Real-time Event Handling in RFID Middleware System

Dissertation

submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

by

Kamlesh Laddhad

(Roll no. 05329014)

under the guidance of

Prof. Bernard Menezes

and

Prof. Krithi Ramamritham

Dept. of Computer Science and Engineering (KReSIT)

Indian Institute of Technology Bombay

2007

Dissertation Approval Sheet

This is to certify that the dissertation entitled

Real-time Event Handling in an RFID Middleware System

by

Kamlesh Laddhad

(Roll no. 05329014)

is approved for the degree of **Master of Technology**.

Prof. Bernard Menezezs (Guide)

Prof. Krithi Ramamritham (Co-guide)

Prof. S. Sudarshan (Internal Examiner)

Prof. Ramesh S. (External Examiner)

Prof.Narayanan (Chairperson)

Date: _____

Place: _____

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
CERTIFICATE OF COURSE WORK

This is to certify that **Mr. Kamlesh Laddhad** was admitted to the candidacy of the M.Tech. Degree and has successfully completed all the courses required for the M.Tech. Programme. The details of the course work done are given below.

Sr.No.	Course No.	Course Name	Credits
		Semester 1 (Jul – Nov 2005)	
1.	CS601	Algorithms & Complexity	6
2.	HS699	Communication and Presentation Skills (P/NP)	4
3.	IT608	Data warehousing and Data Mining	6
4.	IT603	Database Management Systems	6
5.	IT619	IT Foundation Laboratory	8
6.	IT623	Foundation course of IT - Part II	6
		Semester 2 (Jan – Apr 2006)	
7.	IT694	Seminar	4
8.	CS632	Advanced Database Management Systems	6
9.	CS606	Foundations of Parallel Computation	6
10.	CS686	Object Oriented Systems	6
11.	IT680	Systems Laboratory	6
		Semester 3 (Jul – Nov 2006)	
12.	CS705	Statistical Foundation of Machine Learning (Audit)	6
13.	HS847	Algorithms in Astronomy	6
		M.Tech. Project	
14.	IT696	M.Tech. Project Stage - I (Jul 2006)	18
15.	IT697	M.Tech. Project Stage - II (Jan 2007)	30
16.	IT698	M.Tech. Project Stage - III (Jul 2007)	42

I.I.T. Bombay Dy. Registrar(Academic)

Dated:

Abstract

Radio Frequency IDentification (RFID) tags have emerged as a key technology for real-time asset tracking. Wide application of RFID leads to huge amounts of data being generated from scan of each of these RFID tags on individual items, for example, the RFID system of a moderate size retail chain will generate 300 million RFID scans per day. Extracting meaningful information out of this huge amount of scan data is a challenging task. Researchers in database community have presented techniques and models for warehousing as well as cleaning/filtering RFID data. Moreover CIOs are looking for real-time business decision from this RFID scan data. We show how to add value to an RFID middleware system by enabling it to handle a large amount of RFID scan data and execute business rules in real-time. Experimentally we demonstrate that our proposed approach is efficient as compared to similar implementations with existing technologies. We also demonstrate how the proposed system can be deployed in a distributed environment.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction to RFID Technology

Radio Frequency Identification (RFID) is emerging as an all pervasive ubiquitous technology which holds the promise for increasing productivity in business processes that require automatic identification. RFID refers to the automatic identification and tracking of items by "tagging" them with identification chips. RFID components mainly involve tags and readers(sensors). An RFID tag is a small, low-cost device that can hold a limited amount of data and report that data when queried by a reader. Reader is a device that can recognize the presence of RFID tags and read the information stored in them. RFID uses radio-frequency waves to transfer data between readers and movable tagged objects, making it possible to create a physically linked world in which every object is numbered, identified, cataloged. These inexpensive tags can be associated with objects, such as pallets, cases, and even individual items. One of the major advantages of RFIDs over bar codes is that it is non-line-of-sight technology. In addition to that, RFID tags can be read even when they are hidden. By placing RFID tag readers at various locations, one can track the movement of objects. Such item-level tracking can greatly enhance the efficiency of business operations through supply chain networks, namely, from manufacturers to retailers, then to consumers.

1.2 RFID Applications

Radio Frequency Identification (RFID) is emerging as key technology for a wide-range of applications, including supply chain, retail store, and asset management. It promises

to make a big impact on data intensive processes in stock management, warehousing, logistics, healthcare, pharmaceuticals and security which require identification of objects. Some other recent applications of this technology include e-passports, toll collection, automobile tracking, etc.

The technology has gained significant momentum in the past few years with several high-profile adoptions like Wal-Mart, Albertson, Target, U.S. Department of Defense, British airport, etc. The U.S. Department of Defense uses RFID to manage shipments to armed forces worldwide. Several major retail chains, including Wal-Mart, Target, and Albertsons, have asked all their suppliers to tag products at the pallet and case level.

1.3 RFID Data Characteristics

Wide application of RFIDs leads to huge amounts of information being generated from the scan of these RFID tags on individual items. Managing such high volume of data poses challenges to applications as well as back-end databases. Extracting meaningful information out of this huge amount of scan data is a challenging task. Independent of the application, the information generated by RFID system posseses some fundamental characteristics.

1. *Temporal and dynamic*: Applications dynamically generate observation(readings). Object's location as well as containment relationship among objects change with time; as a result of which, RFID data carry state changes.

2. *Inaccuracy of data*: Sometimes non-existing tag may be incorrectly read (False positive reads) or reader may miss tags which were in its vicinity (False negatives). Reader may read a same tag more than once. Certainly such erroneous data has to be semantically filtered.

3. *Continuous Streaming*: Number of tags are proportional to number of items being serviced/tracked and number of readers are proportional to traceable strategic locations/areas. In typical scenario, tagged object stay in place for longer duration and readers record their existence on continuous basis in periodic intervals. Such simple observations keeps piling up in database and produce redundancy. Some

compression techniques need to be employed to remove the redundancy without loss of information.

4. *Granularity*: This factor depends on applications for which RFID system is being implemented. Granularity of data to be collected needs to be determined e.g. if the system is deployed at retail store, the granularity of data collection may be an item in the shelf or store. On the contrary, if system is deployed at airport, the granularity will be unit of luggage/baggage.

Summing it all, one can observe that we need efficient methods for data collection, cleaning and storage.

Challenges in Exploiting RFID Data

A feature that distinguishes an RFID infrastructure from a normal warehousing infrastructure is the greater emphasis on station-local activities in the former. In a typical data warehousing setup (e.g., for a department store), it is uncommon for the data to be queried at its source. The emphasis is on aggregating data at a central warehouse, where it can be indexed and queried using grouping and aggregation. In contrast, data generated by the tag readers at a dock door in a distribution center is more likely to be used within the distribution center than away from it.

Why traditional data cube model would fail: Suppose we view the cleansed RFID data as the *fact table* with dimensions. `(object_epc, location, time_in, time_out : measure)`[1]. The data cube will compute all possible group-bys on this fact table by aggregating records that share the same values at all possible combinations of dimensions. If we use count as measure, we can get for example the number of items that stayed at a given location for a given month. The problem with this form of aggregation is that it does not consider links between the records. For example, if we want to get the number of items of product P that traveled from the distribution center in location L to stores in location U, it is hard to get this information. We have the count of product P for each location but we do not know how many of those items went from the first location to the

[1]EPC is Electronic Product Code which uniquely identifies a tag.

second. We need a more powerful model capable of aggregating data while preserving its path-like structure.

1.4 Organization of report

This chapter sets the stage with introduction to RFID and presenting challenges in handling RFID data. In chapter 2, we motivate the purpose of research and formally present the problem statement. Chapter 3 develop the event based model for RFID system and explains how ECA model can be used in RFID systems. In Chapter 4, we describe the detailed architecture of our system and present algorithms for event identification. We also explain an implementation of RFID event handling system using relational database and compare if with our approach. We experimentally demonstrate the performance of our system. In chapter 5, we first present a literature survey for couple of techniques which address problems in distributed systems. Then we present possible integration of our approach in Sun's open source RFID middleware. We also show how a distributed system can be implemented using Java-RMI. Lastly in Chapter 6, we conclude our work and elaborate on the scope of future work.

Chapter 2

Motivation and Problem Statement

In this chapter, we present two case studies from different domains to motivate the topic. Through these case studies, we try to capture diverse applications where RFIDs can be readily used. First application is a passenger tracking system at airport and another is a case study on deploying RFIDs in retails supply chain. Later one is picked from Harvard Business Case Studies [1]. We explain how our research can help in both these scenarios.

2.1 Motivating Scenarios

Passenger tracking at the Airport: Consider a typical airport with lot of counters or checks. Each of these counters (including entry-exit gates) have RFID readers. Also assume that each ticket/passport is tagged[1] so that passenger can be uniquely identified. When passenger arrives at the airport, the scanner/reader at entrance reads the tag on his/her passport; passenger moves to check-in counter, tag is scanned again, the person moves towards the baggage counter; from there to security check, to waiting room, and finally boards on the aircaft; his/her tag is read all along. At airport, there is a rule which says every passenger boarding the plane has to go through security check but he/she may or may not go through the baggage counter. Also, there might be a rule that allow/disallow more than one visits to a particular counter or there might be rule restricting the order of movement (you cant pick your baggage and leave, you have to go through costume check) or there might be a rule restricting movement on the basis of time span, and so on. As of today, all these rules are verified manually and there is no system wherein we can define these rules and detect violations of these rules automatically.

[1]The term "tagged" from this point onwards means RFID tagged.

Application in retail supply chain: Consider a Supermarket; *Metro Group* (*MG*) [1]. Let us assume supply to *MG* follows following route: Manufacturers make pallets containing cases of same products and send them to *distribution centers* (DC). At DC, these pallets are reassembled into mixed pallets and then sent to different stores of *MG*. Pallets arrive at stores in accordance with the need specified by that store. To avoid manual errors, pallet-level and case-level RFID tagging is used. Each pallet leaving manufacturer's site is tagged. They are loaded on to the truck and the RFID reader counts the number of pallets being loaded. At the DC, pallets are reassembled into mixed pallets. This can be automated with the help of RFIDs. These pallets are now sent to the stores. Pallets arrive at the backroom of the store and are opened and verified for correct configuration using RFID readers. We are able to detect and eventually track the exact point in the supply chain where the object went missing. When the number of items at the shelf for a certain product falls below a threshold, it is refilled from the backroom. An indicator might inform the backroom to refill a certain item as its number falls below the threshold. Ordering for new stocks should happen if the number of cases of that product in the backroom is below a threshold. Also, at the shelf, sensors can monitor the conditions. Alarms can be raised if conditions worsen for a certain item.
These events can be treated as rules in retail supply chain. So *MG* needs middleware system to define, observe and monitor such business rules. Some more rules which can provide reactiveness to the system are

1. If the DC is out of stock for an item, a request for a new supply for the item should be sent to the manufacturer.

2. If the DC has already dispatched the item in the last consignment to the store, recognize this and alert the store manager.

3. If the temperature of a food item is being monitored by a sensor and it observes abnormal temperature variations such that the item placed in that shelf can't be kept in such situations, alert the concerned authority about this problem.

4. If a certain food item kept on a shelf is about to expire[2], identify them for extra discount and make visual announcements that the cost has been reduced.

[2]Assumption: RFID tags are designed so as to maintain item's expiry date.

Potential of RFID system will be realized only when these events can be detected and respective actions can be taken in the least possible time. Some of the key challenges in developing such a system are

1. **Number of events to be handled are huge**: Each scan generates an event. Following the example given in [2], the retailer with 3,000 stores sells 10,000 items a day per store. Assume that each item movement is recorded with a tuple of the form: `(EPC, location, time)`. If each item leaves at least 10 traces before leaving the store by going through different locations, this application will generate at least 300 million tuples per day. In such scenario identifying a particular event e.g., number of items on a shelf goes below a threshold is a challenging task. In general, extracting meaningful information from such huge data is a big challenge.

2. For real time business decisions, each event should be detected in the least possible time. Periodic batch processing of such huge number of data will not serve the purpose. The problem is more complex when the system is distributed, where both generation of events and detection of events happen in various systems which are themselves distributed geographically.

2.2 Problem Statement

The goal of this research is to design RFID specific ECA(event-condition-actions) [3] rule system. This design problem can be broken down into following sub-problems.

1. Development of an event based model for the RFID system.

2. Applying concepts of ECA (event-condition-actions) [3] and state based event management framework [4] in RFID systems.

3. Develop “Knowledge model”, “Execution Model”, and “Rule Management Model” with their dimension and set of possible values for each dimension.

4. Develop a storage scheme for raw RFID events and events of interest(EI) 3.2

5. Development of an efficient scalable approach to identify EIs from a huge RFID scan data.

6. Develop a standalone RFID rule management middleware system and compare it with systems built using relational databases Integrate the RFID system with middleware system from SUN Microsystems.

7. Develop a distributed real time RFID event management middleware system.

2.3 Related Work

Recently, a number of RFID middleware systems have attracted industry attention [5, 6, 7]. The event handling mechanism in these RFID systems is very rudimentary in nature, based on Java's event handling mechanism. In general they can handle very simple basic events e.g., raw RFID scans. They do not have any infrastructure to handle large number of complex events including events that combine RFID scans along with other environmental conditions such as temperature. In real life business scenarios, however, it is very natural to look for complex events. Developing an infrastructure to answer the questions mentioned in Section 2.1 will require elaborate system development effort in these systems and will not be efficient.

Research on active database systems and event-condition-action model seems relevant to large extent. In this, some of the important work to mention are [8, 3, 9]. Because the nature of events in active databases and RFID systems is different, we can't directly apply any of the existing research, however we have borrowed several concepts from these areas to apply in our system, e.g., we borrowed the idea of indexing events on the basis of parameter values as proposed in [10].

With respect to research related to real-world event handling, we borrowed the idea of representing complex RFID events with the help of state diagrams from [4].

In [11, 12, 13], authors proposed pre-processing of raw RFID scan data for cleaning such as identifying missing data and detecting outliers. We show how the data-cleaning can be integrated with our proposed system. In [14], authors proposed a security mechanism for RFID data, which is orthogonal to the research of this paper.

In essence, though we borrow some of the concepts from existing research work, so far there has not been an end-to-end solution proposed in either academics or in industry to handle large number of RFID events generated from RFID scans and hence this research.

Chapter 3

Events in RFID Systems

According to Wikipedia, an event is something that takes place at a particular place and time. For software systems, an event is something that needs to be monitored and may trigger a specific action. Specifying an event is therefore providing a description of the happening. Following [3], each RFID event can be described with some set of *dimensions* which includes *source* of event, event *granularity*, *location* of event, *time* at which the event occurred and a possible set of *operations* for combining events. An Event can be of *primitive* type or *composite* type. A *primitive event* occurs at a particular place and time. A *composite event* is a combination of a number of such primitive events linked by predefined operators (e.g., AND, OR, NOT etc.) [4]. In addition to this classification, in an RFID system, we define two types of events - **Basic Events** and **Events of Interest**.

3.1 Basic Events

A *basic event* (b_e) is an event generated by a source, e.g. an individual scan of RFID tag affixed to an object in the system. Following [4] and [2] a basic event(b_e) can be defined as a tuple (L, S, T) where L is the label dimension containing details of the event, S is the location dimension of the event occurrence and T is the time dimension at which the event occurs. As for example, going back to our example of *MG*'s supply chain system, consider the following situation: An object o is being loaded on a truck in a warehouse at location s at a certain point in time t. This object gets scanned by different readers. Scan of this object by a reader attached to truck at time $T = t$ at location $S = s$ is a basic event generating a tuple (l, s, t). The label l will contain details of the object o such as the scanned RFID tag number of manufacturer details.

In an RFID system a basic event (b_e) can be generated in four ways.

- **Object Scan**: A RFID reader scans a RFID tagged object and generates the basic event, which is termed as RAW RFID event [2]. The RFID scan generates the label L and the scanner id. The location S can be derived from the RFID database [15] that contains the information about the location of the scanner corresponding to the scanner id and the time when the scan happened. The time (T) will tell when the actual event occurred.

- **Clock**: A clock event is raised at some point in time independent of objects and other state of the system. The clock time can be absolute (e.g. 15th of August at 7:55 AM), relative (the next day after the match), or periodic (every day at 11:30 PM). The label of a clock event will just identify it as "CLOCK" event. The location (S) will remain empty in case of Clock event.

- **External**: An external event is raised by a happening outside the system, which includes environmental conditions (e.g., the temperature of the hall/room goes above 30 degrees Celsius). Typically such external events will be generated by various sensors deployed in a system e.g., temperature sensor. The label of the external event will contain sensor data e.g., temperature value in case of temperature sensor. The location will identify the location of the sensor which can be derived from a separate sensor database. The time will identify, when the sensor generated the external event.

- **Internal**: An internal event is related to internal state change of the system at time t. The internal event can be the effect of cascading action of some basic event which got fired by one of the previously defined three ways. These are mostly program generated events. The label(L) of an internal event will identify the details of the event such as "Number of Items $>$ 100" or "Average Price of item sold $>$ 40". The location (S) of an internal event will identify the generator i.e., the particular systems or applications that is generating this internal event. The time(T) will denote the time at which such an event is generated.

3.2 Events of Interest (EI)

Events of Interest (EI) are the events which need to be monitored. EI can be of two types (i) *primitive EI* and (ii) *composite EI*.

3.2.1 Primitive EI

A *primitive event*(p_e) can be defined as a tuple (L, S, T) where L is the label containing details of the event, S is the location of the event occurrence and T is the time at which the event occurs. However, unlike basic events, in case of primitive EIs each of this may or may not be pointedly specified.

The *label of a primitive EI* is an indication of range of products, items or objects for which this event has been defined. This may include a particular supplier, a particular product from a particular supplier, etc. Since the EPC is a hierarchical representation of entities in a supply-chain, we can use bits of EPC code to define label for primitive EI in case RFID tag contains the EPC code.

The *time of a primitive EI* is an indication of a range of time e.g. morning 8AM-10AM or today or month September. As a specific case the time of a primitive EI may be a specific time at the granularity at which it is defined in the basic event, e.g., 7.00 AM 29 Sept 2006.

The *location of a primitive EI* is a region that may contain one or more locations at the granularity of basic events, e.g., a basic event may occur at a scanner located at the distribution center in Miami, whereas an EI may be specified as a scan in a distribution center in South Florida region which contains the distribution centers both in Miami and Tampa.

For example, in the following p_e^1 is a primitive EI.

$$p_e^1 = (L = \{product\_type = \#54567\},$$
$$S = Shelf\#583, T =' Morning')$$

Here the primitive EI p_e^1 is looking for product with "product_type" as "#54567" defined in the label (L). The location (S) dimension of EI p_e^1 is defined by the shelf number "#583". The time (T) is morning.

3.2.2 Composite EI

A *composite EI*(o_e) is a combination of multiple primitive EIs (p_e) or multiple composite EIs linked by operators drawn from the following set [8].

- **AND ($\wedge$):** Conjunction of two events E_1 and E_2, denoted as $E_1 \wedge E_2$, occurs when both E_1 and E_2 occur (the order of occurrence of E_1 and E_2 is irrelevant).
- **OR ($\vee$):** Disjunction of two events E_1 and E_2, denoted as $E_1 \vee E_2$, occurs when either E_1 or E_2 occurs.
- **SEQ ($\Rightarrow$):** Sequence of two events E_1 and E_2, denoted by $E_1 \Rightarrow E_2$, is when E_2 occurs provided E_1 has already occurred. This implies that the time of occurrence of E_1 is guaranteed to be less than the time of occurrence of E_2.
- **NOT (!):** The NOT operator, denoted by $!(E_1, E_2, E_3)$, detects the non-occurrence of the event E_2 in the closed interval formed by E_1 and E_3. It is rather similar to the SEQ operator except that E_2 should not occur between E_1 and E_3 [8]

3.3 Event Condition and Actions (ECA)

Composite EIs are reactive. Some action is associated with such events and every time such events occur, the system identifies them and executes these actions. Following ECA model [3], a composite EI has three parts (i) event definition (ii) condition and (iii) action to be executed.

3.3.1 Events

The composition of EIs to create a composite EI is defined here, e.g., $o_e^1 = p_e^1 \wedge p_e^2$ is the event definition of composite EI o_e^1 which links two primitive EIs p_e^1 and p_e^2 by 'AND' operators. The derivation of L, S and T dimensions of composite EI o_e^1 from dimensions of EI p_e^1 and p_e^2 is done as suggested in [4]. Note that, the event definition of a *composite EI* can be expressed as a regular expression of multiple EIs. Thus the event definition of a composite EI can be represented as a state graph (DFA for regular expressions) as shown in [4]. We will use this *event state graph* of an EI in section 4.2.2.

Composite EI as a composition of composite EI: A composite EI may be composed of two more composite events, e.g.

$$o_e^3 = o_e^1 \wedge o_e^2$$
$$o_e^1 = p_e^1$$
$$o_e^2 = p_e^2 \wedge p_e^3$$

o_e^3 is a composite EI composed of two more composite EIs o_e^1 and o_e^2. The event definition of o_e^3 will be composition of o_e^1 and o_e^2. The condition of o_e^3 can be separately defined based on dimensions of o_e^1 and o_e^2, whereas the individual condition of o_e^1 and o_e^2 will be intact which will be evaluated before o_e^1 and o_e^2 triggers. Similarly the action of o_e^3 will be separately defined whereas the individual action of o_e^1 and o_e^2 will remain intact.

3.3.2 Condition

For a given composite EI, condition is a side-effect free boolean computation or set of boolean computations on dimensions (L, S and T) of *two or more* primitive EIs, which when evaluated as true may trigger an action associated with the EI. The condition is not a mandatory specification. The result of condition is presumed to be true if no condition is specified. Formally condition is a boolean combination of multiple conditional elements of the form $x \otimes y$ where,

$$\otimes \in \{>, <, \geq, \leq, =, \neq\}$$
$$x \rightarrow q_e^i.d.v$$
$$y \rightarrow q_e^j.d.v \tag{3.1}$$

where q_e^i and q_e^j are EIs (primitive or composite), d is one of the dimensions $label(L)$, $location(S)$ and $time(T)$ of EIs q_e^i and q_e^j, and v is some attribute value of this dimension, e.g. *company* is one attribute of dimension $label(L)$. Here v is an optional item. If v is not specified, the default value of the dimension is used, e.g. the default value for the dimension label, 'L', corresponding to an RFID event on a product will be the complete EPC code of the product. Note that, join conditions like $p_e^1.L.company = p_e^2.L.company$ is a valid condition for composite EI $o_e^1 = p_e^1 \wedge p_e^2$, whereas $p_e^1.L.company =$ "HP" is not a valid condition for composite EI o_e^1, because this is defined based on single primitive event and should have been represented in the primitive EI p_e^1 itself.

3.3.3 Action

An action is arbitrary sequence of predefined operations which are executed when the corresponding event gets fired on evaluations of associated conditions. Actions depend on the type of business where the system is being deployed.

3.3.4 Example of ECA in *MG*

Here we present some examples of how the business rules regarding *MG*'s supply chain system can be expressed as EI using ECA form.

Rule 1: When the number of items of product "54567" in the shelf "583" falls below a threshold (let us say 5), then the backroom needs to be alerted.

This EI (o_e^1) can be expressed in ECA form as follows.

Event:

$$o_e^1 = p_e^1 = (L = \{product\_type = \#54567\}, S = \#583, T = t_1)$$

Here the composite EI o_e^1 contains single primitive EI p_e^1. The primitive EI p_e^1 is looking for product with "product_type" as "#54567" defined in the label (L). The location (S) dimension of EI p_e^1 is defined by the shelf number "#583". The time (T) is t_1, where t_1 is anytime when the system will look for primitive EI p_e^1.

Condition:

$$p_e^1.L.count < 5$$

Here, we assume RFID reader has an attribute count. The value of count holds the number of items the reader would read for a given product-type.

Action: *Notify backroom*

Rule 2: The store manager wants the system to alert him when the temperature sensor on a shelf "124" finds the temperature to be unsuitable for a certain item "54567".

Event:

$$p_e^1 = (L = TEMP, S = \#124, T = t_1),$$

$$p_e^2 = (L = \{product\_type = \#54567\}, S = \#124, T = t_2)$$

$$o_e^1 = p_e^1 \wedge p_e^2$$

Here, p_e^1 and p_e^2 are primitive EIs with dimensions: (L, S, T), o_e^1 is the composite EI denoting the event corresponding to rule 2. In this example, time denotes the time at which the event occurred. Here we wait for only those events which satisfy the label and location as specified above. The EI p_e^1 has label temperature from an external source (temperature sensor) with location identifier indicating shelf #124, and the EI p_e^2 has label RFID tag with source as RFID scanner located at shelf #124. When both of these events occur the following condition is evaluated.

Condition:

$$(p_e^1.L.temperature > 70 \vee p_e^1.L.temperature < 50) \wedge$$
$$(p_e^1.T < p_e^2.T + timethreshold \wedge p_e^1.T > p_e^2.T - timethreshold)$$

In the condition we check if the temperature is within a certain range and the time difference between the two EIs does not exceed a threshold.

Action: *Notify backroom*

Chapter 4

Event Handler in Centralized System

4.1 Architecture

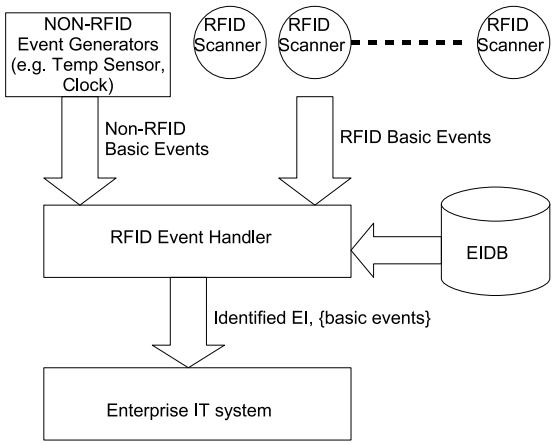

Figure 4.1: RFID Event Handler Architecture

Figure 4.1 describes the overall architecture of our proposed system. The key component in this architecture is the *RFID Event Handler* (EH) and the *Events of Interest database*(EIDB). Basic events are generated by various RFID sources (RFID reader) and non-RFID sources (Clock, External and Internal). These basic events are reported to EH. EIDB contains all EIs of the system. Based on incoming basic events, generated out of RAW RFID scans by RFID readers, EH identifies matching EIs in the EIDB. EH passes the identified EIs and the related basic events to the rest of the IT system. In an RFID system as the number of basic events generated from RFID readers is huge, the scalability of the EH is the biggest concern addressed by our design. In its simplest form,

EH is a centralized system. In complex systems, this EH may be a distributed system. We describe distributed RFID event handling mechanism in chapter 5.

4.2 EIDB and Event Handler

The EI database contains all EIs along with their respective details. The following section explains how composite events are stored in EIDB as state diagrams.

4.2.1 EI Database

In EIDB, EIs are first broken into primitive events of interest (primitive EIs). Primitive EIs are stored and maintained in EIDB in a multi-dimensional R-tree [16] structure. In each of the three dimensions (L, S and T), the tree is formed by the semantic hierarchy similar to R-tree. So we have three R-trees in the structure, e.g., if the label contains EPC code, the hierarchy in the R-tree of label is defined by the EPC hierarchy. In the label dimension, a primitive EI with label "Compaq" will lie above a primitive EI with label "Compaq-laptop". Similarly in the R-tree of location dimension, the hierarchy is defined by physical location based geographical hierarchy. A primitive EI with source (i.e. location of event) "Mumbai" will stay above a primitive EI with source "Powai" (a place in Mumbai). Such a hierarchy indicates, if a primitive EI p_e^1 is identified for "Powai", another primitive EI p_e^2 with source "Mumbai" and location & time same as that of p_e^1 should also be identified. In the R-tree of time(T) dimension, the hierarchy is defined by natural hierarchy of time, e.g. the EIs related to a particular time (e.g. 15th Oct 06, 8.00 PM) will reside below the EIs related to a particular day (e.g. 15th Oct 06). Also against each primitive EIs 'm' in the EI tree, we maintain a list ($\mathcal{Q}_m$) of composite EIs that are composed of the primitive EI 'm' (for simplicity this list is not shown in the Figure 4.2). Consider the following example of a set of primitive EIs.

- p_e^1 = { L=Compaq-laptop, S= Mumbai, , T= Jan, '06 }
- p_e^2 = { L= HP-Printer 3650, S=Powai, T=15th Oct, '06 }
- p_e^3 = {L=HP-Printer , S=San Francisco, T=15th Oct, '06, 8:00 PM }
- p_e^4 = {L=Compaq, S=IIT Bombay, T=*}

- p_e^5 = { L=*, S=IIT Bombay Convocation Hall, T=Every morning-Jan, '06}.

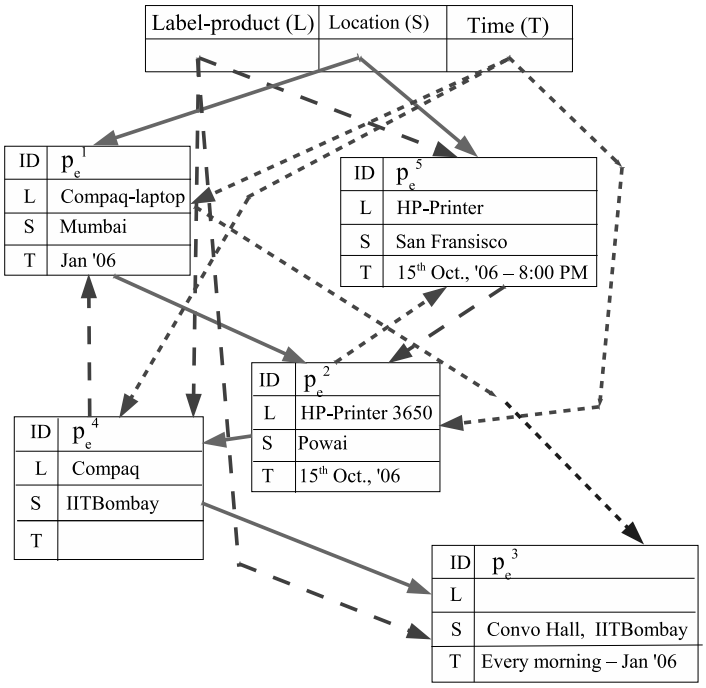

Figure 4.2: R-Tree Structure of Primitive EIs

These events are kept in the EIDB in a multi-dimensional R-tree as shown in Figure 4.2. Whenever a basic event (b_e) arrives at EH, EH looks into the EIDB and does three tree-traversals one on each of the event dimensions - label, source and time. The traversal on each of these dimensions results in three sets $\mathcal{S}_L$, $\mathcal{S}_S$ and $\mathcal{S}_T$ (line 5, 6, 7 of Algorithm 1). Where $\mathcal{S}_L$ is the set of all primitive EIs that match with b_e on the label (L) dimension, $\mathcal{S}_S$ is the set of all primitive EIs that match with b_e on the location (S) dimension, and $\mathcal{S}_T$ is the set of all primitive EIs that match with b_e on the time (t) dimension. Next we compute the set $\mathcal{P} = \mathcal{S}_L \cap \mathcal{S}_S \cap \mathcal{S}_T$, where each primitive EI $p_e \in \mathcal{P}$ matches with the basic event b_e in all three dimensions L, S and T (line 8 of Algorithm 1). Next $\mathcal{Q}_{p_e}$, the list of composite EIs, that are dependent on primitive EI $p_e \in \mathcal{P}$ is computed in line 10 of algorithm 1.

4.2.2 Handling Composite EIs

The semantic of composite EIs can be captured by a state transition diagram. The current state of a composite EI is maintained in the EIDB. The state graph of a composite EI is give by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is set of all the possible states for given composite EI and $\mathcal{E}$ is set of edges indicating occurrence of primitive EIs which leads to state transition. The final states of the state graph $\mathcal{G}$ will denote the occurrence of corresponding composite EI.

For example, a typical composite EI can be a combination of two or more primitive events such as $o_e^1 = p_e^1 \wedge p_e^2$. This means when p_e^1 and p_e^2 both occur, the occurrence of composite EI o_e^1 is identified. However, what will happen if more than one p_e^1 occurs before occurrence of a p_e^2 is not clear. Both the Figure 4.3 and Figure 4.4 represents the composite EI o_e^1. In Figure 4.3, once event p_e^1 has occurred all subsequent occurrence of p_e^1 will be ignored unless a p_e^2 occurs. Whereas in Figure 4.4, once p_e^1 has occurred, subsequent occurrence of event p_e^1 will lead to a state where consecutive two occurrences of event p_e^2 will result in the identification of composite EI o_e^1 twice. Thus the event state graph of a composite EI will also help us to clarify these details of the composite EI.

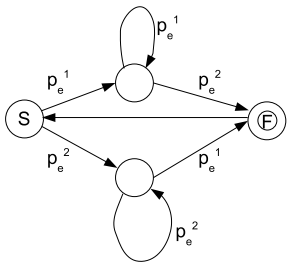

Figure 4.3: State Diagram for Composite EI o_e^1=$p_e^1 \wedge p_e^2$

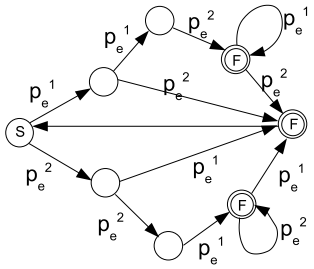

Figure 4.4: State Diagram for Composite EI o_e^1=$p_e^1 \wedge p_e^2$

4.2.3 Handling Conditions in RFID ECA

As explained in section 3.2, the condition part of an EI is described based on dimensions L, S and T of the EI. Conditions are typically join conditions on multiple primitive EIs. Such conditions can result in monitoring the same EI with different parameters. E.g., consider

an EI $o_e^1 = p_e^1 \wedge p_e^2$, with condition $p_e^1.L.company = p_e^2.L.company$, which identifies two primitive events on the product of same company. The composite EI o_e^1 will be identified if two events having the same company on the product label occur. Once primitive EI p_e^1 with $p_e^1.L.company = company_1$ occurs how to store it efficiently so that occurrence of p_e^2 with $p_e^2.L.company = company_1$ at some later time can be linked together very efficiently.

This is an example of multiple instances of the same composite EI (o_e^1) occurring simultaneously for different companies. So we need to maintain multiple instances of the state graph as well.

We maintain single state graph of a composite EI in an indexed structure similar to the one proposed in implementation of "scalable trigger" [10]. We keep each instance of the composite EI (o_e) along with following information - (i) $\mathcal{B}_{o_e}$ a list of all basic events related to a particular instance of the composite EI o_e, e.g. basic events related to a particular company (e.g., *L.company*), (ii) the current state in the event state graph for that instance of the composite EI and (iii) values of fields (e.g., *L.company*) in condition clause (e.g., $p_e^1.L.company = p_e^2.L.company$) in each state transition. The structure is indexed on the field values of condition clause e.g., in the example it will be indexed on *l.company*. A sample of this structure is shown in table 4.1.

Instances of EI	Basic Events	Current State	Field1	Field2	...
Instance 1 of o_e^1	b_e^i, b_e^j, ...	S_a	$p_e^i.L.company$="HP"	$p_e^j.S.city$ = "Mumbai"	...
Instance 2 of o_e^1	b_e^m, b_e^n, ...	S_b	$p_e^m.L.company$="Compaq"	$p_e^n.S.city$ = "Miami"	...

Table 4.1: Multiple Instances of Composite EI

4.3 Algorithm

Following the above description, in this section we present the complete algorithm of identification of EIs when a basic event occurs. The algorithm needs to run every time a basic event is reported to the RFID EH.

In lines 5, 6 and 7 of the Algorithm 1 we identify the primitive EIs that match the incoming basic event(b_e) on dimensions L, S and T respectively. In line 8, the algorithm

identifies primitive EIs that match the basic event b_e in all three dimensions L, S and T. Within the *FOR* loop at line 9, for each of the identified primitive EIs we find the related composite EIs (line 10). For each of the composite EI (line 11), in lines 12, 13, 14 and 15, we identify corresponding instances from the table as shown in Table 4.1 and modify the state to identify whether the final state for that particular instance has been reached. If final state has been reached the algorithm identify the occurrence of the composite EI and pass it to other system for further processing based on the action component of the composite EI.

Algorithm 1 Identification of Composite EIs for a Basic Event

1: **Input:**
2: EI database (EIDB)
3: Basic event b_e
4: **Algorithm:**
5: Identify and create a set $\mathcal{S}_L$ of all primitive EIs p_e^i from the R-tree of primitive EIs in EIDB such that the label of b_e matches with the label of p_e^i
6: Identify and create a set $\mathcal{S}_S$ of all primitive EIs p_e^i from the R-tree of primitive EIs in EIDB such that the location of b_e matches with the location of p_e^i
7: Identify and create a set $\mathcal{S}_T$ of all primitive EIs p_e^i from the R-tree of primitive EIs in EIDB such that the time of b_e matches with the time of p_e^i
8: $\mathcal{P} = \mathcal{S}_L \cap \mathcal{S}_S \cap \mathcal{S}_T$
9: **for** $p_e \in \mathcal{P}$ **do**
10: $\mathcal{Q}_{p_e}$ is the list of all composite EIs that are dependent on the primitive event p_e
11: **for** $o_e^i \in \mathcal{Q}_{p_e}$ **do**
12: Identify the fields for condition evaluation of EI o_e^i
13: Identify/Create (if necessary) instance of EI state graph $\mathcal{G}_{o_e^i}$ for EI o_e^i. Use index on fields identified in the last step
14: Based on the current state (start state if the graph is created newly) of the event state graph and the primitive event p_e change the state in $\mathcal{G}_{o_e^i}$
15: Add b_e in the list of basic events $\mathcal{B}_{o_e^i}$ of EI o_e^i
16: **if** Final state (F) has reached **then**
17: Pass basic event list $\mathcal{B}_{o_e^i}$ and identified EI o_e^i to other IT system for further action
18: **end if**
19: **end for**
20: **end for**

Complexity of Algorithm: Given a proper R-tree on label, source and time dimensions of primitive EIs, each of the lines 5, 6 and 7, can be computed in logarithmic time on the height of the tree. The *FOR* loop in line 9 is linear on the number of primitive EIs (M) identified by per basic event. Similarly *FOR* loop in line 11 is linear on the number of composite EIs (N) identified per primitive events. The line 12 takes again logarithmic time on the number of instances (P) of an EI state graph. Rest of the lines within the *FOR* loop of line 11 can be executed in constant time. Thus total time complexity of the algorithm is $MNlog(P)$. We do not expect M and N to have too large a value, typically 10-20. Thus our time our algorithm is very time efficient and can be run efficiently for each basic event. In section 4.5 we experimentally demonstrate the scalability and efficiency of our proposed approach.

4.4 A Solution using RDBM

We are not aware of any RFID system that can identify complex events from input stream of basic events. So to compare the efficacy of our approach we developed an alternate approach of detecting EIs from input of basic events. In this section we describe this "RDB" approach which primarily exploits and uses existing relational database technology. In Section 4.5 we compare the performance of our approach with this RDB approach. The details of the experiment can be found in section 4.5 where we compare the performance of our approach with this RDB approach.

In the RDB approach all EIs are stored in a relational database and whenever a new basic event occurs, this database is queried using standard SQL to get matching EIs. The database schema to store EIs is as follows.

Composite event table:

$CET = (o_e^{id}$, Composite EI expression, conditions, actions).

Primitive event table:

$PET = (p_e^{id}$, location, time, label)

Primitive-Composite event map:

$PCM = (o_e^{id}, p_e^{id})$

The CET table maintains definitions of all composite EIs. PET maintains all primitive

EIs that are part of these composite EIs and the table PCM maintains which primitive EI is being used by which composite EI. We also maintain three tables namely $Time$, $Label$ and $Space$, each of which keep the dimension related abstractions mapped to actual values of corresponding dimension i.e. each of them has schema as ($abstraction$,$actual\_value$) e.g. $Time$ table will tell us 'morning' is '6:00 AM to 11:59 AM', $Space$ table will tell us 'Powai' is part of 'Mumbai', and so on. Along with these, we maintain a data-structure, an adjacency list (AL) of active composite EIs (refer Figure 4.5) - all composite EIs that have been partially fulfilled based on occurred basic events. So AL maintains the detail of all composite EIs waiting for one or more primitive EIs to occur. AL will be empty when the system is initialized. The details of the RDB algorithm are presented in Algorithm 2. The algorithm runs everytime a basic event arrives at the RFID EH. In line 3, RDB

Algorithm 2 RDB Algorithm for Identification of Composite EIs for a Basic Event

1: **Input:** Primitive Event Table (PET), Composite Event Table(CET), Basic event(b_e), $Time$, $Label$ and $Location$ tables, List of active event(AL) partially filled.
2: **Algorithm:**
3: $\mathcal{P}$ =(Select p_e^{id} from PET where ($PET.location = b_e.location$ or $PET.location$ in (select $abstraction$ from $Location$ connect by prior $abstraction = actual$ start with $actual = b_e.location$)) and ($PET.label = b_e.label$ or $PET.label$ in (Select $abstraction$ from $Label$ connect by prior $abstraction = actual$ start with $actual = b_e.label$)) and ($PET.time = b_e.time$ or $PET.time$ in (Select $abstraction$ from $Time$ connect by prior $abstraction = actual$ start with $actual = b_e.time$))).
4: **for** $p_e \in \mathcal{P}$ **do**
5: $\mathcal{C}$ = Select o_e^{id} from PCM where $p_e^{id} = p_e$
6: **for** $o_e^i \in \mathcal{C}$ **do**
7: **if** o_e^i not present in AL **then**
8: create a new node for o_e^i in AL
9: **end if**
10: Add p_e^{id} to o_e^is list.
11: $PE_set^{o_e^i}$ = Select unique p_e^{id} from $PCMAP$ where $o_e^{id} = o_e^i$;
12: **if** isComplete(o_e^i, $PE_set^{o_e^i}$, AL) **then**
13: Evaluate condition for o_e^i and execute corresponding action.
14: Remove o_e^i from AL
15: **end if**
16: **end for**
17: **end for**

algorithm finds all PEs corresponding to incoming basic event b_e using a SQL statement.

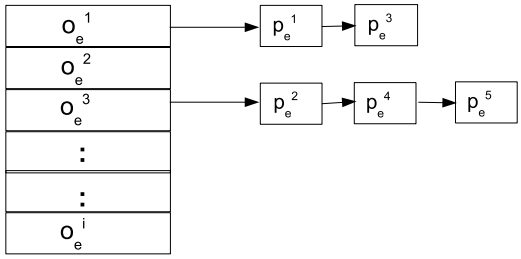

Figure 4.5: Example Adjacency List(AL) of Active EIs

The *FOR* loop on line 4 iterates through each of these PEs taken one at a time. Line 5 finds all EIs dependent on this PE, and for each of these EIs, the *FOR* loop on line 6 checks the effect of occurrence of b_e on EI and takes necessary actions if required. The AL structure is maintained accordingly.

Having described our approach for real-time RFID event handling and a RDB approach, in the next section we compare these two approaches to experimentally demonstrate the efficacy of our approach.

4.5 Comparison and Experimental Results

In this section we experimentally demonstrate the efficacy of our approach of event handling in an RFID middleware system. We study the experimental result from two dimensions (i) performance of our approach (denoted as "R" approach) compared to the RDB approach (ii) how the performance and scalability of our approach varies on various parameters.

We developed the RFID EH system using JDK 1.5. We ran the system on a Windows XP machine with pentium 2.8 GHz processor and 1 GB RAM. The EIDB database is a main-memory data-structure of multi-dimensional R-tree. The input basic events are simulated using a Java program communicating with the RFID EH using shared memory. The basic events are randomly generated by event simulator. The set of primitive EIs in EIDB are randomly generated based on a predefined hierarchy of label, location and time.

The composite EIs are randomly generated by combining multiple primitive EIs with

operators defined in section 3.2.2. In RDB system, the database tables as defined in Section 4.4 are created and populated with EIs in Oracle 10g Express Edition(XE) [17] on the Windows XP machine. The Oracle 10g XE was set to use memory to store tables and indices required for all experiments. This ensures that no disk access was required in Oracle for the RDB approach. To ensure that same set of events are generated by simulator in both RDB and our approach (denoted as "R" approach), same seed was used for generation of random numbers in both the approaches.

The parameters of our experimental analysis are chosen as shown in table 4.2.

Parameter	**Values**					**Base Value**
Number of Composite EIs	2000	4000	6000	8000	10000	10000
Rate of incoming basic events (number/sec)	50	70	90	110	130	50
Number of Primitive EIs per Composite EIs	4	8	12	16	20	4

Table 4.2: Experimental Parameters

During the experiment we measure (i) the time it takes to identify primitive EIs, we denote this time as "Time to identify Primitive EI" (TP) (ii) the time it takes to identify composite EIs, we denote this time as "Time to identify Composite EI" (TC). The total time required to process an input basic event will be the sum of TP and TC.

To compare the performance of our "R" approach with the "RDB" approach, we first initialize the EIDB with 2000 composite EIs and we programmatically simulate the basic event generation and send basic events to RFID EH at a fixed rate (50 basic events per seconds). We continue this for 5 minutes. We note the TP and TC for each basic event. We compute the average value of TP and TC during the run for 5 minutes. We repeat this process for each value of number of composite EIs in the EIDB as given in Table 4.2. In all cases, we keep the rate of incoming basic events constant at the base value 50 per seconds. This experiment done both for "R" and "RDB" approach.

In Figure 4.6 and Figure 4.7, we plot the TP and TC respectively as it varies with the number of composite EIs in EIDB. As can be seen, at any number of composite EIs our approach provides much lower values of TP and TC as compared to RDB approach. Search time for both approaches grows with number of composite EIs. Thus as number

of composite EIs increases, TP increases in both approaches. The improved performance of "R" approach can be explained due to mainly two reasons (i) R Tree - The R-Tree approach of finding primitive EIs is taking $\frac{1}{3}$rd time of that being taken by the SQL query in Oracle in-memory database (ii) State Diagram - Representing composite EIs as state diagram means maximum one operation per basic event to determine whether the composite EI has occurred or not, whereas in "RDB" approach this requires computing the boolean function based on matching primitive EI. As a result TC in "R" approach is $\frac{1}{20}$th of that of "RDB" approach.

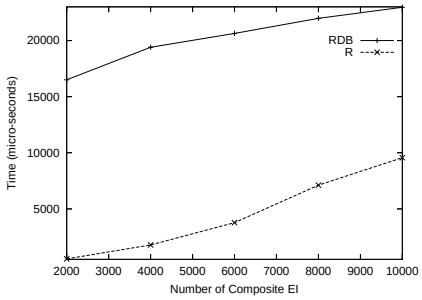

Figure 4.6: Variation of TP with # EIs

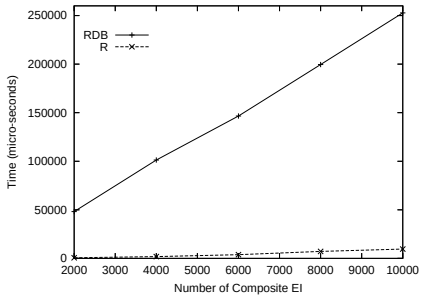

Figure 4.7: Variation of TC with # EIs

To demonstrate the scalability of our approach we kept the number of composite EIs in EIDB constant at its base value 10000. We varied the rate of incoming basic events in our system. For each value of rate of incoming basic events, we measured the total time (TP + TC) required to process each basic event and average it over 5 minutes. We plot the total processing time with the rate of incoming basic events in Figure 4.8. The y-axis denoting "Average Processing Time" (i.e., TP + TC) starts at 18900. As the incoming rate of basic events increases the average processing time for each basic event increases gradually, however one should note that the increase in processing time is very minimal, from 19000 μseconds at 50 basic events per seconds to 19500 μseconds at 130 basic events per seconds.

The complexity of composite EI also affects the scalability of our system. One way to measure the complexity of a composite EI is the number of primitive EIs it depends on as described in Section 3.2.2. We kept the total number of composite EIs in EIDB constant at the base value of 10000, the rate of incoming basic events to 50 per seconds and vary the average number of primitive EIs per composite EI as described in Table 4.2.

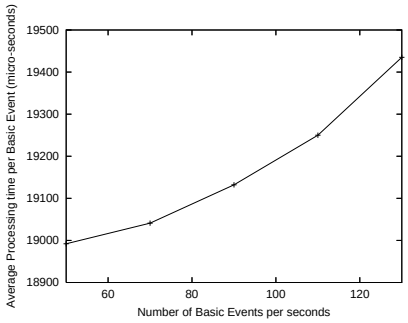

Figure 4.8: Total Performance

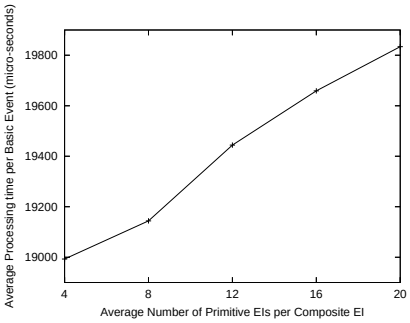

Figure 4.9: Performance with Complexity of Composite EI

For each value of average number of primitive EIs per composite EI, we compute the average processing time of each basic event (TP + TC) over 5 minutes and report it in Figure 4.9. The y-axis denoting the "Average Processing Time" (i.e., TP + TC) starts at 18900 and the x-axis denoting the "Average Number of Primitive EIs per Composite EI" starts at 4. It is obvious that as the complexity of the composite EIs increases the system is taking more time to process each basic event and identify corresponding composite EIs, however the increase in processing time is very minimal compared to the increase in the complexity of composite EI - from 19000 μseconds at 4 primitive EI per composite EI to 19900 μseconds at 20 primitive EI per composite EI.

Thus we can conclude that not only our approach of handling incoming RFID basic events provides much improved performance compared to "RDB" approach based on latest technologies, our approach also scales very well with increased rate of incoming basic events and the complexity of composite EIs giving *real-time performance* (few milliseconds) in all scenarios.

Chapter 5

Distributed RFID Event Handler

5.1 Need

So far we have assumed a single instance of RFID EH where all RFID basic events are gathered. If the number of event sources (i.e. RFID scanners) increases, both the number of basic events and EIs will increase. This may result in performance degradation of the centralized event handling system. Moreover in a centralized system, there is a potential risk of system failing even with single failure. The performance of the system, scalability and reliability can be improved by having a distributed event handling system. Moreover in real life situations raw RFID scans happen at distributed locations e.g. at various DCs or various stores in case of *MG*'s RFID deployment scenario as explaned in section 2.1. Each DC and store will have their own respective system to handle local data. Following this, in this chapter we briefly describe a distributed RFID event handling system that processes local basic events locally and communicates with other remote event handlers to address global business rules.

5.2 Literature Survey

5.2.1 Vanilla Approach

Figure 5.1 depicts a schematic diagram of a naive distributed event handling system, where D_1, D_2,...,D_5 are local RFID event handlers at each DC. Local RFID basic events generated out of local RFID scanners are processed by the corresponding local RFID EH. However there are some events that require monitoring and combining basic events at two or more locations e.g. "the number of a pallets of a particular color in all DCs in

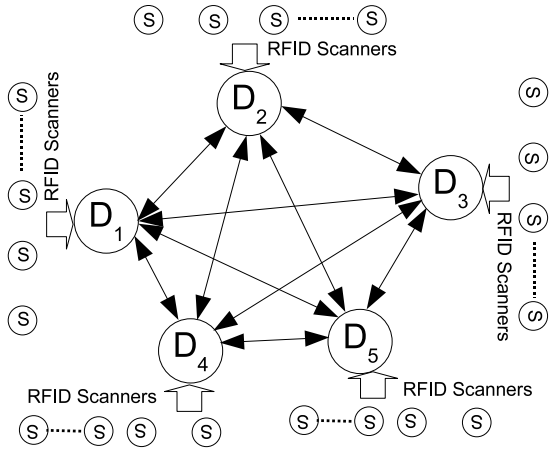

Figure 5.1: Distributed Event Handler

Mumbai should not go below a certain threshold". To identify such EIs it is necessary to consolidate basic events from multiple locations. A simplistic brute force approach is to broadcast all basic events to all other locations as and when they occur. So each system will be aware of basic events happening in other locations. However the obvious drawback for such approach is increased communication cost. However, the communication cost can be significantly reduced by assigning a primary handler for each global EI and maintaining a distributed directory of global EIs along with its primary handler. A number of variations of such approach can be found in distributed cache architecture and invalidation literature [18].

5.2.2 Location Hierarchy based Approach

Typically, geographically distributed enterprise systems follow organizational hierarchy, e.g., in *MG*'s case, store specific systems communicate with the DC responsible for a region. We exploit such a hierarchy in the enterprise distributed system. If such hierarchy does not exist in an enterprise, we can create such hierarchy in the of the enterprise by grouping EHs located at geographically distributed locations based on "location" dimension (L) of an RFID basic event. An example hierarchy is shown in Figure 5.2. Here, we group city based store EHs (Miami and Fort Lauderdale) on the basis of the distribution centers (South Florida region) serving them.

RFID basic events can arrive at any EH in the hierarchy. The EIs can also specified at

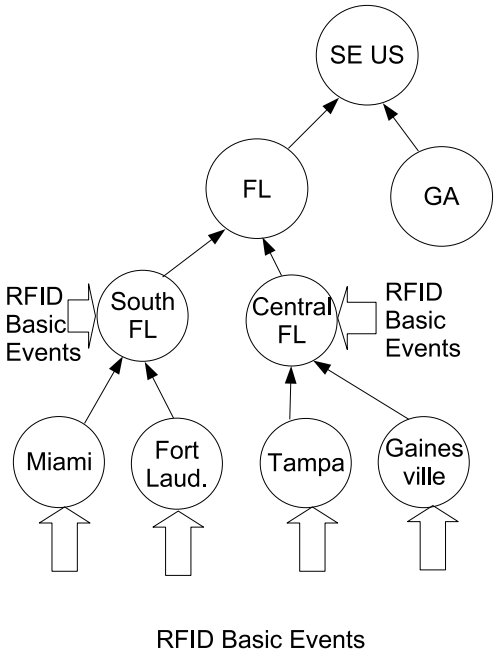

Figure 5.2: Location based Hierarchy in Distributed Event Handling

any EH. The "*location*" dimension (S) of an EI defined at a particular EH can have same value as that of EH or location value of any EH down below in the hierarchy, e.g., location of any EI for EH at South Florida can have one of the following values South Florida, Miami or Fort Lauderdale. When an EI is defined at an EH, corresponding primitive EIs are appropriately propagated to all respective EHs, e.g., when an EI $o_e^i = p_e^1 \wedge p_e^2$ with location $p_e^1.S$='South Florida' and $p_e^2.S$='Central Florida' is added at the EH in FL, the primitive EI p_e^1 is passed to EH in South Florida and subsequently down to EHs in Miami and Fort Lauderdale. Similarly the primitive EI p_e^2 is passed to EH in Central Florida and subsequently to EHs in Tampa and Gainesville.

Assume a basic event b_e with S='Miami' arrives at the EH in Miami. If EH identifies a match between b_e with p_e^1 in all dimensions (L, S and T) following the R-tree structure described in section 4.2.1, the b_e and p_e^1 is passed to the EH in the immediate higher level i.e., South Florida. When EH in South Florida receives this message, it identifies that the p_e^1 is for EH in FL and accordingly it passes this to the higher level until it reaches the appropriate EH in the hierarchy.

- Advantages of the scheme

  1. Basic events are screened at the level where a basic event is first reported. Thus if a basic event does not meet any of the EIs in the system the basic event will not be processed and propagated to the higher level in the hierarchy. This distributes the identification of EIs across all EHs.

  2. Once a primitive EI is identified to match a basic event, intermediate nodes passes the basic event and identified primitive EI to higher level. The identification of a primitive EI for a basic event happens only once at the EH where basic event first arrives in the system.

- Disadvantages of the scheme

  1. Conditions of an EI is evaluated where an EI is originally specified, e.g., in our example any condition related to composite EI o_e^i will be checked in the EH at FL. So all required parameters should be propagated to the originator EH.

  2. A single basic event may identify multiple primitive EIs and accordingly multiple message may need to be passed to higher level. This increases traffic on the channels between EHs.

In this approach, number of basic events processed at each EH is limited by the number of basic events being first reported in the system at that EH, making it a a scheme worth looking at.

5.3 Rule Evaluation Engine

How the events of interests are getting evaluated becomes a critcal issue when the underlying system becomes distributed. We will present couple of techniques for distributed rule evaluation namely, ***single-evaluator*** and ***multi-evaluator***. The basic idea behind both of them is clear from their names. To keep things simple, we assume events at their lowest granularity, in other words, every event is generated at different EH. While defining the ECA rule, any EI which is combination of different primitive EIs can be defined at any EH. Our proposal is to break these EIs so as to represent each of them as set of low level EIs, each having basic events from only one particular EH. Each one of these low level EIs will be evaluated at corresponding EH. This way the load on each EH also gets

distributed. Before describing the procedure for assigning EH to an incoming rule, we will like to explain the concept of `load on EH`.

Load of an EH is defined with certain statistical parameters specific to EH such as (i) Number of EIs the EH is currently evaluating i.e., the size of EIDB, this will be zero initially when there is no EI in EIDB. (ii) Basic event traffic that EH is currently handling i.e., number of basic event added to system per second. (iii) Availability of physical resource like processing power, RAM size, etc at the EH.

5.3.1 Single-evaluator Scheme

Once a rule or EI $O_1 = P_1 \wedge P_2 \wedge P_3$ is added to the system at any EH [1]. We will assign this rule to one of the participant EH and that EH is responsible for evaluation of this rule. Once decided a rule will be evaluated at one location only.

Least Loaded First(LLF): Evaluate newly added EI on the *least loaded* EH amongst its participant EHs. In our example, O_1 is assigned to least loaded EH amongst P_1, P_2 and P_3, say P_1. This means that O_1 will always be evaluated at P_1. This also states that P_2 and P_3 will be informed to forward information about occurence of primitive events to P_1 as and when event occur. Each EH maintains a state graph as proposed in section 4.2.2 for each rule it is supposed to evaluate.

- **Advantages**
  - Rule evaluation is easier as all required information is available at one place.
  - EI can only be evaluated at one EH, so problem of action getting repeatedly triggerd does not arise.
- **Disadvantages**
  - Rule evaluation is no longer one step process.
  - In assigning EH, while using LLF policy, communication overhead is involved in asking load to an EH.
  - This is a greedy approach, we can't achieve global optimum. All the primitive events of a particular EI might have occurred at different places but the EI can't be identified till all information reaches the evaluator EH.

[1] As per assumption, each of P_i are themselves EH. Also, each of P_1, P_2 and P_3 are called "participant EH" for O_1.

5.3.2 Multi-evaluator Evaluation

As opposed to single-evaluator scheme, every participant of the rule (P_1, P_2 and P_3 in O_1) keeps the information about the rule and is capable of evaluating it. When a rule is added at any EH, it sends the rule to all the participant EHs of the rule. Each of these EH maintains a state graph 4.2.2 for this rule. So when any event b_e occurs at P_1, if this completes O_1, then inform P_1 and P_2 to clean their forwarding tables else inform P_2 and P_3 about the occurence of b_e. Thus whenever any of the rules from the rule table is complete either because of event occurring locally or event occurred at remote EH, the rule will be evaluated.

So, apart from EIDB and state diagrams, we need to maintain a list of other EHs to which information about event happening has to be forwarded, we call it *forwarding table*. This table has to be updated when the rule is identified either locally or at some other EH.

- **Advantages**

  – One big advantage of this scheme is that the rule gets evaluated as soon as all its component events happen. There is absolutely no delay.

- **Disadvantages**

  – Each time a event occurs we need to broadcast the information for its observer EHs, this makes it similar to the vanilla approach explained in 5.2.1, which is a big communication overhead.

- **Issues**

  – **Synchronization problem**: Consider a rear possibility when $O_i = P_A \wedge P_B \wedge P_C$ is waiting for only one basic event to occur; this event occurs simultaneously at all EHs, eventually the EI will be identified at all three EHs and action will be trigged thrice. To solve such problem, a distributed locking mechanism has to be devised.

  – **Flushing forwarding table**: One of the critical problem from implementation point of view is how and when to flush the forwarding table which keeps growing both because more and more EIs are getting added and more basic events are generated.

5.3.3 Exploiting Common sub-expressions across EIs

Let us assume we have following 2 rules in the system. E_1 = A $\wedge$ B $\wedge$ C and E_2 = B $\wedge$ C $\wedge$ D. Here B $\wedge$ C is a common subexpression. Most often common patterns can be identified from across all the rules we are observing. For obvious reasons, the common sub-expression should get evaluated only once. Following sub-section explains how this can be achieved in both single-evaluator and multi-evaluator schemes of rule evaluation.

5.3.3.1 Single-evaluator Scheme

With our example of E_1 and E_2, let us consider following two cases

Case 1: E_1 **getting evaluated at A** – When E_2 is added to system at A, we identify that it has a subexpression which is common to E_1. So for A $\wedge$ B can be found at A, therefore E_2 should also be evaluated at A.

Case 2: E_1 **getting evaluated at C** – When E_2 is added to system at A, though it has a subexpression which is common to E_1, we wont be able to deduce that because we do not have information about E_1 at A.

So to take advantage of common subexpression, we need to have all the rules in the system at one point where the matching occurs. One way to do this is to have an independent centralized rule server which for a given new rule will tell us what other rules have matching sub expressions with this rule. Standard sub-expression matching algorithms can be used for this purpose. So when a new rule comes into the system, we perform two operations simultaneously.

1. Search for any common subexpression - This operation occurs at rule server.

2. Depending on load of participant EHs, select suitable EH to be assigned to this rule.

5.3.3.2 Multi-evaluator Evaluation Scheme

In multi-evaluator scheme of rule evaluation, common subexpressions are inherently getting evaluated only once. In our example, lets assume B occurs first, so B will forward this event to A, C, and D. So state diagrams for both E_1 and E_2 gets updated,. Note that B sends data to C only once though B $\wedge$ C occurs in 2 rules. Thus, common subexpression is evaluated only once.

5.4 Implementation in Sun RFID System

5.4.1 Sun RFID Event Manager

The Sun Java RFID Middleware Consists of (i) the RFID Event Manager, (ii) the RFID Management Console and (iii) the RFID Information Server Modules. The 'Event Manager' communicates with the sensor devices, gathers sensor data, filters it, and passes it on to the Sun RFID Information server for future processing. RFID Event Manger can be instructed on how to process the information and define the subsequent consumers of the processed information. The RFID Event Manager consists of a single Control Station and a number of Execution Agents. Execution Agent communicates with the physical devices, processes the information, and conveys the information to the receiver of the information. Along with the 'Control Station' a 'Configuration Object' is also installed, which specifies one or more device(s) to control and a set of components that process the information obtained from those devices. This set of information processing components are called the Business Processing Semantics unit (BPS) [19].

The Configuration Object is a collection of objects; viz. adapters, filters and connectors. They are linked in a chain in the same order. An adapter acts like a device driver, communicating with the device on one side and passing on the information to one or more filters chained to it. Filters are used to remove noise, excessive read events or to perform other data manipulation. Connectors are at the end of the chain passing on the information to end consumers.

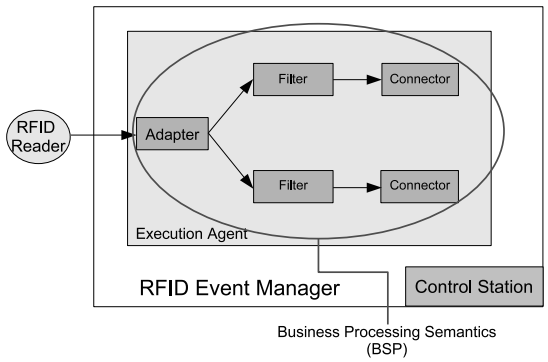

Figure 5.3: Sun RFID Event Manager

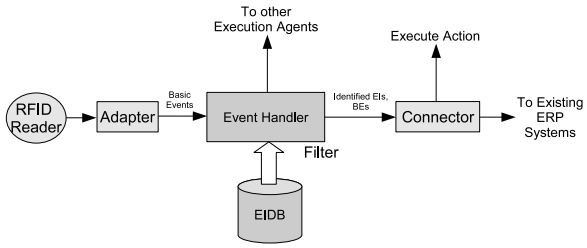

Figure 5.4: Execution Agent

The RFID Information Server is a Java Enterprise application, that servers as an

interface for capture and query of EPC related data. In short, the RFID Information Server converts a set of low level observations into a set of high level business functions. The RFID Configuration Manager or the RFID Management Console provides a means to configure the various parts of the RFID System being deployed. It provides a graphical interface to deploy RFID reader sources, filters and connectors, assign policies, etc.

5.4.2 Proposed Scheme

We integrate the RFID ECA Architecture within the BPS discussed earlier as shown in figure 5.4. We modify the filter from the BPS to accommodate the EIDB as well as the RFID Event Handler. Thus, filter will be responsible for identifying the Events of Interest from the Basic Events coming from the adapters. The Connectors will then be used to perform the action associated with the EIs. The Connectors will also be responsible to carry forward the Basic Events to the existing enterprise system for future use. In the centralized version of the system, the EIDB will contain all the rules corresponding to the visibility of the system. Suppose that the system is being deployed within a store, then all the rules in the EIDB will recognize events from within the store. This is called 'Local Rule'.

We now assume that there exists an ontology based system in place for the distributed system to work. In the distributed scenario, the distributed rules (rules involving sources from different Event Managers) when fed to the system, are broken down into local rules for each of these 'Event Managers'. Thus, each of these EIDBs are sent a part of the rule, which will help the Event Managers in identifying a particular distributed rule. When a basic event (which might trigger a distributed rule) occurs at any of these Event Handlers, then that Event Handler will communicate with the other EIDBs which are part of the rule.

5.5 Implementation using RMI

RMI Primer

Java Remote Method Invocation (Java RMI) enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism.

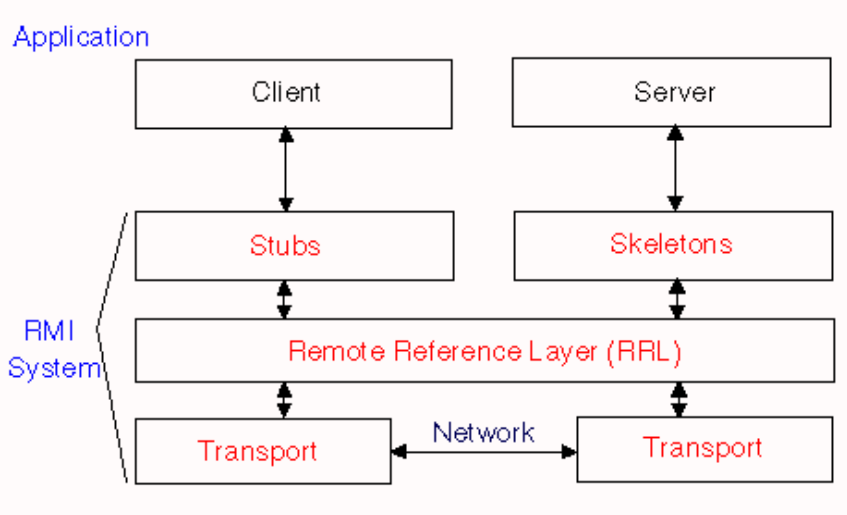

Figure 5.5: Interaction among elementary services (ECA-rule processing)

RMI Registry: Each object that can be accessed remotely, needs to register with RMI registry with unique name. The registry maintains the mapping from names to remote object references.

Stub: This executes on a client machine and acts as a proxy for a remote object and presents the same interfaces as the remote object. It gets arguments from the local object, serialize them and send them to the remote machine; on the return path, it receives results form the remote machine, desearialize them and return to the local object.

Skeleton: This executes on a server machine, receives the remote method call and any associated arguments, invokes the appropriate method of the remote object, receives results, serialize them and send back to the client machine.

5.5.1 Class Diagram and Call Sequences

Figure 5.6 is the class diagram for implementation of single-evaluator engine (section 5.3.1). EIDB is a singleton class which implements EIDBInterface, a common interface for all remote EHs. Call sequence of typical EH is shown in algorithm 3. It first registers the

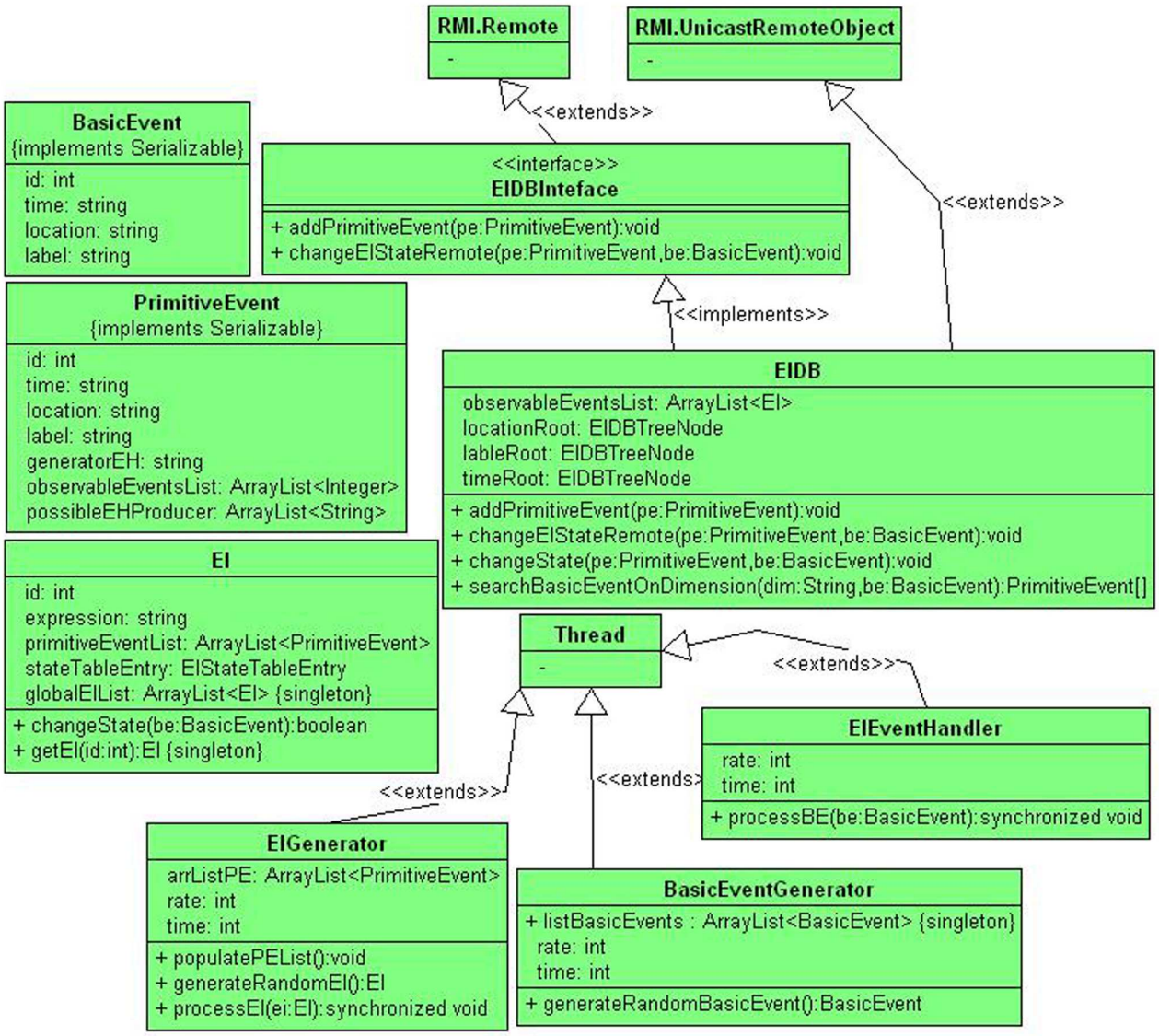

Figure 5.6: Class Diagram for RMI Implementation

EIGenerator object with RMI registry; then builds EIDB at local system, while doing so it makes some RMI calls on other EHs to update remote EIDBs as shown in EIgenrator algorithm 4, then it waits till all other EIDBs are built, finally starts basic event generator and event handler.

Algorithm 4 shows a EIGenerator module where randomly generated EIs are being added

Algorithm 3 Execution sequence of typical EH

1: Assign name to EH
2: Initialize EI Generator
3: Register the EH in Naming Registry
4: Poll till all other EHs are registered
5: Start EI Generator and build EIDB
6: Poll till all other EIDBs are built
7: Start Basic Event Generator
8: Start Event Handler

Algorithm 4 Execution sequence of typical EIGenerator

1: E_i = Generate a random EI
2: Initialize the state diagram for E_i.
3: **for** $p_e \in E_i$.PEList **do**
4: **for** EH $\in p_e$.possibleEHList **do**
5: **if** EH is **remote then**
6: Get the corresponding EH object from registry and Make a RMI call to add p_e to Remote EIDB
7: **else**
8: Add the E_i and p_e to local EIDB
9: **end if**
10: **end for**
11: **end for**

Algorithm 5 Execution sequence of typical EIHandler

1: Pick Basic Event b_e from Queue
2: $\mathcal{P}$ =Search EIDB formed so far for any matching PEs for this b_e.
3: **for** $p_e \in \mathcal{P}$ **do**
4: **for** EH $\in p_e$.possibleEHList **do**
5: **if** EH is **remote then**
6: Get the corresponding EH object from registry and make RMI call to update the state of remote EIs to incorporate the happening of b_e.
7: **else**
8: Change state of associated EIs to incorporate the happening of b_e.
9: **end if**
10: **end for**
11: **end for**

to local as well as remote EIDBs. Algorithm 5 is a typical EIHandler which picks basic event b_e from queue, search EIDB for matching primitive events, and changes state of EIs to capture the happening of b_e. Note that EIs can be local as well as remote.

5.5.2 Experimental Results

In this section, we will demonstrate the efficacy of RMI based implementation of distributed RFID system. We do not have a equivalent system available, so we cannot present any comparative study, but through our experiments we try to show how our system scales with number of basic events, number of EIs and number of EH in system. The experimental setup is same as that of centralized system described in section 4.5 and parameters for experimental analysis are as shown in table 5.1. We first initialize EIDB

Parameter	**Values**						**Base Value**
Number of Composite EIs	2000	4000	6000	8000	10000	-	10000
Rate of incoming basic events (/sec)	2	3	4	5	6	7	2
Number of Primitive EIs per Composite EIs	4	6	8	10	12	14	4
Number of EH nodes in system	4	6	8	10	12	-	4

Table 5.1: Experimental Parameters for distributed system

with 2000 composite EIs evenly distributed among 4 EHs(10000 and 4 being base values of composite EIs and number of EHs in the system respectively). Then we programmatically simulate the basic event generation and send basic events to all RFID EHs at a fixed rate (2 basic events per seconds). We continue this for 5 minutes, note the TP, TC, and queue waiting time for each basic event (TQ). We repeat this process for each value of number of composite EIs in the EIDB as given in table 5.1. In all cases, we keep the rate of incoming basic events constant at the base value 2 per seconds.

In figure 5.8, we plot the TC+TC and total time required to process a basic event (TP+TC+TQ) respectively as it varies with the number of composite EIs in EIDB. We can see, as size of EIDB grows, the TP+TC grow linearly because of increased search time. We also observe that, the time each basic event spends waiting in queue grows

exponentially with size of EIDB. To demonstrate the scalability of the system, we kept the number of composite EIs in EIDB constant at its base value 10000. We varied the rate of incoming basic events from 2 to 7 events per second. For each rate, we measured the total time (TP + TC) required to process each basic event and average it over 5 minutes. We plot the total processing time with the rate of incoming basic events in figure 5.7. As the incoming rate of basic events reaches a threshold, which is actually the average TP+TC for one basic event, the total time required to process event grows exponentially because of the queue build up.

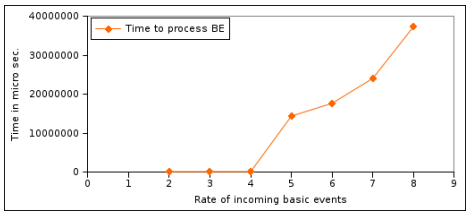

Figure 5.7: Performance with variation of rate of incoming basic events

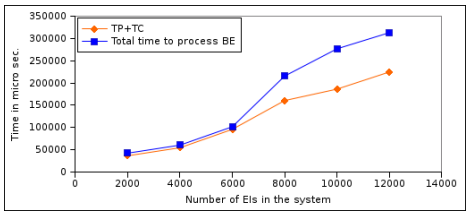

Figure 5.8: Variation of TP+TC and Total processing time for basic event with # EIs

To examine how the complexity of composite EIs will affect the scalability of the system, we kept the total number of composite EIs in EIDB constant at the base value of 10000, the rate of incoming basic events to 2 per seconds and vary the average number of primitive EIs per composite EI as described in table 5.1. For each value of average number of primitive EIs per composite EI, we compute the average processing time of

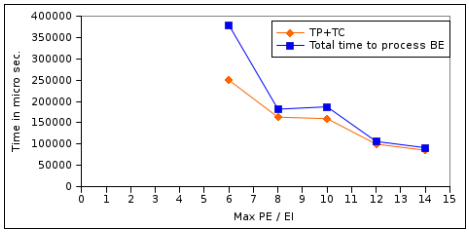

Figure 5.9: Performance with Complexity of Composite EI

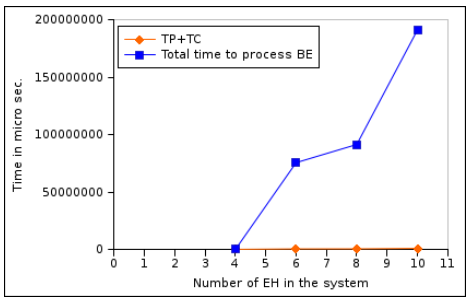

Figure 5.10: Performance with number of nodes(EHs) in the system

each basic event (TP + TC) over 5 minutes and report it in figure 5.9. It is obvious that as the complexity of the composite EIs increases the system is taking more time to process each basic event and identify corresponding composite EIs. We then demonstrate how the system behaves when more and more nodes (EHs) are getting added, as usual we kept the total number of composite EIs in EIDB constant at the base value of 10000, the rate of incoming basic events to 2 per seconds and vary the number of EHs as described in table 5.1. We compute the average processing time of each basic event (TP + TC) over 5 minutes and report it in figure 5.10. It is obvious that as the number of nodes increases, so is the time to process each basic event and identify corresponding composite EIs.

Chapter 6

Summary and Conclusion

Conclusion

We described the architecture of a system that can handle large number of incoming RFID events and identify events of interest in real-time. We developed an event based model for RFID system using the ECA framework. We demonstrated that our approach in identifying RFID events of business interest can perform significantly better than an implementation using latest available technologies. We described how our approach can be extended in a distributed scenario. We implemented the distributed architecture for RFID event handling mechanism and integrated it with Sun's Open source RFID Middleware. Lastly we implemented a distributed architecture using Java-RMI. We conclude the work by showing how the propsed system can be efficiently scaled.

6.1 Future work

Through this project we have explored the possibility of using ECA model for RFID applications. A map of future research directions is presented below:

- **Design Extensions**:
  - Design synchronization protocols for distributed system: As explained in section 5.3.2, we need a synchronization protocol. This is a nice independent problem. Various techniques on distributed locking can be used here.
  - Design protocols to deal with failure recovery: What if one of the nodes in the system goes down because power failure? We need some mechanism to identify the crash and an efficient scheme for making a timely recovery of the system.

- Implementation Extensions
  - Implementation of multi-evaluator scheme: In section 5.5, we explained the implementation level details of single-evaluator scheme as explained in section 5.3.3.1. But the performance analysis for multi-evaluator scheme needs to be done. This can only be done after designing the synchronization protocol.
  - Integrating ontology based structure with proposed system: As we could see, throughout the system, we need some ontology structure to be present; we in our implementation have designed a static ontology. To make this system work in real life situations, we need to integrate an efficient ontology structure to it. Various ontology schemes exists, integrating one of that will be a good point to carry this project on.

Bibliography

[1] Z. Ton, V. Dessain, and M. Stachowiak-Joulain, "Rfids at the metro group," 2005.

[2] H. Gonzalez, J. Han, X. Li, and D. Klabjan, "Warehousing and analyzing massive rfid data sets," in *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, 2006, p. 85.

[3] N. W. Paton and O. Díaz, "Active database systems." *ACM Comput. Surv.*, vol. 31, no. 1, pp. 63–103, 1999.

[4] A. Nagargadde, S. Varadarajan, and K. Ramamritham, "Semantic characterization of real world events," in *DASFAA*, 2005, pp. 675–687.

[5] J. Hoag and C. Thompson, "Architecting rfid middleware," *IEEE Internet Computing*, vol. 10, no. 5, pp. 88– 92, 2006.

[6] Sun Microsystems, "Software solutions: Epc and rfid," 2006.

[7] IBM Inc., "Integrate your enterprise application with ibm websphere rfid middleware," 2006.

[8] S. Chakravarthy, "Sentinel: an object-oriented dbms with event-based rules," in *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, 1997, pp. 572–575.

[9] S. Chakravarthy, R. Le, and R. Dasari, "Eca rule processing in distributed and heterogeneous environments," in *Proceedings of the International Symposium on Distributed Objects and Applications*, 1999, pp. 330 – 339.

[10] E. Hanson, C. Carnes, L. Huang, M. Konyala, L. Noronha, S. Parthasarathy, J. Park, and A. Vernon, "Scalable trigger processing," in *Proceedings. of 15th International Conference on Data Engineering*, 1999, pp. 266–275.

[11] J. Rao, S. Doraiswamy, H. Thakkar, and L. S. Colby, "A deferred cleansing method for rfid data analytics." in *Proceeding of VLDB Conference*, 2006, pp. 175–186.

[12] S. R. Jeffery, M. N. Garofalakis, and M. J. Franklin, "Adaptive cleaning for rfid data streams." in *Proceeding of VLDB Conference*, 2006, pp. 163–174.

[13] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, "Online outlier detection in sensor data using non-parametric models." in *Proceeding of VLDB Conference*, 2006, pp. 187–198.

[14] J. Song and H. Kim, "The rfid middleware system supporting context-aware access control service," in *Proceedings of The 8th International Conference Advanced Communication Technology, 2006. ICACT 2006*, 2006, p. 4 pp.

[15] J. Hoag and C. Thompson, "Architecting rfid middleware," *IEEE Internet Computing*, vol. 10, no. 5, pp. 88 – 92, 2006.

[16] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, 1984, pp. 47–57.

[17] Oracle Inc., "Oracle database 10g express edition," 2006.

[18] F. Pong and M. Dubois, "Verification techniques for cache coherence protocols," *ACM Comput. Surv.*, vol. 29, no. 1, pp. 82–126, 1997.

[19] *Sun Java System RFID Software 3.0 Developer Guide*, Sun Microsystems, Inc., Feb 2006.

[20] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. E. Sarma, "Managing rfid data," in *Proceedings of the 30th International Conference on Very Large Data Bases, Toronto, Canada*, 2004.

[21] F. Wang and P. Liu, "Temporal management of RFID data," in *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway*, 2005.

[22] F. Wang, S. Liu, P. Liu, and Y. Bai, "Bridging physical and virtual worlds: Complex event processing for RFID data streams," in *10th International Conference on Extending Database Technology. Munich, Germany*, 2006.

[23] Y. Hu, S. Sundara, T. Chorma, and J. Srinivasan, "Supporting RFID-based item tracking applications in oracle DBMS using a bitmap datatype," in *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway*, 2005.

[24] S. R. Jeffery, M. Garofalakis, and M. J. Franklin, "Adaptive cleaning for RFID data streams," in *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea*, 2006.

[25] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom, "Declarative support for sensor data cleaning." in *Pervasive*, 2006.

[26] H. Gonzalez, J. Han, X. Li, and D. Klabjan, "Warehousing and analyzing massive RFID data sets," in *The 22nd International Conference on Data Engineering, Atlanta, GA*, 2006.

[27] H. Gonzalez, J. Han, and X. Li, "Flowcube: Constructuing RFID flowcubes for multi-dimensional analysis of commodity flows," in *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea*, 2006.

[28] B. Glover and H. Bhatt, *RFID Essentials - Theory in Practice.*

[29] S. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom, "A pipelined framework for online cleaning of sensor data streams," EECS Department, University of California, Berkeley, Tech. Rep., 2005.

[30] M. Cilia, C. Bornhövd, and A. Buchmann, "CREAM: an infrastructure for distributed, heterogeneous event-based applications," in *Proceedings of the IFCIS Conference on Cooperative Information Systems (CoopIS'03)*, ser. LNCS, vol. 2888. Catania, Italy: Springer, Nov. 2003, pp. 482–502.

Conference Proceedings

1. Kamlesh Laddhad, Bernard Menezes, Karthik B., and Kaushik Dutta, "Challenges in RFID Deployment - A case study in public Transportation", *Proc. 4th International Conference on E-governance (ICEG)*, New Delhi, India, Dec. 2006.

Acknowledgements

I take this opportunity to express my sincere gratitude towards **Prof. Bernard Menezes** and **Prof. Krithi Ramamritham** for their constant support and encouragement.

I would especially like to thank **Prof. Kaushik Dutta** from the Florida International University for his valuable support and useful insights towards the project and for helping us remotely during the course of the work. At last, I would like to thank my friend **Karthik B.** for all the discussions regarding the project.

Kamlesh Laddhad

I. I. T. Bombay

July 6^{th}, 2007