An RDR-Based Approach for Event Data Analysis

Weisi Chen(✉) and Fethi Rabhi

School of Computer Science and Engineering,
University of New South Wales, Sydney, Australia
chenw@cse.unsw.edu.au, f.rabhi@unsw.edu.au

Abstract. Event data analysis is becoming increasingly of interest to academic researchers looking for patterns in the data, contributing to the emergence and popularity of a new field called "data intensive science". Unlike domain experts working in large companies which have access to IT staff and expensive software infrastructure, researchers find it harder to efficiently manage event processing rules by themselves especially when these rules increase in size and complexity over time. In this paper, we propose an event data analysis platform intended for non-IT experts that facilitates the evolution of event processing rules according to changing requirements. This platform integrates a rule learning framework called Ripple-Down Rules (RDR) operating in conjunction with an event pattern detection process invoked as a service. This solution is demonstrated on real-life scenario involving financial data analysis.

Keywords: Event-based data · Event processing · Event data model · Data intensive science · Ripple down rules

1 Introduction

An event is "anything that happens, or is contemplated as happening" [1] at a certain time. Examples of events in the real-world are very diverse and include financial trades and quotes, banking transactions (ATM, online, credit card use, etc.), news broadcast, aircraft movements, sensor outputs, updates in social media sites (e.g. Facebook), network communication message deliveries or computer systems management activities. We refer to large collections of event occurrences recorded in the form of data as "event data" or "event-based data". For many years, event data analysis has been conducted by the business sector for many purposes such as studying market trends, improving the efficiency of operational processes and gathering business intelligence.

To conduct event analysis tasks, domain experts have to rely on IT experts either to implement a bespoke program/service or to customize an event processing system (EPS) according to their needs. Because of constant changes in business needs and the environment, domain experts need to communicate their new requirements to IT experts all the time to update and maintain the event data analysis business logic. In terms of rule management, Luckham [2] claims that managing large sets of event processing rules is a challenge which has not yet been effectively tackled. In present

J.G. Davis et al. (Eds.): ASSRI 2013, LNBIP 177, pp. 1–14, 2014.
DOI: 10.1007/978-3-319-07950-9_1,

event processing systems, rule sets are normally very simple. When it comes to large and complex rule sets, one typical way an event data analysis process can be supported is illustrated in Fig. 1. On the one hand, the knowledge engineer manages the rule set and on the other hand, the IT expert implements the rules according the underlying software infrastructure. There could be a multitude of domain experts defining new rules so the knowledge engineer need to constantly cooperate with IT experts to manage event processing rules particularly when the size of the rule set becomes large.

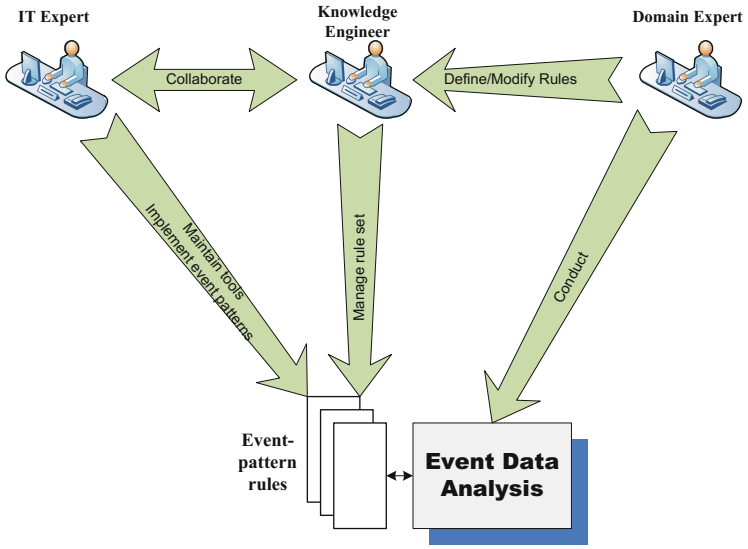

Fig. 1. Evolution of an event data analysis process.

More recently, event data analysis is becoming increasingly of interest to academic researchers looking for patterns in the data, contributing to the emergence and popularity of a new field called "data intensive science" [3]. Unlike domain experts working in large companies and having access to IT staff and expensive software infrastructures, researchers tend to conduct the analysis mostly by themselves using a range of data processing and statistical tools. Therefore, there is a need to enable analysis of event data by domain experts who have limited IT expertise and fewer resources available to them. Whilst the prime motivation of investigating solutions would be of interest to academic researchers, this research avenue would also be relevant to Small and Medium Enterprises (SMEs) looking for simple and cost-effective event data analysis solutions.

In this paper, we propose an approach to enable domain experts to manage event data analysis with little or no IT expert intervention by:

- Integrating a rule learning framework supporting incremental acquisition that enables domain users to define and add rules by themselves;

- Providing event pattern detection as a service (EPDaaS) in a way that allows domain users to conduct event processing without the concern about which event processing language/engine to use.

The rest of the paper is organized as follows. In Sect. 2, we summarize the related research efforts on event data analysis. Section 3 explains our proposed approach. In Sect. 4, we apply our approach to a real-life case study – data cleansing on financial data – to validate our proposed architecture. Section 5 concludes the paper and presents the direction of our future work.

2 Background

2.1 Basic Assumptions

There are several unique characteristics of event data; the primary ones are as follows:

- Time-based: Event data represent or record events, flowing in time-streams. Compared with normal data, event data has a temporal axis in the data schema. To be precise, every event data record is affixed with a timestamp as well as other attributes when it is created. Due to this feature, an event database can also be called time-series database.
- High flow rate and huge volume: Normally, new event data is continuously coming in to guarantee the timeliness of the data. Also, event data records are generated and stored in huge volumes, containing data for years.
- Immutable: On account of the high flow rate of event data, each record comes in and will never be modified.
- Referable: Any event record may be relevant to previous records and can be referred to other relevant records on some conditions such as within a certain time window, several days before or after the current event, etc.
- Influential: Any new event may generate a bunch of new events. For instance, financial market event data represent stimuli, market state transitions and outputs, each of which is issued followed by a chain of responses such as state changes and new outputs.

We view event data analysis as primarily the process of detecting patterns in the data and taking a number of actions accordingly. An example is described next followed by a review of existing work in this area.

2.2 Motivating Example

This example involves the analysis of Sirca's daily data [4] by academic researchers. Often, financial time-series analysis requires the computation of a company returns over a period of time. However, the value of these returns is affected by corporate actions such as the issuing of dividends. A dividend denotes a payment made by a firm out of its current or accumulated retained earnings to its owners, which gives rise to a fall of the stock price by the dividend amount on the executive date. Although the

information on corporate actions is available in the data, processing it is a non-trivial task due to the need to deal with duplicate dividend announcement records. Table 1 illustrates different cases for handling this problem. In most cases (e.g. Table 1(a)), duplicate events are just recording the same dividend announcement for multiple times. The initial logic of duplicate dividend detection is:

If there are two Dividend events issued at the same event time (Date)
Then it is a duplicate so delete the first one

After a while, domain users may find this logic gives wrong decisions and actions in a new case (e.g. Table 1(b)), so they have to change the rule into:

If there are two Dividend events with the same event time (Date) and Div ID
Then it is a duplicate so delete the first one

Soon later, due to another new case (e.g. Table 1(c)), this rule has to be again modified into:

If there are two Dividend events with the same event time, the same Div ID, and the payment status of these two events are both marked 'approved' ('APPD'), and the values of 'Div Delete Marker' are both 0
Then it is a duplicate so delete the first one

Table 1. Different cases in duplicate dividend detection

(a) Case 1 – simple duplicate dividend records

#RIC	Date	Type	Div Ex Date	Div Amt	Div ID	Div Delete Marker	Payment Status
ABC	12/08/2012	Dividend	11/09/2012	0.07	7885540	0	APPD
ABC	12/08/2012	Dividend	11/09/2012	0.07	7885540	0	APPD

(b) Case 2 –although these two dividends are issued at the same time (Date), the Div IDs are different which indicates that these are two different dividends rather than a duplicate.

#RIC	Date	Type	Div Ex Date	Div Amt	Div ID	Div Delete Marker	Payment Status
ABC	12/08/2012	Dividend	11/09/2012	0.07	7885540	0	APPD
ABC	12/08/2012	Dividend	11/10/2012	0.07	7926058	0	APPD

(c) Case 3 –the first dividend is proposed (PROP) and has been deleted (Delete Marker = 1), which is considered to be an out-dated record; the second dividend is an update so this case is not a case of a "duplicate dividend,, to be detected.

#RIC	Date	Type	Div Ex Date	Div Amt	Div ID	Div Delete Marker	Payment Status
ABC	12/08/2012	Dividend	11/09/2012	0.08	7885540	1	PROP
ABC	12/08/2012	Dividend	11/09/2012	0.07	7885540	0	APPD

In real-world event processing, rules are never "perfect" as there are always exceptions against existing rules. As an extension of the example above, the defined data cleansing rules built for Australian stock data may be applied to other country's stock data, e.g. the German stock data, with similar but not exactly the same logic, which requires additional modifications to the rules which can be even more complicated than the example above. This also explains why IT experts and knowledge engineers are always needed to evolve the program or system; in many cases, they have to develop a number of various applications for different or even slightly different event processing tasks.

2.3 Related Work

According to the book ***Event Processing in Action*** [5], event processing is "computing that performs operations on events". The main operations on events include:

- Filter: reducing the overall set of events to be processed by an event-processing system to those events that are actually relevant for the given processing task, e.g., removing erroneous or incomplete data.
- Transformation: changing event instances from one form to another, including translation, splitting, aggregation, composition, etc.
- Pattern Detection: finding a particular pattern by examining a collection of events.

An event processing system is typically a dedicated platform that provides abstractions (event processing logic) for operations on events that are separated from the application logic (event producers and event consumers). This can reduce the cost of development and maintenance. Event processing logic is expressed by event processing languages (EPLs). A stream of event data is fed into the EPS and the event processing language code is executed, then a list of actions is generated (Fig. 2). Table 2 lists different types of event processing languages. These languages all have advantages and disadvantages that reflect the usual tradeoffs between simplicity and expressiveness. Therefore, whatever EPL/EPS the domain user uses, there might always be limitations, and switching is by any means troublesome. This is to say that the performance of the event data analysis largely depends on the selected EPL/EPS.

In the book ***The Power of Events*** [1], Luckham have defined event processing rules as "the foundation for applications of complex event processing (CEP)". Although some work has realized that user customization of the system according to

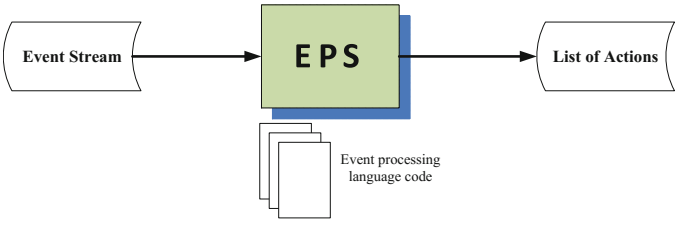

Fig. 2. An event data analysis process.

Table 2. Types and examples of event processing languages

Language type		Language/product [5, 6]
Stream-oriented (SQL extension)		Aleri, Esper, CQL
Rule-oriented	Production rules	DROOLS fusion, TIBCO BusinessEvents
	ECA rules	Amit, IBM WebSphere Business Events
	Logic programming	Etails, Prova
Imperative		MonitorScript, Netcool Impact policy language
Event processing framework or library for a general purpose programming language		Progress Apama

their needs is an important criterion for EPS [7], most efforts are focusing on the operational issues, e.g. event processing language expressiveness and performance [8], rather than event processing rule management. Most work on EPS does not support user-driven event data analysis no matter how good the language expressivity is. Among the very limited discussion on event processing rule management, there are two important insights:

- Reuse of existing event patterns is of great importance for efficiency [9].
- The idea of rule templates are suitable for EPS for completing rules as well as decoupled "building blocks" of rule logic [10].

We agree with these insights. However, the use of rule templates is not sufficient to eliminate rule re-building efforts when modifying the rule. In most cases, the same IT expert and/or knowledge engineer who built the existing rule set (rule base) are needed to re-build the entire rule set. Besides, most literature on event processing rule management tends to focus on building each single rule and disregard the management of the rule base as a whole. It has been proved in the knowledge acquisition community that when the size of the rule base becomes huge, it would be very difficult to maintain the rule base, as any modification of the rule base may cause the system collapse [11]. Thus in most existing EPSs, as rules may be closely associated with each other, it is difficult to keep track of changes effectively.

3 Proposed System

3.1 Architecture

In this paper, we are proposing a novel approach to facilitate incremental event processing rule definition, which is desirable to eliminate event processing rule-rebuilding, to enhance the reuse of existing event processing rules, to keep track of event processing changes, to simplify rule management process, and to avoid rule base collapse. Our architecture enables domain experts to manage rules, define simple event patterns or build event patterns upon existing ones. In addition, the event data analysis system can be incrementally enhanced by domain users. In this case, IT

experts will only play one role in the data analysis process: to define and deploy complex event patterns.

The proposed architecture illustrated in Fig. 3 has two components – a rule-based system and an event pattern detection as a service (EPDaaS). In order to achieve "incremental management" and eliminate clashes, we utilize a framework called Ripple Down Rules (RDR) in the rule-based system to route the event processing logic, which can organize and maintain the rule base more effectively [11]. Unlike other rule management systems, RDR is an error-driven, incremental rule acquisition framework, which enables domain experts to evolve the system solely by themselves and eliminates the risk of corrupting the rule base because all existing rules are never changed, which reserve the existing decision logic of the event processing. When errors occur, RDR allows users to capture the characteristic of the new case as an "exception", and add a new rule to quickly recover the degraded performance. The case that prompted the addition of a rule is called a cornerstone case, which is stored along with the rule and is used to be compared with new cases by the domain experts. It has been proved that the whole processing of adding a rule including checking cornerstone cases takes only a couple of minutes [12].

In this architecture, the EPDaaS has a service interface that has the ability to invoke any underlying EPS (using any EPL) to detect event patterns. When invoked with an event pattern type and a reference to an event stream, EPDaaS will select one available/suitable EPS, run the corresponding EPL code and finally return event pattern occurrences in this event stream, abstractions or aggregations of these

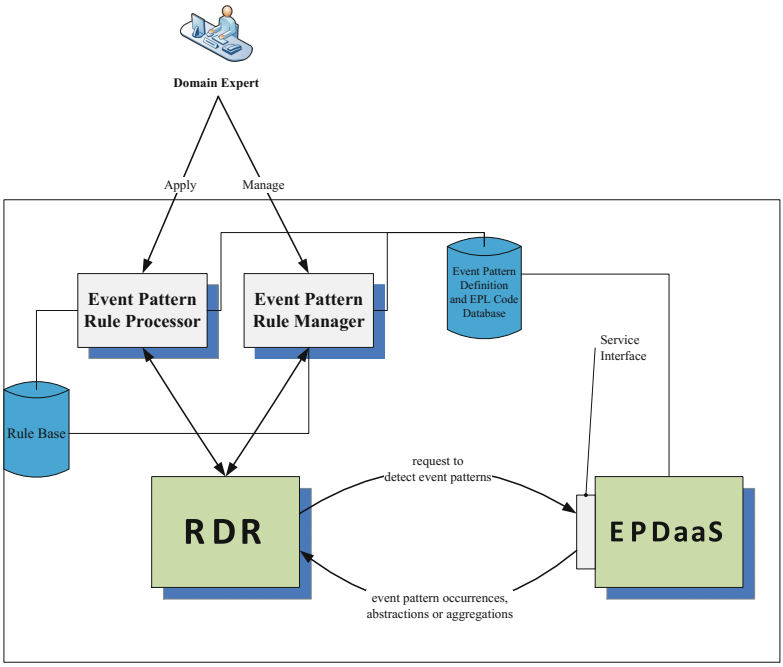

Fig. 3. Proposed EP-RDR architecture

occurrences. The role of the RDR component is to allow incremental definition of rules based on the presence of event patterns – each event "situation" is represented as an event pattern in a rule. The architecture provides the link between the RDR and the EPDaaS components, i.e. the RDR component sends a request to detect an event pattern, and the EPDaaS responds with event pattern occurrences back to the RDR. Finally, RDR will generate a list of actions on the original event stream, which will be inspected by the domain expert. One of the advantages of this architecture is that for different event data analysis, the only things that have to be changed are the rule set and the choice of the EPS invoked by the EPDaaS component; the RDR component, however, does not have to be changed. IT experts are not involved in rule management but managing the EPL code in the database. We call this architecture Event-Processing RDR (EP-RDR).

3.2 Overview of Event Processing Rules

Generally, an event processing RDR rule can be:

If
    an ***event pattern*** *occurs*
Then
    case action*;*
    inference action: *go to rule* ***a***
Else
    inference action: *go to rule* ***b***

If an error is found in a new case, domain experts can use the Event Pattern Rule Manager to associate a new event-based rule to the rule that causes the error. Table 3 illustrates the evolution of duplicate dividend rule in EP-RDR. Table 3(a) is a sample of the initial rule set, in which Rule 4 handles a duplicate dividend case in Table 1(a). The domain user executes this rule set and finds out an error occurs on Rule 4 due to

Table 3. (a) A sample of the original rule base for event data cleansing. (b) The evolved sample of rule base for event data cleansing.

(a)

Rule No.	Event pattern ID	Action	Inf. action (true)	Inf. action (false)
...	...	...	...	...
4	4	Delete duplicate Div	to rule 5	to rule 5
...	...	...	...	...
7	7	Report it as an error	exit	exit

(b)

Rule No.	Event pattern ID	Action	Inf. action (true)	Inf. action (false)
...	...	...	...	...
4	4	Delete duplicate Div	to rule ***8***	to rule 5
...	...	...	...	...
7	7	Report it as an error	exit	exit
8	***8***	***Retrieve the last action***	to rule ***5***	to rule ***5***

the case in Table 1(b); then the domain user can add a new rule (Rule 8) to take the attribute *Div ID* into account. The new rule is attached to Rule 4, whose Inf. action (true) is modified (Table 3(b)). Also, the case in Table 1(b) – the cornerstone case – is stored together with Rule 8. Table 4 shows a summary of the event pattern types used in this example.

Table 4. Event pattern types used in rule base example

Event pattern ID	Event pattern definition
4	Another dividend event is before the current dividend with the same date
7	Missing an End Of Day event on the effective date of a capital change event
8	***Two dividend events have different Div ID***

Figure 4 illustrates how an event processing rule is added. Before defining a rule, an event pattern must be newly defined or selected from the pre-existing event patterns via an event pattern definition GUI. Note that every time a new pattern is defined, it is saved so that all defined event patterns can be re-used when building new rules. After defining or selecting the event pattern, the domain expert can use a rule definition GUI to define the rule and associate the pattern with the condition and action of the rule.

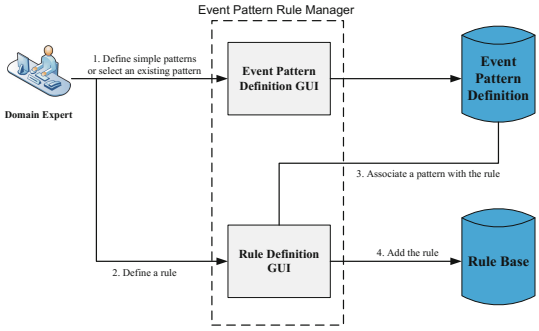

Fig. 4. Event processing rule management.

3.3 A Dynamic View of EP-RDR

Figure 5 demonstrates the business process associated with event-pattern RDR rule execution. For each single rule being processed on an event, the RDR engine sends a request to the EPDaaS with three inputs:

- (1) the key of the event pattern to be detected (the key is used as an index in the event pattern definition and EPL code database);
- (2) a reference to the event stream to be searched from, along with a reference event to be used as the original point to detect the particular event pattern occurrence(s).

For each occurrence sent back by EPDaaS, RDR will then assert the action according to the rule and go to the next rule according to the inference defined in the rule. After processing all rules on all events, a log with a list of actions and a track of

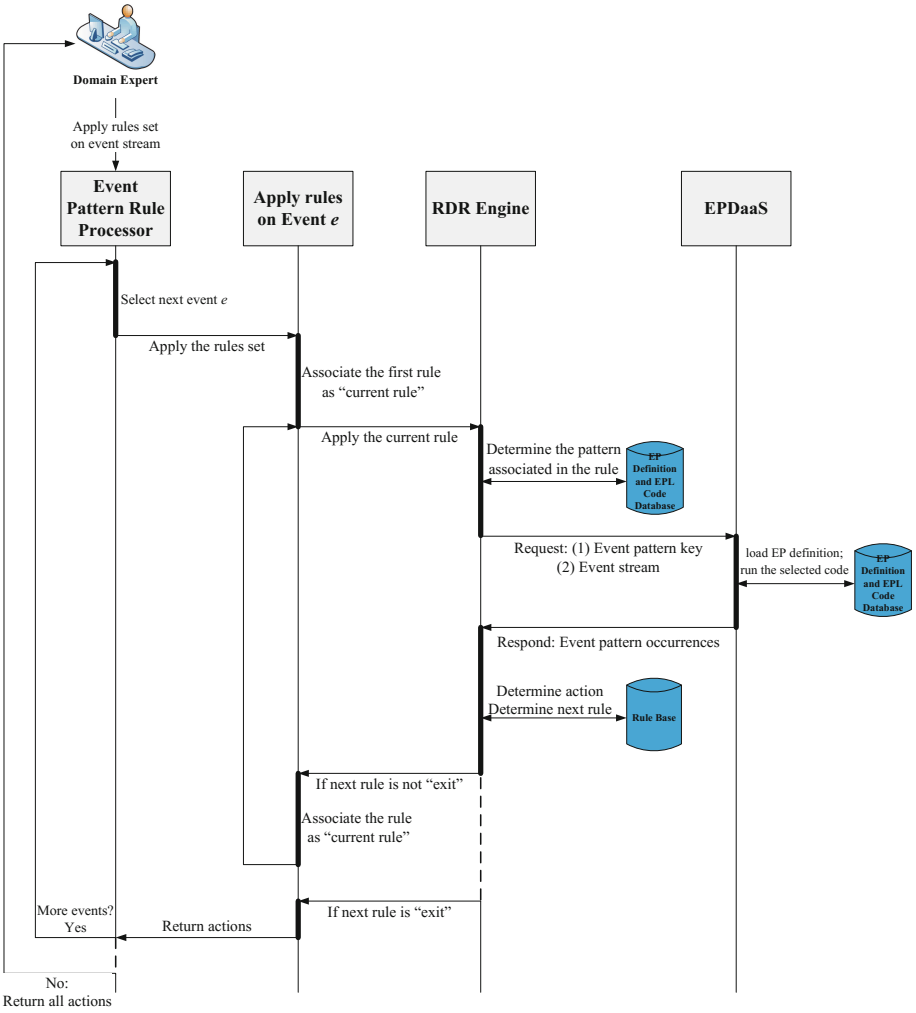

Fig. 5. Apply rules on events.

all "decisions" made during execution will be generated for the domain expert's inspection. Figure 6 shows what the domain expert has to do after the rule execution process: inspect the result; if an error is found due to a case, the domain expert can add a new rule, where the cornerstone case is stored.

To enable communication between the RDR engine and the EPDaaS, we have designed a new event data model (see Fig. 7), which allows the architecture to be independent from any particular event processing language.

There are two sub-types of events, i.e. simple events and complex events. A complex event is generated by an event pattern occurrence that matches a particular event pattern. Each event pattern occurrence involves a number of events, which can be simple events as well as complex events. All events have a number of attributes

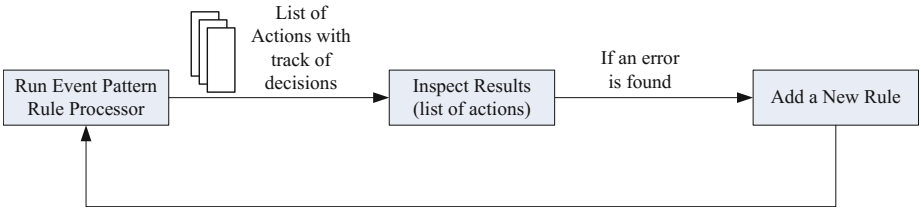

Fig. 6. Rule evolution.

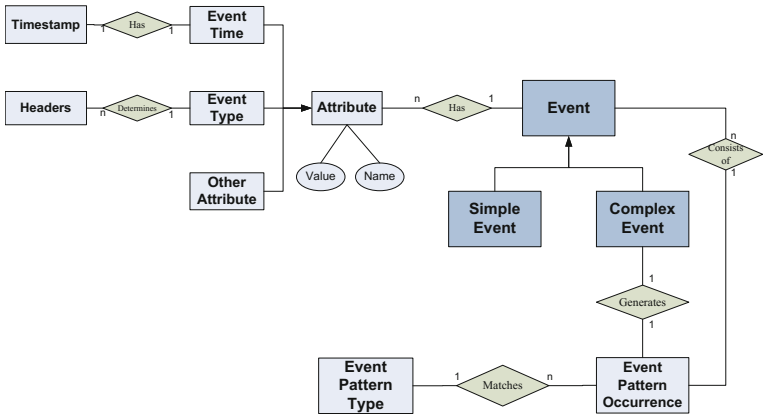

Fig. 7. Event data model.

including "event time" that has a timestamp and "event type" that determines the headers of attributes. For more details about this data model, see [13].

4 Application and Validation

4.1 Implementation

We have implemented a prototype of the proposed architecture in Java with a simple GUI which allows users to define RDR rules as well as simple event pattern types graphically (using JGraph). In this prototype, the underlying event pattern rule processor is a Java program that implements the rule processing logic. Among many structures of RDR, rather than using a tree structure in most RDR applications [14, 15], we apply the linked-production structure, where all rules are at the same level and can be reused. Therefore, event processing logic is separated from the inference logic, and modifications can be made merely on inference logic (which rule to be processed next) rather than on the content of rules (event patterns or actions) when adding rules. This method can reduce rule redundancy and protect existing rules for the sake of rule maintenance. The example previously shown in Table 3 essentially uses the linked-production RDR structure.

The EPDaaS used in this prototype is also a dedicated Java program that implements some simple pattern detection functions. All event pattern definitions/code and rules are stored in PostgreSQL relational database separately.

4.2 Case Study – Event Data Cleansing

Due to its unique characteristics listed in Sect. 2.1, event data cleansing is an important part of event data analysis. Working with a domain expert in the financial area, the case study involves conducting data cleansing in the context of Sirca daily stock data. As preliminary work, we developed a bespoke program which implemented the data cleansing process. However, every time the domain expert has asked for the business logic to be changed, it took several days to modify the program, as changes on one particular rule normally affect some other rules. In total, 8 modifications took more than a year to complete.

We then repeated the process using the prototype. Firstly, we defined 7 initial event patterns using the Event Pattern Definition GUI, each representing one type of event data quality issue respectively: missing value in an end-of-day (EOD) event, missing value in a dividend (Div) event, missing value in a capital change (CC) event, duplicate Div events, duplicate CC events, missing an EOD event on dividend effective date (DED), missing an EOD event on capital change effective date (CCED). Then the domain expert defined totally 7 rules (Table 5) using the Rule Definition GUI, each being associated with one of the defined event patterns.

The domain expert worked iteratively by executing the current rule set, inspecting the resulting list of generated actions and decisions, and adding new rules. Table 6 shows the rule base after the 8 iterations, which only took several hours. The performance is as good as the bespoke program. Note that at each stage of the evolution, each existing rule is true in terms of all previously encountered cases. Compared to the previous approach, the domain expert was able to update the rule base simply and neatly without assistance by the IT expert with significantly less amount of time.

Table 5. Initial data cleansing rule set

Rule no.	Event pattern	Action	Inf. action (true)	Inf. action (false)
1	Missing value in an EOD event	Fill in with previous value	To rule 2	To rule 2
2	Missing value in a Div event	Report missing value	To rule 3	To rule 3
3	Missing value in a CC event	Report missing value	To rule 4	To rule 4
4	Duplicate Div events	Delete the former one	To rule 5	To rule 5
5	Duplicate CC events	Delete the former one	To rule 6	To rule 6
6	Missing an EOD event on DED	Report it as an error	To rule 7	To rule 7
7	Missing an EOD event on CCED	Report it as an error	Exit	Exit

Table 6. Evolved data cleansing rule set

Rule no.	Event pattern	Action	Inf. action (true)	Inf. action (false)
1	Missing value in an EOD event	Fill in with previous value	To rule 2	To rule 2
2	Missing value in a Div event	Report missing value	To rule 3	To rule 3
3	Missing value in a CC event	Report missing value	To rule 4	To rule 4
4	Duplicate Div events	Delete the former one	To rule ***8***	To rule 5
5	Duplicate CC events	Delete the former one	To rule ***11***	To rule 6
6	Missing an EOD event on DED	Report it as an error	To rule 7	To rule ***14***
7	Missing an EOD event on CCED	Report it as an error	Exit	To rule ***15***
8	Different Div ID	***Retrieve the last action***	To rule ***5***	To rule ***9***
9	Status is not "APPD"	***Retrieve the last action***	To rule ***5***	To rule ***10***
10	Delete Marker is not 0	***Retrieve the last action***	To rule ***5***	To rule ***5***
11	Different CC ID	***Retrieve the last action***	To rule ***6***	To rule ***12***
12	Status is not "APPD"	***Retrieve the last action***	To rule ***6***	To rule ***13***
13	Delete Marker is not 0	***Retrieve the last action***	To rule ***6***	To rule ***6***
14	There is a corresponding EOD event but no trading	***Report it as an error***	To rule ***7***	To rule ***7***
15	There is a corresponding EOD event but no trading	***Report it as an error***	Exit	Exit

5 Conclusion and Future Work

In this paper, we have proposed an architecture called EP-RDR for building and maintaining user-driven event data analysis, in which a Ripple-Down Rule (RDR) framework is integrated with an event pattern detection service. Our architecture is designed to optimize the process of rule management from the perspective of managing the whole rule base and to leverage the power of existing Event Processing Systems.

We have implemented a prototype of the architecture, and validated the architecture by applying it to event data cleansing in the context of Sirca daily stock data. The implementation has proved successful in evolving 15 rules, which is not a very huge rule base size. Besides, the implementation has been validated with financial data, even though potentially most event data analysis in other domains can be defined and maintained easily with the proposed architecture. In our future research, we will focus on managing larger rule bases and apply the approach in various domains.

Acknowledgement. We would like to thank the Smart Services Cooperative Research Centre in Australia for sponsoring our research project and Sirca for providing financial data used in the case study. We would also thank Prof. Paul Compton for his valuable advice on the RDR technique.

References

1. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison Wesley Professional, Reading (2002)

2. Luckham, D.: What's the Difference Between ESP and CEP? (2006). http://www.complexevents.com/?p=103
3. Rabhi, F., Yao, L., Guabtni, A.: ADAGE: a framework for supporting user-driven ad hoc data analysis processes. Computing **94**, 489–519 (2012)
4. Sirca. http://www.sirca.org.au/
5. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications Co., Stamford (2011)
6. Cugola, G., Margara, A.: Processing flows of information: from data stream to complex event processing. ACM Comput. Surv. **44**, 1–62 (2012)
7. Chandy, K., Schulte, W.: Event Processing: Designing IT Systems for Agile Companies. McGraw-Hill, New York (2010)
8. Hinze, A., Sachs, K., Buchmann, A.: Event-based applications and enabling technologies. In: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, Nashville, Tennessee (2009)
9. Sen, S., Stojanovic, N.: GRUVe: a methodology for complex event pattern life cycle management. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 209–223. Springer, Heidelberg (2010)
10. Obweger, H., Schiefer, J., Suntinger, M., Kepplinger, P., Rozsnyai, S.: User-Oriented Rule Management for Event-Based Applications. In: Proceedings of the Fifth ACM International Conference on Distributed Event-Based System, New York, USA (2011)
11. Richards, D.: Two decades of ripple down rules research. Knowl. Eng. Rev. **24**(2), 159–184 (2009)
12. Compton, P., Peters, L., Edwards, G., Lavers, T.G.: Experience with ripple-down rules. Knowl. Based Syst. **19**, 356–362 (2006)
13. Rabhi, F.A., Chen, W., Perry, R., Yao, L., Natarajan A.: A new data model for representing events and event patterns. Internal Report, Service Oriented Computing Research Group, School of Computer Science and Engineering, University of New South Wales (2013)
14. Kang, B.H., Compton, P., Preston, P.: Multiple classification ripple down rules: evaluation and possibilities. In: The Ninth Banff Knowledge Acquisition for Knowledge Based Systems Workshop (1995)
15. Prasad, K.H., Faruquie, T.A., Joshi, S., Chaturvedi, S., Subramaniam, L.V., Mohania M.: Data cleansing techniques for large enterprise datasets. In: Annual SRII Global Conference (SRII), pp. 135–144 (2011)