The Study and Application of Tree-based RFID Complex Event Detection Algorithm

Hongying LIU¹, Satoshi GOTO², and Junhuai LI³
¹, ² Graduate school of IPS,Waseda University,Kitakyushu, Japan
³ School of Computer Science & Technology,XiAn University of Technology,Xi’An, China

Email: {liuhongying@fuji., Goto@}@waseda.jp
Email: lijunhuai@xaut.edu.cn

Abstract—With the extension of RFID application, RFID data processing techniques become one of the core techniques. Recently, researchers adopt the event-centric methods to process RFID data and obtain rich semantic information. Complex event processing is a typical data processing technique which becomes the new spotlight of researches. In order to obtain semantic data, in this paper, we propose a complex event detection model to process RFID data. A tree-based complex event detection algorithm which is an improvement to the existed one is realized as the core module in our model. Aimed at improving the detection of non-spontaneous events, we use expected events thus avoid the recursive computation of detection modes and save time cost. The comparative experiments show the algorithm decreases the time complexity of RFID complex event detection. Finally, we implement a production information system based on the complex event detection model and it provides better support for the auto-monitoring of production line.

Index Terms—complex event detection, RFID data processing, production information system, tree-based algorithm

I. INTRODUCTION

The conception of complex event (sometimes it is also called composite event) origins from the research rule processing in active database. The detection of complex events is worthwhile for RFID applications. A number of complex events are frequently used in application systems. By detecting such complex events, they can be reused and the development of related software becomes much easier.

Currently, the complex event detection approaches fall into four categories: finite automata-based, Petri net-based, graph-based and tree-based. Researchers have contributed to these models or methods.

The finite automata-based complex event detection approach: Such as the work of [1, 2]. In the research [3, 4], the author uses a finite automaton Cayuga to detect complex events. The problems of repetition operation and temporal constrains are solved by Cayuga. But many internal naming are added, it is difficult to understand by users. The Petri net-based complex event detection approach: SAMOS system [5] treats event as a time point. The HiFi system [6, 7] discusses general issues in system architecture. However, it is limited in supporting completed temporal events. The graph-based complex event detection approach: Snoop system [9] adopts directed acyclic graphs (DAGs) to detect complex events. But there is no formal framework to express the conception of complex events and to describe the detection process. EVE system [8] uses graphs too.

The main idea of the tree-based complex event detection approach is to construct matching trees from complex event mode. In a matching tree, leaf nodes represent primitive events and other nodes denote the composition of events. The system—Yeast [10] is an early work to use the tree-based structure. Though GEM [11] clearly expresses complex subscription, the subscription tree can not be shared by other subscription. When considering the delay between occurrence and detection of an event, it presumes the existence of a global synchronous clock. READY [12] is an event notification service, which provides effective and loose coupling notification. It is helpful to the information integration among enterprises. However, it uses CORBA notification structure, which limits its application. In [13], the author proposes EPS, in which a subscription tree is used to process events. Nevertheless, a mathematically analysis of the complexity of EPS algorithm is uncompleted.

Among the four types of detection approaches, tree-based complex detection is appropriate for the expression of composite process and it is a sophisticated technology compared with other approaches. So in our work, we use tree-based structure to detect complex event.

The remainder of the paper is organized as follows: Section II introduces the related work. Section III presents framework of our complex event detection model. Section IV discusses important issues in algorithm. Section V explains the complex event detection algorithm in detail. Section VI lists several application examples in our prototype system. Section VII draws conclusions and suggests future research.

II. RELATED WORK

There are several improvements to the RFID data processing in [14]. The author proposes a series temporal constrained event operations and prove its effectiveness with several application examples. In addition, the author believes there are non-spontaneous complex events such as “negation” and “repetition”. They can not judged by the occurrences of primitive events. These are all the contributions. Furthermore, he proposes pseudo events to solve the detection of non-spontaneous events, and the
detection modes are used to aid the detection. However, there are still several flaws with these concepts.

1. The introduction of event detection modes is aimed to detect non-spontaneous events. The detection modes are classified into three types: Push, Pull and Mixed. The detection mode for spontaneous event is always “Push”. While the difference between the definitions of “Pull” and “Mixed” are not clearly shown.

2. When the detection modes are used, the algorithm computes the detection mode of every node. This greatly increases the time cost of the algorithm.

3. Pseudo events are created to detect non-spontaneous events. But the algorithm only considers the backward sliding time window and ignores the forwards. Since the occurrence of event instances at leaf nodes is in an equal probability, it can not be ignored.

In order to solve the above problems, we make improvements to the algorithm RCEDA. The main idea of our strategy is: withdrawing the concept of detection mode; and thus eliminating the needs to compute the detection mode of every node and decreasing execution time; adopting a dual-directional sliding window to detect non-spontaneous events.

Definition 1: primitive event.

A primitive event is an observation read by the reader from tags. It is the basic unit of events. It is defined as:

Primitive Event := (ReaderID, ObjectID, TimeStamp, <AddProperties> )

This is a four-tuple, which contains the following elements: the ID of reader, the ID of tag, the time stamp of the occurrence of event, and the additional properties. The <AddProperties> part includes the data that application recorded on the tag. It is not an obligatory part. Some read-only tags may not contain such properties. Notably, a primitive event is instant. That means for any primitive event instance e, there exists: t Begin(e) = t End(e).

A complex event is the conjunction of one or more primitive events. These primitive events associated together by certain operations. It is defined as:

Complex Event := U (Primitive Event) +

Where “+” denotes one or more times’ operation. We realize them with event constructors, which are the same as the work in [14].

III. COMPLEX EVENT DETECTION MODEL

With the extension of RFID application systems, RFID data which has been primitive processed can not satisfy the needs of users who want data that is full of rich semantic information. So it is necessary to further process the RFID data. We use complex event detection techniques to extract more valuable information from the low-layer RFID data to provide to various applications. To make complex event processing more efficient and effective, we propose a model to carry out the complex event detection algorithm, shown in Fig1.

The core of the model is complex event detector (namely CE detector), which executes the function of complex event detection. It is composed of RFID rule repository, primitive event repository, collector, constructor, analyzer, and expression interface. The process of complex event processing in CE detector is as follows: When the pre-processed RFID data enters CE detector, the collector transforms it into the format of primitive events which has been defined in the system, and stores it into RFID rule repository. This transformed data will be used by constructor later. Applications may request to monitor business events or user may submit interested queries to CE detector. The analyzer parses them and index similar rules from RFID rule repository. If there are such rules, then the analyzer will quote the indexed one. If not, the analyzer will translate them into RFID rules that consistent with the standardizations of CE detector, and store them into RFID rule repository.

The constructor detects complex events according to primitive events and RFID rules. It gets primitive events from collector or primitive event repository and search RFID rules from RFID rule repository. Then it begins to match the events with rules by the complex event detection algorithm and output results to the expression interface. The expression interface represents the results to users in appropriate form.

In the model, the constructor is critical module. The efficiency of CE detector is up to the algorithm adopted by constructor. So in our paper we focus on the complex event detection algorithm.

IV. IMPORTANT ISSUES

A. Important Definitions

Definition 3: Complex event expressions.
The followings are complex event expressions:

1. Primitive events;
2. Resulted events that operated from primitive events;
3. Resulted events that operated from complex events;

The experssional ability of complex events is determined by the event operators defined in event system. Notably, the construction process of complex event tree is the same with the process in [14]. But different with the previous work, we add the following structures to assist the detection process.

1. Node counter: we set such counter for each non-leaf node in the event tree to record the number of events occurred at the leaf node.
2. Node event queue: each node of the event tree keeps one node event queue to store the events arrived.
3. Expected event queue: for each non-spontaneous event node, such queues are to store the expected events.
These structures will be set up at the initialization period.

B. The Detection of Non-spontaneous Events

Now firstly we clarify the concept of the non-spontaneous events. In the following discussion, $e_i^j$ denotes an event instance that takes place at time $j$ and is of event type $E_i$.

Non-spontaneous events are such events that they can not detect their occurrences by themselves unless they either get expired. However whether an event is non-spontaneous is doubtful. We only consider the events from negation constructor as non-spontaneous events. In the following discussion, $e_i^{\alpha}$ denotes an event instance that takes place at time $j$ and is of event type $E_i$.

In a general RFID application system, the events from “SEQ+” constructor can detect themselves. An instance is like this: in the vicinity of a RFID reader, one tag may be recognized many times. Then there is the complex event $E=SEQ+(E_1)$. Every time the instance of $E_1$ occurs, the occurrence of $E$ is detected. Similarly, $E=TSEQ+(E_2, \delta_1, \delta_2)$ can be detected.

Expected events are such events that computed from non-spontaneous events by time-period operation. They are created to aid the detection of complex event which contains negation operation. We adopt the following computation approach, namely time-period operation.

For a non-spontaneous node $V_e$, when the instance of $E$ occurs, we create the expected event $e_{exp}$. It is computed as follows:

$$e_{exp} = \mathbf{E}(\{t-\delta, t+\delta\})$$

where $\delta$ denotes interval constraint, and $(t-\delta, t+\delta) > 0$.

Since there are two types of temporally constrained events: interval constraint and distance constraint, we list the approach respectively. Suppose there is an event history: $\{e_1, e_2, e_3\}$. An complex event $E=WITHIN(\neg E_1 \land E_2, 10sec)$ is detected in the following steps, and the graphical description is shown in Fig2.

Step1: On arrival of $e_2^{13}$, it enters event queue of $V_{E_1}$, and $V_{E_2}$ propagates $e_2$ to its parent node $V_E$, the counter of $E_2$ increases.

Step2: On arrival of $e_1^{0}$, it enters the event queue of $V_E$, and the expected event $e_1^{0-19}$ is created and put into expected event queue of node $V_E$.

Step3: The expected event $e_1^{0-19}$ propagates itself to its parent node $V_E$, counter of $E_1$ increases. Both of the counters are no longer zero. Then detection is triggered. $E_21$ matches with $e_1^{0-19}$, a result is outputted as “Null”.

Step4: On arrival of event $e_1^{13}$, it enters the event queue of $V_{E_1}$, and the expected event $e_1^{3-23}$ is created and put into expected event queue of node $V_{E_1}$.

Step5: The expected event $e_1^{3-23}$ propagates itself to its parent node $V_E$, the counter of $E_1$ increases. Both of the counters are no longer zero. The detection is triggered again. $e_1^{3-23}$ dis-matches with $e_1^{3-23}$, a result is outputted as “$e_1^{3-23}$”.

We illustrate the detection process with expected events in another example. Suppose there is an event history: $\{e_5, e_6, e_7, e_8, e_9, e_10\}$. And to explain the distance constraints, we will give detection of the complex event $E=TSEQ (E_1 : \neg E_2, 1sec, 3sec)$ in the following steps, and the graphical description is shown in Fig3.

Fig2. The detection process of $E=WITHIN(\neg E_1 \land E_2, 10sec)$ with expected events

Fig3. The detection process of $E=TSEQ (E_1 : \neg E_2, 1sec, 3sec)$ with expected events
Step1: On arrival of $e_1^0$, it enters the event queue of $V_{E_1}$, and $V_{E_1}$ propagates $e_1^0$ to its parent node $V_{E}$, the counter of $E1$ increases.

Step2: On arrival of $e_2^1$, it enters the event queue of $V_{E_2}$, and the expected event $e_2^{2,4}$ is created and put into expected event queue of node $V_{E_2}$.

Step3: The expected event $e_2^{2,4}$ propagates itself to its parent node $V_{E_2}$, the counter of $E2$ increases. Both of the counters are no longer zero. Then detection is triggered. $e_2^0$ matches with $e_2^{2,4}$, a result is outputted as “Null”.

Step4: On arrival of event $e_9^0$, $e_1^{13}$, they enter the event queue of $V_{E_1}$. According to the properties of TSEQ, they occur after event $e_1^0$, so they do not trigger the match process.

Step5: On arrival of $e_2^{14}$, it enters the event queue of $V_{E_2}$, and the expected event $e_2^{1,13,15-17}$ is created and put into expected event queue of node $V_{E_2}$.

Step6: The expected event $e_2^{1,13,15-17}$ propagates itself to its parent node $V_{E_2}$, the counter of $E2$ increases. Both of the counters are no longer zero. Then the detection is triggered. $e_2^9$ dis-matches with $e_2^{1,13,15-17}$, and $e_2^0$ dis-matches with $e_2^{1,13,15-17}$, the result is outputted as “$e_2^9 e_2^{14} e_2^{15-17}”.

V. TREE-BASED COMPLEX EVENT DETECTION

A. Algorithm Description

Based on the above definition and rules, we describe the RFID complex event detection algorithm, namely IM-RCEDA, shown in Fig4. The input of the algorithm: RFID rules set: $R=\{r_1, r_2, ..., r_m\}$, primitive event set: $E=\{e_1, e_2, ..., e_n\}$, Output: The set of detection result $H$.

At the beginning of algorithm, we construct event tree $T$ according to the RFID rule $r$ in set $R$, then we initialize $T$ by the following steps. (1) Propagate interval constraints from the root node of $T$ in a top-down way. (2) Set counters for every non-leaf node. And the number of counters depends on the number of child nodes. These counters increases when the node is activated, while decreases when a match occur. (3) Build node event queues for each leaf node in $T$. (4) Assign flags for each non-spontaneous node, and build expected event queues for them. Then we can start to detect the occurrence of complex events with this algorithm.

B. Performance Evaluation

To evaluate the performance of the algorithm, we compare it with RCEDA [14]. The testing environment is an Acer 7700G-M computer with 3.2GHz Pentium 4 CPU, and 1GB memory. The operation system is Windows Server2003. And we realize our algorithm IM-RCEDA with C#.

The time complexity of algorithm is related to three factors: the number of rules, the amount of event instances and the type of events. We tested the event processing time versus the number of primitive events in Fig5, and the event processing time versus the number of RFID rules in Fig6. Under both circumstances, we set the event arrival rate as 1000 events per second in order to compare it with RCEDA.

In Fig5, the time cost of the two algorithms increases as the primitive events increase. But apparently, the increment of IM-RCEDA is lower than RCEDA. In Fig6, when the numbers of RFID rule increase, both the time cost of the two algorithms rise up. While the rule number surpass 400, IM-RCEDA is much better than RCEDA.

VI. APPLICATION EXAMPLES

We have developed a prototype system with our complex event detection model in which the IM-RCEDA is implemented and run on the production lines.

A. Case Query

In the warehouse, the manager may query the number of cases passed through the two entries within certain time period. For example, r1 and r2 are installed in the two entry ports of a warehouse. Now within 10 minutes, the warehouse manager wants to know the exact number of cases entered. We can input such RFID rules to Detector, shown in Fig7. And Detector will call IM-RCEDA to detect the occurrence of complex event “WITHIN (E1 OR E2, 10min)” if it is detected, the counter will increase. And finally the result will represent to manager.

B. Production Line Monitoring

A common RFID application in manufacture is to record the production data and monitor the production
work in progress, we could monitor the procedures automatically. For example, if the time period between the first procedure and the seventh procedure exceeds 20 minutes, it is supposed to be abnormal and the Detector will take measures, such as buzzing, showing warning information, and shutting down certain machines etc. Fig 8 illustrates this process.

C. Working State detection

There is another application in monitoring the production line. We can judge the working state of the readers in system with the algorithm. For example, in Fig9, if there are no observations from reader r1 within 1 hour (say from 8 to 9), it means reader r1 is not online. So Detector will show “Offline”.

Fig7. Case counting rule

DEF E1=Observation(r1. o. t1)
DEF E2=Observation(r2. o. t2)
CREATE Rule02 Case_Detection
ON WITHIN (E1 V E2, 10min)
IF true
Do Counter++;

Fig8. Working process detection rule

DEF E1=Observation(r1. o. t1)
DEF E7=Observation(r7. o. t7)
CREATE Rule07 TimeDetection
ON TSEQ (E1; E7, 0, 20min)
IF false
Do Buzz; Warning; Shut down;

Fig9. Reader states detection

VII. CONCLUSIONS

In this paper, after analyzing the four categories of complex event detection algorithms, a complex event detection model is proposed to mine valuable complex events from the huge RFID data. We improve the core complex event detection algorithms, a complex event model we can safely draw conclusion that it greatly decreases the difficulty of program development.

Our future work will focus on reducing the space complexity of the IM-RCEDA. The algorithm takes more space because of the use of expected event queues for the detection of non-spontaneous event in our algorithm. In particular, if it is implemented on the resource-restrained devices, this is necessary to consider. To further optimize this complex event detection model, we need to explore the completeness of the event construction operators.

REFERENCES


