SPIRE: Scalable Processing of RFID Event Streams

Richard Cocci
Department of Computer Science
University of Massachusetts Amherst
rcocci@cs.umass.edu

Yanlei Diao
Department of Computer Science
University of Massachusetts Amherst
yanlei@cs.umass.edu

Prashant Shenoy
Department of Computer Science
University of Massachusetts Amherst
prashant@cs.umass.edu

Abstract—**Radio Frequency Identification (RFID) technology is gaining acceptance in an increasing number of applications for tracking and monitoring purposes. While RFID raises the potential to provide unprecedented visibility in various application domains, data management techniques that are capable of handling massive amounts of data generated by large RFID deployments are still lacking. The sheer volume of data generated in such deployments could easily overwhelm existing information systems. Moreover, the transformation from raw RFID readings into meaningful, actionable information in real-time poses another significant challenge. In this paper, we present the design of the SPIRE system that aims to manage enormous volumes of RFID data and provide fast data information transformation. In addition, we outline our research plan to refine this design and evaluate its performance in a simulated large-scale RFID supply chain scenario.**

I. Introduction

The past few years have witnessed the emergence of an important trend: physical objects are tagged individually and subsequently sensed in various locations at various times. *Radio Frequency Identification* technology lies in the very heart of this trend. RFID tagging and sensing in combination with ubiquitous networking will soon enable an information infrastructure that collects real-time data of physical objects and delivers high-value content to a wide spectrum of user communities. Examples of emerging user communities include supply chain management **[11]**, healthcare **[11]**, pharmaceuticals **[11]**, postal services **[14]**, and surveillance **[15]**, just to name a few. With driving forces such as the Food and Drug Administration (FDA)'s recommendation to use RFID to combat counterfeit drugs **[26]** and Wal-Mart's mandate to tag cases by its suppliers **[21]**, uses of an RFID-based information infrastructure will soon permeate many aspects of everyday life.

Our research focuses on developing an RFID-based information infrastructure that offers two key functions: *real-time monitoring of real-world activities* and *object track-and-trace on large scales*. Real-time monitoring is enabled by the infrastructure's ability to sift relevant information out of the flood of RFID data immediately after it emerges. In retail and inventory management, for example, such timely, relevant information is needed to detect shoplifting activities, out-of-stocks, misplaced inventory, *etc*. This infrastructure also allows information to be collected from various sources and integrated into a broader view to support object track-and-trace, *i.e.*, reporting the history of location and condition of objects. Take food and drug distribution, example track-and-trace queries include "has this medicine been exposed to temperature over its regulatory range during the distribution?" or "does this beef come from an area with a recent outbreak of the mad cow disease?" In the foreseeable future, this infrastructure will further facilitate the development of new everyday applications such as the "smart medicine cabinet" that monitors human access for medical compliance, detects interaction between medicines, measures temperature and humidity, infers expired and spoiled medicines from past and current condition, and alerts users when a type of medicine is recalled by its manufacturer.

While the database community has made significant progress in developing data management systems suited for related domains such as stream processing **[2][5][6][9][23][25]** and information integration **[10][13][17][20]**, an RFID-based information infrastructure presents two fundamental challenges which have not been sufficiently addressed:

Data-information mismatch for monitoring: Data streams emanating from RFID devices carry primitive information about the tag affixed to an object, its location, and the time of sensing, which is essentially a core dump of the sensing of a physical world. RFID-based monitoring applications, however, require meaningful, actionable information (*e.g.*, shoplifting, out-of-stocks) that is defined by unique complex logic involving filtering, pattern matching, aggregation, and recursive pattern matching. Such logic has not been a focus of existing stream systems **[2][5][6][9][23][25]** and sensor networks **[7][18][19][29]**, and remains under-addressed in recent event systems **[1][3][8][15][25]**. To resolve the mismatch between data and information, it is critical to have a processing component residing on RFID streams that performs complex data-information transformation.

Incomplete, insufficient data for track-and-trace: To support track-and-trace, an RFID-based information infrastructure needs to integrate data from various types of devices, such as RFID readers and wireless sensors, and numerous distributed data sources. Unlike traditional data warehousing **[4][12]** and information integration **[10][13][17][20]**, this process is significantly complicated by two unique problems of RFID technology: a) missing data is common, and b) readers often overlap in read ranges resulting in duplicated readings of the same tag at different locations. These problems result in a tremendous need for filtering and smoothing techniques to improve the quality of data that is initially incomplete.

Besides the above challenges, the information infrastructure also faces the following performance requirements:

Scalability: Large deployments of RFID devices will create unprecedented volumes of data. For example, when Wal-Mart tags goods at the individual item level, it could create as much as 7 million terabytes of data in a single day **[24]**. This sheer volume of data is an obstacle in itself and all aspects of RFID data management must be designed to account for this challenge.

Low-latency: Despite the volume of data, RFID data processing needs to be fast. This is crucial for monitoring applications that require up-to-the-second information to prevent loss in value and mitigate harm to life, property, and the environment. Data integration, inference, and track-and-trace querying are also often required to be performed with low-latencies.

In this paper we present the architecture and technical overview of SPIRE, a distributed system designed to address the challenges discussed above through the following methods:

Data Cleaning: To improve data quality and reader reliability we introduce a layer directly above the readers, the purpose of which is to filter out abnormal and corrupted readings, remove duplicate readings, and smooth readings to assist in recreating missing data from imperfect readers.

Data Compression: To reduce overall data volume originating from the readers, we introduce compression techniques designed to reduce the number of tag readings recorded, while at the same time supporting precise location tracking and event detection.

Event Processing: To extract meaningful information from the raw RFID tag data, we employ an event processor operating over a stream of tag readings to search for user specified trends. For instance, a user could define a query to monitor the incoming compressed tag readings to detect patterns that indicate a specific tag may have disappeared from a warehouse. If sufficient events are observed to satisfy the query, the event processor reports a warning indicating the object attached to that specific tag might have been misplaced or stolen.

Another key feature of SPIRE is our effort to resolve an inherent tension between compression and event processing for anomaly detection: the former aims to remove large amounts of raw data for scalable processing, while the latter requires sufficiently detailed data to provide timely, precise information. One focus of our research is to devise architectural and algorithmic solutions to resolve this tension.

Distributed Event Processing: Scalability is achieved through the combination of our compression techniques with a distributed architecture that allows for each installation of the system at a different location to operate independently. Compression and initial event processing are carried out at the local level, and then integrated on a global scale where further event processing can occur to monitor data at the enterprise level. By compressing the local tag data and transforming it into a series of events, the data volume is reduced to a scale which can become manageable for a large central repository.

Section 2 of this paper details the overall system architecture of SPIRE, while section 3 provides a more detailed description of the various system components. Section 4 describes the design of a simulator we have created to generate sample RFID data from a supply chain scenario in order to test the features of SPIRE. Since this is a sample application which we have chosen as an initial trial for SPIRE, some of the system specifics have been developed with a supply chain scenario in mind. Our techniques, however, are general enough to be applicable to a wide spectrum of monitoring and tracking applications.

II. Architecture

Our architecture described here is an extension to that used for a demo application in **[13]**. The architecture of a local SPIRE system is shown in Figure 1 and consists of three distinct layers.

The bottom layer contains physical RFID devices (*e.g.*, tags, readers). The RFID data returned from the readers is passed to the middle layer, which contains multiple sub-layers for data cleaning, compression, and event generation. The output from the middle layer is a stream of events, which is then fed to the third layer where event processing takes place. A key component of the third layer is a complex event processor that monitors the event stream to deliver timely notifications to the user and archive events into the event database. SPIRE allows the user to query the resulting event database by either sending ad-hoc queries or writing continuous queries that combine stream processing and database access. These components are described in more detail below. This initial discussion focuses on a single localized instance of SPIRE, such as a single warehouse in terms of a supply chain. Section D elaborates further on extending this architecture to a global distributed system capable of enterprise scale querying and data management.

Physical Device Layer: The physical device layer consists of RFID readers, antennas, and tags. RFID readers scan their surrounding areas in regular intervals and return a reading for each tag detected in the form of (Tag ID, Reader ID, Timestamp).

Cleaning, Compression, and Association Layer: The middle layer serves three important functions. First, it copes with idiosyncrasies of readers and performs data cleaning, such as filtering and smoothing. This is important

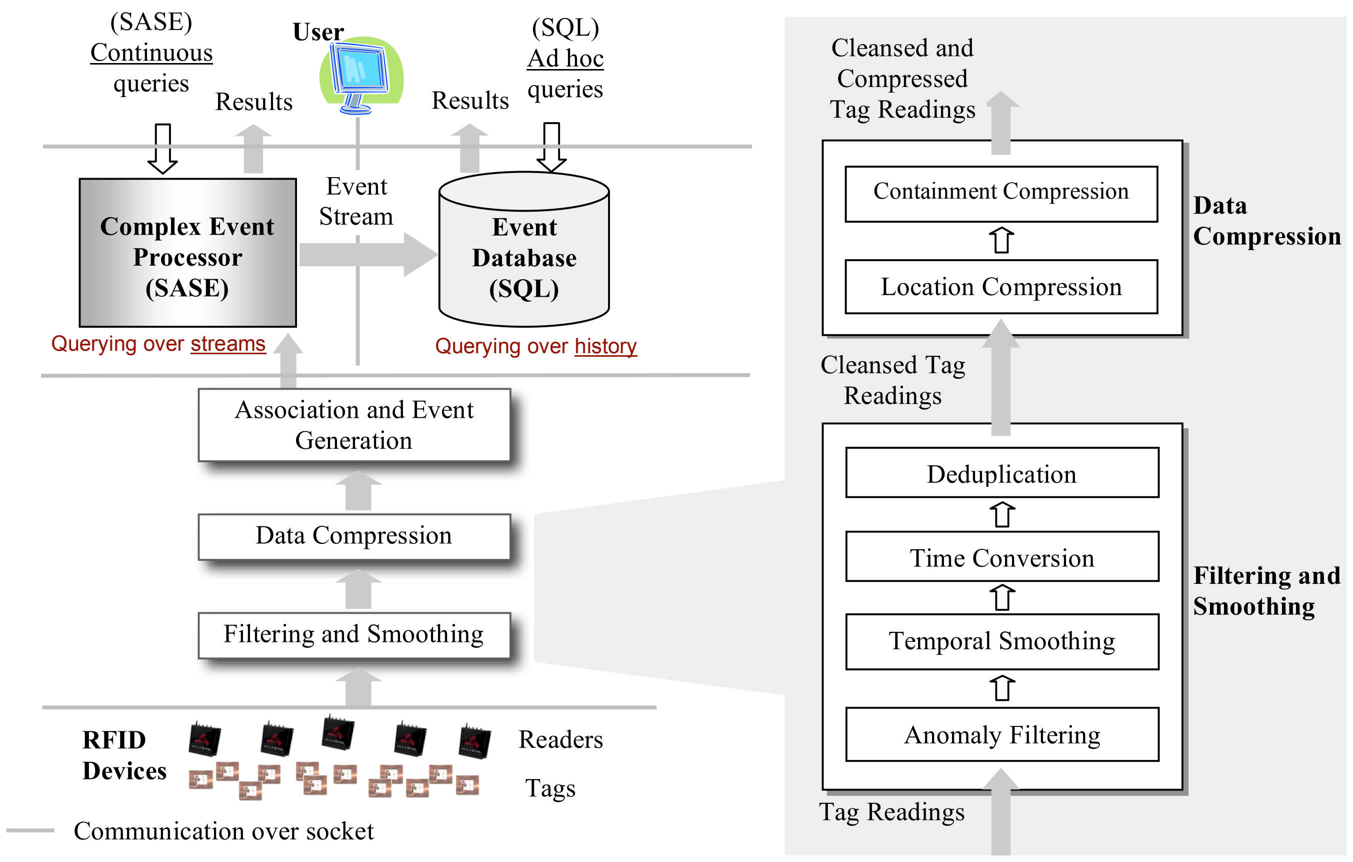

Figure 1: SPIRE Architecture at the Local Level

as RFID readings are known to be inaccurate and lossy. Our data cleaning component leverages state-of-the-art solutions for data cleaning described in **[16]**, **[27],** and **[28]**. Second, it uses two compression techniques to effectively reduce data volume between the readers and the event processor, which otherwise could be quickly overwhelmed by the sheer volume of data originating from the readers. Third, it uses attributes such as product name, expiration date, and saleable state to create events. These additional attributes are necessary for supporting various classes of tracking and monitoring queries which may be run over the events. Technical details of these three sub-layers are elaborated further in section III.

Complex Event Processor: The complex event processor supports continuous long-running queries over an input data stream. In our system we are utilizing the SASE **[28]** stream processing engine and query language to express and process these queries. Our complex event processor performs three important functions:

- For each monitoring task, such as detection of missing items, the user writes a query and registers it as a continuous query with the complex event processor. The event processor immediately starts executing the query over the RFID stream and returns a result (*e.g.*, a notification) to the user every time the query is satisfied. Such processing continues until the query is deleted by the user.
- Transformation rules for data archiving are also registered as continuous queries with the event processor. These queries can be used to remove duplicate data and transform data to the format required for archival. The resulting events are streamed to the event database for storage.
- The event processor can further handle complex continuous queries that integrate stream processing and database lookup; upon detection of an event of interest, these queries require database access to retrieve additional information. The event processor supports these queries by first detecting the event, then sending a subquery to the database, combining information retrieved from the database with that obtained from the stream, and finally returning a complete result to the user.

Sections 3.3 and 4 elaborate further upon how we plan to utilize the complex event processor to automatically detect anomalous supply chain conditions, such as missing and copied tags.

Event Database: SPIRE contains a persistent storage component to both support querying over historical data and allow query results from the stream processor to be combined with this historical data. As mentioned in the previous section, RFID stream data is transformed using rules declared with the complex event processor for archiving. SPIRE supports rules that create location and containment records in the event database. More details about these rules and events can be found under the compression and event generation portions of section 3.

III. Technical Components

After presenting an overview of the SPIRE system architecture, we now focus on several key technical components described below.

A. Filtering and Smoothing

Given that RFID readers are inherently imperfect at recording all tags present, SPIRE implements cleaning techniques to improve the quality of the raw data. This Filtering and Smoothing sub-layer accomplishes two goals, filtering out invalid tag readings and smoothing tag readings to fill missing gaps in the data. Internally, this sub-layer consists of four components:

1) **Anomaly Filtering:** Removal of readings which are spurious or contain truncated tag IDs.
2) **Temporal Smoothing:** The system decides whether an object was present at time *t* based not only on the reading at time *t*, but also on the readings of this object in a window size of *w* before *t*. Using this heuristic a new reading may be created, helping to fill in gaps where a tag was missed by a reader. Readings created from previous smoothings are not considered when determining whether or not to create a new smoothed reading at a given time *t*.
3) **Time Conversion:** A timestamp is appended to each reading based on a logical time unit that is set as a system configuration parameter, ensuring that all readings are recorded using the same unit of time measurement.
4) **Deduplication:** Removal of duplicates, which can be caused either by a redundant setup, where two readers monitor the same logical area, or when an item resides in overlapping read ranges of two separate readers.

While cleaning provides a more reliable data stream, it does introduce potential implications towards data accuracy. A short window size has a propensity towards false negatives, a missed reading for a tag which was actually present, as it becomes more likely for a tag to miss being read for the entirety of the window. Likewise, a long window size has a propensity towards creating false positives, a created reading for a tag which was not actually present, as a single reading during a long window could result in many added readings for a tag which has already moved from the reader. The interested reader may refer to **[16]** for an in-depth discussion on the effects of window size on RFID tag smoothing. Any configuration for SPIRE should balance the effects of the window size against application requirements to determine what window setting is best suited for the intended use.

B. Data Compression

After the raw tag readings have been cleansed they are passed to the Compression sub-layer. This sub-layer examines the readings and attempts to reduce the volume of data as much as possible, while minimizing inaccuracies, before passing the tag data onto the event generation phase. The Compression sub-layer utilizes two varieties of compression, location compression and containment compression.

1) Location Compression

When a tag is first observed at a new reader, an event is generated to indicate the tag's arrival. If the tag sits at the same physical location for an extended period of time, it will be repeatedly read by the same reader for the extent of its stay. Each of these readings generates no new information, other than to confirm that the tag has not moved from its most recent location. As such, it is possible to condense this entire series of readings into a single reading with a time range indicated by a start and end timestamp. To avoid unnecessary access to the event database, a cache is used to record RFID tags currently located at each reader according to most recent observations. When new readings are performed, this cache is used to identify tags which might no longer be located at each reader. If a tag is missed at a reader *x* times, an event is generated to indicate that the tag has left the reader. Here *x* is a configurable system parameter that represents the threshold for the number of times a tag may be missed at a reader before a location timeout occurs.

This method of compression is particularly effective for a supply chain in situations such as shelved tags, where a given tag may remain at the same location for a long period of time. The challenge presented by this compression method is establishing a good value for the timeout threshold. If the threshold is set too low, then too many false movements will be reported and more events will be generated than necessary. If the threshold is set too high, the tag will be recorded as remaining at a location for potentially many readings after it has moved on. The value of the window size chosen for smoothing should also be taken into account when choosing the threshold. A large smoothing window will reduce the likelihood that a false movement will occur, but may also cause a tag to extend its location entry beyond the point at which the tag actually left the reader. An effective balance between window size and location timeout threshold will need to be established for any application, based on the application's specific requirements and tolerances.

2) Containment Compression

Our second compression method exploits the fact that tagged objects often move together in groupings. For example, in a supply chain a case of products may be packaged where both the case and every product inside have an RFID tag attached. If this containment relationship between the case and its products can be appropriately captured, it becomes possible to use just the case tag to represent the location for both the case and all of the products within. In order to create these containment relationships, one of two options can be used: 1) specific readers physically configured so that only one tag of a highest containment level will be present at a time (i.e. there can be multiple product tags present, but only one case tag when creating the containment relationship described above), 2) containment relationships will have to be manually entered as they are created. Obviously option 1 is more desirable than option 2, and SPIRE is being designed so that output from these identified readers can be utilized to automatically create containment relationships through generated events.

After a containment relationship has been established, the benefit provided is that location records no longer need be directly kept for all tagged objects contained within another tagged object. Instead, a location record for the containing tag is sufficient to indicate that both itself and all tags contained within it were present. In order to keep containment records current, when a new reading is performed the containment records for each observed tag are validated based on what other tags are currently located at the reader. For example, if tag *A* is currently recorded in the event database as containing tag *B*, when a reading is observed for tag *A* a check will be performed to ensure that tag *B* is also present. If enough consecutive readings are observed where *A* is observed without *B*, or vice versa, this containment relationship will be considered stale and an event will be generated to end the relationship. As with the location compression caching mechanisms will be used for checking the containment relationships to reduce the amount of direct queries handled by the event database. There is also a possibility that specific locations could be configured to automatically end containment relationships, such as at areas where pallets are unloaded in a supply chain warehouse, but the details of this need to be further explored.

There are some tradeoffs to consider when utilizing containment compression. First, the ability to create containment relationships depends on the capacity to configure a special reader setup such that only one tag of a highest containment level is read at a time. In more controllable situations such as a supply chain warehouse this may be a reasonable assumption. However, in some applications it may not be possible to guarantee that such a setup is possible, making it challenging to establish new containment relationships.

Second, it is not always possible to directly verify a containment relationship. For instance, if two cases with products are located on a shelf by the same reader, it is impossible to tell if a product moved from the first case to the second based solely on the reader output. All that is able to be confirmed is that the containment relationship has not definitely changed. The containment change would not be noticed until one of the cases left the shelf and the containment relationship became marked as stale.

Finally, similar to location compression, the number of consecutive readings before a relationship is marked as stale needs to be carefully chosen. If the value is too low, many false containment changes might be unnecessarily recorded. If the value is too high, a stale containment relationship may persist for too long and introduce inaccuracies into the data record.

C. *EVENT PROCESSING*

The event processing phase of SPIRE contains two stages. The first receives the cleansed and compressed tag readings and transforms them into an event stream. The second phase is the complex event processor, which monitors this event stream to search for events which satisfy user defined continuous queries. Provided below is a further description of each of these phases.

1) *EVENT GENERATION*

After compression of the tag readings has completed, events can be generated and sent to the complex event processor to record useful RFID tag observations that have been made. For example, when a tag first arrives at a reader, an event will be created such as StartLocation(Tag A, Location B, Timestamp). Likewise, when this tag leaves this reader an EndLocation event will be created to mark this change in location as well. Similar events can also be used to start and end containment relationships. These events are monitored by the complex event processor to attempt matches with the user defined queries. Furthermore, they are also stored directly in the event database, after being transformed into the desired schema, to create a historical record for each tag of the movements and containment changes that it has experienced. Directly querying these events for a specific tag ID allows for track-and-trace information regarding the entire history of that ID.

An important step in event generation is to obtain additional attributes beyond tag ID and location defined in the schema. Potentially, attributes (e.g., product name, expiration date, etc.) can be retrieved from a tag's user-memory bank, from a service such as the EPC Object Name Service (ONS) **[22]**, or through the use of a localized repository that holds a copy of the information. As we are not yet sure which option we will explore for retrieving these attributes, this area remains an open issue.

2) *COMPLEX EVENT PROCESSING*

As described above in section 2, the complex event processor is used to allow a user to specify custom continuous queries over both the incoming event stream and historical data. Queries specified in the SASE language adhere to the following syntax:

```
[FROM      <stream name>]
EVENT      <event pattern>
[WHERE     <qualification>]
[WITHIN    <window>]
[RETURN    <return event pattern>]
```

The semantics of the language are briefly described as follows: The FROM clause provides the name of an input stream. If it is omitted, the query refers to a default system input. The EVENT, WHERE and WITHIN clauses form the event matching block. The event clause specifies an event pattern to be matched against the input stream. The WHERE clause, if present, imposes value-based constraints on the events addressed by the pattern. The WITHIN clause further specifies a sliding window over the event pattern. The event matching block transforms a stream of input events to a stream of new composite events.

Finally, the RETURN clause transforms the stream of composite events into the desired format for final output. It can select a subset of attributes and compute aggregate values, similar to the SQL SELECT clause. It can also specify the output stream and the type of events in the output. Additionally, the RETURN clause is capable of invoking further database operations for retrieval and update functionality.

One specific use that we intend for the complex event processor in our system is the automated detection of anomalies. As a supply chain example, consider the following query in the SASE language which could be used to identify objects that have potentially disappeared from a warehouse:

```
EVENT   SEQ(PACKAGING_READING x,
        !(EXIT_READING y))
WHERE   x.TagId = y.TagId
WITHIN  12 hours
RETURN  x.TagId, x.ProductName, x.TimeStamp
```

This query would monitor for any tag which entered the packaging stage in a warehouse, where shipments are grouped together, and created a location reading at the packaging reader, but did not create a location reading at the exit door within the next 12 hours. Thus, this tag disappeared from the packaging area and did not leave the warehouse through a standard exit point, indicating that the object attached to this tag may potentially have been misplaced or stolen. The RETURN clause specifies the required information for a notification regarding this event. For further information on the SASE query language and the complex event processor itself, or a detailed demonstration of SASE being used in a retail store scenario, refer to **[13]** and **[28]**.

The above query provides only one example of an automated anomaly which a user might be interested in monitoring. Other supply chain anomalies that could be monitored include the appearance of unknown tags, the appearance of duplicated tag IDs, tagged objects being incorrectly located near each other (e.g., items containing nuts being located next to an item without a nut allergy warning), or any of a multitude of other possibilities.

A potential issue which requires further investigation is the tension between our compression techniques and the ability of our complex event processor to detect anomalies. For instance, if a continuous query was being run to monitor a supply chain for duplicated tag IDs, the complexity would be greatly increased due to containment compression. Since location update events are no longer being created for all tags, the performance of the complex event processor could be compromised. If the original version of the duplicated tag was currently stored inside another container, in order to detect this cloned tag a search would have to performed on not only the location records but on the containment records as well. The appropriate containment record would then need to be joined with additional containment and location records to finally obtain the location of the contained tag. This tension between our compression techniques and anomaly detection is a salient challenge to address. We plan to explore additional algorithms and solutions to strike a balance between the compression and performance so that significant amounts of data can be compressed while allowing efficient detection of anomalies.

D. Global Processing

Up to this point the architecture and processing of SPIRE has been described solely in the terms of a single localized system. However, an additional extension upon this infrastructure would be to combine various disparate locations in a distributed system, allowing for location tracking and automated query processing at an enterprise level. Through the use of a web service, a centralized query system could provide for easy accessibility to real time and historical data across an entire distributed network of locations. At the enterprise level would be an additional centralized event processor to consolidate events from various local data stores and run continuous queries at the enterprise level over these events. An example of such a query would be to detect a copied EPC tag which was present simultaneously in more than one warehouse. The central repository would also serve as a service to provide global track-and-trace data that could show the history of a tag moving through different localized data stores, such as an item moving through multiple warehouses in a supply chain.

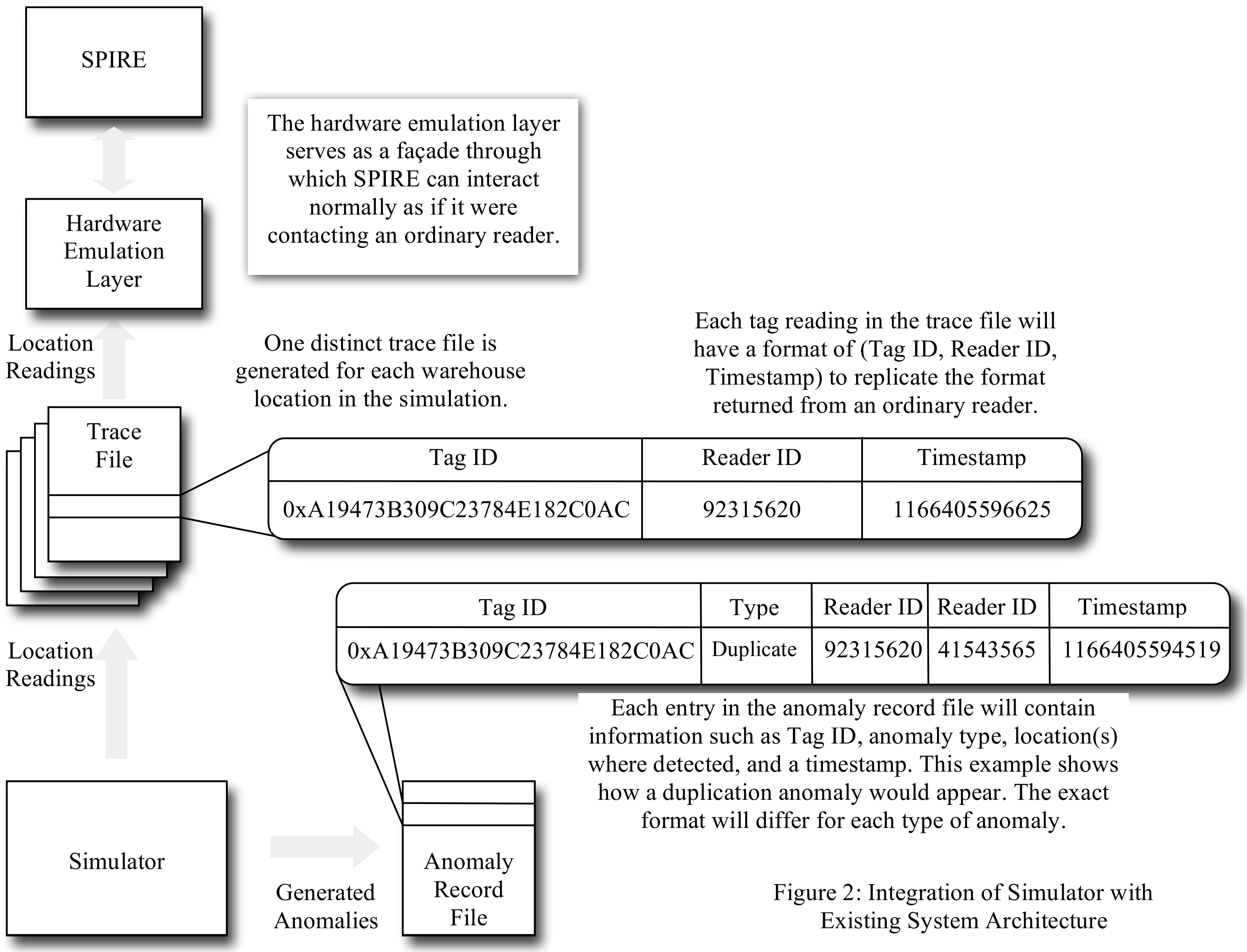
SPIRE
The hardware emulation layer serves as a façade through which SPIRE can interact normally as if it were contacting an ordinary reader.
Hardware Emulation Layer
Location Readings
One distinct trace file is generated for each warehouse location in the simulation.
Each tag reading in the trace file will have a format of (Tag ID, Reader ID, Timestamp) to replicate the format returned from an ordinary reader.
Trace File
Tag ID
Reader ID
Timestamp
0xA19473B309C23784E182C0AC
92315620
1166405596625
Tag ID
Type
Reader ID
Reader ID
Timestamp
0xA19473B309C23784E182C0AC
Duplicate
92315620
41543565
1166405594519
Location Readings
Each entry in the anomaly record file will contain information such as Tag ID, anomaly type, location(s) where detected, and a timestamp. This example shows how a duplication anomaly would appear. The exact format will differ for each type of anomaly.
Simulator
Generated Anomalies
Anomaly Record File

Figure 2: Integration of Simulator with Existing System Architecture

In order to perform these continuous queries at the global level it will be necessary to replicate the event information from the local event databases to the global event database in a timely fashion. This is an area of ongoing research, but our initial findings indicate that the compression techniques may be effective enough at reducing data volume at the local level to make feasible direct replication of the events from the local event databases to a global event database. These results are preliminary, but encouraging towards the general scalability of SPIRE.

IV. RFID SUPPLY CHAIN SIMULATOR

To assist in testing the capabilities of SPIRE, we have created a simulator capable of generating artificial RFID tag data from an entire retail supply chain. The following lists some of the assumptions made in our simulation:

- Three types of RFID tagged objects are used: pallets, cases, and products.
- Pallets contain cases which contain products.
- When a pallet reaches a warehouse the cases are removed from the pallet and placed onto shelves, where they will wait for some user specified amount of time.
- When the cases are ready to be removed from the shelves, they are sent to a packaging area and grouped into new pallets.
- These new pallets are recorded and then sent off on a simulated truck to another warehouse.
- The products are not removed from the cases at this time, though enabling this feature would require only a simple change to the simulator.

The user is able to specify locations in the supply chain where RFID tagged objects originate, where they are drained from the system, and the links between all of the warehouses in the system. Each link between two warehouse locations is given a specific distance and probability that it will be used. The simulation will run for an allotted amount of time, placing new objects into the supply chain at the designated creation locations and routing the existing items through the warehouses until they reach locations that are designated to drain objects from the supply chain. All readers in the supply chain have a customizable read rate and record to a trace file a raw tag reading containing (Tag ID, Reader ID, Timestamp) for every tag located at the reader whenever a reading is performed.

In order to test the complex event processing capabilities of SPIRE, the simulator allows for the user to add anomalies into the supply chain at either the warehouse level or chain wide level. Some currently supported anomaly types are unexpected removal of tags, insertion of unknown tags, and duplication of existing tags. We will add support for more anomaly types in the future as needed. For each instance of an anomaly created, the user is able to specify the amount of time between occurrences of the anomaly, the probability that the anomaly occurs after this amount of time, the warehouse location where the anomaly occurs (if applicable), and the type of tag that the anomaly will occur with (either pallet, case or product). The anomalous readings will be included in the raw trace files with all other readings and an additional anomaly record file is also produced to maintain a listing of all anomalies which occurred.

As a motivation for future research we intend to take trace files generated from this simulator to test the cleansing, compression, and event processing capabilities of SPIRE. Figure 2 provides a depiction of how the simulator fits into our existing system architecture. Below is a summary of how we plan to use the simulator data to test SPIRE:

- The cleaning functionality can be tested by introducing randomized imperfections into the simulated readers and examining how effective our cleaning techniques are at compensating for the faulty data.
- The compression techniques can be assessed by examining the volume of events sent to the complex event processor from the trace, as well as the accuracy with which these events represent the movements and containments of the simulated tags.
- The complex event processor can be assessed by creating continuous queries designed to detect the anomalies generated by the simulator and comparing the anomalies detected against the anomaly record log file.

V. Status and Future Work

In summary, SPIRE is designed to address the key challenges that arise in large scale RFID based information systems. Our techniques for data cleaning, compression, and event processing collectively deliver an RFID data event stream that is more reliable, manageable, and informative than raw readings produced from numerous RFID readers. While much work still lies ahead to achieve full implementation of our system, upon completion we anticipate that SPIRE will provide a scalable solution to both track-and-trace and user defined continuous queries.

Currently, we are in the process of implementing the compression sub-layer and finalizing the schema for entries in the event database. When finished, we will integrate these features with existing work that has already been done on the cleaning functionality and the SASE event processor **[28]**. At this point we will be able to test a single warehouse implementation of our system by incorporating reader data generated by the simulator. With the simulator trace files as described above in section 4, we will test the performance, accuracy, and anomaly detection capabilities of SPIRE using available cluster computing resources in our department.

As we progress with our implementation we will reassess our cleaning and compression techniques, altering them as needed if it is discovered that they interfere with the ability of the complex event processor to automatically detect anomalies from the incoming event stream.

Our eventual goal is to extend the single warehouse design to incorporate multiple local warehouses and one or more centralized global locations, similar to what one might find in a distributed supply chain network. At this point we will seek to test SPIRE's scalability by utilizing our simulated supply chain to perform track-and-trace queries and automated detection of anomalies on a global scale.

References

[1] Aguilera, M.K., Strom, R.E., Sturman, D.C., Astley, M., and Chandra, T.D. Matching events in a content-based subscription system. In *Proc. of Principles of Distributed Computing*, 1999.

[2] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., et al. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[3] Chandy, K.M., Aydemir, B.E., Karpilovsky, E.M., et al. Event webs for crisis management. In *Proc. of the 2nd IASTED Int'l Conf. on Communications, Internet and Information Technology*, 2003.

[4] Chaudhuri, S., and Dayal, U. An overview of data warehousing and OLAP technology. SIGMOD Record, 26(1), 65-74, March 1997.

[5] Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., et al. Scalable distributed stream processing. In *CIDR*, 2003.

[6] Cranor, C. D., Johnson, T., Spatscheck, O., and Shkapenyuk, V. Gigascope: A Stream Database for Network Applications. In *SIGMOD*, 647-651, 2003.

[7] Deshpande, A., Guestrin, C., Madden, S., and Hellerstein, J.M, and Hong, W. Model-Driven Data Acquisition in Sensor Networks. In *VLDB*, 588-599, 2004.

[8] Fabret, F., Jacobsen, H.A., Llirbat, Pereira, J., Ross, K.A., and Shasha, D. Filtering algorithms and implementation for very fast publish/subscribe systems. In *SIGMOD*, 115-126, 2001.

[9] Franklin, M.J., Jeffery, S., Krishnamurthy, S., Reiss, F., Rizvi, S., Wu, E., Cooper, O., Edakkunni, A., and Hong, W. Design considerations for high fan-in systems: The HiFi approach. In *CIDR*, 2005.

[10] Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., and Widom, J. Integrating and Accessing Heterogeneous Information Sources in TSIMMIS. In *Proc. of the AAAI Symposium on Information Gathering*, 61-64, March 1995.

[11] Garfinkel, S. and Rosenberg, B. RFID: Applications, security, and privacy. Addison-Wesley, 2006.

[12] Gupta, A., and Mumick, I. Maintenance of materialized views: Problems, techniques, and applications. IEEE Data Engineering Bulletin, 18(2), 3-18, June 1995.

[13] Gyllstrom, D., Wu, E., Chae, H., Diao, Y., Stahlberg, P., and Anderson, G. SASE: Complex Event Processing over Streams. In *CIDR*, 2007.

[14] Harrop, P., and Holland, G. RFID for postal and courier services. November 2005. http://www.idtechex.com/pdfs/en/R2646Q5829.pdf

[15] Hinze, A. Efficient filtering of composite events. In *Proc. of the British National Database Conference*, 207-225, 2003.

[16] Jeffrey, S., Garofalakis, M., and Franklin, M. Adaptive Cleaning for RFID Data Streams. In *VLDB*, 2006.

[17] Levy, A.Y., Rajaraman, A., Ordille, J. Querying heterogeneous information sources using source descriptions. In *VLDB*, 251-262, 1996.

[18] Madden, S., Franklin, M. J., and Hellerstein, J.M, and Hong, W. TAG: A tiny aggregation service for ad-hoc sensor networks. In *OSDI*, 2002.

[19] Madden, S., Franklin, M. J., and Hellerstein, J.M, and Hong, W. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, 491-502, 2003.

[20] Manolescu, I., Florescu, D., and Kossmann, D. Answering XML queries on heterogeneous data sources. In *VLDB*, 241-250, 2001.

[21] MarketWatch. Wal-Mart sees more suppliers adopting RFID. http://www.rfidgazette.org/walmart/index.html.

[22] MIT Auto-ID Lab. EPC network architecture. January 2006. http://autoid.mit.edu/CS/files/3/networkarchitecture.

[23] Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., et al. Query processing, approximation, and resource management in a data stream management system. In *CIDR*, 2003.

[24] RF Code. The data capture infrastructure platform for the new age of Auto-ID. http://www.rfcode.com/white_papers.asp.

[25] Rizvi, S., Jeffery, S.R., Krishnamurthy, S., Franklin, M.J., Burkhart, N., et al. Events on the edge. In *SIGMOD*, 885-887, 2005.

[26] U.S. Food and Drug Administration. Combating Counterfeit Drugs. February 2004. http://www.fda.gov/oc/initiatives/counterfeit/report02_04.html

[27] Wang, F. and Liu, Peiya. Temporal management of RFID data. In *VLDB*, 1128-1139, 2005.

[28] Wu, E., Diao, Y., and Rizvi, S. High-performance complex event processing over streams. In *SIGMOD*, 407-418, 2006.

[29] Yao, Y., and Gehrke, J. Query Processing in Sensor Networks. In *CIDR*, 2003.