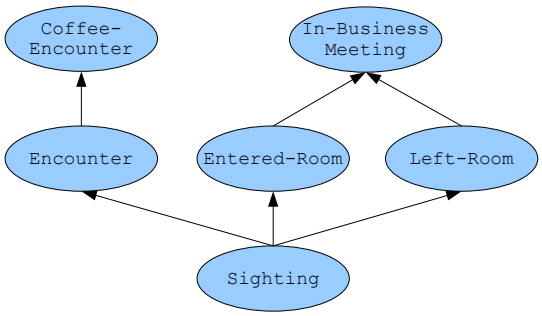


Figure 6: **An event dependency graph.** Edges exist between two nodes if there is a dependency. For example, the `In-Business-Meeting` is dependent on `Entered-Room` events and the non-existence of `Left-Room` events.

mum timestamp we have processed thus far, and the timestamps between which to learn the confidences (i.e. the timestamps between which the training data falls). It also creates a table, `PEEXEvents`, to keep track of all events added to the system, their corresponding confidence tables, and whether the event is active.

After the first run of PEEX, this initialization is no longer executed. Instead, in every subsequent run of PEEX, it loads information about the current state of the system. This involves checking the values of global variables, the events which are active in the system and so on.

Perhaps the most important step in this loading procedure is the construction of the *Event Dependency Graph* (EDG). The EDG is a graph which represents the dependencies that exist between the current event definitions. Each node is an event definition and an edge exists between two event definitions E_1 and E_2 if E_1 is a composite event derived from E_2 (or the non-existence of E_2) and possibly other events. PEEX must ensure that this EDG remains a directed acyclic graph (DAG) at all times. It therefore checks before the addition of an event that no cycles will be formed in the graph. If cycles are to be created, it rejects the event. The EDG is later used to determine in which order to generate events (see Section 4.2). An example EDG is shown in Figure 6.

The second function of the Initializer is adding events. The current system provides the user with a graphical user interface through which to enter event definitions. When a user enters a new event definition, the Initializer parses the definition, and checks to see if the event causes cycles in the current EDG. If no cycles are created in the current EDG, the Initializer adds the event to the EDG, generates the SQL for detecting and generating the event, and generates the SQL for learning the confidences. It then adds the event name, description and corresponding SQL queries to the `PEEXEvents` table. Although the conversion of the event definition to two different SQL statements happens in the Initializer, we describe the details of these procedures in Section 4.2 and Section 4.3.

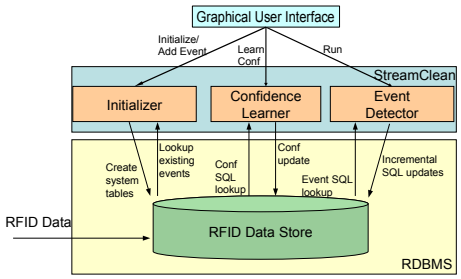


Figure 7: **PEEX software architecture**

4.2 Event Detector

4.2.1 Event Detection

All events (primitive and composite) are stored persistently in the RFID Data Store, using one relation per event type. Primitive events are inserted into the store when they arrive. The Event Detector runs periodically. Every time it runs, it checks if any new events have occurred. For each newly detected event, it inserts a new tuple into the appropriate relation.

The order in which event definitions are processed is critical to the performance of the Event Detector. If the events are not processed in the correct order, the event generation has to be executed recursively until no new events are generated. Therefore, PEEX processes the event definitions in topological ordering from the EDG. A topological ordering is one in which if there is an edge (dependency) from node A to B then A appears before B in the ordering. This ensures that when PEEX generates an event, this new event will not cause any inconsistencies with previously generated events. An example of a topological ordering for the EDG in Figure 6 is `Encounter`, `Entered-Room`, `Left-Room`, `Coffee-Encounter` and `In-Business-Meeting`.

To detect events, the Initializer transforms event definitions into SQL queries that the Event Detector then executes every time it runs. This translation is done when the user first adds the event definition to the system. Once the translation is completed, the SQL query is stored in the database so as to avoid executing the conversion every time PEEX loads.

We begin the description of the conversion process by explaining a simple conversion procedure. At the end of this, we will clarify how our actual SQL conversion differs from the described strategy.

This conversion operation requires three changes to event definitions.

First, all negations that appear in the `FORALL` clause are replaced with `NOT EXISTS` clauses. For example, consider the following event definition from Section 3.4.3 for the `Entered-Room` event.

```
FORALL   SIGHTING I1, SIGHTING ! I2, SIGHTING I3
WHERE    SEQ(I1, I2, I3)
         AND I1.tagID = I2.tagID AND I2.tagID = I3.tagID
         AND I1.antID = 'ant10' AND I3.antID = 'ant20'
CREATE   EVENT ENTERED-ROOM E
SET      E.tagID = I1.tagID,
         E.roomID = room555;
...
```

The definition specifies that event I_2 should NOT exist. It is translated into the following SQL query (where $A_1, \ldots, A_k$ are the values to which the attributes of the event are set):

```
INSERT INTO ENTERED-ROOM (tagID, roomID, time)
SELECT I1.tagID, 'room555', I3.time
FROM SIGHTING I1, SIGHTING I3
WHERE NOT EXISTS (
   SELECT *
   FROM SIGHTING I2
   WHERE SEQ(I1, I2, I3) AND I1.tagID=I2.tagID )
AND SEQ(I1,I3) AND I1.tagID = I3.tagID
AND I1.antID = 'ant10' AND I3.antID = 'ant20'
. . .
```

It is easy to see that the time it takes to run the SQL depends heavily on the number of negations that are involved in the event definition.

Second, all `SEQ(`I_1, I_2`,..)` constructs are transformed into explicit predicates on input event timestamps: *i.e.*, I_1.time $<$ I_2.time AND I_2.time $< \ldots$. This procedure is slightly complicated by negations and `AND` constructs. During this procedure, PEEX also determines the timestamp of the latest event. If the `SEQ` operator imposes only a partial order on the events, then this is simply the `latest()` of all the timestamps. The `latest()` is a user-defined scalar function, which chooses the highest value across columns. This timestamp is then used for setting the timestamp of the created composite event.

This timestamp is also used for the final rewrite. This rewrite is unrelated to the language, but has

to do with the continuous nature of the data and event detection process. To avoid detecting the same events every time it executes, the Event Detector must transform event definitions into *incremental* queries. PEEX achieves this by querying only for combinations of low-level events where at least one event occurred more recently than the Event Detector's previous execution. PEEX uses the `SEQ` construct and all predicates on event times to compute the expected order of the low-level events. If these events are totally ordered, PEEX constrains the last event to occur within a recent time window by adding to the event definition a predicate the form: I_n`.time > now() -` Δ, where Δ is the period between consecutive executions of the Event Detector. If the event definition imposes only a partial order on the lower-level events, PEEX specifies that the maximum timestamp of the events be within the most recent time window.

Hence, PEEX requires that at least one lower-level event for each new composite event occurs within a recent time-window. Additionally, to maintain its performance, PEEX requires that *all* lower-level events occur within some bounded, though much longer, time window. (*e.g.*, the last week worth of RFID readings). This constraint simply ensures that the Event Detector always operates on a small data set.

Even with the above simpler technique, the amount of rewriting is non-trivial. Our current implementation restricts it by constraining event definitions to at most one `SEQ` construct. Other restrictions include that PEEX disallows consecutive negations in a sequence operator, and allows only one event type in the `THEN` clause.

One of the significant restrictions of PEEX is regarding the addition of events which occured in the past. We discuss this in Section 4.2.2.

The main differences between the described conversion process and the implemented process is the way in which negations are handled. Since it is understood that `NOT EXISTS` clauses can cause severe performance problems, we have decided to instead use `OUTER-JOINS`. The `NOT EXISTS` clause can be simulated by a left outer join operation followed by a selection of tuples whose right hand side (i.e. the attributes from the right table) consists of only `nulls` (sometimes known as an anti-semi-join). The following is what the current implementation produces for the earlier example.

```
INSERT INTO ENTERED-ROOM (tagID, roomID, time)
SELECT I1.tagID, 'room555', I3.time
FROM SIGHTING I1 INNER JOIN SIGHTING I3 ON (
   AND SEQ(I1,I3) AND I1.tagID = I3.tagID
   AND I1.antID = 'ant10' AND I3.antID = 'ant20')
   LEFT OUTER JOIN SIGHTING I2 ON (
   SEQ(I1, I2, I3) AND I1.tagID=I2.tagID)
WHERE I2.time IS NULL)
.  .  .
```

By executing the rewritten queries periodically, PEEX extracts new events soon after they occur. For each newly detected event, PEEX computes its probability using the approach presented in Section 3.4.2 and inserts it into the appropriate table.

4.2.2 Events in the Past

A significant challenge in managing and extracting events is handling the addition or correction of past events. The reason for this is that such additions can leave the database in an inconsistent and incomplete state. For example, if a past `Entered-Room` event that occured at time 2 is added at time 10, a number of `In-Business-Meeting` events that should have been captured may be missed due to the late insertion. On the other hand, if a past `Left-Room` event is added late, a number of `In-Business-Meeting` events that should have been discarded are nonetheless included due to the late insertion.

One solution to this problem, is to rerun the event extraction from time t every time a past event is inserted at time t. This strategy can have devastating implications on the performance of the system.

The other extreme is to disallow the insertion of past events completely. Although this solves our problem, it eliminates many reasonable event definitions. This is especially the case since the time of events are defined using normal distributions because all events with a time variance of greater than 0 have some probability to have occured in the past.

PEEX takes a more moderate approach. Firstly, it allows the addition of events which have happened within the current time window. More precisely, an event is accepted if the probability that it occured before the time window is less than ϵ, an application

defined lower bound on probabilities. Such a restriction prevents the two problems listed above because we already ensure the order in which events are generated within a time window causes no inconsistencies (by using the EDG).

The drawback of this approach is that if the time window is only five seconds, we restrict the addition of events to only the last five seconds. This provides us almost no benefit in comparison to disallowing past events completely if our time window is small. However, a neat solution to this is to have several processes running PEEX, each with a different time window size.

For example, one instance can run PEEX with a time window of five seconds providing the user with real time information on events but low accuracy. Another instance uses a time window of ten seconds providing slightly higher accuracy. Another uses a time window of one minute, of one hour and finally of one day. Every time an instance of PEEX with a larger time window finishes processing one window of events, the past events from the instance with a smaller time window can be discarded and replaced with the more accurate events. This provides users with a flexible tradeoff between latency and accuracy. Since we replace events generated by instances with smaller time windows, at the end of the day, the events captured are quite accurate.

This approach is not yet completely implemented in the current prototype of PEEX. Currently, PEEX allows events in the past but does not reprocess them in order to correct or add new events.

4.3 Confidence Learner

To instruct the system to learn event confidences using RFID data collected within a specified time-range, the user issues the command: `LEARN [EventType]`. At that time, the Confidence Learner, learns or updates the confidence values for the given type of event. Unless the global variables are explicitly changed, the time period for which the system learns confidences is set once at the initialization of the PEEX.

Since events are defined decoratively, the *same* event definition can be used both for the event definition and for learning. To learn confidences, the Confidence Learner rewrites event definitions into queries using the same algorithm as the Event Detector with two exceptions. First, the queries need not be incremental as they will process the whole training data at once. Instead, they only restrict the timestamp of the events to be within the training data time-range.

Second, the Confidence Learner actually needs to extract two sets of events: all events that match the definition and all events that match the definition and also have a corresponding high-level event. The ratio of the two sets, grouped by the appropriate attributes, gives the desired confidence value. The Confidence Learner uses a similar approach to compute confidences for partial events. At the end, the Confidence Learner updates the confidence tables.

The example in Figure 12 demonstrates the intricacy of the SQL generated. It populates the confidence tables for `Entered-Room event`. The basic idea is that the temporary result A in the query calculates the numerator for each set of attributes whereas B calculates the denominator value. Each only looks at the data between the times specified for `min-timestamp-training` and `max-timestamp-training`. The numerator is the number of sets of events that match the definition and have a corresponding high-level event, whereas the denominator is simply the number of sets of events which match the definition. For time attributes, since the distribution function is always the normal distribution, the Confidence Learner only needs to compute the mean and the variance from the historical data.

5 Evaluation

We evaluate our approach in two phases. In the first phase we study the feasibility of using PEEX to detect and generate events in near real-time. In the second phase we evaluate and study the recall and precision offered by PEEX in comparison to a deterministic algorithm.

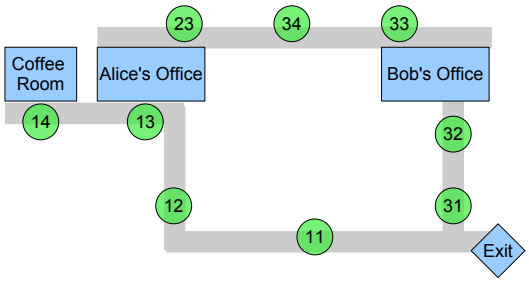


Figure 8: **Test scenario setup.**

EVENT	DESCRIPTION
SIGHTING	Tag id X was sighted at antenna A (deterministic primitive event).
ENTERED-OFFICE	X was sighted at antennas A and B (where A and B are two adjacent antennas with B being the nearest antenna to an office).
LEFT-OFFICE	X was sighted at antennas A and B (where A and B are two adjacent antennas with A being the nearest antenna to an office).
LEFT-BUILDING	X was sighted at antenna 3.2 then at 3.1 with no sightings in between (deterministic event).
MEETING	If X and Y are seen entering the same office with their notebooks, with the first person not leaving before the second person gets to the office.
ENCOUNTER	If X and Y are seen at the same antenna within 3 seconds of each other. Confidence table is manually defined.
COFFEE-ENCOUNTER	If X and Y ENCOUNTER each other and both have their coffee mugs. Confidence table is manually defined.

Table 1: Test scenario events.

5.1 Performance

In order to test PEEX and to study its feasibility we generated synthetic data for a realistic scenario in an office environment. In this section we first explain the scenario, and the events that occur in the scenario. This is followed by a description of the simple measurements we made to study the feasibility of the approach.

5.1.1 Testing Scenario

Our testing scenario consisted of two people named Alice and Bob and their belongings. There are four locations, 'Alice's Office', 'Bob's Office', 'Coffee Room' and the 'Exit', and nine antennas. Figure 8 portrays the layout of the locations and antennas.

The scenario we imagine proceeds as follows. Alice comes into work (through the Exit) with her purse, followed by Bob with his backpack. During the first few hours of the day, Bob comes to Alice's office hoping for a meeting, but each time he misses Alice who goes to the coffee room each time he visits. Finally when Alice discovers that Bob had visited her, she visits his office with her notebook and they have their first meeting for the day. The next meeting occurs soon after in Alice's office when Bob enters her office with his notebook.

During the day, they each make trips to the coffee room and even run into each other once. Later in the day, they both visit each other's office but miss each other. Finally, near the end of the day, Alice visits Bob's office with her notebook. They are both in the office with their notebooks, however, they do not have a meeting. They spend fifteen minutes together and Alice leaves the building. Later, Bob leaves the building.

The events we consider are listed in Table 1. For the training data, the first six types of events are generated. For the testing data, only `Sighting` events are generated and the rest are detected and generated by PEEX. Therefore, confidence tables can be learned for the second event through to the sixth event. The confidence tables for `Encounter` and `Coffee-Encounter` events are populated manually.

5.1.2 Measurements and Results

We tested and made measurements on the test data generated for our example scenario above. The `Sightings` table we worked on had about four hundred tuples.

In order to study the feasibility of our approach we measure the time it takes to detect and generate the events. Listed under the second column in Table 2 are the times it takes to execute the SQL update statements which populate the corresponding confidence table for each event definition. This table includes all those events from our example scenario which do not have their confidence table populating manually.

EVENT	Conf SQL	Event SQL for all	Event SQL for window
ENTERED-OFFICE	663 ms	227 ms	15.2 ms
LEFT-OFFICE	702 ms	197 ms	10.2 ms
LEFT-BUILDING	19 ms	3 ms	2 ms
MEETING	35 ms	58 ms	11 ms

Table 2: Times to execute various SQL statements generated by PEEX.

It is important to note that these times do not include the compile and parse times of the generated SQL. Although these have significant costs the first time the query is run, this time decreases significantly, usually to 1 ms, in subsequent executions. It becomes clear from this table that the running time of the SQL generated depends heavily on the size of the tables involved. For example, since there are many, fifty-six times, more `Entered-Office` events than `Meeting` events, the time for populating the `Entered-Office` event's confidence table is much, eighteen times, greater than the time for populating the `Meeting` event's confidence table.

The more important measurement, however, is the time it takes to detect and generate composite events. This query, unlike the confidence table query, is executed at every time window. Therefore, execution time is more critical. In our experiments we first measured the average time to execute the event SQL over all the data (i.e. without any restrictions on the time of the lower level events) and also the event SQL over one time windows. In Table 2, the third and fourth columns show the results of these measurements. The time window size we used was 5000 ms and the time given in the fourth column is the average over ten runs of the event SQL over ten consequent time windows.

In order to understand what the above results imply in a large scale system, we estimate the time these event SQL statements would take to execute over a larger `Sighting` table. Let's consider a deployment with fifty antennas and 500 tags. Let's assume that each tag is sighted with probability 0.5 and that the read rate for the antennas is one read per second. Now, across a 5000 ms time window, we would expect a total of 2500 `Sighting` events. In our current testing scenario, it takes a total of 485 ms to process a `Sighting` table of only about 400 tuples. Assuming that there is a linear scale up, it would take 3031.25 ms to process one time window of size 5000 ms in our assume deployment. Under these assumptions, our approach is feasible for a moderate-scale deployment of fifty readers and 500 tags.

5.2 Effectiveness

In this section, we evaluate the effectiveness and accuracy of PEEX through a series of experiments on real RFID traces collected on a small antenna deployment (on one floor of our building).

For these experiments, we collected data as a person named Alice who carried a personal RFID tag and also tagged her keys and her purse. In our setup, Alice leaves the building only to go to class (without her purse most of the time), go to lunch (with her purse), or go home (with her purse as well). We made Alice leave the building 20 times, carrying different belonging with her. Hence, we generated 12 `LEFT-FOR-CLASS`, 2 `LEFT-FOR-LUNCH`, and 6 `LEFT-FOR-HOME` events. These events comprise 41 `LEFT-THE-BUILDING` events: 20 for Alice, 13 for her keys and 8 for her purse.

Table 3 summarizes the constraints and event definitions that we use in the evaluation. We use constraints in these experiments, as we believe that the complete PEEX system will have the functionality to handle such constraints. Integrity constraints are further explained in [25].

For each constraint and each event, we indicate both its real confidence (*i.e.*, the fraction of time that it held true) and the confidence computed by the Confidence Learner[4]. The confidences learned are sensitive to the quality of the data. For example, although the confidences learned for the last two constraints are reasonably close to the true confidence, the learned confidence for the `Containment` constraint is more than 10 times smaller than the true confidence. The explanation for this, however, is simple. The tag on the purse is an old, worn out tag, and was hardly sighted (only 11 times out of the true 56 times). Due to the lack of evidence, the confidence of the constraint remained low.

[4]We used half of our traces for confidence learning and half for the actual event detection and data cleaning.

EVENT	DESCRIPTION
SIGHTING	Tag id X was sighted at antenna A (deterministic primitive event).
WALK-IN-HALLWAY	X was sighted at antenna 3.1, 3.2, 3.3, and 3.4 in sequence (deterministic event).
LEFT-THE-BUILDING	X was sighted at antenna 3.2 then at 3.1 with no sightings in between (deterministic event).
LEFT-FOR-LUNCH	X and X's purse are leaving the building at approximately the same time.
CONSTRAINT	**DESCRIPTION**
Containment	If X is sighted at antenna A and X is usually contained in Y, then Y must also be sighted at antenna A. For example, if keys are sighted, then the purse must also be present. Real confidence: 0.46. Learned confidence: 0.04
Pairs	If tag X,Y are sighted at antenna A together, and later tag X is sighted at antenna B then tag Y must also be sighted there. For example, if Alice is walking with her purse, and later the purse is sighted, the Alice should be there too. Real confidence: 0.98. Learned confidence: 0.69
InBetween	If X is sighted at antenna A and later at antenna C then X must also be sighted at antenna B (if A is adjacent to B which is adjacent to C). Real confidence: 1. Learned confidence: 0.7

Table 3: Experimental events and constraints.

In the rest of this section, we present three sets of experiments. First, we look at event extraction from raw data and show the benefits of using partial events. Second, we look at using constraints to clean raw data, comparing deterministic and probabilistic cleaning. Finally, we evaluate the combined benefits of probabilistic cleaning and probabilistic event extractions.

5.3 Detecting Partial Events

In this section, we study event detection performance when high-level events are extracted directly from dirty, raw data. We show how the `WITH m FALSE NEGATIVE/POSITIVE` clauses improve detection rates.

We define recall as the number of detected events divided by the total number of events. We measure recall for the deterministic `WALK-IN-HALLWAY` event using both deterministic and probabilistic extraction. For the latter, we include a `WITH k FALSE NEGATIVE` clause in the event definition, with k varying from 1 to 3. Figure 9 presents the results. The curve labeled "54%" corresponds to our experimental data (because we observed an approximate overall error-rate of 54%). Without allowing any false negatives recall is below 25%. By allowing $k = 1$ false negatives, recall increases to over 50%, and continues to increase with k.

As a comparison, we compute the theoretical detection rates for different error rates. For $k \leq n$, where n is the number of raw events that compose the complex event, recall is given by: $p^n + \binom{n}{1}p^{n-1}(1-p) + \ldots + \binom{n}{k}p^{n-k}(1-p)^k$ where $p = (1 - error\ rate)$. The results show that tolerating even one false negative increases recall by 25% or more depending on error rate, event doubling recall once error rates exceed 25%.

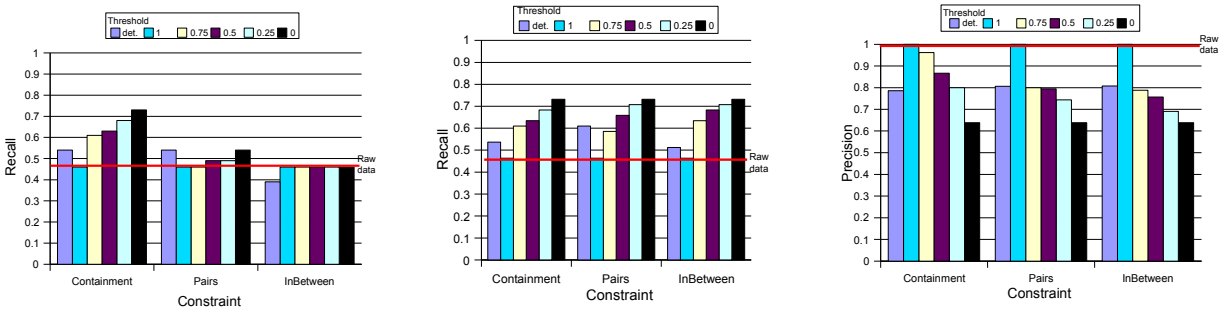
Allowing false positives rather than false negatives in event detection similarly improves performance but for events that require the non-occurrence of some tag reads. We omit these results due to lack of space.

The increased recall comes at the expense of precision, defined as the number of correctly detected events over all detected events. As an extreme example, a single sighting of a person by the elevator would match a large number of events: `LEFT-THE-BUILDING`, `GOT-COFFEE`, `GOT-MAIL` event, etc., but only one of these events would be correct. The precision depends on the level of ambiguity in the data; precision is worse when more events are similar. For example, if three out of four tag readings in our experiment were shared between two events, and both events occurred with the same probability, the precision of event detection with 1 false negative would have been 87.5%. However, in this case, PEEX would have used the historical confidence and would have labeled detected events with a 0.5 probability, signaling to the application the uncertainty of these events.

Key finding: Given the unreliable nature of RFID, PEEX must allow at least a small number of false negatives and false positives during event detection to achieve usable success rates, even if doing so decreases event detection precision.

5.4 Cleaning

In this section, we evaluate the performance of deterministic and probabilistic data cleaning based on



(a) Independent cleaning (b) Cumulative cleaning (c) Cumulative cleaning

Figure 11: **Precision and recall for `LEFT-THE-BUILDING` events extracted from cleaned `SIGHTING` data.** Data is either cleaned by one constraint at a time or it is cleaned cumulatively by multiple constraints. PEEX offers applications a flexible trade-off between precision and recall.

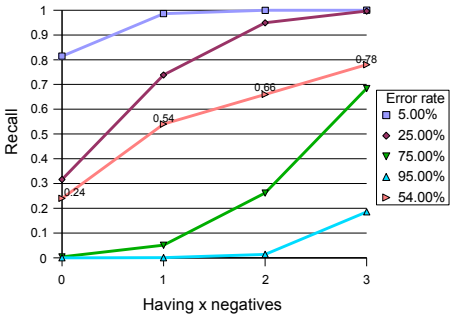


Figure 9: **Event recall from dirty, raw data**. Experimental results shown with 54% error-rate curve. Other results are analytical.

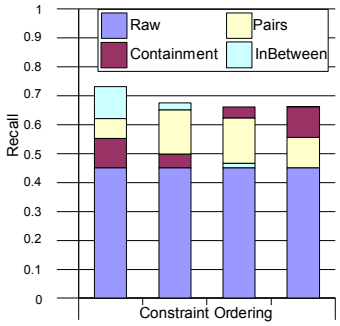


Figure 10: **Effect of constraint order**

integrity constraints. We apply the cleaning to the primitive, deterministic events from the `SIGHTING` table.

We first measure the recall and precision of the raw data. Our experiment includes 281 events where a tagID should have been sighted by an antenna. In the `SIGHTING` table, however, some events are missing and some events appear as bursts of RFID tag reads. We thus compute recall as follows. Every time a correct tag reading is produced within 2.5 seconds of a true event, we consider the event to have been successfully recorded. We use a 2.5 second bound because, although we know exactly how many real events occurred, and in what order, we must approximate the exact time when missing events occurred. The raw data has a recall of only 0.452.

To measure precision, we consider as false positives all tag readings that occur at least 2.5 seconds away from a real event which shares the same `tagID` and `antID` values. Because antennas produce large numbers of tag reads in short periods of time, we compute one false positive per antenna, per tagID, and per burst of tag reads. We approximate a burst of tag reads with a 5 second time-window. This window size captures well the bursts of readings in our experiments. The raw data had a precision of 1. As we show below, cleaning can add false positives.

To compare deterministic and probabilistic data cleaning, we measure the recall and precision they achieve. For probabilistic cleaning, we compute these values separately for events with probabilities above different thresholds. We focus on the `Containment` constraint. For thresholds 1, 0.75, 0.5, 0.25 and 0, the resulting recall values were 0.45, 0.45, 0.52, 0.55, and 0.63 respectively. The precision values were 0.93,

0.74, 0.75, 0.63 and 0.6.

Under deterministic cleaning, we assume that all constraints are always valid. Deterministic cleaning thus aggressively adds tuples resulting in the same post-cleaning recall as the probabilistic technique for events with probability > 0. A more conservative strategy would be to only apply deterministic constraints. In that case, the recall would be equal to that of probabilistic events with probability 1.0. The deterministic approach thus offers a binary trade-off: favor recall or precision. In contrast, the probabilistic approach offers a finer-grain trade-off in the form of a threshold on event probabilities. For example, letting the threshold drop from 1.0 to 0.25 (after cleaning with the `Containment` constraint) improves the recall from 0.45 to 0.55, but drops the precision from 0.93 to 0.63.

Interestingly, for the *Containment* constraint, probabilistic cleaning (recall of 0.63) outperforms deterministic cleaning (recall of 0.53). Indeed, for the deterministic constraint, we simply indicated, through a helper table, that 'keys' were contained in a 'purse'. For the probabilistic technique, we specified a generic *Containment* constraint for *any pair* of objects, expecting the Confidence Learner to find the purse-keys relationship. Instead, the learner found several other such containment relationships, computed their confidences from history, and used them to improve recall.

Finally, Figure 10 shows the impact of constraint ordering on recall. In this experiment, each constraint cleans the data already cleaned by another. The results show recall for events with probability ≥ 0.5. Applying multiple constraints improves recall but the values depend on constraint order. This problem affects both deterministic and probabilistic cleaning. Currently, PEEX applies constraints in the order in which they have been defined.

Key finding: Deterministic data cleaning must always ignore or always apply constraints, leading to either bad recall or bad precision. In contrast, probabilistic constraints enable applications to choose their preferred trade-off between these two important properties.

5.5 Detecting Events over Cleaned Data

Finally, we study the performance of composite `LEFT-THE-BUILDING` event detection from cleaned data. Once again, we compare the deterministic and probabilistic approaches.

For the probabilistic approach, we extract events without tolerating false negatives and false positives (*i.e.*, no `WITH FALSE POSITIVE/NEGATIVE` clauses) to show the effects of data cleaning and event ambiguity.

Figure 11(a), shows the recall of the high-level events after the raw data has been cleaned by each constraints individually. The recall after the `Containment` constraint shows, as expected, that a lower threshold implies a higher recall; as we decrease the threshold we include events with lower probability in addition to the previously counted events.

More interestingly, contrary to the results from Section 5.4, the recall after deterministic cleaning is not equal to the recall after probabilistic cleaning with threshold 0; it is lower. The reason is that we are not looking at primitive events but at the composite `LEFT-THE-BUILDING` event which corresponds to pairs of sightings with *no sightings in between*. This means that the event is non-monotonic and therefore, if the `Containment` constraint causes tuples to be inserted in the wrong place or time, this may actually cause some `LEFT-THE-BUILDING` events to not be detected. However, with probabilistic cleaning and probabilistic event detection, since we insert tuples with probability usually less than 1, the events are still detected but with lower probabilities. The contention between the non-monotonic `LEFT-THE-BUILDING` event and deterministic cleaning is also exemplified in the `InBetweeen` constraint which actually lowers the recall to 0.39. Hence, deterministic cleaning can sometimes lower recall of complex events.

Figure 11(b) shows the recall increase, as we incrementally clean the data with one constraint followed by another. This effect is similar to the effect we measured for raw data cleaning. An interesting anomaly, however, is that the deterministic `InBetween` constraint actually worsens the recall. This is again due

to the non-monotonicity of the `LEFT-THE-BUILDING` event.

Figure 11(c) shows precision results. For deterministic cleaning, the precision depends only on how often the constraint holds in practice because deterministic cleaning always applies the constraint. If the constraint always holds, the precision is one, but if the constraint holds only a fraction of the time, the precision goes down accordingly. In contrast, data precision with probabilistic-constraint cleaning also depends on the training data that was used to populate the confidence tables. The precision then drops for events with increasingly lower probabilities. Because these probabilities are visible to applications, however, the latter can choose their preferred trade-off between precision and recall. For example, an application can achieve a recall of 0.63 with precision 0.79 by ignoring events with less than 0.75 probability. The resulting precision is only 0.02 lower than the precision offered by deterministic cleaning which offers a recall of only 0.51.

Key finding: Because both event-detection confidence and constraint confidence affect the probability of an event, PEEX offers a flexible trade-off between recall and precision. If an application examines high-probability events only, it gets high precision but lower recall. By lowering the threshold, an application can improve the desired recall at the expense of precision. These probabilities capture both the uncertainty due to event ambiguity and the uncertainty due to data errors.

6 Related Work

RFID data management. Several techniques have recently been proposed for compactly representing, summarizing, and efficiently accessing RFID data [19, 21]. These techniques, however, do not perform any event detection. They only focus on information aggregation and compact data representation.

Event detection. There exists a rich literature on event detection and event processing systems. Active databases support the specification of event-condition-action rules [27], with sophisticated event definitions [1, 6, 18], Publish-subscribe systems have started to propose stateful and expressive languages [12, 24]. Recent efforts are investigating extracting complex events specifically from sensor and RFID data [31, 34, 39]. All these systems, however, are deterministic: they ignore event ambiguity and possible input data errors during event extraction.

The Data Furnace project at Berkeley [17] has similar goals to ours, but is investigating a complementary approach to the problem. Unlike our approach, the Data Furnace project envisions using statistical learning techniques to build, maintain, and run inferences over probabilistic models that capture correlations across primitive events.

Sensor and RFID data cleaning. Several techniques have recently been proposed for RFID and sensor data cleaning. The Extensible Sensor stream Processing (ESP) framework [22], part of the HiFi project [16] allows a user to specify, in the form of declarative queries, the sequence of algorithms that the system should use to clean the data. Similarly, Rao *et al.* [29] enable users to specify combinations of patterns over RFID data streams and matching cleaning actions that either delete or keep some of the data. These approaches work well in many scenarios where the user can predict the types of errors that will occur and the appropriate algorithms to correct them, and errors can be cleaned deterministically. More recently, Khoussainova *et al.* [25] proposed to use integrity constraints and a probabilistic model to clean sensor data. Their scheme generates missing tuples and uses entropy maximization to assign probabilities to tuples. PEEX is based on similar principles: it uses integrity constraints to clean the data and a probabilistic data model. Our approach, however, is much more sophisticated: PEEX supports extracting high-level probabilistic events from low-level data and can interleave event extraction and cleaning steps; PEEX also exploits history to clean the data more accurately.

Earlier work has also proposed simple low-level cleaning mechanisms that average measurements within a short time-window [23] and across a group of sensors covering the same area [22]. PEEX could leverage these techniques.

Chawathe *et al.* [7] propose to perform various inferences on RFID data to recover from input data

errors. The envisioned techniques, however, are specific to the supply-chain management domain. In our system, such specific rules could be represented with integrity constraints.

Deshpande *et al.*, [13, 14] propose to handle input errors and inaccuracies by building a probabilistic model of the spatial and temporal correlations between values produced by different sensors. The model then serves to produce approximate answers to queries, predict missing values, and identify outliers. The main challenge of the approach lies in selecting, building, and maintaining appropriate models. The proposed models are also inappropriate for RFID data where events correspond to combinations of observations with specific order and time constraints.

Probabilistic databases. Probabilistic databases have a long history and have mostly focused on the data model and query evaluation approaches. The data model we use for probabilistic composite events corresponds closely to Barbara *et al.*, [3]. Other researchers too have recently used variations on this model. Widom [38] calls tuples with probability < 1 *may-be tuples*, and alternative tuples *or-tuples*; Green and Tannen [20] call this model *pc-tables*, while Dalvi *et al.*, [8] call them *disjoint or independent* tuples. The complexity of query evaluation has been shown in [9] to be #P-hard for queries with duplicate elimination: our queries (e.g. used in the definition of composite events) do not perform duplicate elimination. Probabilistic temporal databases have been introduced in [11] but they use a semantics based on probability intervals, which is different from ours.

Bertossi and Chomicki [4] describe a framework in which queries can be answered over inconsistent databases: the database violates some integrity constraints, but the user still wants to evaluate queries over the database. Andritsos *et al.*, [2] extend this approach to a probabilistic framework, in which the repairing tuples are associated some probabilities. Our approach follows this line of research, in that the RFID data can be viewed as a database violating the constraints given by the users, but our constraints are much more complex than [2] (which only consider key constraints), and have confidences.

7 Conclusion

RFID data enables a new class of applications, but requires the management of often erroneous data and ambiguously defined high-level events. In this paper, we presented an approach that uses a probabilistic model to cope with input data errors and event ambiguities. The approach also applies probabilistic constraints to improve data quality.

We presented PEEX, an implementation of our approach, and several experimental results showing that PEEX offers a high recall for both low-level and high-level events, outperforming deterministic techniques. We showed that PEEX performs at its best when using both probabilistic cleaning and probabilistic event detection. Improved event recall comes at the expense of precision, but PEEX provides applications a flexible trade-off between these two important properties. With PEEX, applications can simply ignore events with a probability below their desired threshold.

The work presented in this paper also presents many challenges that we are currently addressing or planning to address. The first challenge is that of handling constraints. Since integrity constraints provide a wealth of information regarding the dependencies between events, it would be of great benefit to include such capabilities into an RFID data management system. Two distinct ideas we have on handling integrity constraints lie in either using constraints to clean data prior to generating events or in taking into consideration the integrity constraints when answering queries over the data. The implications of the two directions are not clear and this challenge deserves much attention.

Another direction for future work is to extend PEEX to handle other probability distributions for continuous attributes such as the uniform, Zipf and other common distributions. This requires an understanding of how to manipulate these distributions both within one type of distribution and across different types of distributions.

Other issues that we plan to investigate are the drawbacks of the independence assumptions we make, the feasibility of our approach on a large-scale deployment, the issues pertaining to privacy,

the challenges of integrating PEEX with a stream processing engine and extending a stream processing engine to handle probabilistic data.

Our vision is a unified framework for managing and maintaining RFID data effectively. We hope for a framework that can be used by a plethora of applications, each extracting its events either from raw data or from other abstract events, previously defined by this application or even by other related applications. We view the work in this paper as a critical step towards achieving this vision.

References

[1] R. Adaikkalavan and S. Chakravarthy. SnoopIB: Interval-based event specification and detection for active databases. In *Proc. of Advances in Databases and Information Systems (ADBIS)*, 2003.

[2] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases. In *Proc. of the 22nd International Conference on Data Engineering (ICDE)*, 2006.

[3] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.

[4] L. Bertossi and J. Chomicki. Query answering in inconsistent databases. In G. S. J. Chomicki and R. van der Meyden, editors, *Logics for Emerging Applications of Databases*. Springer, 2003.

[5] G. Borriello, W. Brunette, M. Hall, C. Hartung, and C. Tangney. Reminding about tagged objects using passive RFIDs. In *Proc. of the 6th Conference on Ubiquitous Computing (Ubicomp)*, Sept. 2004.

[6] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proc. of the 20th International Conference on Very Large DataBases (VLDB)*, 1994.

[7] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, , and S. Sarma. Managing RFID data. In *Proc. of the 30th International Conference on Very Large DataBases (VLDB)*, 2004.

[8] N. Dalvi, C. Re, and D. Suciu. Query evaluation on probabilistic databases. *IEEE Data Engineering Bulletin*, 29(1):25–31, 2006.

[9] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. of the 30th International Conference on Very Large DataBases (VLDB)*, Toronto, Canada, Sept. 2004.

[10] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[11] A. Dekhtyar, R. Ross, and V. Subrahmanian. Probabilistic temporal databases, I: algebra. *ACM Transactions on Database Systems (TODS)*, 26(1):41–95, March 2001.

[12] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White. Towards expressive publish/subscribe systems. In *Proc. of the 10th International Conference on Extending Database Technology (EDBT)*, 2006.

[13] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-based approximate querying in sensor networks. *The VLDB Journal*, 14(4), 2005.

[14] A. Deshpande, C. Guestrin, and S. R. Madden. Using probabilistic models for data management in acquisitional environments. In *Proc. of the Second Biennial Conference on Innovative Data Systems Research (CIDR)*, Jan. 2005.

[15] C. Floerkemeier and M. Lampe. Issues with RFID usage in ubiquitous computing applications. In *Proc. of the 2nd International Conference on Pervasive Computing (Pervasive)*, Apr. 2004.

[16] M. J. Franklin, S. R. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong. Design considerations for high fan-in systems: The HiFi approach. In *Proc. of the Second Biennial Conference on Innovative Data Systems Research (CIDR)*, Jan. 2005.

[17] M. Garofalakis, K. P. Brown, M. J. Franklin, J. M. Hellerstein, D. Z. Wang, E. Michelakis, L. Tancau, E. Wu, S. R. Jeffery, and R. Aipperspach. Probabilistic data management for pervasive computing: The Data Furnace project. *IEEE Data Engineering Bulletin*, 29(1), 2006.

[18] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model & implementation. In *Proc. of the 18th International Conference on Very Large DataBases (VLDB)*, 1992.

[19] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and analyzing massive RFID data sets. In

Proc. of the 22nd International Conference on Data Engineering (ICDE), 2006.

[20] T. J. Green and V. Tannen. Models for incomplete and probabilistic information. *IEEE Data Engineering Bulletin*, 29(1):17–24, March 2006.

[21] Y. Hu, S. Sundara, T. Chorma, and J. Srinivasan. Supporting RFID-based item tracking applications in Oracle DBMS using a bitmap datatype. In *Proc. of the 31st International Conference on Very Large DataBases (VLDB)*, 2005.

[22] S. Jeffery, G. Alonso, M. J. Franklin, W. Hong, , and J. Widom. Declarative support for sensor data cleaning. In *Proc. of the 4th International Conference on Pervasive Computing (Pervasive)*, Mar. 2006.

[23] S. R. Jeffery, M. Garofalakis, and M. J. Franklin. Adaptive cleaning for RFID data streams. In *Proc. of the 32nd International Conference on Very Large DataBases (VLDB)*, Sept. 2006.

[24] Y. Jin and R. Strom. Relational subscription middleware for Internet-scale publish-subscribe. In *Proc of the 2nd International Workshop on Distributed Event-Based Systems (DEBS)*, 2003.

[25] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *Proc. of the Fifth International ACM Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*, June 2006.

[26] Microsoft Corporation. Microsoft sql server. `http://www.microsoft.com/sql/default.mspx`, 2007.

[27] N. W. Paton and O. Diaz. Active database systems. *ACM Computing Surveys*, 31(1):63–103, 1999.

[28] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proc. of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCom)*, Aug. 2000.

[29] J. Rao, S. Doraiswamy, H. Thakkar, and L. S. Colby. A deferred cleansing method for RFID data analytics. In *Proc. of the 32nd International Conference on Very Large DataBases (VLDB)*, Sept. 2006.

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

[31] S. Rizvi, S. R. Jeffery, S. Krishnamurthy, M. J. Franklin, N. Burkhart, A. Edakkunni, and L. Liang. Events on the edge. In *Proc. of the 2005 ACM SIGMOD International Conference on Management of Data*, 2005. (System demonstration).

[32] M. L. Songini. Wal-Mart details its RFID journey. ComputerWorld. `http://www.computerworld.com/industrytopics/retail/story/0,10801,109132%,00.html`, Mar. 2006.

[33] V. Stanford. Pervasive computing goes the last hundred feet with RFID systems. *IEEE Pervasive Computing*, 2(2), Apr. 2003.

[34] F. Wang and P. Liu. Temporal management of RFID data. In *Proc. of the 31st International Conference on Very Large DataBases (VLDB)*, 2005.

[35] R. Want. The magic of RFID. *ACM Queue*, 2(7), Oct. 2004.

[36] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems (TOIS)*, 10(1), 1992.

[37] E. Welbourne, M. Balazinska, G. Borriello, and W. Brunette. Challenges for pervasive RFID-based infrastructures. Technical Report 2006-10-03, Department of Computer Science and Engineering, University of Washington, Oct. 2006.

[38] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. of the Second Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 262–276, 2005.

[39] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *Proc. of the 2006 ACM SIGMOD International Conference on Management of Data*, 2006.

```
SELECT  X1.ant1, X1.ant2, E. roomID, avg(E.time_mean - X1.time_mean) as timediff,
var(E.time_mean - X1.time_mean), count(X1.time_mean) as count
FROM  Entered-Room E JOIN
(SELECT  DISTINCT  S1.antID as  ant1,  S3.antID as  ant2,
S1.tagID AS S1_tagID, S3.time_mean AS time_mean
 FROM  Sighting S1 JOIN Sighting S3 ON
  ( S1.tagID = S3.tagID AND S1.antID <> S3.antID AND
  S1.time_mean<=S3.time_mean)
  LEFT OUTER JOIN Sighting S2 ON
  ( S2.tagID = S1.tagID AND S1.time_mean<S2.time_mean
  AND S2.time_mean < S3.time_mean)
 WHERE  (S2.prob IS NULL OR S2.prob < 1.0) AND (S3.time_mean <=
  (select curValue
   from  StreamCleanVariables
   where  varName = ''max_timestamp_training'')
  AND S3.time_mean>=  (select curValue
     from  StreamCleanVariables
   where varName = ''min_timestamp_training'') +
  (select curValue
   from  StreamCleanVariables
   where varName = ''window_size''))
 ) as X1
  ON ( E.tagID = X1.S1_tagID AND X1.time_mean<= E.time_mean)
LEFT OUTER JOIN
(SELECT  DISTINCT  S1.antID as  ant1,  S3.antID as  ant2,
S1.tagID AS  S1_tagID, S3.time_mean AS time_mean
 FROM  Sighting S1 JOIN Sighting S3 ON
  ( S1.tagID = S3.tagID AND S1.antID <> S3.antID AND
  S1.time_mean<=S3.time_mean)
  LEFT OUTER JOIN Sighting S2 ON
  ( S2.tagID = S1.tagID AND S1.time_mean<S2.time_mean AND
  S2.time_mean < S3.time_mean)
 WHERE  (S2.prob IS NULL OR S2.prob < 1.0) AND (S3.time_mean<=
  (select curValue
   from StreamCleanVariables
   where varName = ''max_timestamp_training'')
  AND S3.time_mean>=  (select curValue
   from StreamCleanVariables
   where varName = ''min_timestamp_training'') +
   (select curValue
    from StreamCleanVariables
    where varName = ''window_size''))) as X2
  ON  (X2.time_mean > X1.time_mean AND X2.time_mean <=E.time_mean
      AND X1. S1_tagID = X2.S1_tagID)
  LEFT OUTER JOIN Entered-Room X_E
  ON (X_E.time_mean <E.time_mean AND X_E.time_mean > X1.time_mean
      AND X_E.tagID=E.tagID)
WHERE X2.time_mean is null AND X_E.time_mean IS NULL
GROUP BY X1.ant1, X1.ant2,  E . roomID) A,
(SELECT  S1.antID as  ant1,  S3.antID as  ant2, count(S3.time_mean) as count
 FROM  Sighting S1 JOIN Sighting S3 ON
  ( S1.tagID = S3.tagID AND S1.antID <> S3.antID AND S1.time_mean<=S3.time_mean)
  LEFT OUTER JOIN Sighting S2 ON
  ( S2.tagID = S1.tagID AND S1.time_mean<S2.time_mean AND S2.time_mean < S3.time_mean)
 WHERE  (S2.prob IS NULL OR S2.prob < 1.0) AND (S3.time_mean<=
  (select curValue
   from StreamCleanVariables
   where varName = ''max_timestamp_training'')
  AND S3.time_mean>=  (select curValue
   from StreamCleanVariables
   where varName = ''min_timestamp_training'') +
   (select curValue
   from StreamCleanVariables
   where varName = ''window_size''))
 GROUP BY  S1 . antID,  S3 . antID) B
WHERE A. ant1 = B. ant1 AND A. ant2 = B. ant2;
```

Figure 12: SQL query for populating the confidence table for the ENTERED-ROOM event.