Data Management in RFID Applications

Dan Lin[1] Hicham G. Elmongui[1⋆] Elisa Bertino[1] Beng Chin Ooi[2]

[1] Department of Computer Science, Purdue University, USA
{lindan, elmongui, bertino}@cs.purdue.edu
[2] Department of Computer Science, National University of Singapore, Singapore
ooibc@comp.nus.edu.sg

Abstract. Nowadays, RFID applications have attracted a great deal of interest due to their increasing adoptions in supply chain management, logistics and security. They have posed many new challenges to existing underlying database technologies, such as the requirements of supporting big volume data, preserving data transition path and handling new types of queries. In this paper, we propose an efficient method to manage RFID data. We explore and take advantage of the containment relationships in the relational tables in order to support special queries in the RFID applications. The experimental evaluation conducted on an existing RDBMS demonstrates the efficiency of our method.

1 Introduction

Radio frequency identification (RFID) [6] has been around for decades, and recently, there has been greater push from governments for its adoption for more efficient manufacturing, logistics and supply-chain management, and as a measure for security enforcement and weeding out counterfeiting. Take the supply-chain management for example (Figure 1), RFID enables accurate and real-time tracking of inventory by companies throughout an entire supply chain. Specifically, data stored in RFID are captured remotely via radio waves. Information from goods tagged with RFIDs can then be read simultaneously using fixed or mobile readers rather than requiring the scanning of individual bar code. Such a better supply chain visibility with the use of RFID also means that loss of inventory will be minimized during shipment. Businesses are suggested to use RFID for better inventory control since it may reduce excess inventories and free up capital for other activities.

Unfortunately, traditional database cannot efficiently support these new applications. Tracking each individual item causes data input to increase tremendously, and volume of data is enormous. As an example, Venture Development Corporation [4] has predicted that when tags are used at the item level, Walmart supermarket will generate around 7 terabytes of data every day. Though some compression techniques have been proposed (e.g. [8]), none of them fully explore the speciality of the RFID data while supporting online tracking.

For a better understanding of the characteristics of RFID data, consider the following example of the supply-chain management. Suppose there are several warehouses and stores. Products like T-shirts, milk packages are tagged with RFIDs and shipped

⋆ Also affiliated with Alexandria University, Alexandria, Egypt.

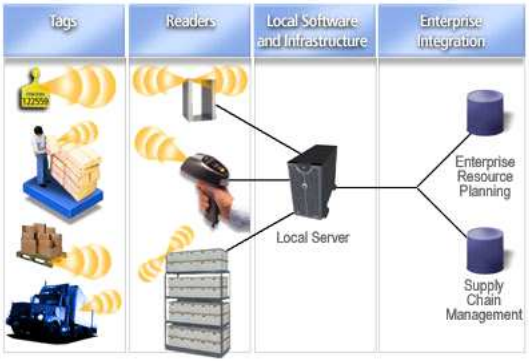


Fig. 1. Supply-Chain Management

respectively from warehouses to stores by trucks. During the shipment, products may be reallocated or reorganized at some intermediate warehouses. All such information is recorded in a central database system when products pass through a warehouse or a store. In this scenario, suppose a type of T-shirts at a store is sold out and a customer wants to know when his order can be completed. To answer such a query, the retailer needs to check current status of the shipment. If he knows from the database that this type of T-shirts is now at a warehouse close to his store and will soon be sent to his store, he can then estimate the arrival time for the customer. Next, let us examine a more interesting but complicated situation. A retailer finds that a box of milk packages in his store is contaminated. He thus asks a query on the path of the shipment: "which place did the box of milk packages stay before arriving at my store?" If it is deemed to be contaminated in a truck, an alerting query may be issued to avoid more losses: "where is the truck now and what goods are in it?" This requires the system to quickly identify suspected trucks (which are possibly heading to other stores), and stop them to prevent possible contamination that may happen in other stores. The scenario would have been more disastrous if the movement of goods or living things causes infectious diseases to spread (for example, the breakout of SARS in Asia in 2003).

In this paper, we tackle the above problems specifically. We summarize our contributions as follows.

- We have explored the path and containment relationships in the RFID data and developed an ER-model based on it. To the best of our knowledge, it is the first time to clearly identify such inherent data connections in RFID applications so that they can be taken into account during the system design.
- We have proposed a real-time tracking system for applications in supply-chain management, manufacturing, logistics and delivery services. Both incremental updates and online queries are supported.
- We have conducted an extensive experimental study. The results demonstrate the efficiency of our system compared with the traditional method.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents our proposed ER-model and discusses queries in the RFID applications.

Section 4 proposes our approaches for the RFID data management. Section 5 reports the experimental results. Finally, Section 6 gives the conclusion.

2 Related Work

RFID technology has posed many new challenges to database management systems [10, 12]. Some IT companies are providing RFID platforms [1–3, 5, 6], through which RFID data are acquired, filtered and normalized, and then dispatched to applications. Thus high level RFID data modelling and management is up to applications. However little research has been observed in this area.

Chawathe et al. [7] presented an overview of RFID data management from a high-level perspective and introduced the idea of an online warehouse but without providing details at the level of data structure or algorithms. Later, Wang et al. [11] proposed a model for RFID data management. This model shares many common principles with the traditional models and hence is still inefficient in representing the specialty of RFID data. Hu et al. [9] proposed a bitmap data type to compactly represent a collection of identifiers, which can significantly reduce the storage overhead. However, the bitmap technique may not work well when the data in the same cluster are not continuous. As also reported by the authors, this approach might not be a good candidate for some applications like postal mail dispatch, because unlike the retail sector, the items in these applications do not lend themselves well to grouping based on a common property, thus precluding the use of bitmap for these cases.

Most recently, Gonzalez et al. [8] have proposed a new warehousing model that preserves object transitions while providing significant compression and path-dependent aggregates. The warehouse is constructed after all data have been collected. Specifically, each object is registered in the database only once at the end of its movement, which is different from traditional method that records each object at each station during its movement. This approach can largely reduce information volume. However, it may not be able to answer online queries on current status of objects, and hence it is not applicable for real-time tracking problems.

3 RFID Data Modeling

In this section, we will first introduce a new ER-model for the RFID data management, and then address the query types. Finally, we discuss a running example to present an overview of functions that are achieved by our approach.

3.1 ER-Model and Query Types

In RFID applications, it is often the case that items tagged with RFIDs move and stay together during their movements or are regrouped at some locations [8]. Consequently, queries on path and containment relationship naturally arise. In order to efficiently support these queries, we propose an ER-model that captures such internal relationships among RFID data.

In our ER-model, there are three main entities: *landmark*, *means* and *moving units*. Landmarks can be warehouses, delivery centers, super markets, etc. Means can be trucks, ships or airplanes. Moving units can be moving objects (goods item), groups

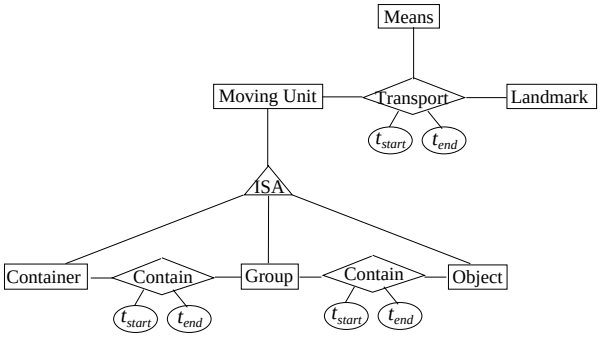


Fig. 2. The ER-Model

of objects, or containers. Figure 2 shows the relationships among the entities, where moving units are transported to some landmarks by some means from time t_{start} to time t_{end}. There exists a hierarchy of containment relationship. That is *Object* is contained in *Group* and *Group* is contained in *Container*. Another implicit containment relationship is that of the containers and locations. Note that there may be multiple levels in the hierarchy, though our example uses only three levels.

Queries on RFID data can be categorized from different aspects. According to the query time, there are three types of queries: current queries, predictive queries and historical queries. According to the query condition, queries can be classified into two categories: ID-based queries and location-based queries. In the ID-based queries, retrieval is based on given ID information. In the location-based queries, retrieval is based on given location information. According to the information being queried, we identify two types of queries: containment-relationship queries and path-preserving queries. The *containment-relationship queries* find all objects contained in a given object at a higher level. The *path-preserving queries* retrieve path information of one or more objects under specified constraints. Queries in the last categorization mostly reflect RFID data characteristic, and hence we will address their processing in details.

3.2 An Illustrative Example

For illustration purpose, we adopt a simple example from the supply-chain management scenario, which will be used throughout the paper. As shown in Figure 3, there are two locations L_1, L_2, three containers C_1, C_2, C_3, and three groups G_1, G_2, G_3. Each group contains one object: G_1 contains O_1, G_2 contains O_2 and G_3 contains O_3. During time 0 to 5, container C_1 stayed at location L_1 and contained two groups G_1 and G_2. C_1 was then shipped from L_1 to L_2. After C_1 arrived at L_2, its containment was changed, where group G_2 was moved to container C_2. At time 50, a new container C_3 arrived at location L_1. Note that this example is only a part of the whole scenario. In the following discussion, we represent different entities by using their IDs. The detailed information of these entities can be stored in a separate information table, which will not affect the efficiency of the proposed method.

Regarding this example, we will examine three representative queries. The first one (denoted as Q_1) is "what objects are (were) in group G (container C) at time t?", which is a containment-relationship query. Second, Q_2 is "where has object O (or group G,

container C) been to?". Third, Q_3 is "what objects (groups, containers) were shipped from L_1 to L_2 via L_3 and L_4 (L_3 and L_4 are intermediate warehouses) during time t_1 to t_2?". The last two are both path-preserving queries.

4 RFID Data Management

Handling a large amount of RFID data as well as providing efficient query services poses new challenges to existing database techniques. To make this point clear, we first study a straightforward method – Time-Line approach, and discuss its limitations. After that, we propose a more efficient approach – Multi-Table approach.

4.1 Time-Line Approach

The Time-Line approach is a naive method that stores all information in one table according to the insertion time. The format of each row in the table is $\langle Ts, Te, LID, CID, GID, OID, Means\rangle$, where Ts is the arrival time, Te is the leaving time, LID, CID, GID and OID correspond to the IDs of the location, container, group and object respectively, and $Means$ is the way the moving units being transported. Figure 4 shows how the data in the previous example is stored by using this Time-Line approach. Once there is an update on a field of the table, a new row is inserted. Here, an update could be a location update (e.g. a container reaches a new station), or a containment update (e.g. reallocation of goods in a container, or an object being delivered).

The aforementioned three queries can all be answered by combination of projection, selection and join operations. For example, Q_2 (to find where has object O been to) can be answered as: SELECT $*$ FROM Table WHERE OID = 'O'.

The main disadvantage of this approach is the data redundancy. Specifically, if the containment of a container (or a group) does not change frequently during the transportation, the Time-Line approach will store a lot of redundant information caused by the containment relationships. As shown in the example (Figure 3), O_1 stayed in the same container C_1 and the group G_1 when being transported from L_1 to L_2. The containment information of O_1 is unchanged but repeatedly stored in two records (1st and 3rd records in Figure 4). Such redundant information will unnecessarily increase the table size and result in poor performance.

4.2 Multi-Table Approach

To alleviate the data redundancy problem and take advantage of the specialty of RFID data, we develop a Multi-Table Approach based on our proposed ER-model. Our ap-

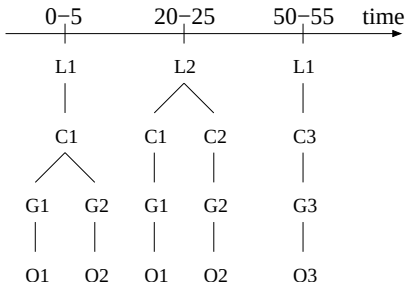


Fig. 3. An Example

Ts	Te	LID	CID	GID	OID	Means
0	5	L1	C1	G1	O1	Truck1
0	5	L1	C1	G2	O2	Truck1
20	25	L2	C1	G1	O1	Truck1
20	25	L2	C2	G2	O2	Truck2
50	55	L1	C3	G3	O3	Truck3
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Fig. 4. Time-Line Approach

proach adopts the following assumptions. Each object only belongs to one group, which means we do not reallocate objects to other groups. This is due to the consideration of the scenario like a box of milk packages, where a single milk package (object) usually stays at the same box (group) during its transportation. Unlike objects which are at the lowest level of the containment relationship hierarchy, groups can be reallocated to other containers, containers can be reallocated to other trucks, and so on. Moreover, groups and objects have their final destinations while containers and trucks can be reused.

In the Multi-Table approach, there are two types of relational tables: the *containment table* and the *path table*. The containment table stores the information of containment relationship and the path table captures the path information of moving units.

Figure 5 gives an overview of the containment tables in our system. There are Location-Container (L-C for short) table, Container-Group (C-G) table and Group-Object (G-O) table. Each row of these tables consists of at least four fields. $[Ts, Te]$ is the time interval during which one moving unit (e.g. GID) stays at the same place (e.g. CID). In the L-C table, there is one more field – *Means*, which indicates the transportation means of the containers. Each table has corresponding history tables. Records are moved to history tables periodically (details will be covered shortly).

Location−Container Table

Ts	Te	LID	CID	Means
0	5	L1	C1	Truck1
20	25	L2	C1	Truck1
20	25	L2	C2	Truck2
50	55	L1	C3	Truck3
⋮	⋮	⋮	⋮	⋮

Container−Group Table

Ts	Te	CID	GID
0	25	C1	G1
0	5	C1	G2
20	25	C2	G2
50	55	C3	G3
⋮	⋮	⋮	⋮

Group−Object Table

Ts	Te	GID	OID
0	25	G1	O1
0	25	G2	O2
50	55	G3	O3
⋮	⋮	⋮	⋮

Fig. 5. Three Main Relational Tables of Multiple-Table Approach

GID	LID−Time List
G1	<L1, 5>, <L2, 25>, ...
G2	<L2, 25>, ...
G3	<L1, 55>, ...
...	...

Fig. 6. An Example of Group_Path Table

Figure 6 shows the structure of the path table, i.e., the Group_Path table. This table is a query-driven table, which is created during the query processing. It stores part of query results in order to facilitate new queries. Each row of this table contains two fields: a group ID GID and an LID-Time list. The LID-Time list records a sequence of $\langle location, Te \rangle$ pairs, which indicates the location that the group has visited and the corresponding departure time.

In the rest of this section, we first present how to update information in the containment and path tables. Then we present the query algorithms.

Construction Consider the scenario at a station, where several containers arrive at time Ts. First, there will be an arrival scan that reports the container IDs to the system. During their stay, their containments will be scanned and checked. If there is any change of the containments, i.e., rearrangement of goods, the system will receive new inventories for the corresponding containers. Finally, when containers leave, a departure scan is carried out and reports the departure time Te to the system. From the above scenario, we identify three types of events: (i) Arrival event; (ii) Containment arrangement event; (iii) Departure event. The algorithm for each event is presented as follows.

The arrival event provides the location information of containers, and hence only the L-C Table is modified at this stage. Specifically, for each container, we will insert a new record $\langle Ts$, _, LID, CID,_$\rangle$ to the L-C Table. The two fields Te and $Means$ will be filled later when more information is received.

The containment arrangement event includes two sub-events corresponding to containers and groups respectively. We first elaborate the management of containment change in containers. If there is a reallocation in container C_1, C_2, ..., C_n, in the C-G table, set the Te of groups that move out of the above containers to be the reallocation time, and insert a set of new records of groups that move into these containers. The event of containment arrangement of groups is triggered by object arrival or delivery. If objects O_1, O_2, ..., O_m are new objects to the system, insert records like $\langle GID$, O_1, Ts, _, $\rangle$ to the G-O table. If objects O has been delivered, move its record from G-O table to history G-O table and set the Te to be the delivery time.

Finally, we handle the departure event. The operation is simple. We only need to update the departure time Te of each departure container as well as its transportation means (e.g. truck ID) in the L-C table.

Apart from the event handling, there is one more step for system optimization, which is the construction of history tables. Every certain time interval T_{int}, we will check L-C and C-G tables to move records with Te older than current time to the history tables. Each history table has a global time interval that indicates the earliest and latest timestamps of its records. As time elapses, there may exist a set of history tables. Here, T_{int} is an application dependent parameter which controls the table size. It can be set according to the speed of information grow. For example, if updates are frequent, a small value of T_{int} may benefit the query retrieval.

Query Processing We proceed to present algorithms for three representative queries (in Section 3.1). Note that other queries are special cases of the techniques used for these three representative queries. To speed up the search in each table, we have a clustered index on one type of ID and an unclustered index on the other.

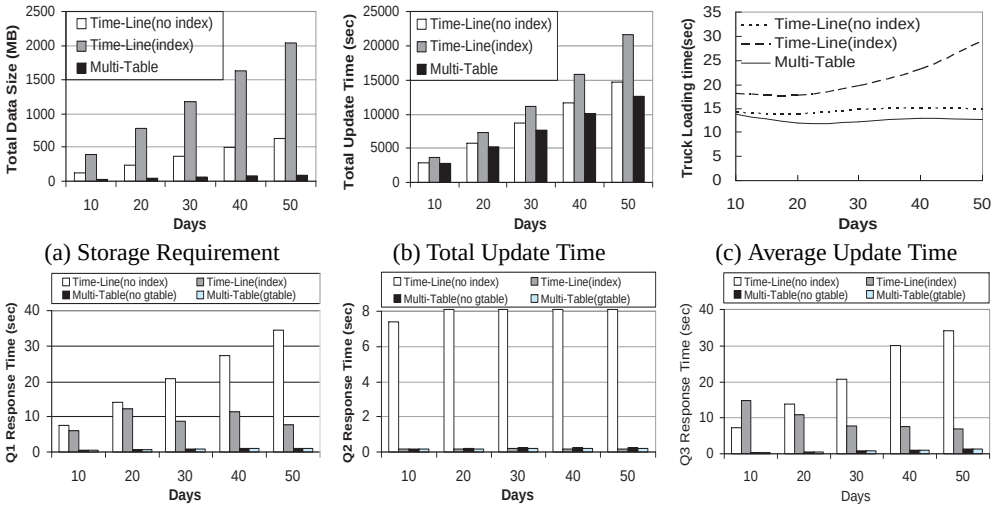
For Q_1 (containment-relationship query) on location L_1, the search starts from the L-C table, where we obtain a list of containers at location L_1. Then we search the C-G table to find the groups of these containers. Finally, we retrieve the G-O table to get the objects at location L_1.

For Q_2 (path-preserving query) on object O, there are two main steps. The first step is to find the group that object O belongs to. According to the object status (delivered or not), we can find its group ID in G-O table or history G-O tables. The second step is to find the locations that this group G has visited within the life time of object O. Here, we may take advantage of the Group_Path table. If there exists a record with respect to the group G in the Group_Path table, we further check whether this record contains sufficient information of object O, i.e., whether the location list contains a location with Te larger than the object delivery time (or the latest update time). If yes, we report locations in the list till the one with Te larger than the query time. If we can not find a corresponding record of group G in the Group_Path table or the table does not contain full path of object O, we have to retrieve C-G table to obtain a set of containers that group G ever belonged to, and then retrieve L-C table to find the locations of the containers. Finally, we need to append the query results to the Group_Path table.

The last query Q_3 is more complicated than previous ones since it requires to retrieve both containment and path information. The algorithm consists of following three steps. First, we find all containers at location L_1 during time t_1 to t_2 by searching the L-C table. Second, we find all groups of these containers and store them in a group list. The Third step is to check the Group_Path table to see if the path of each group in the group list contains a sequence of locations $\langle L_1, L_3, L_4, L_2 \rangle$. If yes, we report the objects in the qualified groups with lifetime cover the query time interval. Otherwise, there could be two situations. One is that the path of the group as recorded in the Group_Path table is different from the query path, which can be safely pruned. The other situation is that the path of the group is not completed or there is not a record of this group. For this case, we need to find the locations of the group by retrieving all the containers that it ever belonged to, and retrieving all the locations of these containers. Then we check if the path of the group matches the query path. Finally, we append the group path information to the Group_Path table for the use of future queries.

5 Performance Study

We implemented the proposed algorithms as stored procedures in MS SQL Server 2005. For all experiments, we use a Xenon 2.0GHz CPU with 1GB of RAM. We created an application that simulates the movement of 18-wheelers between warehouses and stores. The simulated scenario is for 20 trucks and 80 warehouses. Each 18-wheeler contains 8 containers; each container holds up to 8 boxes; each box contains 12 objects. All containers, boxes, and objects are tagged with RFIDs. Upon arrival to a warehouse, the 18-wheeler is filled to completion. Upon arrival to a store, the probability that a container contains boxes for delivery is set to $(1-p)$. The probability that a box in such a container is to be delivered is also $(1-p)$. Thus objects are delivered to stores according to a geometric distribution with average numbers of hops $1/q$, where $q = 1-(1-p)^2$. We set default value of this average to 6 hops. The parameters of this simulation come from real samples of 18-wheelers. The trip from a warehouse to a store is uniform with mean equal to a day and with a standard deviation of 20 minutes.



(a) Storage Requirement (b) Total Update Time (c) Average Update Time

(d) Q1 (e) Q2 (f) Q3

Fig. 7. Experimental Results

We implemented two variants of Time-Line approaches distinguished by having index assistance or not, denoted as "Time-Line(no index)" and "Time-Line(index)" respectively. We also implemented two version of Multi-Table approaches distinguished by using the Group_Path table or not, denoted as "Multi-Table(no gtable)" and "Multi-Table(gtable)" respectively. It is worth noting that the size of the Group_Path table is ignorable compared to the total data size and the table is not involved in the data update process. Therefore we do not distinguish the two variants in the experiments regarding storage space and update performance.

Storage Requirement The total data size that needs to be stored for an application is an important concern in database system design since a small data size can save cost for companies and may also benefit the system performance. To evaluate the storage efficiency, we examine the total data size stored by each approach every 10 days. Figure 7(a) shows the results, in which the Multi-Table approach requires the least storage space than the Time-Line approaches. The main reason is that the Time-Line approach maintains more redundant information. For example, if a container contains 100 items, each time the container reaches a station with the same items inside, the Time-Line approach needs to create 100 new records for all the items, whereas the Multi-Table approach only needs to create one new record corresponding to the container itself. In addition, Time-Line(index) needs more space than Time-Line(no index) to store the indexes.

Update Performance Figure 7(b) plots the total update time every 10 days for each approach. It is not surprising that the insertion time of all approaches increases as time elapses due to the increased table sizes. Among all, the update time of Multi-Table approach is the shortest because it has the smallest table size (as shown in Figure 7(a)). The Time-Line(index) is the slowest approach with respect to the insertion performance. This is because Time-Line(index) needs to maintain its indexes for each update.

Figure 7(c) shows the average update time of each truck. The result again shows that the Multi-Table approach is the best. Moreover, we also observe that both the Multi-Table approach and Time-Line(no index) achieve steady performance, while the Time-Line(index) requires more time to maintain its indexes with the growth of the data size.

Query Performance In the following experiments, we will evaluate three representative queries. Figure 7(d) and (f) show the average response time of Q_1 and Q_3 respectively. We can observe that Multi-Table approaches achieve the best performance, which possibly due to small data sizes that reduce the data retrieval and table join time.

Figure 7(e) shows the performance of query Q_2. We can see that the Time-Line(no index) is extremely slow (more than 100 times slower), and the other three approaches yield the similar performance. The slowness of the Time-Line(no index) is mainly because without any index support, it has to execute "brute-force" join operations. The Time-Line(index) is sometimes a little bit better than the Multi-Table approaches, but we should note that the Time-Line(index) requires much more space and longer update time. Another interesting observation is that the Group_Path table can reduce the query cost and its benefit increases as time passes (this effect is a little hard to be seen from the figure due to the large value of Time-Line approach).

6 Conclusion

In this paper, we study the important features of RFID applications, such as the hierarchy of containment relationships and path preserving in query operations. We propose an expressive ER-model. Based on the ER-model, we develop a simple yet efficient real-time tracking system for RFID data managements. Our extensive experimental results prove the significant performance improvement achieved by our system compared with a naive method.

References

1. Developing auto-id solutions using sun java system rfid software. *http:// java.sun.com/ developer/ technical-Articles/ Ecommerce/ rfid/ sjsrfid/ RFID.html*.
2. Microsoft's rfid 'momentum' includes middleware platform, apps. *http:// www.eweek.com/ article2/ 0,1759,1766050,00.asp*.
3. Oracle sensor edge server. *http:// www.oracle.com/ technology/ products/ sensor_edge_server*.
4. Venture development corporation (vdc). *http://www.vdc-corp.com*.
5. Websphere rfid premises server. *http://www-306.ibm.com/software/pervasive/ ws_rfid_premises_server*.
6. C. Bornhovd, T. Lin, S. Haller, and J. Schaper. Integrating automatic data acquisition with business processes - experiences with sap's auto-id infrastructure. In *Proc. VLDB*, pages 1182–1188, 2004.
7. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. Sarma. Managing rfid data. In *Proc. VLDB*, pages 1189–1195, 2004.
8. H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and analyzing massive rfid data sets. In *Proc. ICDE*, page 83, 2006.
9. Y. Hu, S. Sundara, T. Chorma, and J. Srinivasan. Supporting rfid-based item tracking applications in oracle dbms using a bitmap datatype. In *Proc. VLDB*, pages 1140–1151, 2005.
10. M. Lampe and C. Flrkemeier. The smart box application model. In *Proc. Int. Conf. of Pervasive Computing*, 2004.
11. F. Wang and P. Liu. Temporal management of rfid data. In *Proc. VLDB*, pages 1128–1139, 2005.
12. R. Want. The magic of rfid. *ACM Queue*, 2(7):40–48, 2004.