Early Detection of Changes Using Event Sequence Extractor

● Koji Wakio ● Toru Yoshibayashi ● Naoki Akaboshi

(Manuscript received January 14, 2004)

Today's fast-changing business environment makes high demands on organizations for timely strategic decisions and business actions. If it was possible to analyze all the business events and promptly identify those conditions that require the attention of decision makers, the necessary adjustments and improvements could be made at the right time. In this paper, we present Event Sequence Extractor, which is our efficient solution for promptly detecting problematic sequences of business events that require action by decision makers. Event Sequence Extractor integrates the Interstage Navigator family of products, which provide faster Business Intelligence to enable decision-making in business process monitoring. The result is a more efficient and reliable business operation that leads to lower costs, protection from lost revenues, and a sustainable competitive advantage.

1. Introduction

The pace of change in the business world continues to increase, and in order to survive and flourish, organizations and companies must make strategic decisions and take business actions at the right time. However, today's business complexity makes it difficult to analyze all the business events that should be taken into account for these decisions and actions.

In a complex analysis, examination of only a single business event is usually not enough. On-Line Analytical Processing (OLAP) groups business events based on the values of "dimension" data and provides aggregations of "fact" data. For example, OLAP for a Claim Management process can promptly provide the number of complaints per product per day or the number of complaints per customer per week. This kind of aggregated information can be very useful for executive managers who need a global view of the situation.

However, for more complex tasks, for instance, the identification of customers who do not seem to be satisfied, an analysis of the detailed sequence of events is necessary. For example, in the Claim Management process, the detection of customers who made more than four complaints about a product within a week after its purchase or customers whose complaint interval is getting shorter indicates potential cases of customer dissatisfaction and loss that should be promptly handled. In such an analysis, an important feature is the sequential nature of the events. In this paper we use the term "sequence" to refer to events that are ordered due to a temporal relationship. Traditional relational databases provide no abstraction of ordering in the data model, so they cannot efficiently handle queries based on logical sequentiality in the data.

In this paper, we present Event Sequence Extractor, which is our solution for efficiently detecting pre-specified sequential patterns of events that represent situations that may affect a company's business performance. Event Sequence

Extractor integrates Fujitsu's Interstage Navigator family of products, which provide fast and effective Business Intelligence. By combining Event Sequence Extractor with OLAP and data mining, Fujitsu provides a complete Business Intelligence solution that supports real-time decision-making in response to changes in the business environment and optimization of the business processes of an entire organization.[1)]

This paper is organized as follows. In Section 2 we briefly review the data model and show how a pattern is specified in Event Sequence Extractor. In Section 3 we introduce the algorithm used in Event Sequence Extractor for efficiently detecting the sequential pattern of events. This algorithm dynamically constructs an on-memory data structure that stores the events of a given pattern by scanning the sequence of events only once. In Section 4 we outline some example applications of Event Sequence Extractor, and in Section 5 we show how Event Sequence Extractor integrates the Interstage family of products. Finally, Section 6 presents some concluding remarks.

2. Pattern specification using Event Sequence Extractor

Let R be a set of records composed of n attributes $\{a_1, a_2, ..., a_n\}$. By specifying attribute a_k as the Group Coordinate (GC) and a_l as the Order Coordinate (OC), where $1 \le k \le n$, $1 \le l \le n$, and $k \ne l$, R is partitioned according to the GC into a set of partitions and the partition records are sorted according to the OC. This results in a set of sequences of events that are ordered on the a_l values, where there is one sequence for each value of a_k.

For each sequence of events, Event Sequence Extractor searches for an m-pattern $[e_1 \Theta_{1,2} e_2 \Theta_{2,3} e_3 \ ... \ e_{m-1} \Theta_{m-1,m} e_m]$, where e_i represents an event and $\Theta_{i,i+1}$ represents the distance relationship between consecutive events e_i and e_{i+1}. This distance relationship specifies whether two consecutive events e_i and e_{i+1} of the pattern occur simultaneously, contiguously, or non-contiguously in the input sequence of events. Specifically:

1) $e_i = e_{i+1}$ means that both events occur simultaneously.
2) $e_i - e_{i+1}$ means that the events are contiguous; that is, e_{i+1} occurs immediately after e_i.
3) $e_i < e_{i+1}$ means that e_{i+1} occurs after but not immediately after e_i.

Each event e_i $(1 \le i \le m)$ of the pattern satisfies some intra-event conditions; that is, some conditions for the values of any of the attributes a_j of event e_i $(1 \le j \le n)$ that require only the examination of the event itself. Besides these intra-event conditions, there are also inter-event conditions; that is, conditions that specify how different events in the pattern are interrelated with specifications for the values of any of the attributes a_j for $1 \le j \le n$, $j \ne k$, where a_k is the GC. A special inter-event condition gives the window size ws of the pattern; that is, the maximum distance allowed between the first event e_1 and the last event e_m of the pattern: i.e., $ws \ge e_m.a_l - e_1.a_l$, where a_l is the OC.

Consider a simple example that searches for customers who complained more than twice about a product within a week after its purchase. As illustrated in **Figure 1**, for a given set of purchase and complaint events, by specifying customerID as the GC and specifying day as the OC, we logically have two sequences of events. The pattern of events $[e_1 < e_2 < e_3]$ that we want to detect has to satisfy the following conditions:

1) Intra-event conditions:
- e_1.type = "purchase"
- e_2.type = "complaint"
- e_3.type = "complaint"

2) Inter-event conditions:
- e_1.productID = e_2.productID = e_3.productID
- window size ws = 7 days

Because of a lack of space, more detailed descriptions of the data model and the query language supported by Event Sequence Extractor will not be presented here. However, they can be found in a previous publication.[2)] **Figure 2** illustrates the Event Sequence Extractor's Graph-

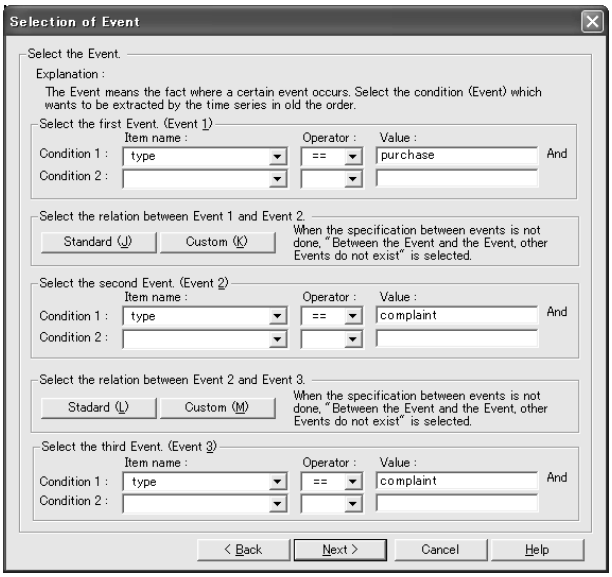
Grouping Coordinate (GC)

Ordering Coordinate (OC)

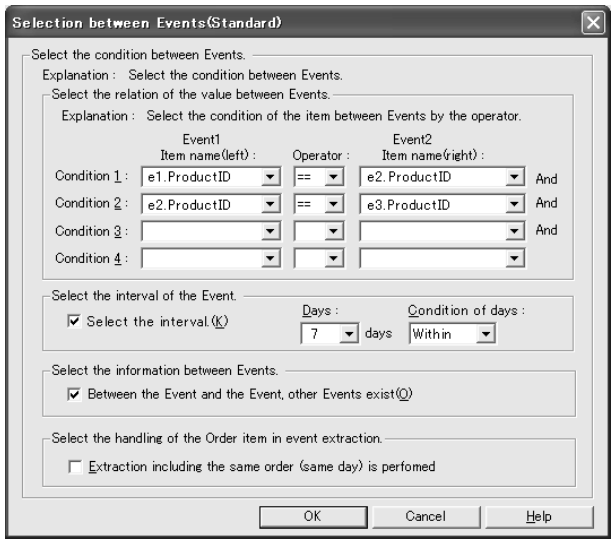
customer ID	day	type	product ID
customer1	10/1	purchase	product1
customer2	10/4	purchase	product1
customer2	10/4	purchase	product2
customer2	10/5	complaint	product1
customer2	10/10	complaint	product1
customer1	10/10	complaint	product1

customer1	10/1	purchase	product1
customer1	10/10	complaint	product1
customer2	10/4	purchase	product1
customer2	10/4	purchase	product2
customer2	10/5	complaint	product1
customer2	10/10	complaint	product1

Figure 1
Transforming a set of events into two sequences of events.



(a) Intra-event conditions



(b) Inter-event conditions

Figure 2
Pattern specification using Event Sequence Extractor's GUI.

ical User Interface (GUI), which makes it easy to specify the above pattern. The specification of the intra-event conditions and the inter-event conditions between e_1 and e_2 are shown in Figures 2(a) and 2(b), respectively.

In the above example, customer1 has complained about product1 only once. On the other hand, customer2 has complained twice about product1 within a week from its purchase but has not complained about product2. Therefore, Event Sequence Extractor will detect {customer2, product1} as the only potential case in which customer satisfaction is not achieved.

3. Pattern detection using Event Sequence Extractor

A naive approach for the search of events composing the complex patterns specified by Event Sequence Extractor would require multiple scans of the entire sequence of events. Event Sequence Extractor, however, can efficiently detect those patterns by scanning the event's sequence just once. Note that the patterns specified by Event Sequence Extractor can be very complex. They can specify conditions for multiple attributes, and those conditions can have inter-relationships with previous events in the pattern. Also, multiple events can occur simultaneously (i.e., have the

same OC value), and consecutive events of the pattern do not necessarily occur contiguously in the sequence. Therefore, previous approaches that extend traditional string-matching algorithms[3),4)] cannot be applied to searches for the more complex patterns of events supported by Event Sequence Extractor.

In order to detect these more-complex patterns, Event Sequence Extractor dynamically constructs an on-memory graph structure whose nodes represent the events that satisfy the conditions specified in the pattern. This algorithm is basically composed of the following steps:

1) Filtering of an event that satisfies the pattern conditions,
2) introduction of the filtered event as a node in the graph structure,
3) output of a detected pattern, and
4) deletion of unnecessary nodes from the graph structure.

Figure 3 presents a simplified pseudocode for the detection of an m-pattern over a sequence in which the events cannot have the same order value and therefore $\Theta_{ei,ei+1} \in \{-, <\}$. Except the first event e_1, which is specified by an intra-event condition (i.e., $e_1 = f_{intra1}(C_1)$, where C_1 is a constant), each event e_i can be specified by a conjunction of intra-event and inter-event conditions (i.e., $e_i = f_{intrai}(C_i)$ AND $f_{interi}(e_{i-1})$, where $2 \leq i \leq m$). A special inter-event condition that we will consider separately is the maximum distance allowed between two consecutive events e_i and e_{i+1}, which we denote as $\Delta_{i,i+1}$. For instance, when the OC attribute is the day attribute, then $\Delta_{i,i+1} \geq e_{i+1}.day - e_i.day$. When the distance $\Delta_{i,i+1}$ is not specified, we take $\Delta_{i,i+1}$ as the window size ws.

Due to space limitations in this paper, we will not present details of the extensions that are necessary for the more complex case in which multiple events can have the same order value (e.g., multi-

```
While there is a new input event r to process
   /* level 1 */
    If r satisfies f_intra1(C_1)
       Create a node e_1 that represents r
   /* levels 2 to m-1 */
    For ( i = 2, …, m-1 )
       Take the newest node e_i-1
       While r satisfies Δ_ei-1,ei AND Θ_ei-1,ei
          If r satisties f_intrai(C_i) AND f_intrai(e_i-1)
                /*f_intrai( ) and f_intrai( ) = TRUE if not explicitly specified in the pattern */
                Create a node e_i that represents r (if it does not exist yet), and link it to the node e_i-1
                Take the previous node e_i-1
   /* last level m */
    While the distance between r and the oldest node e_1 is longer than the window size ws
        Discard the oldest node e_1 and all nodes linked only to it
    Take the newest node e_m-1
    While r satisfies Δ_em-1,em AND Θ_em-1,em
          If r satisties f_intram(C_m) AND f_interm(e_m-1)
                /* f_intram( ) and f_interm( ) = TRUE if not explicitly specified in the pattern */
                Create a node e_m that represents r (if it does not exist yet) and link it to the node e_m-1
                Take the previous node e_m-1
Output all patterns composed by the subgraph rooted at node e_m
Discard the node e_m
```

Figure 3
Simplified pseudocode for pattern detection.

ple simultaneous purchases) and therefore $\Theta_{ei,ei+1} \in \{-, <, =, \leq\}$. Basically, all simultaneous input events must be checked once against each of the pattern events e_i and a check must be made to prevent the same input event *r* from appearing in different events e_i of the pattern.

4. Application examples

Event Sequence Extractor can efficiently detect changes in the business environment so that timely adjustments and improvements can be made by decision makers. As we illustrated for the Claim Management process, by detecting changes in customer behavior, prompt actions can be taken to improve customer satisfaction. Next, we present two examples in which Event Sequence Extractor has been applied to promptly detect sequences of events that were then appropriately acted on by the decision maker.

4.1 Brand switch analysis

Detecting recent sales trends and changes from purchase history data in early stages makes it possible to promptly adapt to those changes, or in unfavorable cases, to take actions to avoid and repair those changes. When a user switches between brands in the same category, the user is said to have made a "brand switch."

In order to detect brand switching, we use Event Sequence Extractor to search for the pattern of events in which 1) the sales of one brand increases and the sales of another brand decreases over a specified number of consecutive months and 2) the two brands in each of the brand pairs that satisfy the above conditions have a difference in sales below a given threshold value.

As a result, we detected brand-switching behavior for some brands of beer and also for some brands of bottled water by analyzing the sales data of a supermarket. A marketing group, for example, could use this information in a sales campaign to promote bottled water from which people have switched.

Interstage Navigator Explorer Server provides template queries for the described brand switch analysis. Users can easily modify and apply this template in their applications.

4.2 Clinical analysis

Electronic patient-records register information concerning symptoms, drugs, clinical tests, and results for patients. Analogous to business process analysis, clinical process analysis is necessary to, for example, determine the evolution of a disease, the effectiveness of a treatment, and the side effects of drug combinations. In this type of analysis, the clinical information is treated in sequential order and can be efficiently handled by Event Sequence Extractor.

For instance, it is not rare that serious side effects resulting from certain drug combinations become public knowledge months or even years after those drugs have been approved. As soon as this information is released, it is important that every patient who has taken one of those combinations for more than the period believed to be harmless is examined to check for side effects. We found that these patients can be efficiently identified using Event Sequence Extractor. We are now planning to apply data mining analysis to identify which of these patients have side effects caused by those combinations.

We have also used Event Sequence Extractor in the analysis of treatments adopted for some diseases. The order in which the medications were given and how they affected the test results were analyzed. We are now planning to extend this analysis to a broader range of diseases and medications.

5. Event Sequence Extractor and Fujitsu's Business Intelligence solution

Fujitsu's Interstage Suite is a broad family of modular and flexible software products. These products are used to build applications that accelerate business processes, maximize revenue, lower operating costs, improve customer service

and time-to-market, and help businesses react quickly to changing market requirements and customer trends.

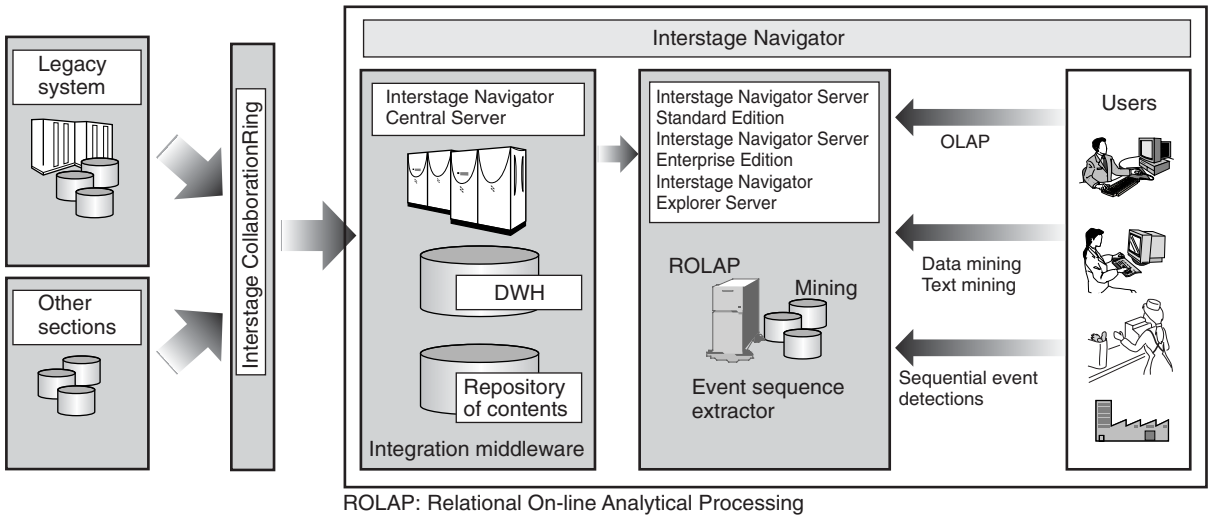
The term "Business Intelligence" covers a broad range of applications and technologies for gathering, storing, analyzing, and providing access to data in order to help enterprise users make better business decisions. Companies are starting to make use of Business Intelligence solutions to understand their businesses better, identify prospective customers and retain existing ones, and keep inventories and production orders under control. Business Intelligence solutions enable them to not only respond to the ever-increasing demands of today's business environment, but also to anticipate its trends.

As illustrated in **Figure 4**, Interstage Navigator is Fujitsu's Business Intelligence software platform and one of the core applications of Interstage Suite. Interstage Navigator was created to facilitate efficient and speedy analysis with OLAP technology. Interstage Navigator Central Server provides data warehouse (DWH) building functions such as aggregation of operational data and XML data importing. Interstage Navigator consists of interrelated components that closely collaborate to offer complete and integrated Business Intelligence solutions, for example, OLAP and data mining solutions, as well as efficient sequential event detection provided by Event Sequence Extractor.

Event Sequence Extractor makes use of Interstage Navigator data-access functions that can collect and provide operational data through Interstage Navigator. End users do not need to know Structured Query Language (SQL) or software programming to retrieve and filter the input data. All they need to do is drag-and-drop the desired dimensions into a report layout using a WYSIWYG interface. As we described in Section 2, Event Sequence Extractor also provides a GUI that makes it easy to specify a pattern of interest. By using the pattern detection algorithm presented in Section 3, Event Sequence Extractor can efficiently detect a specified pattern in the sequence of events provided by Interstage Navigator.

6. Conclusions

In this paper we presented Event Sequence Extractor, which is a Fujitsu solution for efficiently searching for patterns in sequences of events so that changes in the business environment can be quickly detected. Event Sequence Extractor can be used to monitor business processes. When



ROLAP: Relational On-line Analytical Processing

Figure 4
Fujitsu's Interstage Navigator.

Event Sequence Extractor detects a specified pattern, it can promptly notify the decision maker through an alarm and automatically perform further analyses. To enable timely decision-making, Event Sequence Extractor is integrated into the Interstage Navigator family of products, which provide Business Intelligence. This integration results in lower costs, protection from lost revenues, and a sustainable competitive advantage for a business organization.

We plan to provide a real-time monitoring framework for detecting changes in a future version of the Interstage family of products. Real-time monitoring functions will be implemented by applying Event Sequence Extractor to real-time data.

References

1) M. Nakagawa: Business Process Management with Web-Service Integration Technology. *FUJITSU Sci. Tech. J.*, **40**, 1, p.17-21 (2004).
2) L. Harada et al.: Event Analyzer: a Tool for Sequential Data Processing. Proceedings of 12th ACM International Conference on Information and Knowledge Management (CIKM2003), New Orleans, November 2003, p.172-174.
3) L. Harada: An Efficient Sliding Window Algorithm for Detection of Sequential Patterns. Proceedings of 8th International Conference on Database Systems for Advanced Applications (DASFAA2003), Kyoto, IEEE Computer Society, March 2003, p.73-80.
4) R. Sadri et al.: Optimization of Sequence Queries in Database Systems. Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems (PODS2001), Santa Barbara, May 2001, p.71-81.

Koji Wakio received the B.S. degree in Human Science in 1980 from Osaka University, Japan. He joined Fujitsu Limited, Numazu, Japan in 1980, where he has been engaged in development of Business Intelligence software products, beginning with DSS software products.

Toru Yoshibayashi received the B.S. degree in Electrical Engineering in 1994 and the M.S. degree in Electrical Engineering in 1996 from Waseda University, Japan. He joined Fujitsu Limited, Numazu, Japan in 1996, where he has been engaged in development of Fujitsu middleware products for Business Intelligence.

Naoki Akaboshi received the B.S. degree in Electrical Engineering in 1989 and the M.S. degree in Computer Science in 1991 from Kyoto University, Japan. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1991, where he has been engaged in research and development of database systems. His research interests include data mining and data stream analysis. He is a member of the Information Processing Society of Japan (IPSJ), the IEEE Computer Society, and the ACM.