A Data Model for EPC Information Services

Tuyen Nguyen, Young-Koo Lee, Rezwanul Huq, Byeong-Soo Jeong, Sungyoung Lee

Department of Computer Engineering, KyungHee University, Korea
{ntttuyen, rhuq}@oslab.khu.ac.kr, {yklee, jeong}@khu.ac.kr, sylee@oslab.khu.ac.kr

Abstract

The RFID system with the capability for unique item-level tracking promises a tremendous change in the automation process of identifying, managing and tracking commodities along supply chains. One of the heaviest burdens in the RFID system is laid on its repository that has to handle an enormous amount of timestamped RFID data that are generated continuously from many complex relationships. This motivates us to propose an event-based data model for EPC Information Service (EPCIS) repository that reduces the data volume and handles well all kinds of entity relationships. We also exploit both object-relational and relational strength to make data rich in semantics and build a generalization hierarchy of events to answer successfully queries about historical events recommended by EPCGlobal's EPCIS specification.

1 Introduction

With the great ability of identifying each object by a unique Electronic Product Code(EPC), RFID technology has enormous potentiality that can make significant changes in many areas such as supply chain automation, asset tracking, medical applications and warehouse. Despite of this diversity, all RFID applications share one of the biggest challenges that we have to fully consider in the effort to make RFID a reality: how to build a strong repository that can maintain enormous incoming stream data logically and efficiently allowing partners' applications to query data easily with no need to concern themselves in the internal technologies. Different from ordinary applications' repositories, this repository has to handle both static data describing objects and different aspects of business processes, and temporal data coming from each individual EPC-tagged object's lifecycle throughout the business processes it involves in. (In this paper, we use the terms RFID and EPC tag interchangeably.) To leverage EPC data sharing between disparate applications using different database technologies, EPCGlobal[1], an organization that develops world-wide standards for RFID technology, proposed the EPC Information Service (EPCIS)[2], a data repository that covers most issues related to RFID data management and uniform programming interfaces for data acquisition and sharing. Our work focuses on building a repository that meets the requirements in this specification.

Static data are similar to traditional regular business data and easy to handle by relational databases. However, in a large system like RFID network, we should think about a general meta-data model with high formality so that different applications can easily query any entity in the same way. Temporal data actually describe the dynamic relationships between EPC objects and other static entities. The amount of data clearly increases over time and easily overloads our repository. Moreover, in some cases, we also have to maintain the implied relationship between the objects experiencing the same business process. For example, when some milk boxes are moved from a warehouse to a shipping house and then packed into a container for shipping, their relationships with entities *location* and *business step* change, new relationships between these boxes and that container begin and there also exists an implied relationship between these boxes indicating that from this time, they are always at the same locations and conditions. Obviously, when a change occurs in an EPC object life, it can include many changes in the relationships of that object with other entities. In other words, the change itself, which is described as an event in the EPCIS specification[2], contains much semantic information that the data model should well represent. The EPCIS data model have to consider all of these problems.

There are several products and projects that build their own RFID architectures in which there is a part operating similar to EPCIS, such as Sun RFID Software[3], Siemens RFID middleware[4], etc. Their data models are efficient for separate EPC management and some queries on tracking and monitoring commodities. However, these systems have not fully considered the EPCIS specification that is really helpful for different components developed by different organizations to easily integrate with each other. In addition, since in their relational repositories, one event or entity in the real world is dispersed in several tables makes these models weak in semantics and difficult for us looking back on history of events pushed up from underlying component.

Based on the EPCGlobal's specification, our paper just focuses on one important component in RFID system, the

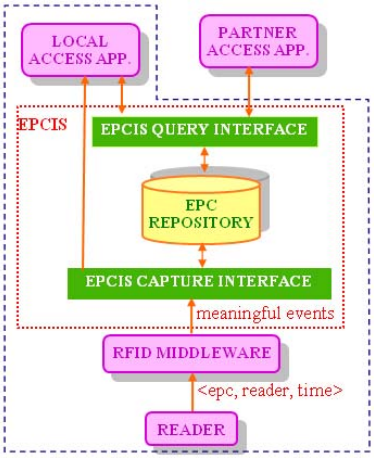

Figure 1. EPCIS sits at the top of EPCGlobal architecture framework, serving as a data bridge between the RFID physical world and high-level applications

EPCIS repository. We built an event-based repository that satisfies the requirements specified above. Our data model is different from others in storing RFID data in form of meaningful timestamped events and exploiting the strength of object-oriented concepts while still keeping all prominent features of traditional relational approaches. We built the event generalization hierarchy so that it is simple to specialize more events and easy to look back on the event history. All EPCs experiencing the same business context at the same time are in the same event so that we can reduce the data volume and also preserve the association between EPCs implied their appearing in the same event.

As a result, our repository is efficient when objects are moving together in large groups. With the event-based approach, we can answer successfully the two predefined queries recommended in EPCIS Specification[2]: SimpleEventQuery on the event history and SimpleMasterDataQuery on static entities. One example of SimpleEventQuery is "find all events that occurred at location L1 or L2 from time T1 to T2". The difficulty of this kind of queries is the ability of retrieving all events along the event generalization hierarchy and recovering the all semantic information the events had when they were sent to EPCIS. This difficulty is easily solved by our data model but previous models.

This paper is organized as follows. Section 2 introduces the EPC system architecture, EPCIS, its specification and previous work related to EPCIS repository. Section 3 proposes an extended ER diagram for the repository. Section 4 applies object-relational concepts on the model while keeping relational best features. Section 5 answers two predefined queries. Section 6 reports the experiments and future work. Section 7 overviews some related works. Section 8 concludes our paper.

2 EPC Information Services (EPCIS)

Before proposing our Extended ER diagram, we overview the EPC system architecture [1], the position of EPCIS and the purpose of EPCIS specification in our effort to build an EPC Network.

2.1 EPC General Architecture

Figure 1 shows a common architecture illustrating how an RFID application works. Each object is tagged with an RFID chip that contains a unique Electronic Product Code (EPC). Data from these tags are then collected periodically by RFID readers and sent to RFID Middleware in form of tuple *<epc, location, time>*. Instead of storing these massive, uncleaned, poor-semantic reads directly in repository, the RFID Middleware will filter (e.g., eliminate duplicate reads and missing reads) and correlate them with the business context to generate all clean and meaningful events. These events are then passed to EPCIS through Capture Interface and stored permanently in its repository or pushed to some applications interested in real-time information. The data can be queried from partners' accessing applications through Query Interface.

2.2 What is EPCIS?

In the EPC architecture, EPCIS is the primary bridge connecting RFID physical world with the high level applications and facilitating the data exchange between trading partners. It consists of an EPC-enabled repository for storing data persistently and two standard interfaces: *EPC Capturing Interface* that gets EPC events from underlying applications and *EPC Query Interface* that allows authorized enterprises to retrieve data in repository.

Data in EPC repository fall into two categories which will be described more detailed in section 3.1

- *Timestamped event data* collected throughout the lifecycle of EPC-tagged objects while business processes are carried out
- *Master data* providing necessary context information for interpreting event data and rarely changed over time.

The repository has all rights to decide how to store master and event data as long as it preserves all semantic information and supports Query interface to answer various queries from clients including queries on the event history.

2.3 EPCIS Specification

Despite the diversity of RFID applications, the EPCIS functionalities are almost similar. Therefore, EPCglobal in-

troduces the EPCIS Specification that provides a uniform programmatic interface (not an implementation) to allow various clients to capture and access EPC-related data and the business transactions with which data are associated. This specification specifies generic structures for representing EPCIS data; what data is exchanged through EPCIS, what its abstract structure is, what it means and its binding to a XML Schema; and service operations through which EPCIS clients interact. Actually, EPCIS Specification just provides a standard data sharing interface to facilitate the communication between different applications that deal with EPC-related data. How these standards should be implemented as well as how to model the data in EPCIS are up to each organization.

2.4 Previous EPCIS Data Models

This section summarizes some data models that were proposed before. We can not but mention one of the earliest ones from Mark Harrison[5] that summarizes EPC data into two categories: static attribute data and timestamped historical data. The timestamped data includes observation, transaction, containment, and measurement. Each type is represented in a three-dimensional space which has a timestamped axis and other two axes representing two entities joining that dynamic relation. Collected temporal data are viewed as points in the corresponding 3D spaces. The simple temporal query in the space is the process of looking for appropriate plane, line, or point. The complex temporal query is decomposed into numerous simple queries whose results are joined and filtered later. Clearly, this model is not efficient enough for complex queries and queries related to business context because obviously, three dimensions are just enough for describing the minimum information about a dynamic relationship.

Fusheng Wang et al. proposed a data model for Siemens RFID Middleware[4]. This model includes four primary static entities: objects, sensors/readers, locations, and transactions. They interact with each other to generate state changes (*location change*, *containment change*) and event changes (*observations* and *transaction items*). These changes are modeled as dynamic binary relationships which associate with timestamps (event changes) or time intervals [*tstart*, *tend*] (state changes). This data model highlights the state history and temporal semantics of business processes. However, we lose the completeness of EPC data because intermediate observations are eliminated in the state history. Therefore, it is hard to answer such queries as "the last time object 123 was seen at location L" and the *SimpleEventQuery* recommended by the EPCIS Specification. Moreover, it takes time to process events from underlying middleware and store them in the appropriate tuples. Finally, the four primary entities above are not enough for describing business processes.

Sun RFID software[3] models each static entity type in three different tables: the first for the entity type itself including identifier and the attributes that all entities share in common, the second for specific attributes intended for each entity, the third for parent/child relationships. Dynamic data are stored in form of different logs: container log, observation log , transaction log, tag allocation log (history of all tags allocated). Each log entry has a timestamp. Sun provides us a useful meta schema for static entity. It preserves the completeness of RFID data history. However, like other models, it is weak in business process semantics because it does not include different aspects of a business process but location and shipping.

In general, three models mentioned above track EPC-tagged objects separately. Next section, we propose our own data model for both static and dynamic data that covers most requirements specified by the EPCIS Specification. Our model is event-based and rich in semantics and represents all kind of relationships. Objects that experience the same process will be tracked together so that we can reduce the data volume and keep the internal relations between them.

3 Extended Entity Relationship (EER) Diagram

3.1 Data in EPCIS

As mentioned, data in EPCIS fall into two categories: master data (static data) and event data (dynamic data). Master data include (1) *class-level Data* describing properties for all objects of the same object class (product name, manufacturer, SKU, etc), (2) *instance-level Data* describing properties of each individual object (date of manufacture, lot number, etc), and (3) *business context data* providing necessary business context (business locations, transaction type, etc).

Event data refer to things that happen at a specific time. There are four kinds of events:

- *Object Event* which carries information about actual observations of or assertions about EPC-tagged objects (e.g., "EPC X was shipped from store Y at time Z").
- *Aggregation Event* which announces a specific group of EPC-tagged objects was contained in another one (e.g., "at time T, objects of EPCs A, B, C was aggregated to the case EPC X at factory L").
- *Quantity Event* which inventorially reports the number of instances of a specific object class (e.g., "at time T, there are 400 instances of object class X observed at location L").

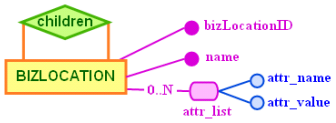

Figure 2. ER diagram for Vocabulary *Business Location*

- *Transaction Event* which describes the association of EPC-tagged objects with one or more business transactions (e.g., "at time T, the cases of EPC X, Y, Z were shipped for purchase order #123".

3.2 Characteristics of Master Data

Master data consist of a collection of vocabularies. Each vocabulary contains a list of elements, each of which has its own attribute set. For example, to represent information about all the products, we list each product as an element in vocabulary *Products*. But these products may not share the same set of attributes beacause attributes for product *Razor* are different from those for *Shirt*.

According to the particularity of business, each enterprise builds up its own set of vocabularies. However, some important ones needed for interpreting business processes are:

- *Object, Product, Organization* – instant-level and class-level information about EPC-tagged objects
- *Container* – different types of containers into which objects can be packed (case, pallet, truck, etc.)
- *Readpoint* – places where EPCIS events took place.
- *Business Location* – business locations where objects are assumed to be following an events.
- *BizTransaction, BizTransactionType* – information about transactions and their types.
- *BizStep* – steps in business process (shipping, receiving, etc.)
- *Disposition* – business state of EPC objects (available for sale, sold, etc.)

Elements in the same vocabulary can have hierarchical relationship with each other with an arbitrary number of levels of depth. For example, location "Walmart. at Seoul" can have "Walmart. at Seoul floor#1" as its child. Besides, a given child can be a direct child of more than one parents.

Based on these characteristics, we model each vocabulary in the similar way like we do for vocabulary *BIZLOCATION (business location)* illustrated in Figure 2. An entity type called *BIZLOCATION* represents for the vocabulary. Besides the identifier *vocabulary_name*ID

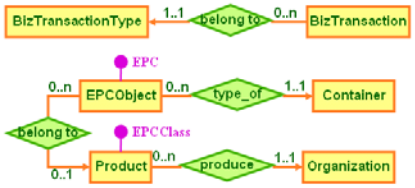

Figure 3. Relationships between vocabularies

(e.g.*bizLocationID*) and *Name* for each entity, this entity type has a special attribute called *attr_list*, which is a list of pairs of (*attr_name, attr_value*), to maintain the attribute list associating with each entity. The entity type relates to itself in a parent/child relationship.

There also exists other relationships among different vocabularies in master data that can be modeled easily as illustrated in Figure 3. For example, each EPC-tagged object (EPCObject) can be a product item or a container and belongs to a container type (e.g. item, case, pallet, truck, etc.). And associating with each business transaction is its corresponding transaction type.

3.3 Characteristics of Event Data

Every event type carries temporal information in nature. Therefore, first a base event type, EPCISEvent, consists of two base attributes: the timestamp when event occurred, called *event time*, and the timestamp when event was recorded in repository, called *record time*. Other event types inherit from this root and encapsulate more attributes about the business context in which they occurred, including:

- the objects or entities observed which are EPCs, EPCClass (Quantity Event), list of business transactions (Transaction Event).
- the location (Readpoint and Business Location)
- the business context (Business Step and Disposition)

Though the EPCIS specification recommends four kinds of events that provide necessary information about the commodity flow in most business processes. However, each enterprise may come up with its own events or extensions from these ones to meet their specific requirements. This generalization hierarchy allows us to specialize more events easily as needed and perform temporal operation on the root while searching for specific attribute in the specialization events. Therefore, it answers successfully those queries asking for the event history (e.g., the SimpleEventDataQuery recommended by the EPCIS specification). The hierarchy of event types and the relationship between them and other vocabularies are described in detail in Figure 4. To avoid line overlap, in this figure, we considered that event types with the same name are the same entity type.

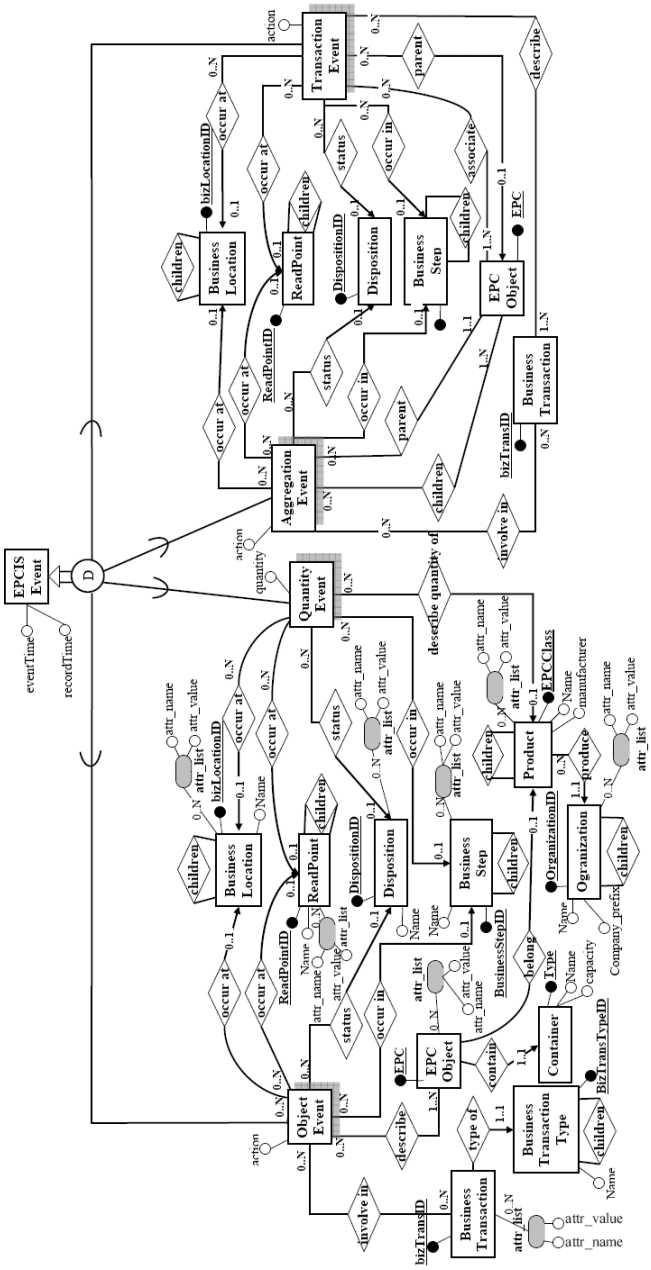

Figure 4. Extended entity relationship diagram for our proposed data model. Events are temporal and hierarchical and interpreted by other vocabularies

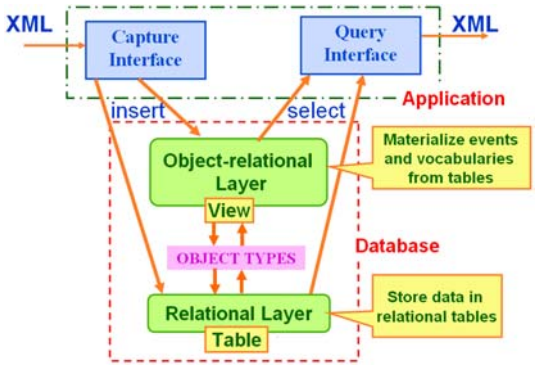

Figure 5. Proposed object-relational model consisted of two layers, one based on tables and the other based on views

BIZLOCATION_EXT

bizLocationID	attr_name	attr_value
dc#1	address	123 Yongin St
dc#1	city	Suwon
dc#1	postal code	449-701
…	…	…

BIZLOCATION

bizLocationID	name	….
dc#1	ABC distribution center #1	…
sec1A	ABC distribution center # 1 sec. A	…
…	…	…

BIZLOCATION_HREF

parent	children
dc#1	sec1A
….	…

Figure 6. Vocabulary BizLocation and its tables in relational layer

4 Object-Relational Model

Relational database has proved itself over years a good ability of storing large amount of data and protecting integrity through normalization and constraints. Furthermore, all sorts of optimizations have been built into engines so that they are very fast. However, relational database has a lack of semantic features and an inability to represent complex structures and operations. Therefore, it is not nearly strong enough to model all kinds of relationships, both static and dynamic. Meanwhile, the object-oriented database is good at semantic data modeling but complex for query processing and unable to support large-scale systems. These reasons have stimulated the emergence of object-relational approach to benefit aspects from both models. Data are stored in tables, and we can define our own data structures and relevant operations for tabular entries, and benefit different kinds of object-oriented relationships between objects such as aggregation, association and inheritance.

Unfortunately, current database systems, such as Oracle, depend heavily on relational tables when building object-relational features. If we directly use these object-relational features, the performance of our system is much less efficient than purely relational approach. Therefore, before applying object-relational approach, we employ a relational

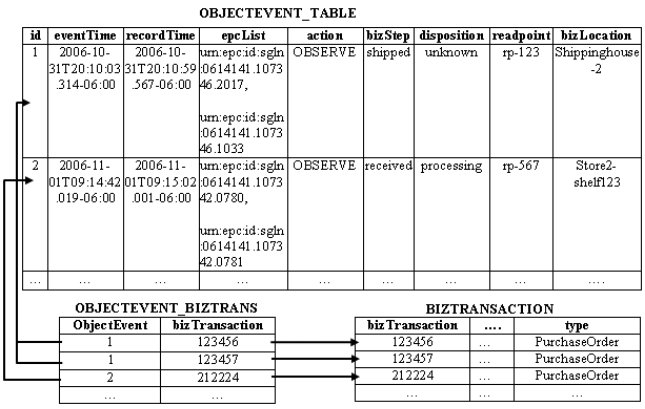

Figure 7. Tables for ObjectEvent in relational layer

layer (Figure 5) in which we map the EER diagram in previous section into relational tables. Normalization and all constraints are deployed at this layer as in a traditional database. Then, we create object views that materialize events and vocabularies in object-relational approach so that they carry as much semantics as we expect. Query Interface can query repository from views in object-relational layer. Besides, we place instead-of triggers on these views so that incoming data from capture interface can be inserted into repository via object-relational layer. And in fact, application also can access the data via relational layer directly.

4.1 Relational Layer

We store each vocabulary in a table, in which each common attribute of all vocabulary elements corresponds with a table field, and the distinct set of attributes, which is specific for individual element, in another table (*vocabularyname*_EXT) whose field named *attr_value* can accept any type of value. Similarly, the parent/children relationships among elements are maintained in the third table (*vocabularyname*_HREF) which consists of two fields as the composite primary key: *parent* and *child*. **Example** in Figure 6, vocabulary *BizLocation* in Figure 2 are dispersed in three tables *BizLocation*, *BizLocation_Ext* and *BizLocation_Href*.

In EER diagram (Figure 4), we can see that there are a number of vocabularies to describe each event type. Therefore, besides a table designed for the event itself, we have additional tables for one-many or many-many relationships between that event and other vocabularies. As an example, Figure 7 demonstrates how to maintain information about *ObjectEvent* in tables. *ObjectEvent_bizTrans* is the table for the the business transaction list one object event can involve in.

4.2 Object-Relational Layer

Above the well-constrained and normalized relational layer, we deploy an object-relational layer, which consists of object views, to materialize rich semantic vocabularies and events. Because each vocabulary and event can be dispersed in some tables in relational layers, first, for each vocabularies and events, we create an corresponding object type encapsulating all the necessary information from several tables that relate to that vocabulary or event. For example, the object type for vocabulary *BizLocaiton* is created as follows

```
create type BizLocation_Type as object(
bizLocationID varchar2(50),name varchar2(50),
attr_list attr_list_type,children children_table);
```

Especially, when we create object types for events, we also define the inheritance relationships among them. We begin with EPCISEvent_Type representing the root EPCIS event that just carries the temporal information

```
create type EPCISEvent_Type as object(
eventTime timestamp with time zone,
recordTime timestamp with time zone)not final;
```

Other event types inherit EPCIS event type and continue this inheritance hierarchy as deep as we expect. For example:

```
create type ObjectEvent_Type
            under EPCISEvent_Type}(
EPCList EPCList_type,action varchar2(10),
bizStep varchar2(100),disposition varchar2(100),
readpoint varchar2(100),bizLocation varchar2(100),
bizTransactionList bizTransaction_Ref_table) not final;
```

with *EPCList_type* - a collection or nested tables of *string* and *bizTransaction_Ref_table* - a nested table of an object type consisting of two attributes: *bizTransaction* and *bizTransactionType*.

Next step, we create object views based on these objects types and materialize information from separate tables.

```
create view bizLocation_view of bizLocation_type
WITH OBJECT IDENTIFIER(bizLocationID) as select
b.*,cast(multiset(select attr_name,attr_value)
            from bizLocation_ext al
           where al.bizlocationid=b.bizLocationID)
         as attr_list_type) as attr_list,
    cast(multiset(select child
                from bizLocation_href bh
                where b.bizLocationID = bh.parent)
          as children_table) as children
from bizLocation b;
```

Again, we apply inheritance relationship among the object views materializing events. The root object view *EPCISEvent* is created based on *EPCISEvent_Type* and populated by the data coming from table *EPCISEvent_Table* (which is nominal and supposed to contain no data records). Other event views base themselves on their corresponding object types and tables and inherit this the root view.

```
create or replace view EPCISEvent of EPCISEvent_Type
as select * from EPCISEvent_Table;

create view ObjectEvent of ObjectEvent_Type
            under EPCISEvent as
select eventTime,recordTime,epclist,action,
 bizStep,disposition,readpoint,bizLocation,
 cast(multiset(select distinct l.bizTransactionId,
```

```
 t.type from ObjectEvent_TransList l,BizTransaction t
        where bl.ObjectEventID = l.ObjectEventID and
              l.biztransactionid=t.biztransactionid)
    as bizTransaction_Ref_table) BizTransactionList
from ObjectEvent_table bl;
```

We continue this way until we have the generation hierarchy we want.

5 EPCIS Queries

The EPCIS specification requires to handle two predefined queries that support simple ways for accessing application to retrieve EPCIS event instances and vocabulary elements: SimpleEventQuery and SimpleMasterDataQuery. When clients want to use one of the predefined queries, they specify its name and an appropriate list of parameters in form of pairs *(attribute name, attribute value)*. EPCIS based itself on the query name and then processes the parameter list to return the expected data. For queries on EPC events, EPCIS has to have the ability to retrieve instances in an inheritance chain. The returned events carry as much semantics as they were supposed to have when they were pushed to EPCIS. In this section, we prove how our model successfully answered these requirements by using a simple illustrated query: "find all events that occurred at location L1 or L2 from time T1 to T2".

5.1 Simple Queries from EPCIS specification

5.1.1 SimpleEventQuery

This kind of query retrieves all the event instances that satisfy the predicates implied in the parameter list which can be summarized as below:

- EventType – the event types we need to query
- (GE(GT)(LT)(LE)(EQ)_*fieldName*,*value*)– events whose field named *fieldName* is greater than or equal to the value specified in the parameter
- (MATCH_*fieldName*, *value_list*)– events whose field (which is a collection of EPC) named *fieldName* contains one of the values in the *value_list*.
- EQ_*fieldName* is similar to MATCH, except that *fieldName* is not a collection.
- EQ_bizTransaction_*type* – events which have a transaction list containing a transaction whose type is equal to *type*.
- WD_*fieldName* – events whose field named *fieldName* matches one of the specific values or is a direct or indirect descendant of one of the specific values.
- HASATTR_*fieldName*, EQATTR_*fieldName_attributeName* – events whose field named *fieldName* is a vocabulary element which has an attribute whose name matches one of the specific value. Or the attribute name is *attributeName* and its value matches one of the specific values.

To answer the illustrated query, we specify the query name as SimpleEventQuery and the parameter list as (EQ_bizLocation, 'L1'),(GE_eventTime, 'T1'), (LE_eventTime, 'T2'). We place the query on the root view *EPCISEvent* to retrieve all events in the inheritance chain. The corresponding query statement has this following form: SELECT VALUE(e) FROM EPCISEvent e WHERE *predicates*. *Predicates* are generated specifically for each specialized event types. Therefore, the first thing we do is to retrieve all event types in the generalization hierarchy. We query the data dictionary recursively to have this job done.

```
SELECT DISTINCT TYPE_NAME FROM ALL_TYPES
START WITH owner = 'EPCISADMIN' AND
       type_name = 'EPCISEVENT_TYPE'
CONNECT BY PRIOR type_name = supertype_name
       AND PRIOR owner = supertype_owner;
```

Then we eliminate all event types that do not have event fields *bizLocation* and *eventTime* to prevent our system from wasting resource for processing unnecessary event types. The next step, we generate the predicates to filter out events that does not meet the requirements.

With parameter (GE_*eventTime*,*'T1'*), we map the uppercase operator string to appropriate logical operator and form the predicate for each event type, for example (TREAT VALUE(p) AS ObjectEvent_Type).*eventTime*>=*'T1'*. Similarly, parameter(EQ_*bizLocation*,'L1') is mapped to this predicate

(TREAT VALUE(p) AS ObjectEvent_Type).*bizLocation='L1'*

By starting the query from the root event and placing predicates on each specialized event, we can answer easily the SimpleEventQuery.

5.1.2 SimpleMasterQuery

SimpleMasterQuery retrieves the information about the vocabularies. Generating query statements for this query can be achieved in a similar manner as for SimpleEventDataQuery.

6 Experiments

We finished the first prototype of EPCIS for industrial enterprises based on the requirements posed in the EPCIS specification. This prototype included the two standard interfaces and a efficient repository which are mentioned mainly in this paper. Currently we have no existing RFID benchmark for our work. Therefore, we implemented an event generator in Java that produced events encoded in XML documents and pushed them up to EPCIS. EPCIS

catched and stored these events in the repository. We used Oracle 10g for the data repository and Java for implementation. We predefined two simple queries: SimpleEventQuery and SimpleMasterDataQuery (section 5.1) as recommended in the EPCIS Specification and let the EPCIS-accessing applications request these queries and have the answers on-demand. These applications can also register the queries and retrieve the result periodically according to the schedules provided.

7 RELATED WORK

RFID technology makes a fast progress in recent years and promises a bright future of labor cost reduction, business process automation and inventory inaccuracy reduction, etc. However, there are many challenges for data management we need to consider when deploying this technology [6][7][8]. Many researchers devotes their time to find solutions for these problems including building warehouse model [9][10], cleansing anomalies in RFID reads[11], using bitmap datatype for representing collections of EPCs [12]. There is also works on building a efficient data model for EPCIS repository that we already summarized in section 2.4.

Besides researchers, many IT organizations have invested in building RFID platforms including Sun Java System RFID Software [3], Oracle Sensor EdgeServer [13], IBM WebSphere RFID Premises Server [14], Sybase RFID Solutions [15], etc. These platforms have a similar architecture and handle all the complexity from the RFID physical world, presenting to the higher level applications a simple interface. Nowadays, these platforms also includes their own repositories for data storage. However, the data models are up to each organization as long as EPCIS can respond the requirements for system integration and data exchange.

Mentioned mostly in this paper is the EPCIS Specification proposed by EPCGlobal, the current EPC standard Group, that gave solution to most issues related to EPCIS data and standardized interfaces for disparate applications to share data, both within and across enterprise.

8 Conclusions

In this paper, we have proposed a strong data model for the EPCIS repository that is rich in semantics and has all prominent features from both relational and object-relational approaches. This model handles various kinds of data and relationships efficiently. Vocabularies in master data share the same general meta-data model with high formality. Dynamic data are modeled as temporal events in generalization hierarchy that is easy to extend more event types and look back on the event history. We proved that with this efficient model we can answer successfully the two recommended queries. Moreover, objects moving together and experiencing the same business processes are stored in the same event so that we can reduce the data volume, especially in the system that objects usually move in large groups.

Acknowledgement

This research was supported by the Ministry of Commerce, Industry, and Energy (MOCIE), Korea (10016466).

References

[1] EPCGlobal, *http://www.epcglobal.com/*, 2006.

[2] EPCglobal, *EPC Information Services (EPCIS) Version 1.0 Specification*, Mar. 2006.

[3] "Sun java system RFID software," *http://sun.com/rfid*, 2004.

[4] P. L. Fusheng Wang, "Temporal management of rfid data," In *Proc. of VLDB*, 2005.

[5] M. Harrison, "EPC information service - data model and queries," White paper, Auto-ID Centre for Manufacturing, University of Cambridge, February 2003.

[6] S. R. Suarshan S. Chawathe, Venkat krishnamurthy and S. Sarma, "Mananaging rfid data," in *Proc. of VLDB*, 2004.

[7] C. Hanebeck, "Managing data from rfid and sensor-based networks," GlobeRanger Corporation, Tech. Rep., 2003.

[8] M. Palmer, "Seven principles of effective RFID data management," Progress Software's Real Time Division, http://www.progress.com.mx/realtime/docs/articles/, Tech. Rep., 2004.

[9] X. L. D. K. Hector Gonzalez, Jiawei Han, "Warehousing and analyzing massive RFID data sets," in *Proc. of VLDB*, 2006.

[10] X. L. Hector Gonzalez, Jiawei Han, "Flowcube: Constructing RFID flowcubes for multi-dimensional analysis of commodity flows," in *Proceedings of the 22nd International Conference on Data Engineering*, 2006.

[11] M. J. F. Shawn R. Jeffery, Minos Garofalakis, "Adaptive cleaning for RFID data streams," in *Proc. of VLDB*, 2006.

[12] T. C. J. S. Ying Hu, Seema Sundara, "Supporting rfid-based item tracking applications in oracle DBMS using a bitmap datatype," In *Proc. of VLDB*, 2005.

[13] Oracle sensor edge server, *http://www.oracle.com/technology/products/iaswe/edge_server/index.html*, 2006.

[14] Websphere rfid premises server, *http://www.ibm.com/software/pervasive/ws_rfid_premises_server/*, December 2004.

[15] Sybase rfid solutions, *http://www.sybase.com/rfid*, 2005.