# PEEX: Extracting Probabilistic Events from RFID Data

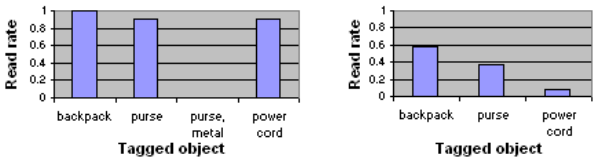
Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu
Department of Computer Science and Engineering
University of Washington, Seattle, WA
{nodira,magda,suciu}@cs.washington.edu

## Abstract

*Radio-Frequency Identification (RFID) technology is increasingly being used to support various industrial and ubiquitous computing applications. Although this technology holds the promise to facilitate many of our every day activities, the noisy and low-level data produced by RFID readers today is extremely difficult to use or comprehend in most but the simplest settings. In this paper, we present PEEX, a system that enables applications to easily define, extract, and manage meaningful* probabilistic high-level events *from low-level RFID data. By using a declarative query language, the system simplifies definitions of new events. By using probabilities, the system copes with the noise and errors in the data and the inherent ambiguity in the event extraction. We have built PEEX as a layer on top of a traditional RDBMS, thus enabling applications not only to detect events but also manage them further as necessary. Through experiments with RFID traces collected on a real, building-wide RFID deployment, we demonstrate the performance and practicality of PEEX.*

## 1. Introduction

In the past several years, Radio Frequency Identification (RFID) technology has become an increasingly popular solution for tagging and wireless identification [34]. Unlike other location systems that tag objects with expensive devices [27, 35], RFID-based systems can rely on inexpensive, passive RFID tags (the typical cost of a tag is approximately 20 to 40 cents [29]). Low-cost tags make it feasible to track large numbers of people and objects, opening the door to many new types of applications. Currently, the main use of RFID technology is in the supply-chain management domain [31]. 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 readers are scattered through the environment. Such deployments hold the promise of enabling many user-oriented applications [3, 32] from simple asset tracking and alerting services, to sophisticated studies of social phenomena and user behavior.




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

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

Exploiting data produced by RFID readers, however, presents significant challenges. The data needs not only to be stored in a database and properly managed [19], but it must also be transformed in sophisticated ways [38]. Indeed, 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". In theory, high-level events are simply spatio-temporal combinations of raw RFID tag sightings or other previously defined events. In practice, however, two important issues make the mapping from RFID data to high-level events challenging.

The first issue is reliability. RFID antennas frequently fail to read tags in their vicinity [13, 20] and nearby antennas can detect the same tags at the same time [20]. Figure 1 illustrates some sample read-rates that we observed in laboratory experiments and in a 2-week pilot study on a small deployment [36]. As the figure shows, failures are frequent, especially on metal objects, or when participants move around freely. Such data errors can cause complex events to go undetected.

The second issue in transforming raw 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.*, Bob is printing a paper or sending a fax. Am-

biguity can make it impossible to determine which from a set of high-level events actually occurred.

Most previous schemes for RFID event detection ignore ambiguity and input data errors [38]. We show that ambiguity and data errors make deterministic event extraction unworkable, leading to event recalls near zero (Section 5.2.1, Figure 6). Schemes that do consider data errors propose to clean the data deterministically before or while processing it [20, 21, 28, 33]. Deterministic cleaning can indeed improve data quality, but rarely cleans away all errors. For example, if a set of keys appear to be in two offices at the same time, it is not always clear which location is correct.

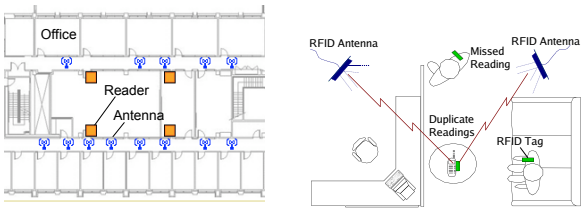
Given the limitations of the deterministic data cleaning and event detection techniques, to enable useful extraction of high-level events in face of uncertainty in the observed data, we propose to use a probabilistic model. More specifically, our contributions are as follows:

1. **Probabilistic event model**. We propose a probabilistic model for high-level events extracted from RFID data. We also propose an extended event language that supports our probabilistic model in a natural way (Section 3).

2. **Probabilistic event extraction system**. We propose a simple technique for extracting probabilistic events by executing SQL queries over a relational database management system (RDBMS). To handle ambiguity and data errors, our approach relies on two key techniques: (1) confidence tables that capture the historical probability that different combinations of tag sightings correspond to a high-level event and (2) partial events, where the system only needs to detect subsets of all specified tag sightings to infer that a higher-level event occurred. We implement this approach in a system called *Probabilistic Event EXtractor (PEEX)*, which we present in Section 4.

3. **Experiments with real RFID data traces**. Finally, we demonstrate the power and practicality of PEEX through experiments with data collected from a real building-wide RFID deployment (Section 5). We show that a probabilistic approach to event detection significantly improves detection rates compared with deterministic techniques. For example, we show in Section 5.2.1 that PEEX can provide a 93% recall for a non-trivial `STARTED-MEETING` event over the 18% recall of a deterministic approach. Additionally, since probabilities are associated with detected events, applications can choose their desired trade-off between detection rate and precision by considering only events above some probability threshold. We also show that our RDBMS-based event detection approach is sufficiently fast to be practical for most ubiquitous computing applications. For example, in Section 5.2.5, we show that the `STARTED-MEETING` event can be detected in near real-time on top of our collected data which consists of more than ten thousand raw events.

Since our system is based on an RDBMS and the output it produces follows standard probabilistic models, existing probabilistic databases [6, 7, 37, 8] could be used to execute queries over our system's output and further manage the extracted events. However, we restrict the focus of this paper on the actual event extraction from RFID data. We do not address query execution over the detected events.




(a) Sample deployment (b) Types of errors

**Figure 2. RFID deployment and errors**

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

To experiment with RFID technology and applications, we have deployed 150 RFID antennas in all hallways in our department building as illustrated in Figure 2(a). RFID readers located in nearby offices poll antennas continuously and send information about detected tags to a back-end database. In our setting, the read-range for each antenna is approximately 2 to 3 meters. 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. Another application, which is the one we use throughout the paper, simply enables users to determine the current location of co-workers. [1] In most cases, users do not want to see low-level information such as "Alice was last seen at antenna 37 at 11:56am" but higher level information such as `STARTED-MEETING(Alice, Bob, Rm303, 11.57am)`. 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 meeting event occurs when Alice and Bob both walk into room 303 together, each with their laptop. 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 laptop has entered the same room and that they have not left the room before the other attendee arrived. We may be even more certain of our conclusion if this pair has a history of meeting together in room 303.

[1] Although paramount, privacy issues are outside the scope of the paper.

This scenario illustrates the main challenges we face. (1) 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 meeting together, Alice may have stopped by to invite Bob to lunch or chat over coffee. (2) 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 and the orientation of tags and antennas [36]. 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.

The high-level events that can be extracted from RFID data are thus probabilistic rather than deterministic in nature. The uncertainty propagates as 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 STARTED-MEETING event will be accordingly lower. In the following sections, we present PEEX, our probabilistic event extraction system, and show how it addresses the above challenges.

# 3. Probabilistic Event Model and Language

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

## 3.1. Probabilistic Event Model

The PEEX probabilistic event model borrows elements from the event model proposed by Demers *et al.* [10] and from recent probabilistic data models [6, 18, 37].

### 3.1.1. Deterministic Events

Although PEEX handles probabilistic events, we first define a deterministic event which is a named tuple of the form: s

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

where EventType is the type name of that event: PEEX stores all events of this type in a relation called EventType. 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 indicating the time when the event occurred. Unlike [10], 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)
ENTERED-ROOM(evID, tagID, roomID, time),
LEFT-ROOM(evID, tagID, roomID, time)
GOT-COFFEE(evID, tagID, time)
STARTED-MEETING(evID, tagID1, tagID2, roomID, time)
FINISHED-MEETING(evID, tagID1, tagID2, roomID, time)
```

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

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 base relation SIGHTING and stored persistently before being processed.

A *primitive event* in PEEX thus corresponds to a tuple in the SIGHTING relation. 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.

Note that tagID plays no special role among an event's attributes, and events may refer to more than one tag (*e.g.*, STARTED-MEETING 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

All events, including those in SIGHTING are probabilistic: they have an associated probability that represents PEEX's confidence in the occurrence of the event. More formally, a *probabilistic event* consists of a type 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 that 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 another location. In this illustration one 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, varied between 0.7 and 2.5 sec-

onds 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 PEEX, we capture this uncertainty by specifying the time attribute independently with a probability distribution function in the form of a histogram. Each bucket corresponds to a small time-window and contains the probability that the event occurs within that window. The size of the buckets and the number of buckets to use for each event are set by the administrator. Assuming that we use only three five-second buckets, the ENTERED-ROOM event for tagID 404 and room 505 could be represented as:

```
ENTERED-ROOM(5394,tag404,room505,295,0.05)
ENTERED-ROOM(5394,tag404,room505,300,0.30)
ENTERED-ROOM(5394,tag404,room505,305,0.05)
```

In general, we expect administrators to use a small number of course-grain buckets (*e.g.*, 30 or 60 second buckets) and most events to fall within a single or a few buckets. For example, ENTERED-ROOM events would almost always occur within 30 seconds of the last antenna sighting.

For simplicity, all primitive SIGHTING events generated by the RFID infrastructure have a probability of 1.0 and a deterministic time.

## 3.2. Probabilistic 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
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 the next section.

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 Figure 3(a). Given raw events SIGHTING(8778, tag432, ant035, 420) and SIGHTING(8778, tag432, ant036, 425) with no SIGHTING for tag432 in between, the system generates a composite event ENTERED-ROOM(238, tag432, Room555, 425).

### 3.2.1. Temporal Aspects of Events

Because our event definition language is primarily based on SQL, we can borrow (and extend) the probabilistic data model described in [6, 18, 37].

Our language also includes two powerful constructs for predicates on event ordering, which we borrow from [4, 38]. The first is SEQ($I_1$, $I_2$, ..., $I_n$). This is a predicate stating that the events $I_1, I_2$, ..., $I_n$ come in this order: *i.e.*, $(I_1.\text{time} < I_2.\text{time}) \wedge (I_2.\text{time} < I_3.\text{time}) \wedge \ldots \wedge (I_{n-1}.\text{time} < I_n.\text{time})$. The second construct is the bang ! in front of a variable in the FORALL clause, which specifies the non-occurrence of the event. For example, the event definition in Figure 3(a) indicates that S2 occurs after S1 and that event S should NOT exist.

### 3.2.2. Expressing Probabilistic Events

Figure 3(a) defines a deterministic event: *i.e.*, its probability is 1.0 (since there is no explicit clause indicating otherwise and the probability of the underlying SIGHTING events is always 1.0), and its time attribute is simply equal to the time of the tag sighting at antenna 36.

Typically, however, a given combination of RFID tag readings can define 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 CONF 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(b), which defines the composite event ENTERED-ROOM. We assume that two consecutive readings of the same tag by antennas 35 and 36 signal that the tag moves towards a cluster of three rooms, most likely towards room 555, but maybe towards 505 or 501. These alternatives are captured in the event definition by assigning different values to the room attribute, with different confidences. Such choices define a joint probability distribution on the room attribute of the new event. Similarly, users could define joint distributions on multiple attributes: *e.g.*, SET ($A_1$ := $v_1$, $A_2$ = $v_2$ CONFIDENCE c | $A_1$ := $v_1$', $A_2$ = $v_2$' CONFIDENCE c'), assigns probability c to values ($v_1$,$v_2$) and probability c' to ($v_1$',$v_2$').

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). 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($A_1$, $A_2$, ..., $A_n$, time-bucket, conf).

Figure 3(c) illustrates the approach, by showing the specification of the ENTERED-ROOM event with a confidence value

```
FORALL SIGHTING S1, !SIGHTING S,
      SIGHTING S2
WHERE SEQ(S1, S, S2)
      AND S1.antID = 'ant035'
      AND S2.antID =  'ant036'
      AND S1.tagID = S2.tagID
      AND S.tagID = S1.tagID

CREATE EVENT  ENTERED-ROOM E
SET E.tagID = S1.tagID,
   E.room  = 'Room555',
   E.time  = S2.time;
```

(a) Deterministic

```
FORALL SIGHTING S1, !SIGHTING S,
      SIGHTING S2
WHERE  SEQ(S1, S, S2)
      AND S1.antID = 'ant035'
      AND S2.antID =  'ant036'
      AND S1.tagID = S2.tagID
      AND S.tagID = S1.tagID

CREATE EVENT  ENTERED-ROOM E
SET E.tagID = S1.tagID,
    ( E.room = 'room555' CONF 0.4|
      E.room = 'room505' CONF 0.3|
      E.room = 'room501' CONF 0.1),
   E.time  = S2.time;
```

(b) No confidence table

```
FORALL SIGHTING S1, !SIGHTING S,
      SIGHTING S2
CTABLE FLOOR5-STATS C
WHERE  SEQ(S1, S, S2)
      AND S1.antID = 'ant035'
      AND S2.antID =  'ant036'
      AND S1.tagID = S2.tagID
      AND S.tagID = S1.tagID
      AND S1.tagID = C.tagID

CREATE EVENT  ENTERED-ROOM E
SET E.tagID = S1.tagID,
   E.room = C.room CONF C.conf
   E.time  = S2.time + C.time-bucket;
```

(c) Using a confidence table

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

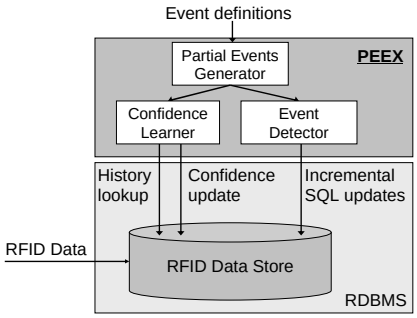



**Figure 4. PEEX software architecture**

```
[01] SELECT S1.tagID, 'Rm555', S2.time
[02] FROM   SIGHTING S1, SIGHTING S2
[03] WHERE  S1.time < S2.time
[04]       AND S1.antID = 'ant035'
[05]       AND S2.antID = 'ant036'
[06]       AND S1.tagID = S2.tagID
[07]       AND NOT EXISTS (
[08]             SELECT *
[09]             FROM SIGHTING S
[10]             WHERE S.time > S1.time
[11]               AND S.time < S2.time
[12]               AND S.tagID=S1.tagID)
[13]       AND S2.timestamp>now()-delta
```

**Figure 5. SQL for detecting** `ENTERED-ROOM` **events.**

computed from a `FLOOR5-STATS` confidence table, which appears in the `WHERE` clause. The schema of `FLOOR5-STATS` is simply `(tagID, room, time-bucket, conf)`: *i.e.*, the confidence depends on the room number and the tag identifier. Note that the time when the event occurred is now also computed from the confidence table.

An important consequence of using confidence tables is that the system is now able to *learn* event probabilities and event times as we discuss in Section 4.2.

# 4. PEEX System

We have designed PEEX as a layer on top of a traditional RDBMS (Microsoft SQL Server [24] in our prototype implementation). This design enables us to demonstrate the benefits of a probabilistic RFID DBMS, while leveraging all the features of an existing RDBMS. As illustrated in Figure 4, the core components of PEEX are the Event Detector, the Confidence Learner and the Partial Events Generator. We now describe these three components.

## 4.1. Event Detector

The Event Detector extracts events specified by the event definitions. 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. For each newly detected event, it inserts a tuple into the appropriate relation and computes its probability appropriately.

### 4.1.1. Extracting Events

To leverage the underlying DBMS, the Event Detector transforms event definitions into SQL queries that it executes each time it runs. Interestingly, we managed to keep this transformation relatively straightforward in spite of the sophistication of our language.

For deterministic events, the transformation requires three key changes to event definitions. First, all negations that appear in the `FORALL` clause are replaced with `NOT EXISTS` clauses. Second, all `SEQ`$(I_1, I_2, ..)$ constructs are transformed into explicit predicates on input event timestamps. As an example, Figure 5 shows the SQL query for the event from Figure 3(a).

The third and final rewrite 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 transforms event definitions into *incremental* queries. These queries look-up only combinations of low-level events where at least one event occurred in the time-window $\Delta$ since the Event Detector's last execution. PEEX uses the `SEQ` construct and all predicates on event times to compute the expected order of the low-level events. If the event definition imposes only a partial order on the lower-level events, PEEX specifies that the maximum timestamp of the events should be within the most recent time window. Line 13 in Figure 5 illustrates this transformation for the case where all events are ordered.

To maintain constant performance, PEEX also 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.

With the above 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 in the `CREATE EVENT` clause.

There are two major changes that we make in order to detect probabilistic events. The first is to replace the not-exists clauses with anti-semi-joins (which in turn requires us to alter other joins too). This allows us to detect, for example, an `Entered-Room` event, $E$, that consists of two `Sighting` events $e_1$ and $e_2$ even if there exists some $e''$ in between the two with a non-zero probability. Of course, the probability of $E$ is lower than had the probability of $e''$ been zero. The second change is due to the addition of confidence tables into event definitions when defining probabilistic events. We need to add the confidence table to the join.

Finally, to avoid the proliferation of very low-probability events, PEEX generates only events that have a confidence higher than some $\epsilon$ threshold defined by the administrator.

#### 4.1.2. Assigning Probabilities to Composite Events

Once it detects an event, the Event Detector must also assign it the appropriate probability. This problem is challenging because the probability of a composite event must take into account the probabilities of the underlying, lower-level events, which can be correlated. For example, a `LUNCH` event could be defined as the sequence of an `IMPROMPTU-MEETING` event between two people followed by a `LEAVING-TOGETHER` event. Let's denote with $p_1$ and $p_2$ the probabilities of these lower-level events. A straightforward approach would be to assume that the events are independent and return $p_1 p_2$ 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* (probabilistic) event. For example, suppose that the `IMPROMPTU-MEETING` event comprises two `LEFT-OFFICE` events followed by an `ENCOUNTER` event (hence $p_1 = p_A p_B p_C$). Suppose now that the `LEAVING-TOGETHER` event depends on an `ENCOUNTER` event followed by two `LEFT-BUILDING` events (with $p_2 = p_C p_D p_E$). It may happen that the two events are correlated because they depend on the same `ENCOUNTER` event and the probability that they both happen is no longer $p_1 p_2$.

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 the `ENCOUNTER` event twice, which PEEX minimizes. PEEX now computes the correct probability $p_A p_B p_C p_D p_E$ (here $p_C$, the probability of the `ENCOUNTER` event, 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.

### 4.2. Confidence Learner

As we discussed in Section 3.2.2, the schema of a confidence table takes the form: `CONF_TABLE(`$A_1$`, `$A_2$`, ..., `$A_n$`, time-bucket, conf)`. If `CONF_TABLE` serves to generate events of type $E$, each entry in `CONF_TABLE` indicates the probability, `conf`, that given the parameters $A_1$ through $A_n$, $E$ occurs within the time bucket. Time buckets are defined relative to the time of the most recent lower-level event. For example, a confidence table may indicate that there is a 20% chance user 444 enters room 505 within thirty seconds of having been sighted at antenna Ant35 followed by Ant36.

To automatically populate these confidence tables, PEEX uses annotated historical data that includes input primitive events and output composite events. Primitive events are generated by RFID readers. To obtain historical composite events, uses must annotate, over some time period, their movements through the building with the corresponding activities. To do so, users typically associate activity labels with the times when the activities occurred (annotation precision affects event detection precision).

The confidence for an event is then determined by two sets: the set of historical composite events that match the event definition and the subset of those events that are also associated with the appropriate label. The ratio of the two sets, grouped by the appropriate attributes (including `time-bucket`), gives the desired confidence value. For the learning process to be accurate, PEEX must thus carefully match event patterns with the appropriate event labels. To do this, PEEX matches each pattern with the closest following event label for the event of interest. If there is a real correlation between the pattern and the label, many historical events occur clustered within one or a few time buckets, yielding a high confidence. In contrast, if the association is spurious, events appear scattered across a large number of time buckets, each one with a probability below $\epsilon$ that PEEX discards.

PEEX learns event confidences using RFID data from a given time-period when the user issues the command: `LEARN [EventType] begin_time, end_time`. If the EventType is omitted, all declared confidence tables are updated. Since events are specified declaratively, the *same* event specification can be used both for the event definition and for learning. The Confidence Learner rewrites event definitions into queries using transformations similar to those used by the Event Detector.

### 4.3. Partial Event Generator

RFID errors, both false negatives (where readings are missed) and false positives (where neighboring antennas detect tags), dramatically impact applications that rely on complex events, because error rates are amplified at each level in the event hierarchy. To illustrate, consider the following STARTED-MEETING event

```
FORALL ENTERED-ROOM E1, ENTERED-ROOM E2, !LEFT-ROOM L,
       ENTERED-ROOM E3, ENTERED-ROOM E4
WHERE  SEQ(AND(E1, E2), L, AND(E3, E4))
       AND  ...
CREATE EVENT STARTED-MEETING E
SET E.tag1 = E1.tagID,
    E.tag2 = E3.tagID,
    E.room = E1.room,
    E.time = max(E3.time, E4.time)
```

The composite event depends on four positive RFID readings E1, E2, E3, E4, and one negative reading L. If each event has an error rate of 15%, the STARTED-MEETING event has an error rate of 56%. If three such composite events are now needed to detect an even higher-level event, that event has an error rate of 91%! Deterministic event detection is thus unworkable in an error-prone environment.

We observe, however, that it is often possible to detect composite events even when some errors occur. For example, if only $E1$, $E2$, and $E3$ are detected, the system might still be able to conclude that a STARTED-MEETING event occurred, although with lower confidence.

PEEX captures this intuition through the use of *partial events*. Given a definition of a composite event consisting of $n$ lower-level events (some may be negated), PEEX detects the composite event as soon as some non-empty subset of the $n$ events occur (or do not occur). In our example, PEEX still detects the STARTED-MEETING event even if up to three of the events E1, E2, E3, E4 are missing and even if L is present. Of course, the more errors occur, the lower the confidence that the high-level event occurs.

To detect partial events, PEEX relies on the Partial Event Generator. Given an event definition that depends on lower level events $E_1, E_2, ..., E_n$, the Partial Event Generator generates a partial event for each subset of events with at least one positive event $E_i$ and no consecutive negated events $E_j, E_{j+1}$. The Partial Event Generator adds each partial event to the system and from there on, these events are handled like regular events, each with its own confidence table. We analyze the performance of partial events in Section 5.2.2.

## 5. Evaluation

In order to evaluate PEEX we collected data for one hour with ten participants in our building-wide RFID deployment. Each participant had with them several tags including their person, keys, laptop, and mug tags. Each was given a schedule that consisted of the times for entering the building, attending meetings, taking lunch breaks and exiting the building. Participants also took coffee breaks and trips to the printer at their discretion. Overall, we collected 11585 SIGHTING events.

### 5.1. Methodology

We looked at the extraction of many events in our experiments including those listed in Section 3. However, for brevity we discuss only the ENTERED-ROOM, LEFT-ROOM, and STARTED-MEETING events.

The ENTERED-ROOM event is defined as in Figure 3(c) with one change. We replace the negated S from the definition with a condition requiring that the two SIGHTING events occur within 30 seconds of one another and at adjacent antennas. We make this alteration for simplicity's sake as the STARTED-MEETING event already demonstrates a negation. The LEFT-ROOM event is defined with a single underlying SIGHTING event, just to get a greater diversity of event definitions. The STARTED-MEETING event is a simplified version of the definition in Section 4.3 without E2 and E4.

Participants labeled the times they traveled between rooms and also the objects that they carried. This information was sufficient for us to label all high level events. We use the labeled data for the first half-hour to populate confidence tables and the rest of the data to measure recall and precision. Given a variable $c$, we say that a generated event $E$ correctly captures a labeled event $E'$ if $E$ is within $c$ seconds of $E'$. The following equations measure recall and precision using this definition.

$$\text{Recall} = \frac{|\{E \in S : \exists E_1 \in T \text{ s.t. } |E_1.time - E.time| \leq c\}|}{|\{E \in S\}|}$$

$$\text{Precision} = \frac{|\{E \in T : \exists E_1 \in S \text{ s.t. } |E_1.time - E.time| \leq c\}|}{|\{E \in T\}|}$$

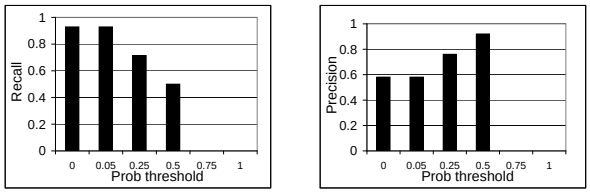
where $S$ is the set of labeled events and $T$ is the set of events generated by PEEX. We use $c$ to cope with the varying degrees of inaccuracy in the labeled data. For example, in the labeled Entered-Room data, the times people input varied from immediately before entering the office to ten seconds after entering. Henceforth, $c$ is 60 seconds.

### 5.2. Results

We present our evaluation results in four parts. First, we compare PEEX to a deterministic approach. Second, we demonstrate the need for and the benefit of using partial events for the ENTERED-ROOM event. Third, we show the effect of using partial-events on a higher-level event STARTED-MEETING. Finally, we study PEEX's ability to detect and generate events in near real-time.

#### 5.2.1. Probabilistic vs. Deterministic Approach

Figure 6 shows the recall and precision achieved by PEEX for STARTED-MEETING events. The results include probabilistic




(a) Recall (b) Precision

**Figure 6. Detecting `STARTED-MEETING` events.**

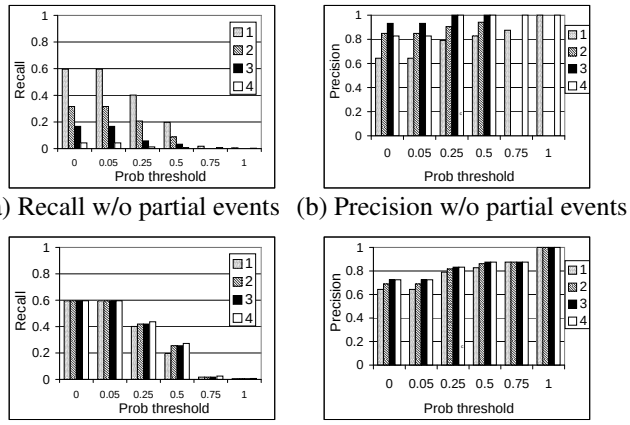
cleaning (see Section 5.2.4). For increasingly high confidence thresholds (on the x-axis), the figure shows the corresponding recall and precision (on the y-axis) for all events detected with higher or equal probability.

We compare PEEX to a deterministic approach. There are two possible deterministic approaches, to extract only events that have occurred with certainty or to generate all events that have any chance of having occurred. These approaches are equivalent to setting the probability threshold to 1 and 0 respectively. It is clear that PEEX can provide a higher recall than the first deterministic approach, from 0% to 93% (when the probability threshold is 0.05). PEEX can also deliver higher precision than the second deterministic approach to applications that require it. Hence, unlike a deterministic approach, because probabilities are associated with each event, PEEX allows applications to choose their desired trade-off between recall and precision by considering only events above some probability threshold.

### 5.2.2. Partial Events

To evaluate the benefits of partial events, we take four definitions of the `ENTERED-ROOM` event, each with 1, 2, 3 or 4 `SIGHTING` events. For a definition containing $x$ `SIGHTING` events, the `ENTERED-ROOM` event is generated when a tag is sighted at $x$ adjacent antennas. The confidence table is also altered to the form `C(`$ant_1$`, `$ant_2$`, ..., `$ant_x$`, tagID, time-bucket, conf)`. Figure 7(a) and (b) show the recall and precision respectively for these four definitions. As expected, without partial events, as $x$ increases, the recall of PEEX decreases due to false negatives (see Section 4.3). The precision increases because the longer the sequence of tag sightings, the more confident we can be that the person is indeed entering a specific office.

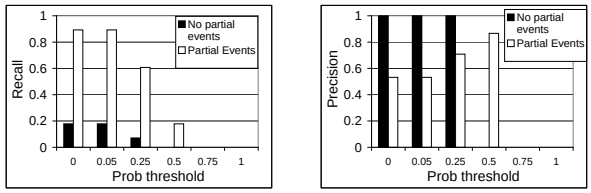
For $x \geq 2$, with partial events (Figure 7(c) and (d)) the recall *increases* as there are now more combinations of `SIGHTING` events that can trigger an `ENTERED-ROOM` event. Thus PEEX has a higher tolerance to missed readings. For example, for $x = 4$ and probability threshold 0.5 the recall increases from 0.8% to 27%. Again, however, the precision drops due to the lower confidence events that PEEX now generates. The drop in precision, however, is small in comparison to the increase in recall. At the 0.5 threshold, precision drops only to 88%. Note that with partial events, precision and recall are similar independent of $x$ because our rooms were placed unambiguously on the corridor. More complex room layouts, or higher-level events (*e.g.*`STARTED-MEETING` must be defined in terms of multiple base events, otherwise they result in low precision.




(a) Recall w/o partial events (b) Precision w/o partial events

(c) Recall w. partial events (d) Precision w. partial events

**Figure 7. Effect of partial events on recall/precision for the `ENTERED-ROOM` event where the event is defined on a varying number underlying `SIGHTING` events**



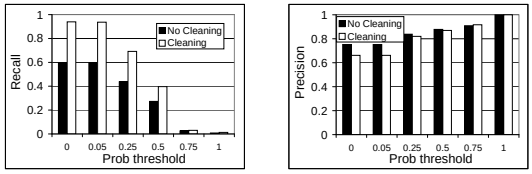

(a) Recall (b) Precision

**Figure 8. Effect of partial events on recall/precision for the `STARTED-MEETING` event.**

### 5.2.3. Higher-level Events

As discussed in Section 4.3, higher-level events are prone to lower recalls due to the error rate amplifying at each level of the event hierarchy. Thus, we also explore the benefits of partial events for the higher-level `STARTED-MEETING` event (Figure 8). In this experiment, `ENTERED-ROOM` events are defined as a sequence of two `SIGHTING` events. We need not rewrite the `STARTED-MEETING` event in terms of `SIGHTING` events because the underlying `LEFT-ROOM` and `ENTERED-ROOM` events can never be correlated.

As the figure shows, partial events offer a higher recall. For example, for probability threshold 0.25 there is about a seven-fold increase in recall. Of course, there is also a decrease in precision. In our example above, the decrease is under two-fold.




(a) Recall (b) Precision

**Figure 9. Effect of data cleaning with only one constraint for the `Entered-Room` event.**

| *Event* | *Without* | *With* |
|---|---|---|
| ENTERED-ROOM | 8 msec | 110 msec |
| STARTED-MEETING | 3 msec | 110 msec |

**Table 1. Running times of PEEX with and without partial events**

#### 5.2.4. Cleaning

In previous work [23], we looked at cleaning sensor data probabilistically through the use of integrity constraints. In this section, we show that the two techniques are complementary and can be integrated to deliver an improved event detection performance. Figure 9 shows the result of applying a single cleaning constraint on ENTERED-ROOM events after they have been detected by PEEX with the use of partial events. The cleaning constraint states that if a person's keys enter an office, then the person also probably enters the office. We learn this constraint and its degree of accuracy from the training data for all pairs of tags owned by the same person. The results show that probabilistic data cleaning significantly improves recall: *e.g.*, for probability threshold 0.05 the recall increases from 0.6 to 0.94 while precision drops only from 0.75 to 0.66. We plan to address the full integration of these two techniques in future work.

#### 5.2.5. PEEX Performance

PEEX operates over a sliding window (5 seconds in our experiment). Every time it runs, PEEX detects events that occurred since its last execution. Our dataset contains a total of 11585 SIGHTING events. Although this is a rate of only 16 new SIGHTING events per time-window, one must remember that older SIGHTING events also participate in the detection of composite events: event detection queries run over a much larger time window, the entire database in our experiment, (see Section 4.1). PEEX simply requires that the most recent lower-level event occurs within the current interval. There were also a total of 792 ENTERED-ROOM events and 56 STARTED-MEETING events.

In order to determine whether PEEX is able to run in near real-time, we measure the average time it takes to detect events within a five second window. Table 1 shows the results for ENTERED-ROOM and STARTED-MEETING events. In this experiment, we do not do any cleaning, but the table shows results both with and without partial events.

Since the time for detection is significantly less than five seconds for each event, we conclude that PEEX is easily able to run in near real-time for a deployment with tens of users. Another important component is the learning process. However, since this process is done offline, the performance is less important. The running time for learning the confidences were 1.58 and 1.71 seconds (without partial events) and 4.02 and 13.67 seconds (with partial events).

## 6. Related Work

**RFID data management**. Several techniques have recently been proposed for managing RFID data [17, 19]. These techniques, however, do not perform any event detection. Instead, they focus on the complementary problem of compactly representing, summarizing, and efficiently accessing RFID data.

**Event detection**. Event detection and processing has previously been addressed in three main research areas: active databases [1, 4, 16], publish-subscribe systems [10, 22], and more recently complex event extraction from sensor and RFID data [30, 38]. In all these systems, however, event detection is deterministic: these approaches ignore event ambiguity and possible input data errors.

The Data Furnace project at Berkeley [15] 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.

Previous work [25, 26] also proposed to build dynamic Bayesian networks to infer human activities from RFID and other sensor data. These systems can infer the most likely activity that a user is performing. In contrast, PEEX extracts the entire set of possible activities. Furthermore, because PEEX is based on an RDBMS, it makes it easy for users to query and manage the generated events.

**Sensor and RFID data cleaning**. Because our goal is to detect events from erroneous and inaccurate RFID data, the area of sensor and RFID data cleaning is complementary to our work. Several techniques have recently been proposed, where the user declaratively specifies either the algorithms to clean the data [20, 14] or patterns over the data with matching cleaning actions [28, 33]. In contrast, our system operates directly on the inaccurate and dirty data, without requiring the user to specify how to clean the data.

In previous work [23], we showed that integrity constraints can serve to clean sensor data probabilistically. As we showed in Section 5.2.4, integrity constraints can easily be integrated into PEEX and can help improve event recall for only a small decrease in precision. Similarly, PEEX can also leverage simple low-level cleaning mechanisms that average measurements within a short time-window [21] and across a group of sensors covering the same area [20].

Chawathe *et al.* [5] 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. Such specific rules could be represented in PEEX with integrity constraints.

Deshpande *et al.*, [11, 12] propose to handle sensor data errors and inaccuracies by building a probabilistic model of the spatial and temporal correlations between values produced by different sensors. In contrast to PEEX, their goal is to answer approximate queries directly over the low-level data rather than extract high-level events from the data.

**Probabilistic databases**. They have a long history, starting with Barbara *et al.*, [2]. Our probabilistic events are similar to *maybe-or* tuples in Trio [37], *pc-tables* in Green and Tannen [18], or *disjoint-independent* tuples in Dalvi *et al.*, [6]. The query complexity on such databases has been studied in [7, 6]. Probabilistic temporal databases have been introduced in [9], but they use a semantics based on probability intervals, which is different from ours.

## 7. Conclusion

RFID technology enables many new types of applications, but requires the management of often erroneous data and ambiguously defined high-level events. In this paper, we presented a probabilistic model and language for high-level events extracted from RFID data. We also presented the design, implementation, and evaluation of PEEX, a data management system that effectively extracts probabilistic events from RFID data using three key techniques: it translates event definitions into SQL queries, it relies on confidence tables to determine the probability of ambiguous events, and it uses partial events to handle data errors.

Through experiments with data from a real RFID deployment, we showed that PEEX offers a better event recall than deterministic techniques. Improved recall comes at the expense of precision, but PEEX provides applications a flexible trade-off between these two important properties: applications can simply ignore events with a probability below their desired threshold.

Our long-term goal is to build an RFID data management system, where a plethora of applications can extract, manage, and possibly share high-level events. We view the work in this paper as an important step towards this goal.

## 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 Conf. (ADBIS)*, Sept. 2003.

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

[3] G. Borriello, W. Brunette, M. Hall, C. Hartung, and C. Tangney. Reminding about tagged objects using passive RFIDs. In *Proc. of the 6th Ubicomp Conf.*, Sept. 2004.

[4] 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.

[5] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, , and S. Sarma. Managing RFID data. In *Proc. of the 30th VLDB Conf.*, Sept. 2004.

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

[7] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. of the 30th VLDB Conf.*, Sept. 2004.

[8] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *Proc. of the 22nd ICDE Conf.*, Apr. 2006.

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

[10] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White. Towards expressive publish/subscribe systems. In *Proc. of the 10th EDBT Conf.*, 2006.

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

[12] A. Deshpande, C. Guestrin, and S. R. Madden. Using probabilistic models for data management in acquisitional environments. In *Proc. of the Second CIDR Conf.*, Jan. 2005.

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

[14] 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 CIDR Conf.*, Jan. 2005.

[15] Garofalakis et. al. Probabilistic data management for pervasive computing: The Data Furnace project. *IEEE Data Engineering Bulletin*, 29(1), Mar. 2006.

[16] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model & implementation. In *Proc. of the 18th VLDB Conf.*, Aug. 1992.

[17] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and analyzing massive RFID data sets. In *Proc. of the 22nd ICDE Conf.*, Apr. 2006.

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

[19] 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 VLDB Conf.*, Sept. 2005.

[20] 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.

[21] S. R. Jeffery, M. Garofalakis, and M. J. Franklin. Adaptive cleaning for RFID data streams. In *Proc. of the 32nd VLDB Conf.*, Sept. 2006.

[22] Y. Jin and R. Strom. Relational subscription middleware for Internet-scale publish-subscribe. In *Proc of the 2nd DEBS Workshop*, June 2003.

[23] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *Proc. of the Fifth MobiDE Workshop*, June 2006.

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

[25] D. J. Patterson, L. Liao, D. Fox, and H. Kautz. Inferring high-level behavior from low-level sensors. In *Proc. of the 5th Ubicomp Conf.*, Oct. 2003.

[26] M. Philipose, K. P. Fishkin, M. Perkowitz, D. J. Patterson, D. Fox, H. Kautz, and D. Hahnel. Inferring activities from interactions with objects. *IEEE Pervasive Computing*, 3(4), 2004.

[27] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proc. of the 6th MOBICOM Conf.*, Aug. 2000.

[28] J. Rao, S. Doraiswamy, H. Thakkar, and L. S. Colby. A deferred cleansing method for RFID data analytics. In *Proc. of the 32nd VLDB Conf.*, Sept. 2006.

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

[30] Rizvi et. al. Events on the edge. In *Proc. of the 2005 SIGMOD Conf.*, June 2005. (System demonstration).

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

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

[33] F. Wang and P. Liu. Temporal management of RFID data. In *Proc. of the 31st VLDB Conf.*, Sept. 2005.

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

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

[36] E. Welbourne, M. Balazinska, G. Borriello, and W. Brunette. Challenges for pervasive RFID-based infrastructures. In *Proc. of PERTEC Workshop*, 2007.

[37] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. of the Second CIDR Conf.*, pages 262–276, Jan. 2005.

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