Towards Traceability across Sovereign, Distributed RFID Databases

Rakesh Agrawal[†] Alvin Cheung[‡] Karin Kailing[‡] Stefan Schönauer[‡]

[†]Microsoft Search Labs
1065 La Avenida
Mountain View, CA
rakesh.agrawal@microsoft.com

[‡]IBM Almaden Research Center
650 Harry Road
San Jose, CA
{alvin, kkailin, sschoena}@us.ibm.com

Abstract

Tracking and tracing individual items is a new and emerging trend in many industries. Driven by maturing technologies such as Radio-Frequency Identification (RFID) and upcoming standards such as the Electronic Product Code (EPC), a rapidly increasing number of enterprises are collecting vast amounts of tracking data. To enable traceability over the entire life-cycle of items data has to be shared across independent and possibly competing enterprises. The need to simultaneously compete and cooperate requires a traceability system design that allows companies to share their traceability data while maintaining complete sovereignty over what is shared and with whom.

Based on an extensive study of traceability applications, we introduce the formal concept of traceability networks and highlight the technical challenges involved in sharing data in such a network. To address these challenges, we present an innovative combination of query processing techniques from P2P networks and distributed as well as parallel databases with confidentiality enforcement techniques.

Keywords: traceability, RFID, query processing, data sovereignty, distributed databases

1 Introduction

Tracking and tracing individual objects throughout their lifetime is key to many applications, such as supply chain management, counterfeit detection, or targeted product recall. There are two main enablers for such tracking today: unique identification of objects, and automated recording of object movements. For example, the Electronic Product Code (EPC) [13] provides a standard to uniquely identify individual objects globally. Maturing technologies such as Radio-Frequency Identification (RFID) and new standards such as RuBee [25] allow automated recording of information about movement of objects.

The pharmaceutical industry provides a good example for the importance of accurate and efficient tracking and tracing of individual objects. The World Health Organization estimates that as much as 10% of the half trillion dollar pharmaceutical market is counterfeit [11]. To address this issue some states in the USA have introduced pedigree laws [8, 16]. These laws require that the complete history of a drug including its movement throughout the supply chain is recorded and verifiable at any point in time. This data can also be used to do timely and targeted recalls of defective drug batches instead of indiscriminate recalls of everything on the market as done today.

The required level of object traceability can only be achieved if all participants in the production and distribution process share object-related information. To enable end-to-end traceability, (possibly competing) enterprises have to cooperate. However, object movement and related data are valuable business information which companies are reluctant to share freely. Thus, any traceability data sharing system will have to grant enterprises the ability to define and enforce a policy that specifies the data to be shared, with whom and under what circumstances.

Reconstructing the trace of an object is complicated by the fact that objects may get assembled into other objects or may be packed into or unpacked from a container. Some RFID readers may record the appearance of only the outermost object (e.g. a crate) but may not read the tags of the contained objects (e.g. smaller boxes inside the crate). Thus information about associations has to be taken into account when constructing the complete history of an object.

Finally, the scalability of the data management system is a crucial factor. The volume of data generated as a consequence of having RFID-tagged objects is enormous. For example, a major electronics manufacturer estimates that the RFID implementations at three of its manufacturing plants alone will generate between 1 to 5 Terabytes of data per day [17]. In addition, the number of organizations that adopt RFID technology to enable traceability is growing rapidly.

The large amount of data and the strong confidentiality

concerns prohibit the use of centralized indexing or querying services and demand new approaches to the problem of data sharing and query processing in distributed environments. In this paper, we take the first step towards such a system. We will describe a system design for data management in traceability networks, geared towards efficiently and effectively supporting traceability applications while maintaining the sovereignty of each participant over its data. Specifically, we address the following challenges: unifying traceability functionality to simplify application development, balancing benefits and risks of sharing traceability data, and providing a scalable system design to cope with large-scale RFID deployments.

We assume in our work that traceability data, such as RFID readings, is already captured, cleansed, and stored in one or more databases. There is rich literature on these topics (*e.g.* [12, 26]); we build upon this body of prior work. We also do not discuss issues related to time synchronization among different systems, which has been studied in sensor networks (*e.g.* [35]) as well as in ad-hoc networks (*e.g.* [32]).

The rest of this paper is organized as follows. Section 2 reviews related work on RFID data management. Section 3 gives a definition of traceability networks and introduces a conceptual and logical data model for traceability applications. Section 4 describes a new system architecture for traceability query processing, which is discussed and evaluated in Section 5. Section 6 concludes the paper.

2 Related Work

Traceability data management touches many active research areas. In this section, we give a brief overview of research and industry approaches that address different parts of the outlined challenges.

2.1 Related Research Areas

Distributed, Federated, and P2P Databases. Distributed [29] and federated [33] database systems approach query processing across multiple sites by leveraging global knowledge about the data distribution. P2P content-sharing networks [4, 19] either flood queries or use globally available information about data distribution (*e.g.* in the form of distributed hash-tables [34]) to route queries. In traceability networks, data distribution, *i.e.* movement of objects between companies, is confidential information and not to be shared globally.

Confidentiality Enforcement. Increasing legal requirements to keep personally identifiable information private inspired the development of policy-based, privacy-preserving databases [20, 2]. Similarly, confidentiality requirements can be expressed as a policy and technologies such as Hippocratic Database Active Enforcement [28] can ensure that based on the query issuer and the purpose of a query, information is only revealed in accordance with the confidentiality policy.

Moving Object Databases. Querying large sets of moving objects is a well-studied research topic in the context of spatio-temporal databases [23]. Moving object databases were mostly designed for application scenarios with a defined set of moving objects (e.g. all taxis) whose location is recorded continuously. In contrast, supply chains monitor a constantly changing set of objects whose location is only recorded at the few places at which RFID readers are installed.

RFID Data Management. An RFID cube is introduced in [21] to support warehousing and analysis of massive RFID data sets. Oracle presented a new bitmap data type for Oracle DBMS to support RFID-based item tracking applications in [24]. Both approaches assume that RFID data is stored within a single data repository. Apart from this work, academic research has mostly focused on privacy and security issues surrounding RFID technology. An overview can be found in [5]. However, confidentiality of RFID data once it is read and stored has so far not been addressed.

2.2 Industry Status

Despite all the work in academic research there is currently no industry solution that fully supports tracking of items across independent organizations.

In a 2004 paper, SAP presented an overview of their existing RFID infrastructure pointing out that a distributed infrastructure that supports sharing and synchronization of data across multiple nodes is still an open issue [7]. In 2005, Siemens described their middleware for RFID data management [36]. Their data model uses relations to store how long an object stayed at a certain location and how long an object was associated with another object. This would require that independent enterprises update each other's data which clearly violates the data sovereignty requirement.

Lacking a compelling solution, large retail companies have started to build traceability warehouses. However, as competing retailers are not wiling to store their data in a shared repository, suppliers are forced to continuously upload their respective RFID data to the warehouse of every retailer they deliver to. As the amount of RFID data increases, the total amount of data that needs to be published may put serious constraints on a central warehouse approach.

Standardization efforts by industry consortia such as EPCglobal (formerly Auto-ID Center) [13] have promoted the adoption of RFID technology. EPCglobal has proposed an architecture for a network of RFID databases where each

database provides a standardized query interface [14, 15, 9]. To enable sharing of RFID data across organizations, central registries are envisioned that keep track of each object movement between organizations. At this point it is unclear how to realize these central registries in a way that allows companies to enforce their confidentiality requirements. While the EPCglobal model supports storing and retrieving of traceability data it does not provide support for cross-organizational traceability functionalities such as recursion, aggregation, and joins across databases.

3 Traceability Networks

As neither research nor industry have sofar formalized the concept, we introduce the notion of traceability networks. The definitions outlined in this section are based on an in-depth study of a number of different traceability applications including supply chain optimization, pedigree generation, product recall, patient surveillance, and cargo tracing.

3.1 Definitions

We define *traceability* as the ability to track the current and all previously recorded states of an object. The *state* of an object changes over time and includes its spatial location, association with other objects, and other properties such as temperature.

We call locations that are equipped with RFID readers *sensing locations* (*e.g.* conveyor belt #1, dock door #2). Each location belongs exclusively to one *organization* (*e.g.* manufacturer, distributor, hospital).

An *event* is the recording of the state of an object at a sensing location at a certain time.

Recording the spatial location of objects establishes an *object movement graph*. The nodes are locations and a directed edge $e(l_1, l_2)$ between locations l_1 and l_2 labeled $\{o_1, ..., o_n\}$ denotes that objects $o_1, ..., o_n$ moved from location l_1 to location l_2. Figure 1 shows an example of an object movement graph. Dotted lines indicate object movement, and dotted smaller circles denote sensing locations.

If the locations involved in building a complete object movement graph are distributed across multiple organizations, only a subgraph of the object movement graph is available at each organization. The subgraph associated with organization X contains only locations that belong to X.

Let Org be a set of organizations, Loc the set of all locations belonging to any organization in Org, Obj the set of all objects sensed at any location in Loc and OMG the object movement graph associated with Loc and Obj.

A *traceability network* is a network where the nodes are organizations in Org, and an edge $e(org_1, org_2)$ between two organizations org_1 and org_2 denotes that there is at least one edge $e(loc_i, loc_j)$ in the associated object movement graph OMG with loc_i, loc_j belonging to org_1, org_2 respectively. $e(org_1, org_2)$ is labeled with the union of the labels of all such edges $e(loc_i, loc_j)$.

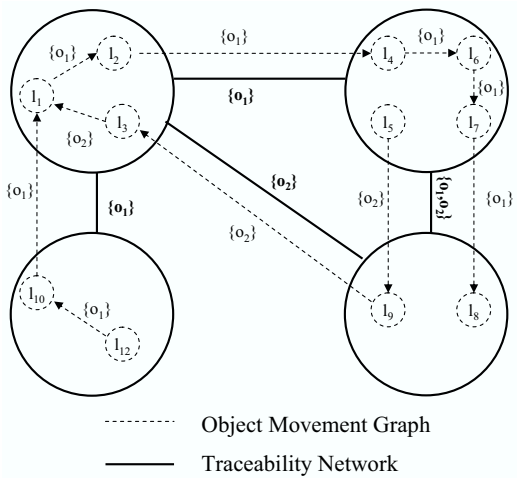

Figure 1. Example of an Object Movement Graph and a Traceability Network

Figure 1 shows an example of a traceability network. Solid larger circles denote organizations, and solid lines denote movement of objects between organizations.

The core of such a traceability network is the data stored at each node. In the remainder of this section we introduce concepts necessary for participants of a traceability network to share their data. We introduce a conceptual data model for traceability, we show how this model can be transformed into a logical data model, and finally demonstrate how traceability queries can be expressed in the derived logical model.

3.2 A Conceptual Data Model for Traceability

A common conceptual model for traceability data provides participants of a traceability network with the ability to specify queries across the entire network. It allows the formulation of a query without knowledge about how the data is stored at each organization, where it is located, and how a query is executed. We use ER constructs [10] to describe the model.

Figure 2 shows our basic traceability model. We represent the essential building blocks of the object movement graph and the traceability network as entity types $Object$, $Location$, and $Organization$. The relationship type $BelongsTo$ associates a location with its organiza-

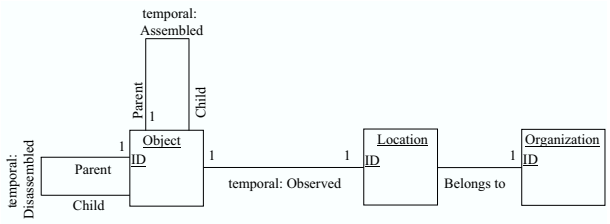

Figure 2. ER Diagram of the Basic Model

tion. To represent events, we use the relationship types $Observed$, $Assembled$, and $Disassembled$.

Observed. The relationship $Observed$ captures that an object was seen at a certain location at a certain time. Thus, we have a ternary relationship between object, location, and a timestamp. For readability we introduce the notation 'temporal:Observed' in the ER diagram to denote that each relationship instance is associated with a timestamp [18].

Objects can be associated with each other such that one object is the parent of another object. Examples of such hierarchical associations are packing (*e.g.* smaller boxes placed inside a crate) and product assembly (*e.g.* an engine block used in a car). We use the term $assembled$ to denote a hierarchical association.

Assembled. The relationship $Assembled$ captures that two objects start a parent-child-relationship at a certain time.

Disassembled. The relationship $Disassembled$ captures that two objects end a parent-child-relationship at a certain time.[1]

The relationship type $Observed$ *implicitly* represents the edges of an object movement graph, thereby mitigating the need for an organization to know about another's locations. This relationship type, together with the relationship type $BelongsTo$, similarly represents the edges of a traceability network. This approach necessitates a join to reconstruct the edges, but it preserves confidentiality.

The basic model captures common requirements of traceability applications (i.e. tracking object movement and recording associations). However, some organizations or traceability applications may need to capture additional information. To satisfy such needs, we extend the basic model with the notion of *properties* (name-value pairs). Figure 3 shows the ER diagram for the extended model. The dotted lines indicate aggregation of a relationship [31].

[1] We assume that assembly or disassembly events cannot be generated without observing the involved objects, i.e any assembly or disassembly event is always preceded or followed by an observation event. In this case, the location where an assembly or disassembly takes place can be computed by joining the observation event with the "closest" timestamp. If assembly or disassembly does not require observation, the relationships $Assembled$ and $Disassembled$ can be extended to a ternary temporal relationship between objects and a location.

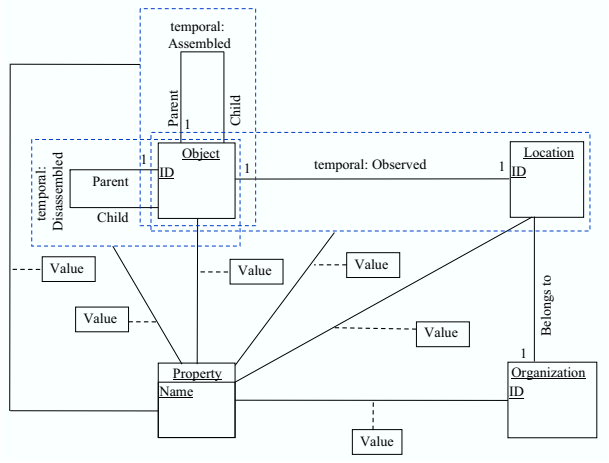

Figure 3. ER Diagram of the Complete Model

With this extension, properties can be associated with an organization, a location, or an object (*e.g.* company name, location address, production date, respectively). Properties can also be associated with any of the three relationships *Observed*, *Assembled*, and *Disassembled*. For example, when a container is observed, its temperature and humidity values may be recorded as properties of the relationship *Observed*.

The extended model allows organizations and applications to represent proprietary information without imposing proprietary schemas on other participants.

3.3 A Logical Data Model for Traceability

For our implementation and to illustrate query execution in a traceability network, we chose to convert the conceptual model to a relational model. We call this relational model the *global schema* and assume that every organization in the network knows about it. We use a subset of SQL to express queries over that schema. (We could have chosen a different logical model, for example an XML-based model similar to that proposed by EPCglobal [13].)

Figure 4 shows the relational schema generated from our conceptual model. The attributes *parent* and *child* are object identifiers (*oid*s) and the attribute *ts* is a timestamp.

This logical model allows organizations to specify a query without prior knowledge about where and how data is stored within the network. Physically, the global relations Observed, ObsPropertySet, Assembled, AsmPropertySet, Disassembled, DsmPropertySet are partitioned horizontally such that each partition belongs exclusively to one organization.

A query specified against the global schema needs to be executed against the local schema and data model of each

Property(name)
Organization(gid)
OrgPropertySet(gid, propertyName, value)
Location(lid, parentLid, gid)
LocPropertySet(lid, propertyName, value)
Object(oid)
ObjPropertySet(oid, propertyName, value)
Observed(oid, lid, ts)
ObsPropertySet(oid, lid, ts, propertyName, value)
Assembled(parent, child, ts)
AsmPropertySet(parent, child, ts, propertyName, value)
Disassembled(parent, child, ts)
DsmPropertySet(parent, child, ts, propertyName, value)

Figure 4. Logical Data Model

organization. An organization is free to choose a local relational schema different from ours, or even use a different data model such as XML. For example, in our experiments we used a different relational schema to improve the efficiency of local query processing. As we only used a fixed number of properties, we chose a flat representation that requires fewer joins than the generic global representation (for details see Section 5). To execute a global query on the local schema an organization can either provide a mechanism to rewrite global queries to conform to their schema or provide a view of their data that fits the global schema.

3.4 Important Traceability Queries

Based on the logical model, we are now able to formulate global traceability queries, *i.e.* traceability queries whose result may depend on data from multiple organizations. In this subsection, we introduce three common traceability query types (pedigree, recall, and bill-of-material) and briefly talk about other queries.

Pedigree Query. Pedigree queries are used to reconstruct the complete history of an object. A pedigree query has the following general form (expressed in relational algebra):

$$Q1 : \pi_{pList}(\sigma_{oid=o}(T \rtimes S))$$

where $\rtimes$ denotes the left outer join, T is one of the relations *Observed, Assembled, Disassembled*, and S is one of the relations *ObsPropertySet, AsmPropertySet, DsmPropertySet* respectively, and $pList$ is a (sub)set of attributes of $T \rtimes S$.

As outlined in Section 3.3, all global relations are partitioned horizontally such that each partition belongs exclusively to one organization. For pedigree queries, the partitions of S and T that fulfill the join condition are always located at the same organization. Thus, it is sufficient to execute the query at each organization where the respective partitions have at least one tuple that matches the selection criteria. A complete result can be computed by a union of the individually retrieved results. The key issue is how to detect all such partitions. We will address this in Section 4 when we describe how query processing is done in our architecture.

Recall Query. Recall queries are used to detect the current location of an object. A recall query has the following general form:

$$Q2 : \pi_{lid}(max_{ts}(\sigma_{oid=o}(Observed)))$$

A recall query needs to be executed at each organization where there is an entry with $oid = o$ in the respective partition of the relation $Observed$. An overall maximum can be built on all retrieved results to detect at which location the object was last seen (because max is distributive).

Bill-of-Material Query. Recursion is needed to express traceability queries that involve assembly and disassembly of objects. We use α-extended relational algebra [1] to express such queries. A bill-of-material (BOM) query asks for everything that is contained in an object.

$$Q3 : \pi_{child}(\sigma_{parent=o}(\alpha(\pi_{parent,child}(Assembled)))$$

where $\alpha(\pi_{parent,child}(Assembled))$ computes the transitive closure of *Assembled* using the attributes *parent* and *child*. Note, that disassembly is not taken into account in this example. How to execute recursive queries in a distributed environment is described in Section 4.

Other Queries. In addition to the three outlined query types, there are a lot of other useful traceability queries.

While the join between event data and event property data in a pedigree query can be computed locally at each organization, there are other traceability queries where the join needs to be computed for data from different organizations. "Which objects have exceeded their maximum allowed storage temperature throughout their lifetime?" is an example of such a query.

$$\begin{aligned} Q4 : \pi_{oid}(\sigma_{ObjectPropertySet.value<ObsPropertySet.value} \\ (\sigma_{propertyName='maxTemp'}(ObjPropertySet) \\ \bowtie_{oid} \\ \sigma_{propertyName='temperature'}(ObsPropertySet))) \end{aligned}$$

In this case, the information about the maximum allowed storage temperature is only stored at the organization that created the object. It needs to be joined with observed temperature readings which are spread across all organizations on the object's path.

Based on the data model and the traceability query types presented in this section, the next section discusses query processing for traceability applications.

4 An Architecture for Traceability Query Processing

In this section, we introduce an architecture to support query processing for arbitrary traceability applications. This architecture combines all query capabilities in a query engine middleware to support information sharing across multiple organizations. The fundamental principle of our approach is to process a query locally to the maximum extent possible and, if necessary, enhance it using locally available information before forwarding it to appropriate remote organizations for further processing. Given a global traceability query, our query engine rewrites it exploiting local data, locates remote data sources, forwards it to remote query engines, and combines local and remote results. This process repeats itself at the other nodes allowing each node to enforce all confidentiality constraints before returning local results. The basic idea is depicted in Figure 5. We show an approach where the query engine is completely independent from any central services to locate the necessary data sources. However, if location services are available the query engine is able to exploit them.

In our architecture, organizations are peers, running the same query-engine middleware. A naming service exists in the network that assigns a unique address to each node for communication purposes (similar to the DNS in the Internet). In the rest of this section, we describe our query processing algorithms.

4.1 Overview of Query Processing

In our approach the complete distribution of data across nodes is not known in advance, instead it is discovered gradually while processing the query. The number of nodes accessed in order to completely process a query is kept minimal. To maintain the information about the path an object took, each organization is required to store two additional properties for each shipping or receiving event, namely *sentTo* and *receivedFrom*. This information can be gained by joining traceability data with other enterprise data such as billing or accounting information. If an organization is not able to provide the precise data, the number of nodes in the network that have to be visited increases, in the worst case all trading partners need to be contacted.

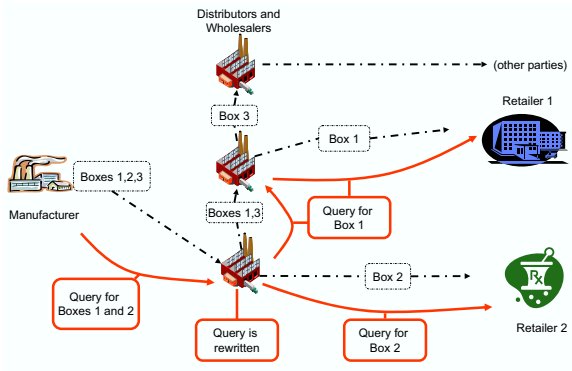

Figure 5. Query Flow in the Process-and-forward Approach.

processAndForward(q)
1 //analyze and rewrite query appropriately
2 $\{q_{remote}, q_{local}, q_{post}\}$:= analyzeAndRewrite(q)
3 //obtain results from appropriate remote nodes
4 if (q_{remote} != null)
5 r := forwardAndCombineResults(q_{remote})
6 //combine local results with remote results
7 if (q_{local} != null)
8 $r = r \cup$ executeQuery(q_{local}, $localData$)
9 //postprocess combined results
10 if (q_{post} != null)
11 r = executeQuery(q_{post}, r)
12 return r

Figure 6. Algorithm processAndForward

Figure 6 is a pseudo code description of the main steps of the algorithm that runs within the query engine at each organization. An incoming query is analyzed and rewritten if necessary. The details for the procedure *analyzeAndRewrite* are given in Figure 8 and discussed below. The rewritten query is then forwarded to other organizations in the network and the results are united. Section 4.2 describes how the relevant organizations are detected. The algorithm for the procedure *forwardAndCombineResults* is described in Figure 7. The global query is also translated to a query that can be executed on the organization's data. Policy enforcement [28] is used to limit data disclosure. This enforcement is done by further rewriting the query based on the query issuer and the purpose of the query. The rewrite ensures that only tuples compliant with the policy are retrieved from the data system. The local result is added to the results retrieved from the network. If necessary, postprocessing is performed on the results. We illustrate the details of the query execution using the example queries given in Section 3.4.

4.2 Query Forwarding

As mentioned in Section 3.4, all that needs to be done for basic pedigree and recall queries is to execute them at all relevant organizations and "combine" the results. Relevant nodes can be detected by analyzing all selection predicates in a query. If a selection predicate specifies locations or organizations (*e.g.* a query asking only for the pedigree at a specific location) the query only needs to be forwarded to

```
forwardAndCombineResults(q)
1   //collect information for query routing
2   orgsRelevant := extractOrganizations(q)
3   objsRelevant := extractObjects(q)
4   //forward query to all relevant organizations
5   for each org in orgsRelevant
6       r = r∪ forward(org, q)
7   for each obj in objsRelevant
8       q' = restrictTo(q,obj)
9       if (obj is known)
10          if (fromOrg(obj) != null)
11              r = r∪ forward(fromOrg(obj), q')
12          if (toOrg(obj) != null)
13              r = r∪ forward(toOrg(obj), q')
14      else
15          r = r ∪ flood(q')
16  return r
```

Figure 7. Algorithm forwardAndCombineResults

the corresponding organizations.

The general pedigree query (Q1) does not specify any location or organization. However, as the query specifies a list of relevant object identifiers, relevant organizations need to be on the paths of these objects. If the objects are known (*i.e.*, at least one observation event per object exists locally) they can be forwarded based on information specified by the properties *sentTo* and *receivedFrom*. For unknown objects the query needs to be flooded to the network. If two objects o_1 and o_2 were received from different organizations, two queries restricted to o_1 or o_2 respectively are sent out to the organizations from which o_1 and o_2 were received. The pseudo code of procedure *forwardAndCombineResults* is given in Figure 7. Note that, although it is not shown in the pseudo code, our implementation groups objects and sends out only one query for each relevant organization.

4.3 Query Rewriting: Non-Local Join

Certain queries cannot be executed as-is at every organization as data needs to be joined across organizations. There are many general query execution methods for distributed query processing (see [27] for a survey). However, these methods are based on the availability of the knowledge about which data sources need to be accessed, in order to optimize the query plan. This knowledge is not available in the process-and-forward approach before starting query execution. A solution is to split the query such that each resulting query has only local joins. A postprocessing query is generated that operates on the results retrieved for the individual queries and produces the final result. For example, query Q4 is split into $q1 = \sigma_{propertyName=maxTemp}(ObjPropertySet)$ and

```
analyzeAndRewrite(q)
1   q_remote := q
2   //handle non-local joins
3   qSet := splitIntoLocalJoinQueries(q)
4   rSet := ∅
5   q_final := buildFinalQuery(q, qSet)
6   for each q_split in qSet
7       rSet = rSet ∪ processAndForward(q_split)
8       return executeQuery(q_final, rSet)
9       //algorithm terminates
10  //handle recursion
11  for each recursive component p in q
12      r_temp := executeQuery(p, db_local)
13      addResultsToRemoteQuery(q_remote, p, r_temp)
14  //handle aggregate functions
15  for each aggregate function f in q
17      if (f is avg(X))
18          //need count to compute overall average
19          addQueryFragement(q_remote, "count(X)")
20          //determine postprocessing
21          addToPostprocess(q_post, f)
22      //process other aggregate functions
23  //build local query
24  q_local := mapToLocalSchema(q_remote)
25  q_local = enforcePolicy(q_local)
26  return {q_remote,q_local,q_post}
```

Figure 8. Algorithm analyzeAndRewrite

$q2 = \sigma_{propertyName=temp}(ObsPropertySet)$. A query $q_{final} = \pi_{oid}(\sigma_{rq1.value<rq2.value}(rq1 \bowtie_{oid} rq2))$ is built and executed on the result $rq1$ and $rq2$. The pseudo code for handling non-local joins is given in Figure 8, lines 2-9. There is a lot of optimization potential for executing non-local joins. However, this is beyond the scope of this paper.

4.4 Query Rewriting: Aggregate Functions

Queries containing aggregate functions require additional treatment. For certain aggregates, such as average, the remote query needs to be extended to retrieve additional information from the remote sites. For all aggregates a postprocessing query needs to be created that computes the overall aggregation based on the individual aggregation values returned. The pseudo code for handling aggregation is given in Figure 8, lines 14-22. Distributive aggregate functions such as summation, algebraic aggregate functions such as standard deviation and holistic aggregate functions such as median have to be handled appropriately [22]. We only show the code for executing the algebraic aggregate function average as an example.

	Observed			ObsPropertySet			Assembled		
	oid	lid	ts	... propertyName	value		parent	child	ts
CYM	cylinder1	l1	t1	sentTo	ENM				
	cylinder2	l1	t1	sentTo	ENM				
ENM	cylinder1	l5	t5	receivedFrom	CYM		engine1	cylinder1	t10
	cylinder2	l5	t5	receivedFrom	CYM		engine1	cylinder2	t10
	cylinder1	l10	t10						
	cylinder2	l10	t10						
	engine1	l10	t10						
	engine1	l15	t15	sentTo	CAM				
CAM	engine1	l20	t20	receivedFrom	ENM		car1	engine1	t25
	engine1	l22	t25						
	car1	l22	t25						

Figure 9. Example Data

4.5 Query Rewriting: Recursion

We handle recursive queries by expanding the recursion locally as much as possible and adding local recursion results to the queries that are forwarded. The pseudo code for handling recursion is given in Figure 8, lines 10-13. For illustration, we use the data shown in Figure 9 produced by three organizations, a cylinder manufacturer CYM, an engine manufacturer ENM, and a car manufacturer CAM. An application at CAM is interested in all parts contained in object *car1*. The query engine at CAM determines that the query has a recursive component p = "$\alpha Assembled$". Executing this query on the locally available table $\alpha Assembled$ returns $rtemp = engine1$. Based on this information q_{remote} is rewritten to $\pi_{child}(\sigma_{parent=`car1` \vee parent=`engine1'}(\alpha Assembled))$. The query is forwarded to ENM. At ENM the same process takes place and ENM detects that *engine1* contains *cylinder1* and *cylinder2*. The query is forwarded to CYM where forwarding terminates as the recursion is no longer expanded.

4.6 Result Routing

To return results in the process-and-forward approach, two possible solutions exist. One possibility is that each node directly sends back any local results to the node that issued the query. In this approach, a node does not know if it has received all answers to a query, since it has no information about how many nodes are supposed to answer. This could be resolved by informing the query issuer to which other nodes a query has been forwarded. However, this results in a large number of additional messages and processing overhead at the query origin. It also reveals to the query issuer which organizations are potentially on the path of a queried object, even if one of those organizations is not willing to reveal itself.

A second approach, which we chose for our implementation, is to propagate the results back along the path the query took. A node knows that it has *complete results*, when *all* nodes it has contacted have answered; otherwise it has only *partial results*. In this approach the path a query took is only revealed to the extent that can be derived from the query result.

5 Evaluation

In this section we discuss how the proposed solution addresses the challenges outlined in Section 1 and give a brief experimental evaluation.

5.1 Addressing Challenges

Unifying Traceability Functionality Our architecture provides a global view of traceability networks. We define a common schema to allow participants in the traceability network to express global queries. However, our query engine middleware is not dependent on the specific schema proposed but will work off any common schema that does not introduce data dependency between organizations. We described our solution based on the traceability queries outlined in Section 3.4, but the approach is not limited to those queries. The outlined principles allow support for arbitrary SQL and thus, support any traceability application. With the ability to execute recursive queries in a distributed manner, recursive relationships such as product assembly or container hierarchies are handled naturally.

Balancing Benefit and Risk Our new solution provides the following benefits. (1) Queries are always executed against the most recent data as no data needs to be uploaded to a central location. (2) Each organization is given complete sovereignty over all its data including information about its trading relationships. With technologies such as Hippocratic Database Active Enforcement [28], confidentiality requirements can be enforced on a query by query basis. Each incoming query is rewritten based upon preinstalled data sharing policies. Policies can be installed and modified without interaction with other parties.

Issues such as security, trust, and reliability need to be further examined in order to mitigate the risks. These topics are also addressed in P2P networks and related areas and some of the results can be directly applied to our solution. An in-depth discussion is beyond the scope of this paper.

Scalability With the increasing amount of RFID data, publication of that data to a central location is not practical. Since only a small fraction of the produced data will actually be queried, shipping queries instead of data appears to be far more efficient. In our approach only the text of the queries and the results are transferred among organizations. In realistic scenarios, the size of the result sets is typically only a tiny fraction of the total amount of RFID data stored.

In the next subsection, we investigate the overhead for query processing incurred by our distributed solution.

5.2 Experiments

To investigate the scalability of our approach, we implemented a prototype system. For comparison purposes, we also implemented a central warehouse solution and the EPCglobal architecture, along with a sample traceability application on top. Throughout this section, we call the three approaches process-and-forward, central warehouse, and EPCglobal.

Each architecture was implemented in Java 1.4 as a webservice using AXIS 1.2 [6] and all communication between nodes was carried out via webservice calls. For query analysis and rewriting, the QGM model [30] was used. The central services in the EPCglobal model as well as the network DNS were implemented using the light-weight Vinci naming service [3]. Each service was running on a separate machine. Each party in the network was represented by a single machine with a Pentium IV processor with 2.4GHz and 1GB RAM. Each machine was running the same software consisting of Linux, a DB2 UDB V8.2 instance and the respective architecture implementation. The times measured are wall-clock times and include the latency of the local area network.

To the best of our knowledge, there is no real RFID data set publicly available. Therefore, we conducted our experiments on artificially generated data. For this purpose we implemented a reference data generator for RFID traceability data which is publicly available on the following website: *http://www.almaden.ibm.com/software/projects/iis/rfid*.

We developed our data generator based on data characteristics we found in our application scenarios. It generates observation events, association events and property data for a given number of objects. The data generator is given a specific network layout and a set of nodes at which objects may be created. It creates data for the following schema:

Observed(oid, lid, ts, obsType, temp)
Location(lid, locationName, locationAddress)
ObjectProperties(oid, color, maxTemp, minTemp)

The data is generated as follows. After an object is created, one of three actions is taken for that object: It is shipped to a neighboring node, associated with an existing object, or its path is terminated. The action is randomly chosen and the probability of shipping an object to a neighbor gradually decreases with the length of the object's path. For the process-and-forward approach appropriate entries for *receivedFrom* and *sentTo*, and for the EPCglobal approach data entries for the central registries are created.

To investigate the effects of the different architectures on query response times, we created data sets of 5 and 10 million objects in a supply chain of length 10. Due to space limitations we only show the results for the data set with 5 million objects. The results for 10 million objects are similar to those with the smaller data set.

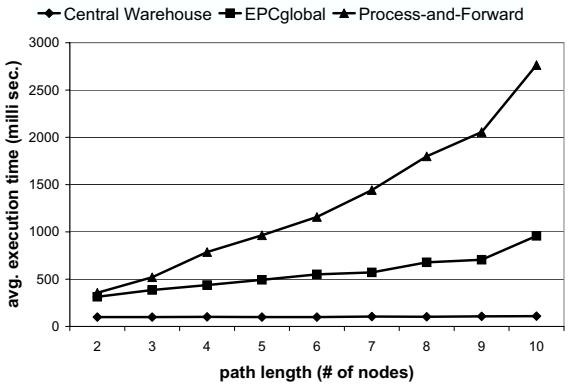

Figure 10. Average runtime for pedigree queries.

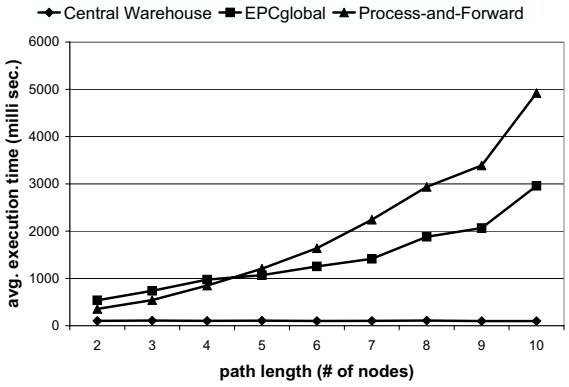

Figure 11. Average runtime for BOM queries.

We measured the overall runtime for pedigree queries depending on the length of the path an object took through the supply chain. Figure 10 shows the average runtime for 100 queries, each asking for all observations of 15 different objects, starting at one end of the supply chain. The experiments show that, as expected, the response time for the process-and-forward approach grows with increasing path length as queries are forwarded and results propagated back along the path. Since response times are dominated by the communication cost, the central warehouse approach does not depend on the path length. The EPCglobal approach is only slightly affected by it, because after retrieving all relevant nodes from the discovery service, all the remote data sources can be contacted in parallel. However, the web service calls are still initiated sequentially.

Figure 11 shows the results for the same experimental setup for BOM queries. In this experiment, the depth of the containment hierarchy was on average one and at

most two. For this experimental setting, the process-and-forward approach outperforms the EPCglobal approach for path lengths up to 4. This is due to the fact that for the EPCglobal model in each recursion step a discovery service lookup is necessary to determine the remote data sources for all contained objects retrieved in the previous step. Subsequently, all these data sources need to be queried to find out which objects are contained in the object identified in the previous step. This process continuous until no further contained objects are found. For the experimental setup this results in an average of 4 sequential web service calls for the EPCglobal approach. The process-and-forward approach, on the other hand, computes the contained objects on the fly while forwarding the query along the path and thus, is faster than the EPCglobal approach for a path length of up to 4. Note, that the break even point varies with the depth of the containment hierarchy.

Our experiments show that the choice of architecture has noticeable effects on the query response times. The higher response times for EPCglobal are mostly due to the overhead incurred by processing multiple web service requests. Propagating queries and results sequentially leads to waiting times and thereby further increase the overall query response time in the process-and-forward approach. Note, that the load on each node is similar for EPCglobal as well as process-and-forward approach. In a practical application of traceability networks, these query performance implications have to be carefully weighed against data uploading and data confidentiality requirements.

6 Conclusion

Sharing RFID data across independent organizations is essential for traceability applications. In this paper, we introduced the formal notion of a traceability network. We described a generic and extensible conceptual data model for RFID data, along with a realization of the model in relational schemas. We proposed a new architecture and algorithms for processing traceability queries across organizations. Our architecture and middleware enable rapid development of traceability applications on top of it. To the best of our knowledge this is the first solution to traceability in a completely distributed setting. Our approach allows enterprises to get the full benefit of participating in a traceability network while offering them complete sovereignty to decide on a query by query basis which data will be shared.

This work can be extended in many directions; we highlight two.

Hippocratic database technology allows each participant within the RFID network to exactly define and enforce which data will be released to which query issuer. However, a company has no control over what happens to its data once it is released. Ensuring the privacy and confidentiality of data once it has been disclosed, is an issue not only in our approach, but in any type of data sharing network. To reach this goal the confidentiality requirements would have to be attached to the disclosed data while it travels through the network.

Another important field of research in this area is data analytics. Electronic pedigree generation is a first step towards traceability network analysis. However, more sophisticated mining algorithms are necessary to automatically detect malicious activities in the network such as counterfeiting. A few of the interesting new mining tasks in this field are: detecting 'suspicious' events in a traceability network; given a 'problem pattern', detecting individual items whose traces resemble that pattern; given a set of problematic items, detecting commonalities among them or even identifying the source of the problem.

These are only two examples of open research problems in the area of traceability networks. By providing a new and more comprehensive approach to traceability data management, we hope to inspire more research in this field.

References

[1] R. Agrawal. Alpha: An Extension of Relational Algebra to Express a Class of Recursive Queries. *IEEE Transactions on Software Engineering*, 14(7):879–885, 1988.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic Databases. In *VLDB '02: Proc. of 28th Int. Conf. on Very Large Data Bases*, pages 143–154, 2002.

[3] R. Agrawal, J. Roberto J. Bayardo, D. Gruhl, and S. Papadimitriou. Vinci: a Service-Oriented Architecture for Rapid Development of Web Applications. In *WWW '01: Proc. of the 10th Int. World Wide Web Conference*, pages 355–365, 2001.

[4] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Survey*, 36(4):335–371, 2004.

[5] G. Avoine. Radio Frequency Identification: Adversary Model and Attacks on Existing Protocols. Technical Report LASEC-REPORT-2005-001, Swiss Federal Institute of Technology in Lausanne, 2005.

[6] Apache Axis. `http://ws.apache.org/axis/`.

[7] C. Bornhövd, T. Lin, S. Haller, and J. Schaper. Integrating Automatic Data Acquisition with Business Processes - Experiences with SAP's Auto-ID Infrastructure. In *VLDB '04: Proc. of the 30th Int. Conf. on Very Large Data Bases*, pages 1182–1188, 2004.

[8] California Business and Professions Code Sections 4163. http://www.leginfo.ca.gov/cgi-bin/calawquery?codesection=bpc&codebody=4163&hits=20.

[9] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. Sarma. Managing RFID Data (Extended Abstract). In *VLDB '04: Proc. of the 30th Int. Conf. on Very Large Data Bases*, pages 1189–1195, 2004.

[10] P. P.-S. Chen. The Entity-Relationship Model Toward a Unified View of Data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[11] T. Datz. Drug Busters. *CSO Magazine*, 2005.

[12] A. Deshpande, C. Guestrin, and S. R. Madden. Using Probabilistic Models for Data Management in Acquisitional Environments. In *CIDR '05: 2nd Biennial Conf. on Innovative Data Systems Research*, pages 317–328, 2005.

[13] EPCglobal. http://www.epcglobalinc.org/.

[14] EPCglobal. The EPCglobal Architecture Framework. *Final Version of 1 July*, 2005.

[15] EPCglobal. EPC Information Sevices (EPCIS) Version 1.0 Specification. *Working Draft Version of 6 March*, 2006.

[16] Florida Statutes, Section 499.0121. http://election.dos.state.fl.us/laws/04laws/ch_2004-328.pdf.

[17] Forrester Research. RFID: The Complete Guide. *Forrester Collection*, 2005.

[18] M. Fowler. *Analysis Patterns: Reusable Object Models*. Addison Wesley, 1997.

[19] F. Giunchiglia and I. Zaihrayeu. Implementing Database Coordination in P2P Networks. Technical Report DIT-03-035, University of Trento, Department of Information and Communication Technology, 2003.

[20] S. Godik and T. Moses. eXtensible Access Control Markup Language (XACML) Version 1.0, OASIS Standard, 18 February, 2003.

[21] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and Analyzing Massive RFID Data Sets. In *ICDE '06: Proc. of the 22nd Int. Conf. on Data Engineering*, page 83, 2006.

[22] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1), 1997.

[23] R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann Publishers, 2005.

[24] Y. Hu, S. Sundara, T. Chorma, and J. Srinivasan. Supporting RFID-based Item Tracking Applications in ORACLE DBMS Using a Bitmap Datatype. In *VLDB '05: Proc. of the 31st Int. Conf. on Very Large Data Bases*, 2005.

[25] IEEE Begins Wireless, Long-Wavelength Standard for Healthcare, Retail and Livestock Visibility Networks. http://standards.ieee.org/announcements/pr_p19021Rubee.html.

[26] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. A Pipelined Framework for Online Cleaning of Sensor Data Streams. Technical Report UCB/CSD-5-1413, University of California Berkeley, 2005.

[27] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.

[28] K. LeFevre, R. Agrawal, Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt. Limiting Disclosure in Hippocratic Databases. In *VLDB '04: Proc. of the 30th Int. Conf. on Very Large Data Bases*, pages 108–119, 2004.

[29] T. Özsu and P. Valduriez. *Principles of distributed database sytems*. Prentice Hall, 2 edition, 1999.

[30] H. Pirahesh, J. M. Hellerstein, and W. Hasan. Extensible/Rulebased Query Rewrite Optimization in Starburst. In *ACM SIGMOD Conference on Management of Data*, pages 39–48, 1992.

[31] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 3 edition, 2003.

[32] K. Römer. Time Synchronization in Ad Hoc Networks. In *MobiHoc '01*, pages 173–182, 2001.

[33] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[34] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the ACM SIGCOMM 2001 Conference*, pages 149–160, 2001.

[35] Sundararaman, B. and Buy, U. and Kshemkalyani, D. Clock Synchronization for Wireless Sensor Networks: A Survey. *Ad-hoc Networks*, 3(3), 2005.

[36] F. Wang and P. Liu. Temporal Management of RFID Data. In *VLDB '05: Proc. of the 31st Int. Conf. on Very Large Data Bases*, pages 1128 – 1139, 2005.